Practica 1: Gramáticas independientes del contexto

1 Objectivo

El objetivo de la práctica consiste en implementar un conjunto de algoritmos, que dada una *gramática independiente del contexto,* producen una segunda gramática *equivalente* a la primera que cumple uno de los siguientes criterios:

1. todos los símbolos son *generadores*.
2. todos los símbolos son *alcanzables*.
3. no contiene *producciones-ϵ* (o contiene una sola producción-ϵ cuya parte izquierda es el simbolo inicial).
4. no contiene producciónes *unitarias*.
5. todos los criterios anteriores.
6. está en forma normal de Chomsky (Opcional. En la práctica 2 implementarás un analizador sintáctico cuya entrada debe estar en forma normal de Chomsky).

2 Especificaciónes

Un ejemplo de gramática independiente del contexto.

$$
\begin{array}{ll}
SN \rightarrow Det\ Name\ Adj & Name \rightarrow \text{república} \\
Det \rightarrow \text{la} & Adj \rightarrow \text{independiente} \\
Name \rightarrow \text{gramática} &
\end{array}
$$

Una *gramática independiente del contexto* es una cuadrupla $\langle V, T, P, S\rangle$ donde

1. V es un conjunto de variables, también llamados símbolos *no terminales*. En el ejemplo los símboloss no terminales son SN, Det, $Name$ y Adj.
2. T es un conjunto de símbolos *terminales* que serán el vocabulario sobre el que se forme el lenguaje. En el ejemplo son las palabras “la”, “gramática”, “república” y “independiente”.
3. P es un conjunto de *producciones*, cada una de las cuales es un par formado por un simbolo no terminal y un cadena de símbolos (terminales o no terminales). En el ejemplo es cada una de las reglas $SN \rightarrow Det\ Name\ Adj$, $Det \rightarrow$ la, etc.
4. S es un simbolo no terminal llamado *inicial*. En el ejemplo es elsímboloo SN.

Una *cadena* es una secuencia de símbolos. Una cadena *deriva* otra cadena si esta puede obtenerse reescribiendo símbolos que aparecen en la parte izquierda de alguna producción por la parte derecha. Por ejemplo de la cadena $Det\ Name$ se deriva las cadenas "la gramatica" y "la república" y se escribe

$$\begin{aligned} Det\ Name &\xrightarrow{*} \text{la gramatica} \\ Det &\xrightarrow{*} \text{la} \\ Name &\xrightarrow{*} \text{gramatica} \\ SN &\xrightarrow{*} Det\ Name\ Adj \\ &\xrightarrow{*} \text{la gramática independiente} \\ SN &\xrightarrow{*} \text{la república independiente} \end{aligned}$$

El conjunto de cadenas que se pueden derivar del simbolo de una gramática es el *lenguaje* generado por una gramática. En el ejemplo el lenguaje generado contiene dos cadenas: "la gramática independiente" y "la república independiente". Una gramática es *equivalente* a otra si generan el mismo lenguaje.

Un símbolo es *generador* si de él se deriva una cadena que solo contiene símbolos terminales. Un símbolo es *alcanzable* si hay alguna cadena que se derive del simbolo inicial que lo contiene. Una *producción-*ϵ es una producción cuya parte derecha es la cadena vacía, normalmente representada por ϵ. Una producción es *unitaria* si su parte derecha solo contiene un símbolo y este es un símbolo no terminal.

3 Ejemplo

Obtener una gramática equivalente cuyos símbolos son todos generadores. Un símbolo es *generador* si de él se deriva una cadena que solo contiene no terminales. Por ejemplo considera la gramática que contine las siguientes producciónes

$$\begin{aligned} S &\to AB \mid AC \mid bA \\ A &\to a \\ B &\to b \\ C &\to D \end{aligned}$$

donde a y b son los símbolos terminales, S, A, B y C son los símbolos no terminales y S es el simbolo inicial. Claramente todos símbolos terminales, a y b, son generadores (se derivan la cadena que los contiene a ellos mismos). De A y B se derivan respectivamente a y b y por lo tanto son generadores. De S se deriva ab y ba y por lo tanto también es generador, pero de C y D no se deriva ninguna cadena de terminales y por lo tanto son símbolos *inutiles*. Por lo tanto la gramática que contiene las prodicciónes

$$\begin{aligned} S &\to AB \mid bA \\ A &\to a \\ B &\to b \end{aligned}$$

es equivalente a la anterior, el leguaje generado por ambas contiene las cadenas ab y ba, y solo contiene simbolos generadores.

El algoritmo seguido para obtener esta gramática es el siguiente:

1. Todo símbolo no terminal es generador.
2. Si una producción es de la forma $A \rightarrow \alpha$ y todos los símbolos de α son generadores entonces A también es un generador.
3. La nueva gramática contiene solo los símbolos generadores y aquellas producciones $A \rightarrow \alpha$ donde todos los símbolos en α son generadores.

Implementación utilizando un lenguaje funcional (Scala):

```
def simplifyGeneratores() : Grammar = {
  var generators:    Set[Symbol]     = this.terminals
  var productions:   Set[Production] = null
  var oldgenerators: Set[Symbol]     = null
  do{
    oldgenerators = generators
    productions   = this.productions filter (p => p.right subsetOf generators)
    generators    = generators union ( productions map (p => p.left) )
  }while(oldgenerators != generators)
  var (terminals, noTerminals) = Grammar.symbolsPartition(generators)
  return Grammar(terminals,noTerminals,initial,productions)
}
```

Nota: `symbolsPartition` es un metodo que divide un conjunto de simbolos en terminales y no terminales.

Implementación utilizando un leguaje imperativo (Java):

```
public Grammar simplifyGeneratores(){
    Set<Symbol> generators = new HashSet<Symbol>(this.terminals);
    boolean newgenerators  = true;
    while(newgenerators){
        newgenerators = false;
        for(Production p: this.productions){
            if(generators.containsAll(p.right())){
                newgenerators = generators.add(p.left()) || newgenerators;
            }
        }
    }
    Set<Production> productions = new HashSet<>(this.productions.size());
    for(Production p: this.productions){
        if(generators.containsAll(p.right()))
            productions.add(p);
    }
    Tuple2<Set<Terminal>,Set<NoTerminal>> tuple = Grammar.symbolsPartition(generators);
    Set<Terminal>   terminals   = tuple._1;
    Set<NoTerminal> noTerminals = tuple._2;
    return new Grammar(terminals,noTerminals,initial,productions);
}
```

La versión 8 de Java también incluye la posibilidad de utilizar algunas caracteristicas funcionales, perminitiendo reescribir el codigo anterior de la siguiente forma:

```
public Grammar simplifyGeneratores(){
    Set<Symbol>        generators = new HashSet<Symbol>(this.terminals);
    Set<Symbol> newGenerators;
    do{
        newGenerators = this.productions.stream()
                    .filter(p -> generators.containsAll(p.right()))
                    .map(p -> p.left())
                    .collect(Collectors.toSet());
    }while( generators.addAll(newGenerators) );

    Set<Production> productions =
        this.productions.stream()
            .filter(p -> generators.containsAll(p.right()))
            .collect(Collectors.toSet());

    Tuple2<Set<Terminal>,Set<NoTerminal>> tuple = Grammar.symbolsPartition(generators);
    Set<Terminal>   terminals   = tuple._1;
    Set<NoTerminal> noTerminals = tuple._2;
    return new Grammar(terminals,noTerminals,initial, productions);
}
```

Podéis encontrar una descripción más detallada de este algoritmo y otros para cada uno de los otros criterios descritos en el objetivo de la práctica en *"Introducción a la teoría de autómatas, lenguajes y computación"* (pag. 217-231).

4 Plazo de entrega

La practica debe ser entregada antes de la clase de prácticas (11:30) del 10 de marzo.