Práctica de Modelos de Computación

Ingeniería de Sistemas de Información

Ignacio Arvilla de Caralt

1. Diseño de los métodos

1) **Método addNonTerminal():**

Los elementos no terminales que se vayan añadiendo a la gramática se guardaran en un ArrayList llamado “nonTerminals”. Como los elementos no terminales los representamos con letras mayúsculas, si el carácter introducido no es una letra mayúscula, se lanza la excepcion CYKAlgorithmException(). En caso contrario se añade al conjunto.

```
public void addNonTerminal(char nonterminal) throws CYKAlgorithmException {
    if(!Character.isUpperCase(ch: nonterminal)){
        throw new CYKAlgorithmException();
    }else if(nonTerminals.contains(o: nonterminal)){
        throw new CYKAlgorithmException();
    }else if(Character.isUpperCase(ch: nonterminal)){
        nonTerminals.add(e: nonterminal);
    }
}
```

2) **Método addTerminal():**

Los elementos terminales añadidos se van guardando en un ArrayList denominado “terminals”. Estos elementos los representamos con letras minúsculas por lo que si el carácter de entrada es una letra mayúscula o un digito, lanzara la excepcion CYKAlgorithmException().

```
public void addTerminal(char terminal) throws CYKAlgorithmException {
    if(!Character.isLowerCase(ch: terminal)){
        throw new CYKAlgorithmException();
    }else if(terminals.contains(o: terminal)){
        throw new CYKAlgorithmException();
    }else if(Character.isLowerCase(ch: terminal)){
        terminals.add(e: terminal);
    }
}
```

3) **Método setStartSymbol():**

Se pasa un elemento no terminal, si este no pertenece al conjunto de los ni ¡terminales lanza la excepción. Si pertenece se asigna a la variable startSymbol.

```
public void setStartSymbol(char nonterminal) throws CYKAlgorithmException {
    if(nonTerminals.contains(o: nonterminal)){
        startSymbol = nonterminal;
    }else if(!nonTerminals.contains(o: nonterminal)){
        throw new CYKAlgorithmException();
    }
}
```

4) **Método addProduction():**

Este método lo que hace es utilizar un HashMap “productions” para almacenar las producciones ya que en estas debemos relacionar el elemento no terminal con la producción. Entonces se comprueba si el elemento no terminal pertenece al conjunto y se tratan las producciones de longitud 1 y 2 por separado, ya que en Forma Normal de Chomsky son “AB” o “a”.

Además de rellenar el HashMap, se hace un HashMap inverso “inverseProductions” para que a la hora de programar el algoritmo CYK en el método isDerived() sea más fácil acceder a las producciones y rellenar las casillas con los correspondientes símbolos no terminales. Si el HashMap original tomaba como claves los no terminales y como valores listas de Strings correspondientes a las producciones de cada elemento. El inverso toma como claves producciones y como valores listas de caracteres no terminales.

5) **Método isDerived():**

Para empezar, se crea una lista de listas de conjuntos que representará la tabla del algoritmo.

Para rellenar la primera fila de la tabla con los elementos no terminales de cada carácter se llama al HashMap inverso creado en el método addProduction().

Para rellenar el resto de la tabla se inicia un bucle. En cada iteración, se considera una ventana deslizante que se desplaza diagonalmente en la tabla. Se combinan los no terminales de las subcadenas de la palabra dentro de la casilla de la ventana y se buscan los no terminales correspondientes a esa producción formada en el HashMap inverso. En el caso de que se encuentren se añaden a la casilla correspondiente en la tabla.

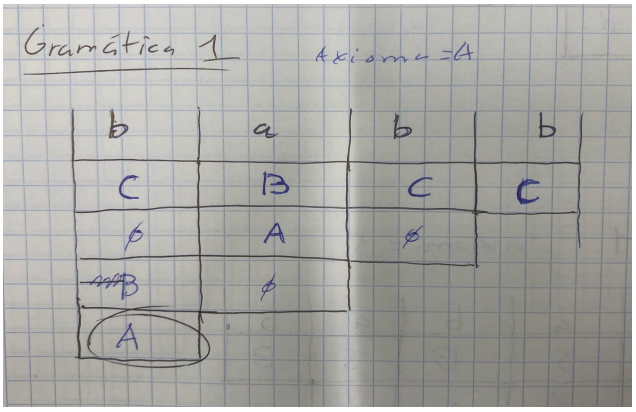
Después de completar el bucle se comprueba si es axioma “startSymbol” se encuentra en la última casilla de la tabla y entonces devuelve true o false.

Finalmente, las estructuras de datos utilizadas son listas de caracteres para los elementos no terminales y terminales, y HashMaps para las producciones. En el caso del algoritmo se utiliza la lista de listas para representar la tabla del algoritmo.

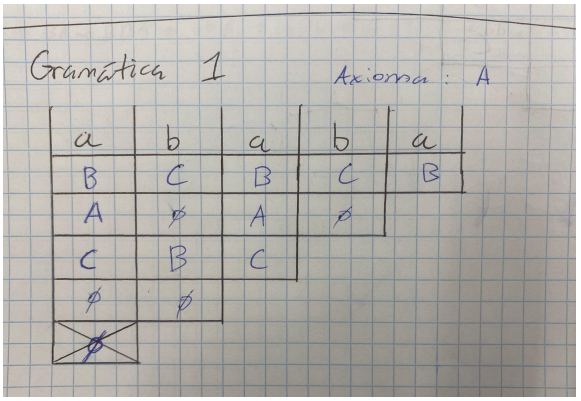
2. Algoritmo CYK aplicado

En esta sección se demuestra que el algoritmo CYK está correctamente programado y devuelve el resultado correcto sobre la pertenencia de la palabra al lenguaje de cada gramática.

Gramática 1

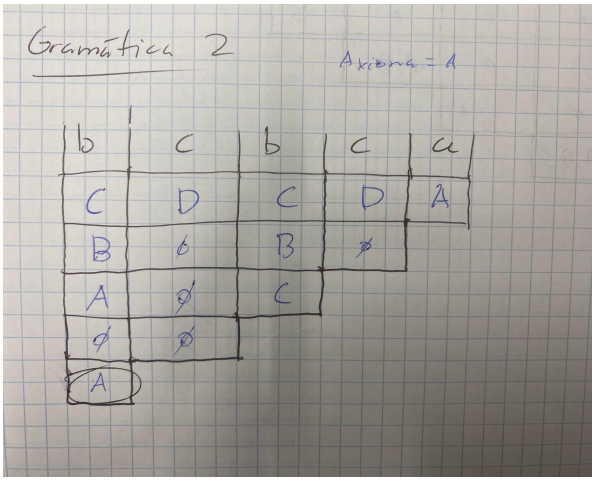


válida
palabra: babb

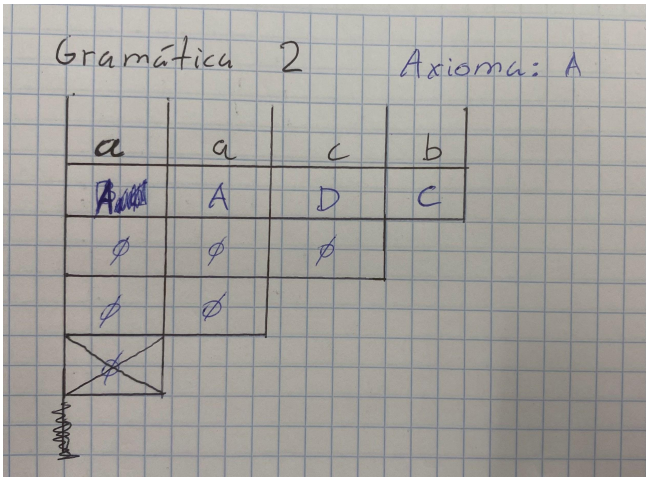


no válida
palabra: ababa

Gramática 2

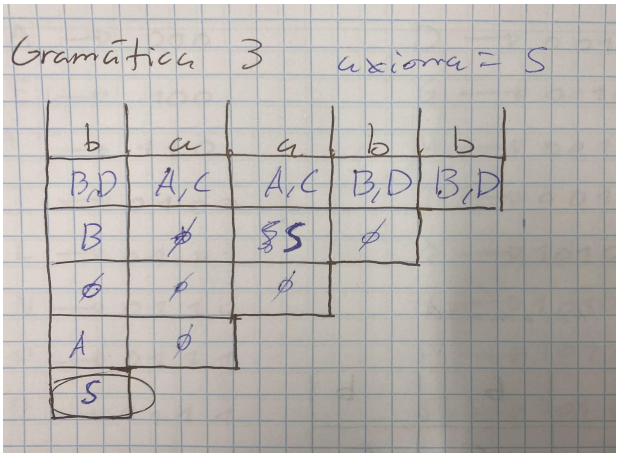


válida
palabra: bcbca

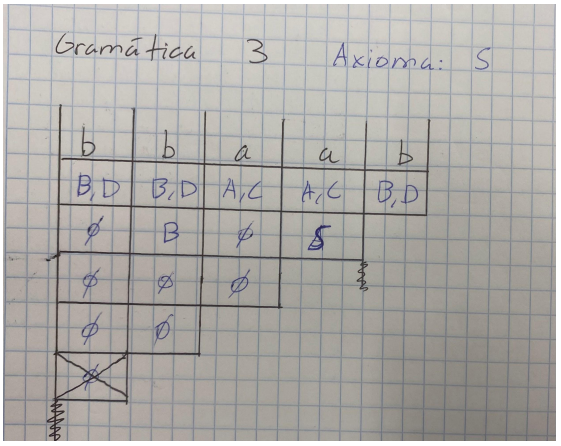


no válida
palabra: aacb

Gramática 3

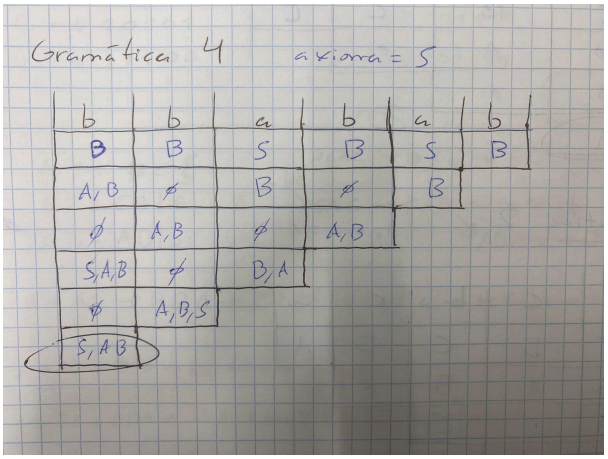


válida
palabra: baabb

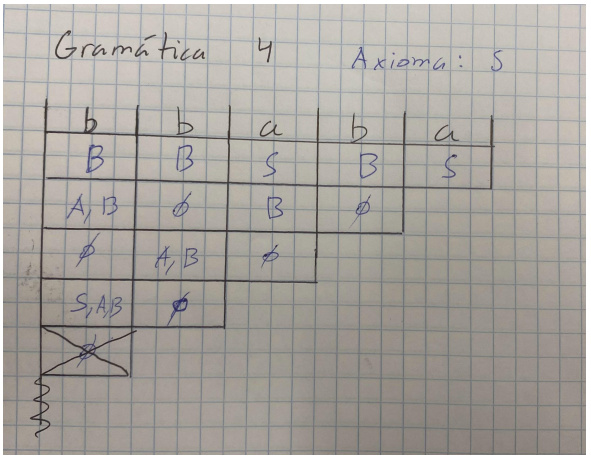


no válida
palabra: bbaab

Gramática 4

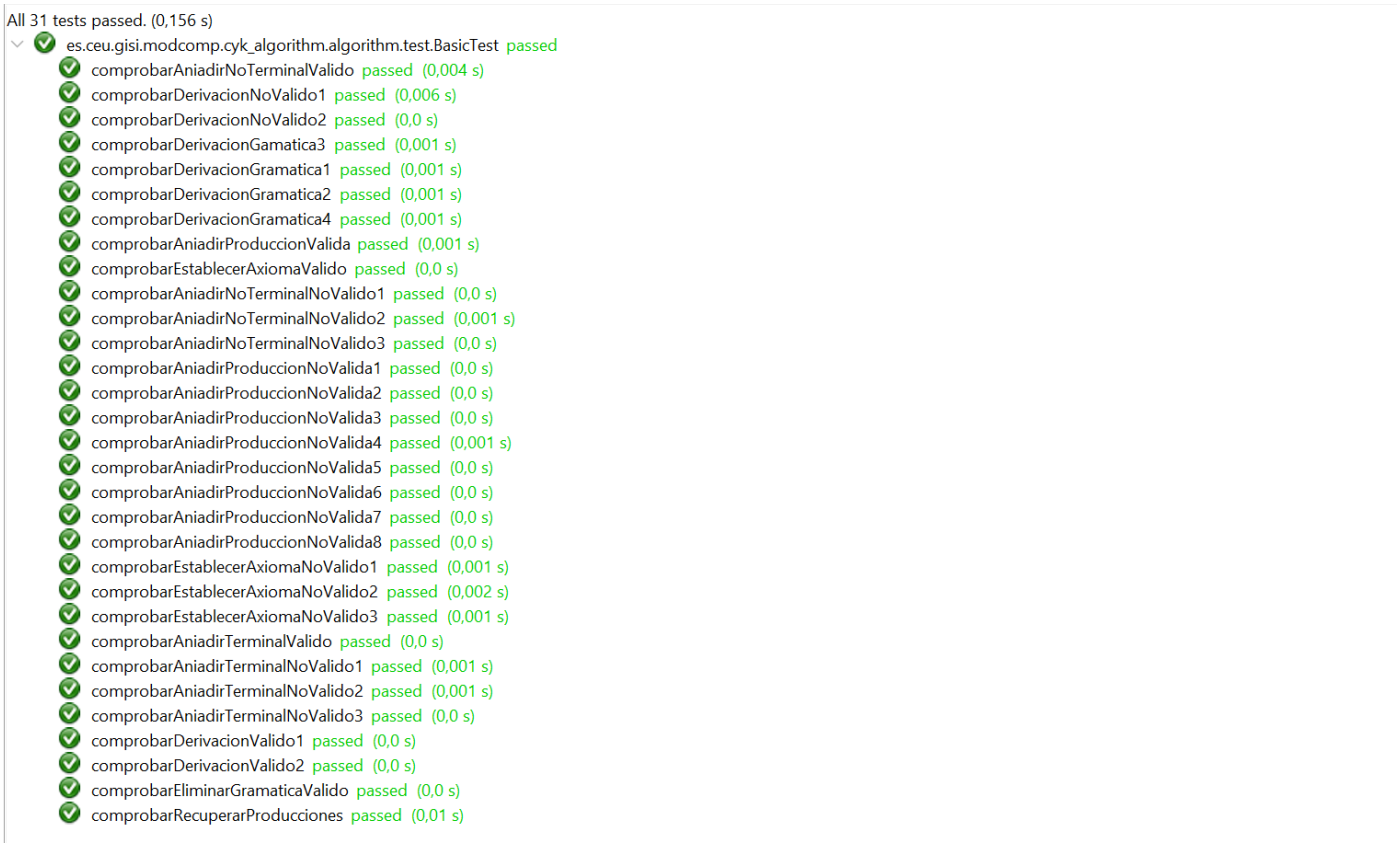


válida
palabra: bbabab



no válida
palabra: bbaba

3. Completitud de los tests



4. Conclusiones

En cuanto al tiempo invertido en la práctica, han sido unas 20 horas de trabajo en total durante dos semanas. Esto es debido a que esta práctica me ha parecido bastante complicada en cuanto al nivel de programación que se necesita, esto ha hecho que me lleve más tiempo en hacer el algoritmo, sin embargo, pienso que he aprendido mucho y me ha parecido una de las prácticas con la que más orgulloso estoy con mi trabajo.