Contenido

**No se encontraron entradas de tabla de contenido.**

## Clase DescendantLL V0.1

De momento solo llama a la clase ll con el path de la gramática a convertir.

## Clase LL V0.1

**Atributos:**

**private HashMap<String, ArrayList<String>> grammarWithActions:** representa la gramática con acciones semánticas y con no terminales con índices. Los índices del mapa son los antecedentes y guarda todas las produccionesdicho antecedente.

**private HashMap<String, ArrayList<String>> gramar:** representa la gramática sin acciones semánticas y con no terminales sin índices. Los índices del mapa son los antecedentes y guarda todas las produccionesdicho antecedente.

**private String path:** ruta del archivo que guarda la gramatica

**private HashMap<String,ArrayList<String>> head:** guarda las cabeceras de los antecedentes. Los índices del mapa son los antecedentes.

**private HashMap<String,ArrayList<String>> headPrime:** guarda las cabeceras prima de los antecedentes. Los índices del mapa son los antecedentes.

**private HashMap<String,ArrayList<String>> nexts:** guarda los siguientes de los antecedentes. Los índices del mapa son los antecedentes.

**private HashMap<String,ArrayList<String>> directors:** guarda los directores de las producciones. Los índices del mapa son los antecedentes más la producción.

**private String axioma:** axioma de la gramática.

**private Integer[][] table:** tabla usada para poder reconocer la cadena de entrada.

**private ArrayList<String> terminals:** lista de terminales.

**private ArrayList<String> noTerminals:** lista de no terminales.

**Constructor:**

**public LL( String path):**

Recoge el path del archivo que contiene la gramática, inicializa los mapas y las listas, añade todos los antecedentes a la lista de terminales, crea una tabla con tantas filas como no terminales y tantas columnas como terminales, calcula las cabeceras, los siguientes y los directores.

**Métodos:**

**private void grammarWithoutActions():**

Crea una nueva gramática sin acciones y sin índices en los no terminales.

Por cada antecedente en la gramática, por cada producción, eliminamos las acciones y los índices de la producción y lo añadimos a una lista. Cuando no hay más producciones se guarda en un mapa la lista usando como índice el antecedente.

**private String removeActions(String production):**

Elimina las acciones semánticas y los índices de los no terminales de una producción.

Se divide la producción en símbolos y por cada símbolo si no empieza con “{”, si es un no terminal sin índice se añade a la nueva String que será la nueva producción, si no se añade una subcadena del símbolo desde el inicio has que aparecen dígitos. Devuelve la nueva producción.

Para saber si un elemento es un digito:

<http://lineadecodigo.com/java/%C2%BFes-ese-caracter-un-numero/>

**private Integer getNumberIndex(String symbol):**

Dado un símbolo en forma de String devuelve la posición donde comienzan los dígitos.

Se pasa el símbolo a un array de caracteres y por cada uno si es un digito devuelve la posición. Si se terminan los símbolos y no hay dígitos devuelve el tamaño de la cadena.

**private void readFile():**

Lee línea a línea el archivo con la gramática y la guarda.

Por cada línea la divide en producciones y antecedentes, si el contador es 0 se asigna el antecedente al axioma, se divide la String con todas las producciones en otras String con cada producción, se crea una lista y se guarda en el mapa con el antecedente como índice, se añaden todas las producciones a la lista y se suma uno al contador.

Para leer archivos con lambda:

<https://stackoverflow.com/questions/979932/read-unicode-text-files-with-java>

**public void directors():**

Calcula los directores de todas las producciones.

Por cada antecedente, por cada producción calcula el director.

**public void director(String production,String antecedent):**

Calcula los directores de una producción.

Separa la producción en símbolos,

Se inicializa la lista donde se van a guardar los directores.

Si el primer carácter del primer símbolo está en mayúscula:

Si la cabecera contiene landa se insertan los símbolos de la cabecera prima en la lista y los directores de la producción a partir del símbolo si no están ya en la lista.

Si no, inserta los símbolos de la cabecera si no están ya en la lista.

Si el primer símbolo es landa se insertan los siguientes del antecedente de la producción

Si no se añade el símbolo si no está ya.

Por último, se añaden los directores al mapa con el antecedente más la producción como índice.

Para distinguir entre mayúscula y miniscula:

<https://brainly.lat/tarea/7075675>

**public ArrayList<String> directorAux(String production,String antecedent):**

Calcula los directores de una producción que no se vayan a guardar en el mapa de directores. Es igual al anterior solo que en vez de guardar el resultado en un mapa lo devuelve.

**public void heads():**

Calcula las cabeceras de todos los antecedentes.

Por cada antecedente calcula su cabecera.

**public void head(String antecedent):**

Calcula la cabecera de un antecedente.

Si la cabecera no se ha calculado ya obtiene las producciones del antecedente e inicializa la lista donde va a guardar los símbolos de las cabeceras.

Por cada producción:

Se obtiene la primera letra, Si está en mayúscula:

Divide la producción en símbolos y si ya existe la cabecera prima del antecedente le añade los símbolos a la lista, si no calcula la cabecera prima y se los añade a la lista.

Si no divide la producción en símbolos y añade el primero a la lista

Añade la lista al mapa de cabeceras con el antecedente como índice.

**public void headPrime(String antecedent):**

Calcula la cabecera prima del antecedente.

Funciona igual que **head(String antecedent)**  pero no añade lambda a la lista.

**public void nexts():**

Calcula los siguientes de todos los antecedentes.

Por cada antecedente calcula los siguientes.

**private void ifNoRepeatInsert(ArrayList<String> oldElements,ArrayList<String> newElements ):**

Asigna los valores de la seguanda lista a la primera si esta no los contiene ya.

Por todos los nuevos elementos si la primera lista no los contiene los añade.

**public void next(String antecedente):**

Calcula los siguientes del antecedente.

Se inicializa la lista que va a contener los siguientes, si el antecedente es el axioma se añade “$” a la lista.

Se busca todas las producciones que contengan el antecedente, por cada producción:

Se divide la producción en símbolos, si a la derecha del antecedente en la producción no hay ningún símbolo y el antecedente de la producción no es igual al antecedente para el cual está calculando los siguientes:

Si se ha calculado el siguiente del antecedente de la producción se añaden a la lista los siguientes del antecedente de la producción.

Si no se calcula los siguientes del antecedente de la producción y se añaden si no están ya.

Si el elemento después del antecedente en la producción está en mayúscula:

Se calcula la posición del siguiente símbolo del antecedente en la producción, si la cabecera prima de dicho símbolo se ha calculado ya se insertan los símbolos en la lista si no están ya.

Si no se calcula y se añaden a la lista si no están ya.

Si el siguiente contiene lambda en sus producciones y el siguiente del siguiente existe:

Si el antecedente de la producción no es igual al antecedente para el cual estamos calculando los siguientes y los siguientes del antecedente de la producción se han calculado se añaden a la lista si no están ya.

Si no se calculan y se añaden si no existen ya.

Si el siguiente contiene landa:

Si el siguiente del siguiente empieza con mayúscula calcula los siguientes de este y los añade a la lista sin no están ya.

Si no añade el siguiente del siguiente a la lista si no está ya.

Si no se añade el siguiente a la lista si no está ya.

Por último, guarda la lista en el mapa de siguientes usando como índice el antecedente.

**public ArrayList<String> productionsWith(String noTerminal):**

Busca todas las producciones donde aparece el no terminal dado.

Por cada antecedente, por cada producción, si la producción contiene el no terminal la añaden a la lista que se va a devolver. Por último, se devuelve la lista con las producciones.

**private boolean containLambda(String symbol):**

comprueba si alguna de las producciones del símbolo dado es lambda.

Se obtiene las producciones del símbolo y por cada una si es lambda se devuelve true, si ninguna es landa se devuelve false.

**private boolean containLambda(ArrayList<String> symbols):**

Comprueba si la lista contiene lambda.

Por cada símbolo si uno es igual a lambda devuelve true, si ninguno es lambda devuelve false.

**private String getAntecedent(String production):**

Busca el antecedente de la producción dada.

Por cada antecedente si una de sus producciones contiene la producción que se ha dado devuelve el antecedente, sino devuelve null.

**private void groupTerminals():**

Busaca y almacena todos los terminales.

Por cada antecedente, por cada producción divide la producción en símbolos y por cada símbolo si no empieza por mayúscula se añade a la lista de terminales.