Teoría de Lenguajes
Curso 2020
Laboratorio 2 – Gramáticas Libres de Contexto

El propósito de este laboratorio es trabajar con gramáticas libres de contexto de una manera práctica, para lo cual se propone escribir programas en el lenguaje Python.

Cada programa recibe una entrada y despliega una salida. En cada programa se deberá construir una gramática para reconocer la entrada en función de la descripción del programa y ejemplos de entradas y salidas. En algunos casos, además, se deberá construir una rutina que recorra el árbol de *parsing* y genere determinada salida.

Para realizar el laboratorio se imparte un archivo lab2.zip que contiene ejemplos de entradas y salidas, un ejemplo de programa implementado y el script ejecutar.py que ejecuta cada programa con las entradas y compara las salidas obtenidas con las esperadas.

Entrega

La entrega es el **30 de junio a las 23:55** y se realizará mediante un formulario en EVA que estará disponible próximo a la fecha de entrega.

Se debe entregar:
- los archivos programa{1,2,3}.py con los programas implementados
- un archivo integrantes.txt con las cédulas de los integrantes del grupo (una por línea) sin puntos ni dígito de verificación.

Programas

A continuación se describen los programas a implementar. Cada programa es un módulo **Python 3** ejecutable que recibe los nombre de los archivos de entrada y salida como parámetros.

Todos los programas tienen lo siguiente en común:
- Analizan la entrada con una gramática libre de contexto
- En caso de que la entrada no pertenezca al lenguaje generado por la gramática, el programa devuelve: **NO PERTENECE**
- En caso de que el conjunto de terminales de la gramática no cubra el conjunto de símbolos de la entrada el programa devuelve: **NO CUBRE**
- En caso de que la entrada pertenezca al lenguaje generado por la gramática el programa devuelve **PERTENECE** o una salida construida a partir del árbol de derivación de la entrada.

programa0.py (implementado)

Este programa verifica que la tira de entrada pertenezca al lenguaje: {$a^n b^n$ con n>0}

Este programa se encuentra implementado como ayuda para la implementación de los programas restantes.

La gramática que se entrega implementada (con la sintáxis de NLTK) es:

```
S -> 'a' S 'b' | 'a' 'b'
```

Ejemplo de entrada 1:

```
aabb
```

Ejemplo de salida 1:

```
PERTENECE
```

Ejemplo de entrada 2:

```
aab
```

Ejemplo de salida 2:

```
NO PERTENECE
```

Ejemplo de entrada 3:

```
aa11
```

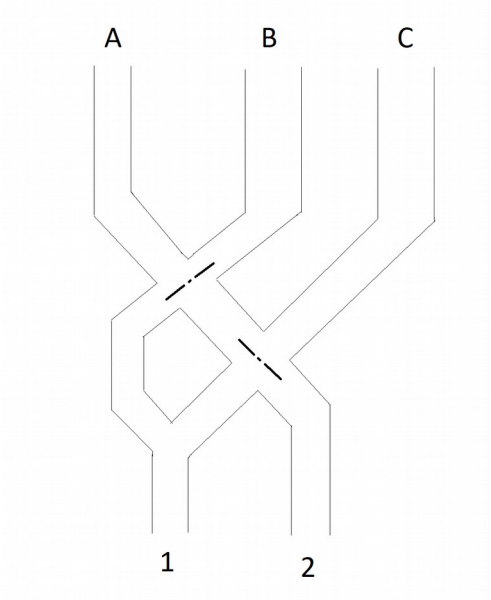
Ejemplo de salida 3:

```
NO CUBRE
```

programa1.py

La gramática generada para resolver este programa se exige que sea una gramática lineal derecha.

Considere el siguiente juego:



Una ficha es ingresada por **A**, **B** o **C**. Las dos llaves hacen que la ficha sea enviada a la derecha o a la izquierda según la posición en que se encuentran (la ficha cae en la dirección paralela a la llave). Cuando una ficha pasa por una llave, ocasiona que esta cambie de posición, por lo tanto, la próxima ficha que la encuentre será enviada en dirección opuesta.

El programa debe verificar, dada una secuencia de fichas ingresadas por las entradas, si la última salió por la **salida 1.**

Sugerencias:

Para resolver este problema se sugiere construir primero el autómata que permite validar tiras del lenguaje y luego deducir la gramática.

Ejemplo de entrada 1:

```
C
```

Ejemplo de salida 1:

```
NO PERTENECE
```

Ejemplo de entrada 2:

```
ACB
```

Ejemplo de salida 2:

```
PERTENECE
```

Ejemplo de entrada 3:

```
123
```

Ejemplo de salida 3:

```
NO CUBRE
```

programa2.py

Este programa debe reconocer las tiras que representan una expresión regular sobre el alfabeto {a,b}. En caso de que la entrada sea válida (es decir, que sea una expresión regular), se debe desplegar como salida la misma expresión, modificando los '|' por '.' y viceversa.

El lenguaje generado por la gramática no puede tener espacios, pero sí paréntesis. Las clausuras de Kleene siempre serán aplicadas sobre una expresión parentizada, y la concatenación será siempre explícita, utilizando el operador '.'.

Ejemplo de entrada 1:

```
(a)|(b.a)*
```

Ejemplo de salida 1:

```
(a).(b|a)*
```

Ejemplo de entrada 2:

```
a*
```

Ejemplo de salida 2:

```
NO PERTENECE
```

Ejemplo de entrada 3:

```
ab
```

Ejemplo de salida 3:

```
NO PERTENECE
```

Ejemplo de entrada 4:

```
72
```

Ejemplo de salida 4:

```
NO CUBRE
```

programa3.py

En este programa consideraremos dos lenguajes utilizados para representar texto que puede tener secciones en negrita o cursiva.

En ambos lenguajes, los únicos caracteres permitidos son las letras (en mayúscula y minúscula pero sin tildes), el espacio, el punto, la coma y los paréntesis curvos. **Cualquier otro caracter no es válido**.

Lenguaje 1

- Negrita: se marca utilizando el caracter '*' al rededor del texto en negrita.
- Cursiva: se marca utilizando el caracter '_' al rededor del texto en cursiva.

Lenguaje 2

- Negrita: se marca encerrando el texto en la tag \textbf{}
- Cursiva: se marca encerrando el texto en la tag \emph{}

En el segundo lenguaje, los paréntesis curvos son caracteres especiales, por lo que deberán ser precedidos por el caracter '\'.

El programa deberá validar mediante una gramática las tiras pertenecientes al Lenguaje 1, y en caso de que la tira de entrada pertenezca a dicho lenguaje, deberá desplegar como salida la misma tira pero escrita en términos del Lenguaje 2.

Ejemplo de entrada 1:

```
Este texto esta en *negrita*, y este esta en _cursiva_. Tambien podemos usar (parentesis).
```

Ejemplo de salida 1:

```
Este texto esta en \textbf{negrita}, y este esta en \emph{cursiva}. Tambien podemos usar \(parentesis\).
```

Ejemplo de entrada 2:

```
No se puede anidar los *at_ribut_os*
```

Ejemplo de salida 2:

```
NO PERTENECE
```

Ejemplo de entrada 3:

```
Esta entrada es invalida porque tiene el siguiente simbolo ;
```

Ejemplo de salida 3:

```
NO CUBRE
```

Referencias

[1] https://www.nltk.org/book/ch08.html
[2] https://docs.python.org/3/