Teoría de Lenguajes
Curso 2019
Laboratorio 2 – Gramática Libre de Contexto

Este laboratorio consiste en utilizar los mecanismo de GLC y árboles de NLTK [1] en el lenguaje de programación Python [2] para abordar un conjunto de programas a resolver.

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 **18 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. Se permite importar funciones entre los módulos para reutilizarlas pero no crear módulos auxiliares.

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 restantes programas.

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

Este programa verifica que la entrada sea un número binario divisible entre 3.

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

Sugerencias:
Para resolver este problema se sugiere construir primero el autómata y luego deducir la gramática.

Observe que al consumir la entrada de izquierda a derecha, si se consume un 0, el número consumido hasta el momento es multiplicado por 2, y al consumir un 1 es multiplicado por dos y al resultado se le suma 1.

Teniendo en cuenta lo anterior es posible discriminar que ocurre al consumir un elemento de la entrada en función del resto de la división entre 3, teniendo en cuenta que todo número natural puede escribirse como 3q+r con r = 0,1 o 2.

Ejemplo de entrada 1:

```
11
```

Ejemplo de salida 1:

```
PERTENECE
```

Ejemplo de entrada 2:

```
101
```

Ejemplo de salida 2:

```
NO PERTENECE
```

Ejemplo de entrada 3:

```
123
```

Ejemplo de salida 3:

```
NO CUBRE
```

programa2.py

Este programa recibe como entrada una expresión aritmética de sumas y productos de números naturales (en base 10) y en caso de que la entrada sea correcta despliega como salida la misma expresión conmutando todas las sumas y productos.

Para simplificar asuma que la expresión contiene paréntesis y no tiene ambigüedades de precedencia entre sumas y productos. Por ejemplo no se probará con entradas como “10+6*21”, sí serán válidas las entradas 10+(6*21) y (10+6)*21.

Sugerencias:
Para resolver esta parte le puede ser útil considerar un parser distinto al provisto en el programa de ejemplo (ej. *LeftCornerChartParser*).

Ejemplo de entrada 1:

```
87
```

Ejemplo de salida 1:

```
87
```

Ejemplo de entrada 2:

```
12+(8*523)
```

Ejemplo de salida 2:

```
(523*8)+12
```

Ejemplo de entrada 3:

```
43**8
```

Ejemplo de salida 3:

```
NO PERTENECE
```

Ejemplo de entrada 4:

```
(5*3)+(2*(52+8))
```

Ejemplo de salida 4:

```
((8+52)*2)+(3*5)
```

programa3.py

Este programa reconoce un sublenguaje de JSON y lo transforma a un sublenguaje de YAML.

El sublenguaje de JSON reconocido considera los siguientes elementos:

- Objetos JSON formados por una secuencia de pares (clave,valor)
- Las claves son string
- Los valores pueden ser:
  - String
  - Naturales
  - Objetos JSON
- Los string son secuencias de caracteres de la “a” a la “z” (mayúsculas y minúsculas) y guión bajo.
- Los naturales son secuencias de dígitos (0,1,2,3,4,5,6,7,8,9)
- La entrada se encuentra en una única línea (sin fines de línea) y sin espacios.

El sublenguaje de YAML desplegado tiene en cuenta lo siguiente:

- Objetos YAML formados por una secuencia de pares (clave,valor)
- Los objetos anidados se encuentran bajo 4 espacios de indentación
- Las strings no llevan las comillas (“)
- Luego del dos puntos (:) de un par clave valor va un espacio

Ejemplo de entrada 1:

```
{"nombre":"Rojo","valor_r":255,"valor_g":0,"valor_b":0}
```

Ejemplo de salida 1:

```
nombre: Rojo
valor_r: 255
valor_g: 0
valor_b: 0
```

Ejemplo de entrada 2:

```
{"colores":{"rojo":{"r":255,"g":0,"b":0},"azul":{"r":0,"g":0,"b":255}}}
```

Ejemplo de salida 2:

```
colores:
    rojo:
        r: 255
        g: 0
        b: 0
    azul:
        r: 0
        g: 0
        b: 255
```

Ejemplo de entrada 3:

```
{"nombre":"Rojo","valor_r:255}
```

Ejemplo de salida 3:

```
NO PERTENECE
```

Referencias

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