Teoría de Lenguajes
Curso 2023
Laboratorio 2 – Gramática Libre de Contexto

Este laboratorio consiste en utilizar las funcionalidades de gramática libre de contexto (GLC) de la librería NLTK [1] para implementar programas en Python [2]. En particular, se implementará un reconocedor de funciones con una sintaxis inspirada en Python.

La propuesta se compone de 3 programas a implementar. Cada programa recibe una entrada que es procesada con una GLC y despliega una salida. En cada programa se deberá construir una gramática para reconocer la entrada, basándose en la descripción en esta letra y un conjunto de ejemplos de entrada y salida.

Los 3 programas a implementar están en secuencia, donde cada programa extiende la gramática del anterior.

Modo de trabajo

Cada programa a implementar es un módulo **Python 3** ejecutable que recibe como parámetros los nombres de los archivos de entrada y salida.

Los programas pedidos, excepto que se indique lo contrario, tienen el siguiente comportamiento:

- 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 PERTENECE - FUERA DE VOCABULARIO**
- En caso de que la entrada pertenezca al lenguaje generado por la gramática el programa devuelve **PERTENECE.**

Se permite importar variables y funciones entre módulos pero no crear módulos auxiliares.

Para realizar este laboratorio se imparte el archivo **lab2.zip** que contiene ejemplos de entrada y salida, el módulo *programa0.py* y el script (*test.py*) que ejecuta cada programa con las entradas disponibles y compara las salidas obtenidas con las esperadas.

programa0.py

El programa 0 se encuentra implementado como guía para la implementación de los restantes programas.

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

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

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

Ejemplo 1:

```
a a a b b b
```

Salida:

```
PERTENECE
```

Ejemplo 2:

```
a a b
```

Salida:

```
NO PERTENECE
```

Ejemplo 3:

```
a a 1 1
```

Salida:

```
NO PERTENECE - FUERA DE VOCABULARIO
```

programa1.py (Expresiones Aritméticas)

Este programa reconoce expresiones aritméticas en notación infija. Las expresiones cuentan con los elementos que se describen a continuación.

- Operadores:
  - \+ (suma)
  - \- (resta)
  - * (producto)
  - / (división)
  - % (módulo)
  - ** (exponenciación)
  - // (división entera)

- Paréntesis: (,)
- Valores numéricos: 0,1,2,..,99,100
- Constantes: k1, …, k10
- Variables x1, ...x10

Puede asumir que no hay problemas de ambigüedad por el orden de los operadores en la entrada.

Ejemplo 1:

```
(17 + x1) * (k3 % (x5 // x7))
```

Salida:

```
PERTENECE
```

Ejemplo 2:

```
(x7 % x5) * ( // (k1 ** k2))
```

Salida:

```
NO PERTENECE
```

Ejemplo 3:

```
3 + a
```

Salida:

```
NO PERTENECE - FUERA DE VOCABULARIO
```

programa2.py (Expresiones booleanas)

Este programa reconoce expresiones booleanas. Las expresiones pueden estar formadas por expresiones atómicas: constantes, variables y comparaciones de expresiones aritméticas; o expresiones compuestas con operadores lógicos.

A continuación se detalla cada elemento.
- Constantes, variables y expresiones aritméticas: (programa 1)
- Paréntesis: (,)
- Operadores de comparación de expresiones arimteticas: ==, <=, >=, !=
- Constantes booleanas: True, False
- Operadores lógicos: and, or, not

Puede asumir que no hay problemas de ambigüedad por el orden de los operadores en la entrada.

Ejemplo 1:

```
( not ( x1 == 1 or x1 == 7 ) ) and x2 != x3
```

Salida:

```
PERTENECE
```

Ejemplo 2:

```
x5 and not
```

Salida:

```
NO PERTENECE
```

Ejemplo 3:

```
True xor
```

Salida:

```
NO PERTENECE - FUERA DE VOCABULARIO
```

programa3.py (Reconocedor de Python)

En este programa se reconoce la definición de funciones en una sintáxis inspirada en Python.

Para la implementación de esta parte, puede y se sugiere reutilizar las gramáticas desarrolladas en las partes anteriores.

El reconocedor a implementar, debe tener en cuenta los siguientes elementos:

Python con:
- Expresiones aritméticas: (programa1)
- Expresiones booleanas: (programa2)
- Variables (además de las de las partes anteriores): x, i, j, k, arr, n, left, right, mid
- Definición de funciones (únicamente con parámetros posicionales no opcionales).
(Ej. def f1(x1,x2,x3))
- Invocación de funciones
- Nombre de funciones: : f1, …,f10, len, range
- Slicing de variables: [e1], [e1:], [:e1], [e1:e2] y [e1:e2:e3] donde e1,e2 y e3 son expresiones aritméticas
- Asignaciones: =
- Estructura if-then, if-then-else
- Estructuras de iteración: for x in expr, while cond
- Bloques de código: { }
- Delimitador de instrucciones: ;

Consideraciones a tener en cuenta:

No considere los bloques delimitados por indentación de Python. Los bloques se definen únicamente con { y }.

La instrucción “return” de las funciones puede ser considerada como una instrucción más, no requiere ninguna consideración especial.

Este programa no se probará con entradas que contengan particularidades adicionales a las contenidas en los ejemplos.

Ejemplo 1:

```
def f1(x):
{
    if x == 1:
        return 1;
    else:
        return (x * f1(x – 1));
}
```

Salida:

```
PERTENECE
```

Ejemplo 2:

```
def f7(x):
{
    if x == 0:

}
```

Salida:

```
NO PERTENECE
```

Ejemplo 3:

```
def f8(x):
{
    if x # 1:
        return 0;
}
```

Salida:

```
NO PERTENECE - FUERA DE VOCABULARIO
```

Entrega

La entrega es el **20 de junio a las 23:59** y se realizará mediante un formulario en EVA que estará disponible próximo a la fecha de entrega.

Los grupos deben ser los mismos que en el laboratorio 1.

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.

Referencias

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