Prácticas de Autómatas y Lenguajes. Curso 2024/25

Práctica 2: Análisis Sintáctico y Semántico

Duración: 5 semanas.

Entrega: viernes 13 de diciembre a las 23h59.

Peso: 45% de la nota de prácticas.

Descripción del enunciado:

En esta práctica estudiaremos distintos conceptos relacionados con el análisis sintáctico y semántico. Empezaremos con nuestra propia implementación de un analizador sintáctico para gramáticas LL(1) y posteriormente utilizaremos las herramientas Lex-Yacc de Python para construir una gramática de atributos.

Los objetivos de la práctica son:

- Calcular los conjuntos *primero* y *siguiente* de cada uno de los símbolos no terminales de una gramática.
- Construir la tabla de análisis para un analizador sintáctico descendente LL(1) a partir de los conjuntos *primero* y *siguiente* de la gramática.
- Entender y programar el algoritmo de análisis sintáctico descendente LL(1) a partir de la descripción de la tabla de análisis.
- Construir la gramática de atributos utilizando las herramientas Lex-Yaac.

Se recuerda que los objetivos de aprendizaje planteados deben ser adquiridos por **ambos** miembros de la pareja. En caso de realizarse una prueba y comprobar que este no es el caso, se podría suspender la práctica y exigir la entrega individual en la modalidad no presencial.

Descripción de los ficheros suministrados:

Se facilitan los siguientes ficheros para los ejercicios 1-4 relacionados con el análisis sintáctico.

- `grammar.py`: Contiene, entre otras, las clases `Grammar` y `LL1Table` que deberán ser modificadas para añadir los métodos pedidos en los ejercicios 1-4.

- `utils.py`: Contiene funciones de utilidad para, por ejemplo, leer una gramática a partir de su descripción en forma de cadena o imprimir la tabla de análisis. No se debe modificar ni entregar este fichero.
- `test_analyze.py`, `test_first.py` y `test_follow.py`: Contienen diversos tests de ejemplo en formato unittest, útiles como punto de partida para desarrollar nuevos tests que permitan comprobar que la funcionalidad implementada es correcta.

Se facilitan los siguientes ficheros para los ejercicios 5-6 relacionados con el análisis semántico.

- `g1_lexer.py` y `g1_parser.py`: Contine los ficheros punto de partida para desarrollar el ejercicio 5.
- `roman_lexer.py` y `roman_parser.py`: Contine los ficheros punto de partida para desarrollar el ejercicio 6.

Ejercicio 1: Análisis sintáctico descendente (2 puntos):

En este ejercicio se implementará el algoritmo de análisis sintáctico descendente LL(1) a partir de una tabla de análisis dada. Para ello es necesario completar el código del método `analyze` de la clase `LL1Table` en el fichero `grammar.py`. El método recibe la cadena a analizar, `input_string`, y el axioma o símbolo inicial, `start`. Debe devolver un árbol de derivación si la cadena es sintácticamente correcta de acuerdo con la tabla de análisis, o generar una excepción de tipo `SyntaxError` si no lo es.

En el fichero `test_analyze.py` se facilitan algunos tests. Recordad no obstante que es vuestra responsabilidad realizar tests adicionales para comprobar que el código funciona de manera correcta.

Ejercicio 2: Cálculo del conjunto *primero* (2 puntos):

Se debe completar, en la clase `Grammar` del fichero `grammar.py`, el código del método `compute_first`, que calcula el conjunto *primero* para una determinada cadena *w* recibida como argumento. Esta cadena debe estar formada únicamente por símbolos terminales y no terminales de la gramática. En caso de que no sea así el método deberá lanzar una excepción del tipo `ValueError`. El método debe devolver un conjunto con todos los símbolos contenidos en *primero(w)*, siendo cada símbolo una cadena con un único carácter, o una cadena vacía si $\lambda \in$ *primero(w)*.

En el fichero `test_first.py` se facilitan algunos tests. Recordad no obstante que es vuestra responsabilidad realizar tests adicionales para comprobar que el código funciona de manera correcta.

Ejercicio 3: Cálculo del conjunto *siguiente* (2 puntos):

Se debe completar, en la clase `Grammar` del fichero `grammar.py`, el código del método `compute_follow`, que calcula el conjunto *siguiente* para un determinado símbolo no terminal *X* recibido como argumento. El método debe generar una excepción del tipo `ValueError` si el símbolo recibido no pertenece al conjunto de símbolos no terminales de la gramática. En caso contrario el método debe devolver un conjunto con todos los símbolos contenidos en *siguiente(X)*, siendo cada símbolo una cadena con un único carácter, que puede incluir al carácter de fin de cadena `'$'`.

En el fichero `test_follow.py` se facilitan algunos tests. Recordad no obstante que es vuestra responsabilidad realizar tests adicionales para comprobar que el código funciona de manera correcta.

Ejercicio 4: Cálculo de la tabla de análisis LL(1) (1 punto):

Completad, en la clase `Grammar` del fichero `grammar.py`, el código del método `get_ll1_table`, que calcula la tabla de análisis LL(1) para la gramática. El método debe devolver un objeto de la clase `LL1Table`, o `None` si la gramática no es LL(1) (es decir, si hay varias partes derechas en una misma casilla de la tabla).

En el fichero `test_analyze.py` se facilitan algunos tests. Recordad no obstante que es vuestra responsabilidad realizar tests adicionales para comprobar que el código funciona de manera correcta.

Ejercicio 5: Implementar una gramática de atributos para el lenguaje (0.5 puntos).

Completad, en el fichero `g1_parser.py`, el código para implementar la gramática de atributos para el siguiente lenguaje L.

$$L = \{a^n b^n c^k \mid k >= n+1\}$$

Se debe incluir atributos para los símbolos no terminales

En el fichero `test_g1.py` se facilitan algunos tests. Recordad no obstante que es vuestra responsabilidad realizar tests adicionales para comprobar que el código funciona de manera correcta.

Ejercicio 6: Implementar una gramática de atributos (2.5 puntos).

Completad, en el fichero `roman_parser.py`, el código para traducir números romanos a números arábigos mediante la implementación de gramáticas independientes del contexto en Python.

Roman	→ Hundrends Tens Units
Hundreds	→ LowHundreds | CD | D LowHundreds | CM
LowHundreds	→ LowHundreds C | λ
Tens	→ LowTens | XL | L LowTens | XC
LowTens	→ LowTens X | λ
Units	→ LowUnits | IV | V LowUnits | IX
LowUnits	→ LowUnits I | λ

Define atributos para esta gramática que lleven a cabo dos tareas:

a) Restringir el número de X en <LowTens>, el de I en <LowUnits> y el de C en <LowHundreds> a no más de tres.

b) Calcular el número para un número romano.

El no terminal Roman (símbolo inicial) se debe representar como un diccionario en python de la siguiente manera:

{“val”: <número arábigo>, “valid”: <True|False>}

“val“:<número arábigo> corresponde a la codificación y “valid”:<True|False> representa si el número está bien construido o no.

Por ejemplo:

Para el siguiente número romano XI tenemos {“val”: 11, “valid”: True}

Para el siguiente número romano XIIII tenemos {“val”: -1, “valid”: False}

En el fichero `test_roman.py` se facilitan algunos tests. Recordad no obstante que es vuestra responsabilidad realizar tests adicionales para comprobar que el código funciona de manera correcta.

Planificación:

Ejercicio 1	Semana 1
Ejercicio 2 y 3	Semana 2
Ejercicio 4	Semana 3
Ejercicio 5	Semana 4
Ejercicio 6	Semana 5

Normas de entrega:

La entrega la realizará sólo uno de los miembros de la pareja a través de la tarea disponible en Moodle.

- Se deben entregar *grammar.py* para los ejercicios 1-4 (Solo es necesario este fichero para los puntos 1-4).
- Se deben entregar los ficheros g1_parser.py y roman_parser.py. Estos ficheros solo pueden ser modificados para codificar las reglas de la gramática de la gramática de atributos.