Procesadores del lenguaje

PRÁCTICA 1: INTRODUCCIÓN AL ENTORNO

10/04/2024

—

Belén Gómez Arnaldo 100472037

Ignacio Fernández Cañedo 100471955

Índice

Introducción

El propósito de esta práctica es utilizar la herramienta PLY para desarrollar analizadores léxico y sintáctico, además de comprobar su validez con una serie de pruebas.

En primer lugar explicaremos las decisiones utilizadas tanto en el analizador léxico como en el analizador sintáctico y finalmente analizaremos la salida con una serie de pruebas que nos permitirán verificar la correcta funcionalidad de nuestros analizadores.

Analizador léxico

En el analizador léxico, definimos una serie de reglas que identifican los tokens de nuestra entrada, estos tokens pueden ser palabras clave, operadores, números, etc. El objetivo de este analizador es que cada token de la entrada sea reconocido correctamente, según una serie de especificaciones indicadas en la práctica.

En primer lugar, hemos dividido los tokens de las palabras reservadas, como NULL, TRUE, o FALSE. Después hemos procedido a especificar cada uno de los tokens del enunciado. En el caso de los números, los hemos separado en funciones según el tipo de cada uno, en lugar de hacer una única función. Hemos decidido hacer esto ya que pensamos que de esta forma se entiende mejor y queda más claro, aún así todos se convierten a enteros.

En el caso de las cadenas de caracteres, estas pueden ser con comillas o sin comillas. En ambas hemos respetado las restricciones del enunciado. Deben comenzar por una letra o una “_” y solo pueden contener letras. Además en el caso de las cadenas con comillas, estas deben comenzar y acabar con comillas.

Finalmente, hemos realizado una función para ignorar las tabulaciones y los saltos de línea, así como una para detectar el error. Las últimas dos funciones son utilizadas para probar el analizador con un archivo externo.

Analizador sintáctico

Una vez que tenemos nuestros tokens, el siguiente paso es el analizador sintáctico. Este componente toma los tokens producidos por el analizador léxico y los organiza en una estructura que refleja la gramática de nuestro lenguaje. Para hacer esto, definiremos la mencionada gramática en las funciones, utilizando los tokens definidos en el analizador léxico como símbolos no terminales.

A diferencia del analizador léxico, que divide la entrada en tokens y detecta un error cuando la entrada no coincide con la regla especificada, este analizador tendrá en cuenta

los errores que el analizador léxico no puede detectar. Este toma los tokens producidos por el analizador léxico y los organiza en una estructura que refleja la gramática de nuestro lenguaje, por lo que puede detectar errores cuando los tokens se combinan de una manera no válida.

Al igual que en el analizador léxico, se han creado funciones que representan todos los niveles de la gramática. La función pair crea un diccionario por cada par clave valor de la entrada, posteriormente la función object se encargará de combinar los diccionarios creados por la función pair en un único diccionario. De esta forma se devolverá un diccionario que será la salida del analizador.

Finalmente se crean las funciones para ejecutar el analizador y que la impresión de la salida sea la especificada por el enunciado. También se distinguen los distintos casos que puede tener la entrada, como que esté vacía o no.

Salida

Para ver la salida de los analizadores hemos creado una batería de pruebas que recorren la mayoría de casos distintos que puede tener el problema. De esta forma comprobamos el correcto funcionamiento de los dos analizadores y podemos observar que la salida es la esperada. Ambos analizadores muestran la salida indicada por el enunciado. Para ejecutar los analizadores, se deberá escribir la línea que se especifica en el enunciado. Además si se quiere ejecutar únicamente el analizador léxico, bastará con añadir el argumento lexer al final.

El analizador léxico imprime cada token seguido por su valor. Si este se encuentra con algún error léxico, imprimirá el carácter que ha generado el error y lo que queda de entrada, y continuará con la ejecución para imprimir el resto de tokens y detectar el resto de errores.

En el caso del analizador sintáctico, la función imprimir tendrá en cuenta los casos que puedan incluir ajson anidados. Esta función recibe un diccionario en el que el valor puede ser otro diccionario, por lo que se utiliza recursividad para tener en cuenta estos casos. La función comienza recorriendo el primer diccionario, y si este tiene otro diccionario en algún valor, se volverá a llamar de forma recursiva a la función con el nuevo diccionario como valor a recorrer y la key anterior como prefijo, para que se pueda imprimir como key1.key2... Si el valor es una lista, ocurrirá algo parecido, se imprimirá la clave de la lista seguido por la posición del array que le corresponde al diccionario en el que se encuentra la variable, de tal forma que se muestra como se especifica en las modificaciones avanzadas del enunciado.

Batería de pruebas

Hemos creado una serie de archivos para comprobar el funcionamiento del programa.

test1.ajson

Este archivo es una entrada válida que tanto el analizador léxico como sintáctico reconocen sin errores. En este archivo hemos incluído claves y valores de todos los tipos que se pueden reconocer. Se pueden destacar los siguientes:

- Hay cadenas de caracteres sin comillas que solo se pueden utilizar como claves y cadenas de caracteres con comillas que se utilizan como claves y como valores.
- Hay números de diferentes tipos y al imprimir la salida se muestra su valor en decimal.
- Hay una clave vacía que en la salida se imprime como "*None*". Además, la mayoría de los pares de clave/valor están separados por salto de línea, pero esto no es necesario. Si se ponen varios pares en una línea, separados por una coma (línea 5 del archivo de entrada), también se reconocen sin problemas.
- Se incluyen valores booleanos y el valor "*null*", que se reconocen tanto en mayúsculas como en minúsculas.
- También se incluye como valor una comparación entre dos números enteros que devuelve el valor *False*.
- Hay claves anidadas, es decir, una clave cuyo valor es otro ajson. Además, dentro de este ajson hay otro valor que es un nuevo ajson. El código puede reconocer cualquier cantidad de claves anidadas.
- Por último, hemos incluído un array como modificación avanzada. Además, al ser el array el último elemento de ajson puede o no terminar con una coma

test2.ajson

Este es un fichero con un ajson vacío. En el enunciado de la práctica se especifica que esto se traducirá como un valor nulo. Por lo tanto, hemos decidido que cuando todo el objeto ajson esté vacío se imprima un mensaje indicando eso.

test3.ajson

Este es un fichero en el que las cadenas de caracteres no tienen el formato correcto. La primera clave empieza con un número y el último valor es una cadena de caracteres sin comillas pero que contiene espacios. Estos no son formatos de cadenas válidos, por lo que el reconocedor sintáctico muestra un error. Además, si se quiere poner como valor una cadena de caracteres tiene que tener comillas.

Cabe destacar que en este caso el reconocedor léxico no muestra errores, porque en realidad no es problema léxico. En la primera clave, el reconocedor léxico separa la clave en dos tokens: un número y una cadena de caracteres sin comillas, porque en realidad no es un problema léxico, son tokens válidos. Es un problema sintáctico y por eso solo el analizador sintáctico devuelve un error.

test4.ajson

Este es un archivo en el que los pares de clave/valor del ajson no están separados por comas. El único caso en el que puede omitirse la coma es el último par, pero el resto deben de ir separados por comas. Por eso este archivo devuelve un error en el analizador sintáctico, pero, por lo mismo que en el anterior ejemplo, no hay ningún error léxico.

test5.ajson

En este caso, este fichero contiene un error léxico porque la clave es una cadena de caracteres con comillas al principio, pero no se llegan a cerrar las comillas. Esto es un error léxico.

test6.ajson

En este ejemplo también hay un error léxico ya que se ha introducido un punto para separar la clave de su valor, y este no es un token válido.

test7.ajson

Este fichero contiene un ajson pero sin el último '}', es decir, no se llega a cerrar el ajson. Este ejemplo devuelve un error sintáctico ya que el programa esperaba ese token pero no encuentra nada.

test8.ajson

En este ejemplo se han introducido números en formatos que no son adecuados, como un binario que empiece por la letra 'b', un número decimal con una letra o un octal que empiece por 'o'. Al igual que en los casos mencionados anteriormente, esto no es un error léxico sino un error que reconoce el analizador sintáctico.

test9.ajson

Este ejemplo es un fichero vacío, por lo que el programa solo imprime un mensaje de que el fichero proporcionado como argumento está vacío, pero no es ningún error ni léxico ni sintáctico.

test10.ajson

Por último, hemos creado un fichero en el que uno de los valores está vacío, por lo que devuelve un error sintáctico. Esto es distinto de que un valor fuera una cadena vacía delimitada por comillas, en ese caso no habría ningún error, simplemente se imprimiría el valor como una cadena vacía. En este caso hay una clave seguida de ':', pero después hay una coma, cuando debería haber un valor.