Actividad Complementaria 1: Fundamentos Teóricos del Análisis Léxico

Isaac Madrigal Silva - C14387

Sección 3.1

La separación del análisis léxico del análisis sintáctico es una decisión de diseño importante porque divide el problema de procesar el código fuente en dos niveles de complejidad distintos. El análisis léxico se encarga de reconocer patrones simples de caracteres (tokens) como identificadores, números, operadores o palabras reservadas (if, else, for, while, etc). Por otro lado, el análisis sintáctico se encarga de tomar estos tokens y verificar que se ajusten a la estructura gramatical del lenguaje de turno.

Esta división permite que el analizador sintáctico trabaje con una secuencia de símbolos ya procesados, en lugar de lidiar con caracteres crudos. Además, por mencionar un ejemplo, permite que el analizador sintáctico no tenga que trabajar con comentarios, ya que estos son procesados e ignorados por el analizador léxico. Si el analizador sintáctico tuviera que trabajar con los comentarios sería mucho más complejo.

Para el proyecto del curso, esta separación simplificará la implementación porque podremos concentrarnos primero en definir un lexer que identifique correctamente todos los tokens del lenguaje, y que en la siguiente entrega, el parser use estos tokens sabiendo que son válidos dentro del lenguaje que se maneja. Esto quiere decir que no se va a topar con nada que no conozca para formar las gramáticas.

Sección 3.3

Las expresiones regulares son la herramienta formal que permite describir de manera precisa los patrones que corresponden a los diferentes tokens del lenguaje. Se usan las expresiones regulares ya que con ellas es posible definir las reglas para

reconocer los diferentes componentes del lenguaje como identificadores, literales numéricos, literales de cadena, así como los distintos operadores aritméticos, lógicos y de comparación. Esto es posible ya que las expresiones regulares abarcan las operaciones básicas de lenguajes (unión, concatenación y clausura). Además, en sistemas modernos cuentan con algunas extensiones como los operadores +, ?, entre otros.

Por ejemplo, para reconocer un número entero en Python se puede usar una expresión regular como:

\d+

Esta expresión hace que coincida con las expresiones que tienen 1 o más caracteres numéricos en su composición.

Sección 3.5

El generador Lex del libro utiliza un modelo basado en reglas patrón-acción, donde cada regla especifica un patrón de caracteres (mediante una expresión regular) y una acción asociada que se ejecuta cuando se encuentra dicho patrón. La librería PLY sigue exactamente la misma filosofía: cada token se define con una expresión regular, y la acción se codifica en una función o en la asignación de la variable correspondiente. Cuando se asigna a una variable tiene una acción implícita (retornar el token). Cuando se usa una función se pueden hacer diversas acciones antes de retornar el token. Si no se retorna nada, se podría decir que se está ignorando la cadena capturada y nunca llegará al analizador sintáctico.

Sección 3.1.4

El libro discute diferentes estrategias para manejar errores en el análisis léxico, como ignorar el carácter inválido, reemplazarlo por un símbolo especial o registrar un error antes de continuar.

En el caso del proyecto del curso, estas ideas deben aplicarse a situaciones específicas, como:

Caracteres desconocidos: cualquier símbolo no definido en la gramática debe reportarse como error léxico y descartarse para no interrumpir todo el proceso. Lo de descartar el carácter se debe a que la idea del lexer no es que se detenga al encontrar un error, sino que siga hasta el final.

Secuencias de escape inválidas en cadenas: deben detectarse y reportarse Una decisión posible sería no hacer una transformación y simplemente reportar el error, igual que con los caracteres desconocidos. De igual forma, se podría realizar alguna transformación en el carácter específico, si se deseara.

En ambos casos se busca que se pueda seguir procesando el programa en lugar de detenerse abruptamente.

Reflexión final sobre la indentación

El manejo de la indentación es el requisito más complejo del lexer. Las expresiones regulares reconocen patrones locales de caracteres, pero la indentación requiere cierto contexto para saber el nivel actual de tabulación. Además, la indentación generalmente se da después de un “:”, pero no siempre tras un signo de dos puntos hay que aplicar indentación.

Por ejemplo, cuando una línea aumenta su indentación, debe generarse un token INDENT; cuando la reduce, deben emitirse uno DEDENT. Esto exige mantener algún tipo de registro (como una pila) para saber los niveles de indentación que hay al momento de insertar los tokens de DEDENT.

Aquí es donde entran las acciones del modelo patrón-acción discutido en el libro: en PLY se podría escribir código dentro de las acciones de las reglas (funciones) para decidir dinámicamente cuántos tokens INDENT o DEDENT devolver. También, se podría intentar una estrategia de usar los operadores de LOOKAHEAD. Aunque esta por sí sola puede que no fuera suficiente.