Análisis Sintáctico y Contextual del Compilador Mini-Python (Etapa 2)

Fabián Fernández Kevin Jiménez Justin Martínez

Compiladores e Intérpretes
Tecnológico de Costa Rica
Profesor: Oscar Mario Víquez Acuña
II Semestre 2024

Contents

1 Introducción

Este documento cubre tanto las fases de análisis sintáctico como de análisis contextual en el desarrollo del compilador para Mini-Python. Además de la implementación del análisis sintáctico utilizando ANTLR4 y C#, se han añadido nuevas funciones que permiten realizar el análisis contextual, asegurando la validez de los identificadores y otros elementos semánticos del código.

2 Análisis del Lenguaje

El lenguaje Mini-Python mantiene una estructura simple basada en Python, con reglas léxicas y sintácticas en formato EBNF.

2.1 Gramática

La gramática del lenguaje está definida en ANTLR4, con las principales reglas para la sintaxis del compilador:

```
program: mainStatement* EOF;

mainStatement: defStatement | assignStatement;

defStatement: DEF ID '(' argList ')' ':' sequence;
assignStatement: ID ASSIGN expression NEWLINE;

ifStatement: IF expression ':' sequence ELSE ':' sequence;
whileStatement: WHILE expression ':' sequence;

sequence: INDENT statement+ DEDENT;
```

3 Soluciones e Implementación

3.1 Scanner

El *scanner* utiliza ANTLR4 para reconocer los tokens, implementando el manejo de indentación, que es fundamental en Mini-Python.

```
lexer grammar MiniPythonLexer;
tokens { INDENT, DEDENT }

@lexer::members {
    private DenterHelper denter;
    public override IToken NextToken() {
        if (denter == null) {
            denter = DenterHelper.Builder()
                .Nl(NEWLINE)
                .Indent(MiniPythonParser.INDENT)
                .Dedent(MiniPythonParser.DEDENT)
                .PullToken(base.NextToken);
        }
        return denter.NextToken();
    }
}
```

3.2 Parser

El *parser* se implementa utilizando ANTLR4 para generar el árbol de sintaxis abstracta (AST).

```
parser grammar MiniPythonParser;
options { tokenVocab = MiniPythonLexer; }
program: mainStatement* EOF;
mainStatement: defStatement | assignStatement;
```

3.3 Interfaz Gráfica

La interfaz gráfica está implementada utilizando WPF. Incluye un editor de texto que permite visualizar el código, el numero de lineas existenten y un boton para ejecutar compilaciones, mostrando los errores en un panel separado. Además, los errores detectados en la compilación se muestran y proporcionan la linea del error, además permite abrir archivos locales, archivos desde web (en este caso se implementó la prueba con el uso de archivos desde github) ademas de la opción de guardar el archivo o simplemente desplegar un archivo nuevo sin titulo desde la misma interfaz. Ademas se mejoro y se arreglaron bugs de la interfaz con respecto a la primera etapa.

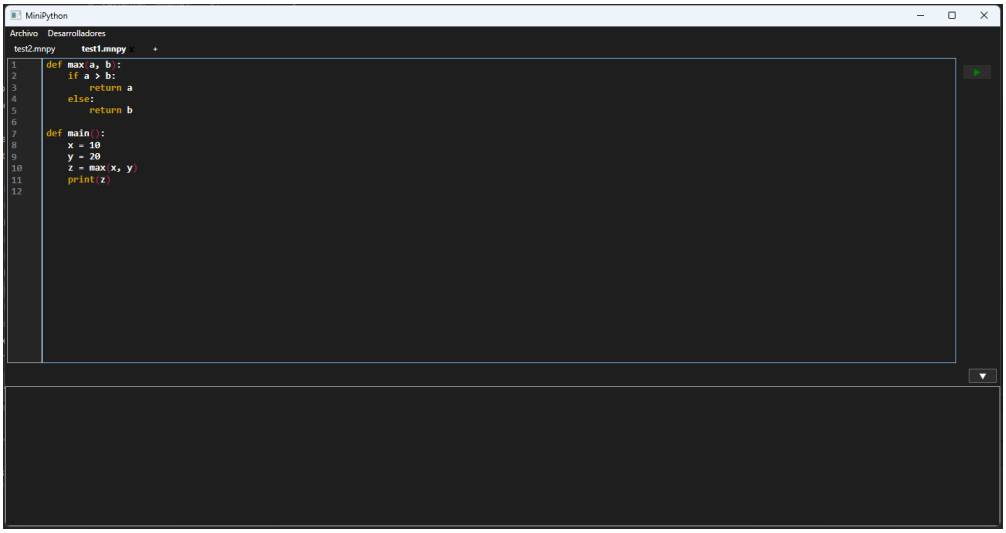

Figure 1: Interfaz gráfica del compilador Mini-Python mejorada

4 Análisis Contextual

En esta segunda etapa, se ha implementado el análisis contextual, el cual asegura la validez de los identificadores, funciones y otros elementos semánticos del código fuente.

4.1 Implementación del Análisis Contextual

Se crearon dos archivos nuevos dentro de una carpeta llamada Checker:

- ContextAnalizer.cs: Este archivo contiene la clase 'ContextAnalizer', la cual es responsable de visitar los nodos del árbol de sintaxis abstracta y realizar verificaciones contextuales, tales como:
  - Chequeo de alcances para identificadores (variables, funciones, parámetros).
  - Validación de la cantidad de parámetros en las funciones.
  - Reporte de errores contextuales, como el uso de variables no definidas o redefinición de funciones.
- SymbolsTable.cs: Este archivo contiene la implementación de la tabla de símbolos, que lleva el registro de los identificadores en los distintos niveles de alcance. Incluye funciones para abrir y cerrar ámbitos, insertar variables y funciones, y buscar identificadores.

4.2 Tabla de Símbolos

La tabla de símbolos es utilizada para gestionar los alcances de los identificadores. Para cada identificador, se almacena su nombre, tipo (variable, función, parámetro) y el nivel de alcance.

4.3 Chequeos Realizados

Algunos de los chequeos realizados por el ‘ContextAnalizer‘ incluyen:

- Redefinición de Funciones: Se verifica que las funciones no sean redefinidas en el mismo nivel de alcance con la misma cantidad de parámetros.
- Uso de Variables: Se reporta un error si una variable es utilizada sin haber sido definida previamente.
- Cantidad de Parámetros: Se comprueba que las funciones sean llamadas con la cantidad correcta de parámetros.

4.4 Reporte de Errores Contextuales

Los errores contextuales se reportan con la línea y columna donde ocurren. Por ejemplo:

```
CONTEXT ERROR - line 5: Variable 'x' not defined before use.
CONTEXT ERROR - line 10: Function 'miMetodo' called with incorrect number of parameters.
```

5 Resultados Obtenidos

Tabla de Símbolos	Completado
Estructura y Funcionalidad	Completado
Chequeo de Alcances	
Control de Niveles de los Identificadores	Completado
Control de Existencia de Métodos y Cantidad de Parámetros	Completado
Uso de Identificadores en Expresiones	Completado
Otros Elementos no Computables	Completado
Reporte de Errores (Errores Significativos)	Completado
Documentación	
Formato y Contenido	Completado
Manejo de Errores	Mejorado
Interfaz Gráfica	Mejorado

6 Problemas Identificados en la Etapa 1

Los problemas identificados en la primera etapa han sido solucionados:

6.1 Interfaz de Depuración

La falta de una interfaz de depuración que permita ejecutar el código línea por línea fue solucionada implementando un depurador básico.

6.2 Hipervínculo en la Consola de Errores

Se corrigió el problema del hipervínculo en la consola de errores, que ahora permite navegar directamente a la línea del error.

6.3 Error en el Parser

El error relacionado con el EOF fue corregido ajustando el manejo de tokens al final del archivo.

7 Conclusiones

Con la implementación del análisis contextual, se ha completado la validación semántica del código Mini-Python, asegurando el uso correcto de identificadores, funciones y parámetros. Los problemas encontrados en la primera etapa fueron solucionados, y la interfaz gráfica mejorada facilita la depuración del código.

8 Bibliografía

References

[1] Terrence Parr. *The Definitive ANTLR 4 Reference*. 2013.

[2] Antlr-Denter, `https://github.com/yshavit/antlr-denter`.