Proyecto Especial

Diseño e implementación de un lenguaje

Tetralepis

Felipe Mindlin	62774	fmindlin@itba.edu.ar
Gianfranco Magliotti	61172	gmagliotti@itba.edu.ar
Francisco Sendot	62351	fsendot@itba.edu.ar
Federico Inti Garcia Lauberer	61374	fedegarcia@itba.edu.ar

Índice

Introducción

Tetralepis fruhstorferi es una serpiente conocida como la "Serpiente de Java", lo cual alude tanto al lenguaje de programación Java como a Python, ya que este último es también el nombre de una serpiente. Por ello, se eligió este nombre para nuestro compilador, cuyo objetivo es traducir un subconjunto de Python a Java.

Nuestro compilador Tetralepis tiene como meta facilitar la vida de los programadores al proporcionar una herramienta eficiente y fácil de usar para convertir código Python en código Java.

En este informe, se explicará la creación del compilador, incluyendo los problemas y limitaciones encontradas durante su desarrollo.

Desarrollo

El desarrollo del compilador Tetralepis se basó en seis etapas principales: Análisis léxico, análisis sintáctico, análisis semántico, chequeo de tipos, tablas de símbolos y generación de código.

Se enfocará en dar detalles acerca de cada etapa implementada. Luego se hablará sobre las dificultades encontradas durante el desarrollo, y se finalizará con unas conclusiones acerca del trabajo realizado, qué cosas se pueden mejorar, qué cosas se aprendieron, etc.

Modelo Computacional

Dominio

El dominio de este proyecto se centra en la traducción automática de código fuente escrito en Python a código fuente en Java. Python es un lenguaje de programación de alto nivel, conocido por su sintaxis clara y su tipificación dinámica, lo que facilita la escritura de código de forma rápida. Java, por otro lado, es un lenguaje de programación de propósito general, concurrente, orientado a objetos y con una tipificación estática estricta. Ambos lenguajes son ampliamente utilizados en diversas aplicaciones, desde desarrollo web hasta inteligencia artificial, pero presentan diferencias significativas en cuanto a su estructura y manejo de tipos.

El compilador Tetralepis se diseñó para manejar un subconjunto específico de Python, enfocándose en las construcciones básicas del lenguaje, como variables, funciones y estructuras de control (if, else, while, for). El objetivo es traducir estas construcciones a sus equivalentes en Java, respetando las reglas de tipificación y estructura propias de Java. El dominio abarca tanto los aspectos sintácticos como semánticos de ambos lenguajes, destacando las diferencias y similitudes entre ellos para facilitar una traducción precisa y funcional.

Lenguaje

El lenguaje de entrada del compilador es un subconjunto de Python que incluye declaraciones de variables, estructuras de control (if, else, while, for), definición y llamada de funciones, manejo básico de tipos de datos (int, float, bool, string) y operaciones aritméticas y lógicas básicas. El lenguaje de salida es Java, que requiere tipificación estática y una estructura más rígida. La traducción implica convertir las declaraciones de variables, estructuras de control y funciones de Python a Java, asegurando la correcta tipificación y adaptando la sintaxis. Se utilizan delimitadores de bloques (@{ y @}) para facilitar el análisis sintáctico debido a la dependencia de Python en la indentación, lo cual no tiene equivalente directo en Java. El uso de un tipo genérico Object en Java permite manejar algunas de las flexibilidades de Python, aunque con limitaciones en el manejo de tipos dinámicos y la inferencia de tipos.

Front-End

Análisis Léxico

El análisis léxico es la primera fase del compilador, donde se escanean las cadenas de caracteres del código fuente para identificar los tokens. Estos tokens son las unidades léxicas del lenguaje, como palabras clave, identificadores, operadores y delimitadores. Para esta fase, se utilizó Flex, una herramienta para generar analizadores léxicos. Se definieron los símbolos terminales del subconjunto de Python que nuestro compilador soporta, como if, else, while, for, def, entre otros.

Python es un lenguaje de programación **muy** grande, lo que significó un gran trabajo para identificar todos los lexemas posibles.

Para facilitar el análisis sintáctico y poder establecer bien luego la sintaxis de los bloques de Python, se hizo una modificación del alfabeto de entrada que debe recibir: Cada bloque (*if-elif-else-def-class*) debe comenzar con el lexema '@{' y cerrarlo con el lexema '@}'.

Análisis Sintático

Una vez identificados los lexemas relevantes del lenguaje, el siguiente paso es el análisis sintáctico, que construye la estructura del programa a partir de los lexemas.

Para esto, se utiliza Bison, una herramienta que genera analizadores sintácticos. La misma trabaja en conjunto con Flex, utilizando a Flex a demanda como analizador léxico para satisfacer las reglas de gramática.

Se definieron reglas de producción para la gramática que representa el subconjunto de Python y cómo estas se traducen en nodos del *Árbol de Sintaxis Abstracta* (AST). Este AST es una representación intermedia del código fuente, que facilita su manipulación y transformación en código Java.

Back-End

Análisis Semántico

El análisis semántico es una etapa crucial en el proceso de compilación que se encarga de verificar que un programa escrito en un lenguaje de programación tenga sentido desde el punto de vista de su significado.

El mismo gatilla distintas subtareas requeridas para poder analizar correctamente el código fuente ingresado. Una vez generado el AST, es posible recorrer cada nodo del mismo, y analizar su formación, inferir los tipos, decidir qué reglas se deben aplicar a partir de los tipos y estructuras obtenidos en el árbol, pues una gramática no es suficiente para analizar un lenguaje como Python.
Es necesario realizar análisis de mayor profundidad para poder hacer inferencias correctas y que el código final obtenido sea uno coherente con el que un usuario hubiese querido realizar para ese código.

El sistema de tipos de Python es uno de los menos estrictos que existe hoy en día, eso deja el desafío de inferir tipos en las expresiones, poder realizar operaciones que resultan equivalentes en Java, pero que en Python vienen garantizadas a un nivel sintáctico, lo cual lo vuelve difícil de analizar.

Generación de Código

La fase final es la generación de código, donde el AST se transforma en código Java.
En Python, a diferencia de Java, no se requiere la declaración de una función inicial en donde la ejecución del programa inicie. El programa simplemente ejecuta el programa desde la primera línea de código. Para la generación de código Java se requirió entonces declarar tanto una clase que contenga el programa, cómo una función de inicio "*Main*" que contenga el código que en Python se encuentra por fuera de declaraciones de funciones.

Ejemplo de Traducción
Considere el siguiente código en Python

```
x = 13

def miFuncion():
       return x * 2


x = miFuncion(x)
```

Teniendo en cuenta que una función no puede declararse dentro de otra función en Java, la definición se debería agregar por fuera de la función "*Main*". El compilador debe traducir lo anterior al siguiente fragmento.

```
public class Main {
	public static void main(String[] args){
		int x = 13;
		 x = miFuncion(x);

	}

	 public static double miFuncion(){
		 return x * 2;
	}
}
```

Cómo se puede ver el código requiere inferir los tipos de datos en múltiples operaciones. Esto presentó dificultades, algunas de las cuales se detallan en el siguiente apartado

Desafíos, Limitaciones y Futuras Extensiones

Con el tiempo limitado y la complejidad nuestro objetivo principal fue poder traducir las variables y las funciones. Uno de los desafíos de esto fue manejar la tipificación estática de Java frente a la tipificación dinámica de Python. En Python, una variable puede cambiar de tipo en tiempo de ejecución, mientras que en Java, el tipo debe ser declarado y no puede cambiar. Esto es algo que dado su complejidad y el tiempo disponible, no logramos implementar.

Además, ciertas características de Python, como la flexibilidad en la definición de listas y diccionarios, no tienen una correspondencia directa en Java. Esto nos obligó a implementar el uso del tipo genérico Object.

Debido a la limitación en la traducción de arrays y otros iterables, solo se dispone del ciclo “for” para iterar sobre objetos de tipo String.

La tabla de símbolos utilizada es global, lo que impide tener variables con el mismo nombre en distintos alcances. Esto puede causar conflictos en programas más complejos que dependen de la capacidad de reutilizar nombres de variables en diferentes contextos.

No hay soporte para clases en el compilador, lo que restringe la capacidad de traducir programas orientados a objetos completos de Python a Java.
También, para poder traducir un programa de Python simple con funciones declaradas a un programa de Java funcional, se decidió utilizar una lista para guardar las funciones y así poder imprimirlas fuera de la función main en Java.

En cuanto a la sintaxis del programa de entrada, se encontraron problemas para interpretar las indentaciones de Python apropiadamente, por lo que finalmente se optó por recibir “@{“ y “@}” como limitadores de bloques de código.

Debido a las dificultades presentadas durante el desarrollo, y la fuerte limitación en funcionalidades que el compilador puede procesar, se han modificado los programas de prueba, de tal manera que estén acorde a las limitaciones del proyecto. Esto incluye la eliminación de programas de pruebas que evalúan la correcta implementación de bloques *try-catch*, por ejemplo, que no están en el alcance del proyecto.

Finalmente, no todas las operaciones aritméticas están implementadas, lo que limita la capacidad de traducir ciertos cálculos matemáticos complejos de Python a Java, a su vez el uso de varios returns en una funcion puede llegar a tener comportamientos i. Esta limitación afecta la funcionalidad y la precisión de algunos programas traducidos.

Estas limitaciones reflejan los desafíos técnicos y las restricciones de tiempo encontradas durante el desarrollo del compilador Tetralepis y sientan a su vez la base para futuras extensiones.

Conclusión

El compilador Tetralepis representa un esfuerzo para facilitar la traducción de código Python a Java, enfrentando los retos de la tipificación y las diferencias entre los lenguajes. Aunque se trata de un subconjunto de Python, este proyecto sienta las bases para futuras expansiones y mejoras en la compatibilidad y funcionalidad.

Bibliografía

- *The Python Standard Library*. (2024, Junio). The Python Software Foundation. Python documentation. https://docs.python.org/3/library/index.html
- *Python: Myths about Indentation*. (2008). Secnetix.de. Consultado en Junio, 2024. https://web.archive.org/web/20080825063858/https://www.secnetix.de/olli/Python/block_indentation.hawk