Construcción de Compiladores - Laboratorio Guiado No. 3

Gabriel Brolo, Bidkar Pojoy

Julio de 2024

1. Introducción

Los conceptos de compiladores todavía son fundamentales hoy en día. Estos conceptos permiten la traducción de lenguajes de alto nivel a código máquina ejecutable, lo cual es esencial para el desarrollo de software y la resolución a nuevos problemas modernos. Hoy en día, los lenguajes específicos de dominio (*DSL: Domain Specific Language*) juegan un papel crucial al permitir a los ingenieros realizar operaciones especializadas de manera eficiente y efectiva.

2. ¿Qué es un DSL?

Es un lenguaje de programación o especificación dedicado a un dominio particular de problemas, a diferencia de un lenguaje de propósito general que es destinado a cualquier tipo de desarrollo de software. Los DSLs están diseñados para ser altamente eficientes y efectivos dentro de su dominio de aplicación.

2.1. Ejemplos de DSLs

1. **SQL (Structured Query Language)**: Utilizado para gestionar y manipular bases de datos relacionales.
2. **HTML (HyperText Markup Language)**: Usado para crear y estructurar secciones, párrafos y enlaces en páginas web.
3. **CSS (Cascading Style Sheets)**: Utilizado para describir la presentación de un documento escrito en HTML o XML.
4. **LaTeX**: Un sistema de preparación de documentos de alta calidad utilizado principalmente para la producción de documentos científicos y técnicos.
5. **Regex (Regular Expressions)**: Un lenguaje para especificar patrones de búsqueda en textos.
6. **Makefile**: Utilizado para definir un conjunto de tareas a ejecutar en un archivo de configuración, comúnmente utilizado en compilación de código.
7. **VHDL (VHSIC Hardware Description Language)**: Utilizado para describir el comportamiento y la estructura de sistemas electrónicos.
8. **Verilog**: Otro lenguaje de descripción de hardware, similar a VHDL.
9. **MATLAB**: Un lenguaje de programación y entorno de desarrollo para análisis numérico y cálculo científico.
10. **R**: Un lenguaje de programación y entorno de software para computación estadística y gráficos.

3. Diseño de un DSL para Reservas de Sala de Conferencias

Imaginemos que necesitamos un lenguaje específico para gestionar las reservas de salas de conferencias en una empresa. Llamaremos a nuestro DSL "**ConfRoomScheduler**".

3.1. Gramática de ConfRoomScheduler

```
grammar ConfRoomScheduler;

prog: stat+ ;

stat: reserve NEWLINE                # reserveStat
    | cancel NEWLINE                 # cancelStat
    | NEWLINE                        # blank
    ;

reserve: 'RESERVAR' ID 'PARA' DATE 'DE' TIME 'A' TIME ;

cancel: 'CANCELAR' ID 'PARA' DATE 'DE' TIME 'A' TIME ;

DATE: [0-9]{2}'/'[0-9]{2}'/'[0-9]{4} ;
TIME: [0-9]{2}':'[0-9]{2} ;
ID  : [a-zA-Z0-9]+ ;
NEWLINE: '\r'? '\n' ;
WS  : [ \t]+ -> skip ;
```

3.2. Compilando el Lenguaje

3.2.1. Generando el Analizador Léxico y Sintáctico

Ejecute el siguiente comando dentro de un contenedor de Docker (o en su entorno) para generar el código del analizador:

```
antlr4 ConfRoomScheduler.g4 -o gen
```

3.2.2. Compilando el Código Generado

Utilice el compilador de Java para compilar el código generado:

```
javac gen/*.java
```

3.3. Probando el Lenguaje

Cree un archivo de prueba 'test.confroomdsl' con el siguiente contenido:

```
RESERVAR Sala101 PARA 12/06/2024 DE 10:00 A 12:00
CANCELAR Sala101 PARA 12/06/2024 DE 10:00 A 12:00
```

Ejecute el siguiente comando para interpretar el archivo de prueba:

```
grun ConfRoomScheduler prog -gui test.confroomdsl
```

4. Código en Python para Analizar el DSL

El siguiente código le permite inicialmente interpretar la gramática de ANTLR proporcionada en Python, para que usted pueda después agregar Reglas Semánticas:

```
import sys
from antlr4 import *
from gen.ConfRoomSchedulerLexer import ConfRoomSchedulerLexer
from gen.ConfRoomSchedulerParser import ConfRoomSchedulerParser

def main():
    input_stream = FileStream(sys.argv[1])
    lexer = ConfRoomSchedulerLexer(input_stream)
    stream = CommonTokenStream(lexer)
    parser = ConfRoomSchedulerParser(stream)
    tree = parser.prog()
    print(tree.toStringTree(recog=parser))

if __name__ == '__main__':
    main()
```

5. Agregando una Regla Semántica

5.1. Concepto breve de Reglas Semánticas

En el diseño y construcción de compiladores, las reglas semánticas se utilizan para garantizar que el programa fuente no solo sea sintácticamente correcto, sino también semánticamente válido. Esto significa que el programa debe tener sentido dentro del contexto del lenguaje y sus reglas.

5.2. Implementación con ANTLR

Agregue una regla semántica para validar que la hora de inicio de una reserva es anterior a la hora de fin. A continuación se le proporciona el código base donde se debe agregar la regla (*en este enfoque utilizaremos un Listener de ANTLR*):

```
import sys
from antlr4 import *
from gen.ConfRoomSchedulerLexer import ConfRoomSchedulerLexer
from gen.ConfRoomSchedulerParser import ConfRoomSchedulerParser
from gen.ConfRoomSchedulerListener import ConfRoomSchedulerListener

class ConfRoomSchedulerSemanticChecker(ConfRoomSchedulerListener):
    def enterReserveStat(self, ctx):
        # Aqui debe colocar su codigo para validar que
        # se cumpla con el requerimiento solicitado
        # Puede quitar el pass despues
        pass

def main():
    input_stream = FileStream(sys.argv[1])
    lexer = ConfRoomSchedulerLexer(input_stream)
    stream = CommonTokenStream(lexer)
```

```
    parser = ConfRoomSchedulerParser(stream)
    tree = parser.prog()

    semantic_checker = ConfRoomSchedulerSemanticChecker()
    walker = ParseTreeWalker()
    walker.walk(semantic_checker, tree)

if __name__ == '__main__':
    main()
```

6. Actividades

1. Cree un programa que reserve una sala de conferencias.
2. Cree un programa que cancele una reserva de sala de conferencias.
3. Experimente con varias reservas y cancelaciones en un mismo programa.
4. Modifique el DSL para incluir el nombre del solicitante de la reserva.
5. Agregue manejo de errores para detectar fechas u horas inválidas.
6. Cree un programa que incluya reservas solapadas y verifique su manejo (para validar reservaciones traslapadas, use un listener de ANTLR en Python; el listener llevará la cuenta de las reservaciones y validará cada nueva reservación en contra de las existentes).
7. Extienda el DSL para soportar descripciones de eventos.
8. Agregue validaciones adicionales como restricciones de tiempo de uso máximo.
9. Implemente una funcionalidad para listar las reservas existentes.
10. Cree un programa que utilice todas las características extendidas del DSL.
11. Añada soporte para diferentes tipos de salas (por ejemplo, sala de juntas, sala de capacitación).
12. Implemente un sistema de notificaciones para reservas próximas.
13. Extienda el DSL para permitir la reprogramación de reservas.
14. Cree un programa que reprograme una reserva existente y valide el cambio (para validar reservaciones traslapadas, use un listener de ANTLR en Python que ya creó en una actividad anterior; el listener llevará la cuenta de las reservaciones y validará cada nueva reservación en contra de las existentes).

7. Entregables

- Documento PDF con capturas de pantalla de la ejecución del ambiente en Docker, o de su propio entorno, para realizar las actividades solicitadas.
- El documento debe de incluir el enlace a un repositorio privado de Github con todo el código utilizado para la realización de este laboratorio guiado. Recuerde que debe de compartir todos los repositorios utilizados en este curso con el catedrático y los auxiliares. La información de los usuarios se encuentra en Canvas.

8. Rúbrica de Evaluación

Tarea	**Puntos**
Crear un programa que reserve una sala de conferencias	5
Crear un programa que cancele una reserva de sala de conferencias	5
Experimentar con varias reservas y cancelaciones	10
Modificar el DSL para incluir el nombre del solicitante	10
Agregar manejo de errores para fechas o horas inválidas	10
Crear un programa que incluya reservas solapadas	10
Extender el DSL para soportar descripciones de eventos	10
Agregar validaciones adicionales como restricciones de tiempo	10
Implementar funcionalidad para listar reservas existentes	10
Crear un programa que utilice todas las características extendidas	10
Añadir soporte para diferentes tipos de salas	10
Implementar un sistema de notificaciones para reservas próximas	10
Extender el DSL para permitir la reprogramación de reservas	10
Crear un programa que reprograma una reserva existente	20
Total	**140**

- **Nota:** Solo se permite obtener un 100 % de la nota total, por lo que elija las actividades que más le interesen.
- La respuesta a la sección 5.2 es **obligatoria**. Sin ella, no hay calificación.

9. Notas adicionales

- Este laboratorio proporciona indicaciones y código para usar ANTLR.
- Si lo desea, puede hacerlo con otras herramientas de su elección (Lex + Yacc, Flex, Bison, etc.)
- En este caso, explique bien y muestre todo su código.
- Este laboratorio se puede hacer en parejas. De ser así ambos deben de subir el trabajo e indicar explícitamente quién es su pareja.