Universidad del valle de Guatemala
Sebastian Huertas 22295

Análisis de una gramática ANTLR y su Driver

1. Estructura típica de un archivo .g4

Encabezado: grammar Nombre; declara el nombre de la gramática; el archivo debe llamarse Nombre.g4.

Opciones: bloque options {} configura el código generado (por ejemplo, el lenguaje destino).

Import: import OtrasGrammars; permite reutilizar reglas de otras gramáticas. **tokens**: lista de tokens literales que no necesitan regla léxica.

Acciones (@header, @members, @parser): código embebido que se copia en las clases generadas.

Reglas del parser: empiezan con minúscula; definen la estructura sintáctica.

Reglas del lexer: empiezan con mayúscula; definen cómo se dividen los caracteres en tokens.

2. Elementos clave de la sintaxis ANTLR

|: separa alternativas dentro de una regla.

()*, ()+, ()?: cuantificadores.

#: etiqueta alternativas para que el visitante/listener produzca nodos con nombres claros.

skip: instrucción de acción léxica para descartar un token (útil para espacios o comentarios).

fragment: subregla léxica reutilizable que no genera token propio.

3. Flujo de trabajo con ANTLR (python)

- **Generar**: antlr4 -Dlanguage=Python3 Nombre.g4 crea NombreLexer.py, NombreParser.py y otros archivos auxiliares.
- **Instalar/compilar**: pip install antlr4-python3-runtime (no se compila con javac; solo se necesita el runtime de ANTLR para Python).
- **Ejecutar**: lanzar un *driver* con python main.py <archivo_entrada>; el script alimenta la entrada al lexer y al parser, obtiene el ParseTree y (opcionalmente) lo recorre con un Listener o Visitor para realizar acciones semánticas.

4. Driver típico (driver.py)

```
input_stream = FileStream(sys.argv[1]); lexer = NombreLexer(input_stream); tokens = CommonTokenStream(lexer); parser = NombreParser(tokens); tree = parser.program(); ParseTreeWalker().walk(MyListener(), tree);
```

El *driver* (versión main.py):

- Crea un flujo de caracteres (**FileStream**).
- Instancia el lexer para transformar caracteres en tokens.
- Genera un **CommonTokenStream** que alimenta al parser.
- Invoca la regla inicial del parser y obtiene un **ParseTree**.
- Opcionalmente recorre el árbol con un **Listener** o **Visitor** para ejecutar acciones semánticas.

5. Resumen

- Un archivo .g4 agrupa reglas léxicas y sintácticas más acciones de usuario.
- Los cuantificadores y operadores de unión modelan la ambigüedad y repetición en la sintaxis. Las etiquetas # nombran las alternativas para producir nodos claros en el AST.
- El Driver orquesta lexer •→ tokens → parser → árbol → acciones (listener/visitor).