Informe del Compilador

1. Introducción

El presente informe detalla el funcionamiento y las características del compilador desarrollado, destacando su estructura, funcionamiento y manejo de errores. Además, se incluye un análisis sobre los resultados obtenidos y posibles mejoras futuras para optimizar el sistema.

2. Estructura del sistema

El sistema consta de los siguientes componentes principales:

- **App.java**: Es el punto de entrada del programa, encargado de cargar el archivo de entrada (`codigo.txt`) y procesarlo mediante las reglas definidas en el archivo `compilador.g4`.
- **compilador.g4**: Archivo que define la gramática utilizada por ANTLR para generar el analizador léxico y sintáctico. Contiene las reglas necesarias para validar y analizar el código fuente.
- **codigo.txt**: Archivo que contiene el código fuente a analizar. Este código debe cumplir con la gramática especificada en `compilador.g4` para ser procesado correctamente.

3. Funcionamiento

El proceso de compilación se lleva a cabo en los siguientes pasos:

1. **Lectura del archivo**: `App.java` se encarga de leer el contenido de `codigo.txt` y pasarlo al analizador.
2. **Análisis léxico y sintáctico**: Usando las reglas definidas en `compilador.g4`, ANTLR genera analizadores que verifican que el código cumpla con la gramática establecida.
3. **Generación de código intermedio**: Si el análisis es exitoso, el sistema genera un código de tres direcciones como representación intermedia.

4. **Optimización**: El código intermedio se optimiza aplicando técnicas como eliminación de redundancias, simplificación de expresiones y reorganización de instrucciones para mejorar el rendimiento del código final.

4. Manejo de errores

El sistema está diseñado para detectar y reportar errores léxicos y sintácticos. Cuando se encuentra un error, el compilador genera un mensaje descriptivo indicando la línea y la posición donde ocurrió. Esto permite al usuario corregir el código de entrada de manera eficiente.

Tipos de errores manejados:

- **Errores léxicos**: Caracteres no reconocidos por la gramática.
- **Errores sintácticos**: Secuencias de tokens que no cumplen con las reglas definidas.

5. Código de tres direcciones y optimización

El sistema genera un código de tres direcciones como representación intermedia. Este tipo de código es fácil de interpretar y optimizar, ya que descompone las instrucciones en operaciones simples. A continuación, se detallan las principales optimizaciones aplicadas:

1. **Eliminación de redundancias comunes**:
    - Identificación de subexpresiones repetidas para calcularlas una sola vez y reutilizar el resultado.
2. **Simplificación de expresiones**:
    - Reducción de expresiones algebraicas triviales, como multiplicaciones o sumas con valores constantes.
3. **Reordenamiento de instrucciones**:
    - Reorganización de operaciones para minimizar el tiempo de ejecución, respetando la semántica del programa.
4. **Propagación de constantes**:
    - Sustitución de variables cuyo valor es constante por dicho valor, reduciendo el número de accesos a memoria.

Ejemplo de optimización

Código original:

```
t1 = a + b
t2 = a + b
c = t1 + d
```

Código optimizado:

```
t1 = a + b
c = t1 + d
```

Estas optimizaciones no solo mejoran el rendimiento, sino que también reducen el tamaño del código generado, facilitando su ejecución en sistemas con recursos limitados.

6. Resultados obtenidos

El compilador ha demostrado ser efectivo en la detección de errores y en la generación de código optimizado. Durante las pruebas, se logró procesar códigos de ejemplo correctamente, detectando errores cuando fue necesario y generando código intermedio optimizado.

7. Conclusión y mejoras futuras

Si bien el sistema cumple con los objetivos planteados, se identificaron posibles mejoras:

1. **Ampliación de la gramática**: Incluir soporte para más estructuras del lenguaje, como bucles y funciones.
2. **Optimización avanzada**: Implementar técnicas más complejas, como análisis de flujo de datos o paralelización de código.
3. **Generación de código máquina**: Extender el sistema para traducir el código intermedio a instrucciones específicas de una arquitectura.
4. **Interfaz gráfica**: Crear una herramienta visual para facilitar la interacción con el compilador y mostrar el código optimizado de manera intuitiva.

En conclusión, el sistema desarrollado representa un paso importante hacia la construcción de un compilador completo, con potencial para adaptarse a proyectos más ambiciosos.