Compilador para Kotlin-Parte 2

0. Identificação

Grupo: T02_G05

Nomes: André Filipe Carvalho Guedes Borges Adrêgo
Bruno Alexandre Magalhães Costa

1. Introdução

O projeto incorporou diversas melhorias e ajustes na primeira parte para suportar melhor o fluxo do compilador, incluindo:

1. Remoção de Strings: Decidimos simplificar o compilador nesta etapa, eliminando o suporte a strings tanto no analisador léxico quanto na análise semântica.
2. Descontinuação do Tipo Float:Embora planejado, o suporte a float foi descartado, focando apenas em int para facilitar a implementação.
3. Adição do Unary Minus: Implementamos o suporte ao operador de negação unária (-), tanto na gramática quanto nas fases de geração de código, permitindo o uso de valores negativos nas expressões.
4. Melhoria da Gramática para Comandos If: Refinamos a gramática do comando if para maior flexibilidade, adicionando suporte a blocos opcionais e simplificando a estrutura de comandos.

Este projeto implementa as fases finais de um compilador para um subconjunto da linguagem **Kotlin**, focando nos seguintes aspectos:

1. **Construção da Tabela de Símbolos:** Gerenciar informações semânticas, como tipos e mutabilidade de variáveis.
2. **Geração de Código Intermediário:** Traduzir a Árvore Sintática Abstrata (AST) para código intermédio de 3 endereços **(TAC).**
3. **Geração de Código MIPS:** Converter o TAC em código MIPS, permitindo execução em simuladores como **MARS**.

2. Estrutura do Projeto

O projeto é organizado em módulos, cada um correspondendo a uma etapa específica do processo de compilação. Essa estrutura modular facilita a manutenção, o entendimento e a extensão do compilador. A seguir, detalhamos cada módulo e sua responsabilidade:

1. **Lexer (Analisador Léxico)** feito na 1ª parte do projeto.
2. **Parser (Analisador Sintático)** feito na 1ª parte do projeto.
3. **Semantics (Análise Semântica):**
    - Implementa a verificação semântica usando uma **Tabela de Símbolos**.
    - Verifica:
        - Declaração e uso de variáveis.
        - Compatibilidade de tipos em operações e atribuições.
    - Armazena informações sobre os identificadores.
4. **CodeGen (Gerador de Código Intermediário):**
    - Produz um conjunto de instruções intermediárias no formato de **Três Endereços (TAC)**.
    - Cria rótulos e reutiliza temporários sempre que possível, otimizando a geração de código intermediário e facilitando a tradução para o código de máquina.
5. **MipsGen (Gerador de Código MIPS):**
    - Converte as instruções intermediárias (TAC) em código MIPS.
    - Implementa operações aritméticas, lógicas, condicionais e controle de fluxo (saltos e rótulos).
    - Gera código para manipulação de entrada e saída (ex. leitura de valores e impressão).
    - Produz o código MIPS final, que pode ser executado em simuladores como MARS.
6. **Main:**
    - Desempenha um papel central, pois é responsável por integrar as diferentes fases do processo de compilação e apresentar os resultados de cada etapa.
7. **Arquivos de Configuração e Testes:**
    - Contém exemplos de código-fonte na linguagem de entrada, usados para testar as etapas do compilador.
    - Scripts para validar o pipeline completo, do código-fonte até a execução do código MIPS.

3. Instruções para Compilação e Execução:

- Escrever o código Kotlin em um arquivo de texto, por exemplo, program.kt.
- Redirecione o arquivo como entrada para o programa compilador usando **cat program.kt | runhaskell Main.hs**

Conclusão

O projeto de compilação desenvolvido permite transformar um subconjunto da linguagem Kotlin em código executável, passando por várias fases essenciais do processo de compilação. Desde a análise léxica e sintática até a geração de código intermediário e, finalmente, a tradução para código MIPS, o trabalho engloba os principais conceitos de um compilador. Em resumo, este projeto não só demonstrou o funcionamento básico de um compilador, mas também forneceu uma compreensão profunda das fases envolvidas na tradução de programas de alto nível para código de máquina.