Projeto de Compiladores - Parte 2: Geração de Código Intermediário e Código MIPS

Descrição Geral

Esta é a segunda parte do projeto de desenvolvimento de um compilador para um subconjunto da linguagem Kotlin. O foco desta etapa está na geração de código intermédio de 3 endereços (IR) e na tradução deste para código de máquina em assembly MIPS, permitindo a execução dos programas Kotlin em um simulador como o MARS.

Ficheiros Principais

1. IR.hs

Define as instruções da Representação Intermédia (IR) para representar o programa de forma mais próxima ao código de máquina neste caso no código intermédio de 3 endereços como referido nas aulas.

- **Principais Instruções:**
    - **MOVE Temp Temp:** Move o valor de um temporário para outro.
    - **MOVEI Temp Int:** Atribui um valor inteiro a um temporário.
    - **OP BinOp Temp Temp Temp:** Executa operações artiméticas, booleanas, comparações, etc...
    - **COND Temp BinOp Temp Label Label:** Avalia uma condição e realiza saltos condicionais.
    - **CALL_NOARGS Temp String:** Usada principalmente para o readln(), pois esta função nao recebe argumentos.
    - **CALL1 Temp Temp:** Representa chamadas simples de funções no programa como por exemplo: print(x).

2. CodeGen.hs

Responsável por traduzir a AST para código intermédio de 3 endereços.

- **Principais Funcionalidades:**
    - Gerar Temporários e as labels: Garante identificadores únicos para variáveis temporárias e labels.

- **Tradução de Instruções**: Suporta comandos como val, var, atribuições, expressões condicionais, loops (if, while) e impressão de inteiros ou strings (print).
- **Gestão da Tabela de Símbolos:** Mapeia variáveis para identificar os temporários correspondentes a estas.
- **Análise Semântica:** Verifica se um var ou val foi declarado antes de ser utilizado no código e caso não aconteça gera um erro e identifica a variável que não foi declarada.

3. MachineGen.hs

Converte a lista de instruções de 3 endereços em código assembly MIPS.

- **Principais Funcionalidades:**
  - **Traduzir as Instruções:** Tradução de operações aritméticas, lógicas e controlar o fluxo para a linguagem MIPS.
  - **Secção .data:** Define strings e buffers se necessário, pois só vão ocorrer no código caso seja invocado um readln() daí ser necessário um buffer ou quando existirem strings no programa, graças a uma função que faz essa verificação no programa.
  - **Secção .text:** Gera o corpo principal e as funções do programa**.**
  - **Manipulação da Pilha:** Manipula os registos do programa.

4. Ficheiro de Teste (Main.hs)

- **Análise Lexical (Lexer.x):** Gera tokens a partir do código fornecido de exemplo.
- **Análise Sintática (Parser.y):** Constrói a AST a partir do parser utilizando os tokens anteriores gerados (estes dois ficheiros pertenciam à primeira parte do projeto).
- **Código intermédio gerado (CodeGen.hs):** Transforma a AST em código intermédio de 3 endereços.
- **Código MIPS gerado (MachineGen.hs):** Guarda o código assembly correspondente ao exemplo recebido como input no ficheiro criado “machine_code.txt”.

Como Executar

Pré-requisitos:

- Haskell instalado no sistema.
- Simulador MARS para testar o código MIPS.

Passos:

1. **Compile e execute o programa:**

ghc -o compiler Main.hs

./compiler < exs/ex1.kt (por exemplo, exs corresponde à pasta de exemplos em src)

2. **Após a execução:**
    - Tokens, AST e o Código Intermédio de 3 endereços serão exibidos no terminal.
    - O código máquina (MIPS) será guardado no ficheiro criado pela Main.hs "machine_code.txt".
3. **Para testar o código que foi salvo no "machine_code.txt" no MARS:**
    - Abrir a aplicação, criar um novo ficheiro.asm e colar o código do ficheiro gerado com o código MIPS correspondente do exemplo.
    - Execute o programa no simulador e verificar se corresponde ao esperado pelo programa em kotlin inicial.

Estrutura das Pastas

- src/: Contém os módulos principais (Lexer.x, AST.hs, Parser.hs, IR.hs, CodeGen.hs, MachineGen.hs e a Main.hs).
- exs/: Exemplos de programas na linguagem Kotlin usados para testes com utilização da instrução if then else, apenas if, while loop, expressões aritméticas ou booleanas, função print e readln e declaração e atribuição de vals ou vars sendo que também aceita códigos de exemplo com ou sem ";" no fim de cada linha. Contém também dois ficheiros que geram erro devido à análise semântica do programa que foi desenvolvida no CodeGen.hs (compiler: Variable not defined: z em erro2.kt).

Referências

- **Documentação Kotlin:** Kotlin Basics
- **Onde instalar o Simulador MARS para verificação:** GitHub