Compilador para Subconjunto de Kotlin

Trabalho Prático – Parte 1

TP1_T01_G12

Descrição

Esta primeira parte do projeto implementa um compilador para um subconjunto da linguagem de programação Kotlin. A implementação inclui as fases de análise léxica (scanner) e análise sintática (parser), e retorna a Abstract Syntax Tree (AST) representando o código Kotlin de entrada.

O subconjunto da linguagem Kotlin suportado pelo compilador inclui:

- **Expressões**: expressões aritméticas, expressões booleanas, e as funções `print` e `readln`.
- **Comandos**: função `main`, declarações e atribuições de variáveis, expressões condicionais (`if-else if-else`) e laços `while`.

O código foi desenvolvido em Haskell e conta com três arquivos principais: Lexer.x, Parser.y e AST.hs.

Estrutura do Código

- **Main.hs**: É o ponto de entrada para o programa, que vai orquestrar o processo de leitura de um arquivo Kotlin (testes), passar o conteúdo através do analisador léxico e sintático, e então gerar a AST.
- **TestN.kt**: Arquivos numerados de 1 a 5 (N) com exemplos em Kotlin para serem usados como input e gerarem a árvore AST.
- **Lexer.x**: Define a gramática léxica de um subconjunto da linguagem Kotlin. Ele especifica como os tokens são identificados, incluindo literais, operadores, delimitadores, palavras-chave e identificadores. O Lexer também ignora espaços em branco e comentários. Dessa forma, Ao compilar o arquivo Lexer.x com a ferramenta Alex, é gerado um arquivo em Haskell, Lexer.hs, que é responsável por processar o código-fonte de entrada e retornar uma lista de tokens correspondentes.
- **Parser.y**: Define a gramática sintática da linguagem Kotlin utilizada, especificando as regras para identificar declarações de variáveis, expressões aritméticas, expressões booleanas, entre outras. Usa

operadores de precedência para resolver ambiguidades na análise. O Parser processa a lista de tokens gerada pelo Lexer e constrói a AST, interpretando as estruturas de controle e expressões especificadas no subconjunto de Kotlin. As instruções são escritas utilizando a ferramenta Happy, que, ao ser compilada, gera o arquivo Parser.hs em Haskell, responsável por transformar a lista de tokens em uma AST.

- **AST.hs**: Define a estrutura da Abstract Syntax Tree (AST), que é uma representação hierárquica e simplificada do código-fonte de um programa. A AST desenvolvida organiza o programa em componentes essenciais, como declarações (*Decl*), comandos (*Stmt*) e expressões (*Exp*). Essas construções formam a base para futura análise semântica e, eventualmente, para a geração de código.
- **arquivoX.o, arquivoX.hi**: são arquivos adicionais criados e utilizados durante a compilação. Portanto, são gerados automaticamente pelo compilador, não sendo desenvolvidos pelo nosso grupo.

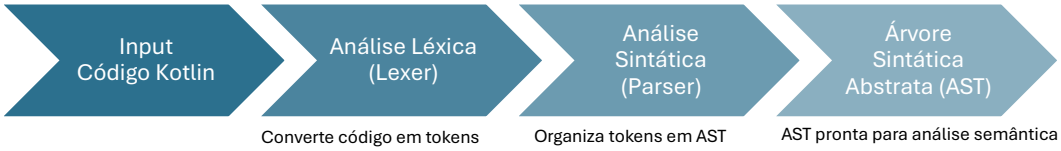

Imagem 1: Fluxograma do Compilador para Subconjunto de Kotlin

Instalação e Execução

Pré-requisitos

- GHC (Glasgow Haskell Compiler) para executar o código em Haskell.
- Cabal para gerenciamento de dependências.
- Alex (gerador léxico) para criar o analisar léxico.
- Happy (gerador de parser) para criar a AST.

1. Instalação GHC e Cabal

Para Windows:

Execute em uma sessão do PowerShell (como um usuário não-administrador):

```
$ Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePoi
ntManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityP
rotocol -bor 3072; try { & ([ScriptBlock]::Create((Invoke-WebRequest https
://www.haskell.org/ghcup/sh/bootstrap-haskell.ps1 -UseBasicParsing))) -Int
eractive -DisableCurl } catch { Write-Error $_ }
```

Para Linux/Ubuntu:

Execute no terminal:

```
$ sudo add-apt-repository -y ppa:hvr/ghc
$ sudo apt-get update
$ sudo apt-get install -y ghc cabal-install
```

2. Instalação Alex e Happy:

Usando Cabal:

```
$ cabal update
$ cabal install alex
$ cabal install happy
```

3. Verificação da instalação:

```
$ ghc --version
$ cabal --version
$ alex --version
$ happy --version
```

4. Executar o Compilador:

Gerar o analisador léxico a partir do arquivo de especificação Lexer.x, criando o arquivo Lexer.hs. a partir do Alex:

```
$ alex Lexer.x -o Lexer.hs
```

Gerar o código para o analisador sintático (parser) a partir do arquivo de especificação Parser.y, criando o arquivo Parser.hs. a partir do Happy:

```
$ happy Parser.y -o Parser.hs
```

Compilar o arquivo Main.hs usando o GHC, gerando o executável Main:

```
$ ghc Main.hs -o Main
```

Fornecer um arquivo de entrada com código Kotlin para que o compilador retorne a AST correspondente:

Para Windows:

```
$ Main.exe < Test1.kt
```

Para Linux/Ubuntu:

```
$ .\Main Test1.kt
```

Exemplo de Código

Exemplo de código Kotlin que pode ser interpretado pelo compilador:

```
fun main() {
    val x: Int = 10
    var y: Double = 20.5
    val isTrue: Boolean = true

    if (x > 5 && y < 25.0) {
        print("Condition met")
    } else {
        print("Condition not met")
    }
}
```

Esse código gera uma AST representando o fluxo de controle e operações declaradas. AST gerada:

```
Program [FunDecl [ValDeclCT "x" TpInt (Num 10),VarDeclCT "y" TpDouble
(DoubleLit 20.5),ValDeclCT "isTrue" TpBool (BoolLit True),IfElse (And
(GreaterThan (Var "x") (Num 5)) (LessThan (Var "y") (DoubleLit 25.0)))
(StmtLi [Print (StringLit "Condition met")]) (StmtLi [Print (StringLit
"Condition not met")])]]
```

Conclusão

Nessa primeira etapa do projeto, foram trabalhados os conceitos e analisadores léxico e sintático (parser), sendo que o objetivo é gerar a AST a partir dos exemplos fornecidos. Dessa forma, é possível extrair uma lista de tokens gerando uma árvore abstrata para um subconjunto de Kotlin.

Autores

- Gustavo Valandro da Rocha – up202202043
- Marthina Oliveira de Castro – up201711094