Relatório do Projeto Compiladores – 2023.2

ParMini - Interpretador MiniPar

Jonatan Leite Alves,
José Victor Dias da Silva Oliveira,
Lucas De Min e
Rayssa M. Roseno.

1. Enunciado

O presente relatório tem como objetivo documentar o desenvolvimento do interpretador MiniPar, uma ferramenta destinada a executar programas escritos na linguagem MiniPar. Esta linguagem foi concebida para suportar a execução de instruções tanto sequenciais quanto paralelas, viabilizando, adicionalmente, a comunicação entre computadores por meio de canais específicos.

Contexto

O interpretador MiniPar foi concebido como parte de um projeto acadêmico que explora conceitos avançados em compiladores e linguagens de programação. O MiniPar oferece um ambiente flexível para a execução de programas concorrentes, promovendo a eficiência na execução de tarefas distribuídas.

Características Principais

- Execução Sequencial e Paralela: O MiniPar permite a execução de blocos de código de maneira sequencial e paralela, proporcionando ao programador a capacidade de criar aplicações concorrentes.
- Comunicação entre Computadores: A linguagem MiniPar implementa canais de comunicação, permitindo a troca de mensagens entre diferentes computadores. Esse recurso é implementado por meio de sockets em Python, garantindo uma comunicação eficaz.
- Suporte a Threads: A execução paralela no MiniPar é implementada utilizando threads, o que possibilita a execução simultânea de blocos de código em um mesmo computador.
- Tipos de Variáveis: O interpretador suporta variáveis booleanas, inteiros e strings, proporcionando versatilidade para o desenvolvimento de uma ampla gama de programas.

Desafios e Metodologia

O desenvolvimento do MiniPar envolve a conclusão da Gramática BNF, a implementação dos analisadores léxico, sintático e semântico, além da execução de programas de teste específicos. Detalhes sobre a gramática e as produções podem ser encontrados na documentação fornecida.

2. Introdução

O interpretador MiniPar, carinhosamente chamado de "ParMini", foi concebido para atender à demanda por uma linguagem de programação que oferece suporte a execução sequencial e paralela. Este relatório explora a estrutura do ParMini, desde sua gramática até a implementação de funcionalidades como a execução simultânea de threads.

3. Gramática

```
programa_minipar ::= bloco_stmt



bloco_stmt ::= bloco_SEQ | bloco_PAR



bloco_SEQ ::= SEQ stmts



bloco_PAR ::= PAR stmts



stmts ::= stmt stmts | ε



stmt ::= atribuicao

        | if ( bool ) stmt

        | if ( bool ) stmt else stmt

        | while ( bool ) stmt

        | c_channel_decl

        | execucao
```

```
atribuicao ::= id = expr ;



expr ::= bool_expr | arith_expr | id | literal



bool_expr ::= bool_op arith_expr bool_expr_tail

bool_expr_tail ::= bool_op arith_expr bool_expr_tail | ε



arith_expr ::= term arith_expr_tail

arith_expr_tail ::= add_op term arith_expr_tail | ε



term ::= factor term_tail

term_tail ::= mul_op factor term_tail | ε



factor ::= ( expr ) | id | literal | c_channel_access



c_channel_decl ::= c_channel id id_comp1 id_comp2 ;



c_channel_access ::= c_channel . id_comp ;



execucao ::= SEQ { stmts }

            {

                # Lógica para execução sequencial

                # ...
```

```
            }
        | PAR { stmts }
            {
                # Lógica para execução simultânea das threads
                # incluir chamadas para a execução de  threads
                # Exemplo:
                def thread1():
                    # Lógica para Thread 1
                    print("Thread 1")

                def thread2():
                    # Lógica para Thread 2
                    print("Thread 2")

                # Iniciar as threads simultaneamente
                thread1.start()
                thread2.start()
            }

bool_op ::= && | || | == | != | < | <= | > | >=

add_op ::= + | -
```

```
mul_op ::= * | /



id ::= [a-zA-Z][a-zA-Z0-9_]*



literal ::= true | false | NUM | STRING



NUM ::= [0-9]+
```

4. Arquitetura de Software

A arquitetura do interpretador MiniPar foi concebida de maneira modular, sendo composta por quatro principais módulos que desempenham papéis cruciais durante o ciclo de vida de interpretação:

- **Lexer (Analisador Léxico)**: Responsável por examinar o código fonte e convertê-lo em uma sequência estruturada de tokens. Cada token representa uma unidade léxica, facilitando as fases subsequentes de análise.
- **Parser (Analisador Sintático)**: Encarregado de analisar a estrutura sintática do código, construindo uma árvore sintática que reflete a organização e relacionamentos entre os diversos elementos da linguagem MiniPar.
- **Interpreter (Interpretador)**: Assume a tarefa de executar as instruções representadas pela árvore sintática gerada pelo Parser. Aqui, o código-fonte é transformado em ações concretas, permitindo a execução sequencial ou paralela das instruções.
- **Error Handler (Gerenciador de Erros)**: Desenvolvido para gerenciar e tratar eventuais erros que possam ocorrer durante as fases de análise e execução. Este módulo contribui significativamente para a robustez e estabilidade do interpretador, fornecendo mensagens de erro elucidativas para facilitar a resolução eficiente de problemas.

5. Pseudocódigo do Analisador Léxico (Lexer)

```
class Lexer:

    def __init__(self, input_code):

        # Implementação do inicializador

    def tokenize(self):

        # Implementação da tokenização

# Exemplo de uso:

lexer = Lexer(input_code)

tokens = lexer.tokenize()
```

6. Pseudocódigo do Analisador Sintático (Parser)

```
class Parser:

    def __init__(self, tokens):

        # Implementação do inicializador

    def parse(self):

        # Implementação da análise sintática

# Exemplo de uso:

parser = Parser(tokens)

abstract_syntax_tree = parser.parse()
```

7. Tratamento de Erros

O interpretador MiniPar é dotado de um sistema de tratamento de erros robusto, projetado para assegurar uma experiência de usuário aprimorada. Esse mecanismo proporciona mensagens de erro precisas, claras e informativas, facilitando a identificação e resolução eficiente de potenciais problemas durante a execução dos programas em MiniPar.

8. Linguagem de Programação, IDE e GitHub

O ParMini foi desenvolvido utilizando a linguagem de programação Python, e a codificação foi realizada nos ambientes de desenvolvimento Sublime Text e Visual Studio Code. O código-fonte, juntamente com outros arquivos relevantes, está disponível para acesso público no repositório GitHub do projeto. Este repositório serve como uma fonte centralizada para colaboração, revisão de código e acompanhamento do desenvolvimento, proporcionando transparência e acessibilidade para todos os membros da equipe.