Universidade Federal do Mato Grosso
Instituto de Computação

TRABALHO DE TELEFAC

ANTHONY RICARDO RODRIGUES REZENDE
ALAN BRUNO MORAIS COSTA
VINICIUS PADILHA VIEIRA
ANDREY LUIGGI DA CRUZ

CUIABÁ, MATO GROSSO 2023

ANTHONY RICARDO RODRIGUES REZENDE
ALAN BRUNO MORAIS COSTA
VINICIUS PADILHA VIEIRA
ANDREY LUIGGI DA CRUZ

TRABALHO DE TELEFAC

Trabalho sobre "Analisador Sintático"conforme a disciplina de Teoria das Linguagens Formais e Autômatos, apresentado como requerido para a obtenção de nota na Universidade Federal de Mato Grosso - UFMT

Professor: Dr. Eduardo

CUIABÁ, MATO GROSSO 2023

Resumo

Apresentação da atividade avaliativa sobre Analisador Sintático, no formato de relatório.

1 Introdução

Neste trabalho, exploramos o universo da análise sintática através da implementação de um analisador sintático utilizando C++. A análise sintática é fundamental na compreensão de como sequências de símbolos se alinham às regras gramaticais de uma linguagem, similar a montar um quebra-cabeça seguindo um guia específico.

Adotamos a abordagem de construção de uma tabela de análise sintática, um recurso valioso que direciona as decisões durante a análise de cadeias de símbolos. O desenvolvimento em C++ foi escolhido por sua robustez e precisão, elementos chave para enfrentarmos este desafio computacional.

O objetivo deste relatório é mais do que apresentar um programa funcional; é compartilhar nossa trajetória de aprendizado, incluindo as adversidades encontradas e as soluções adotadas.

2 Relatório de Implementação (Criação)

2.1 Estruturas de Dados

O código implementa diversas estruturas de dados fundamentais para o processo de análise sintática, sendo elas:

- **Enumeração Simbolo**: Define os tipos de símbolos utilizados na gramática, incluindo terminais como ‘a‘ e ‘b‘, não-terminais como ‘S‘, ‘A‘, e ‘B‘, um símbolo especial para representar a produção vazia (‘epsilon‘), e o símbolo de fim de cadeia (‘cifrao‘).
- **Mapa tabelaDeAnalise**: Um mapeamento que associa pares de não-terminal e terminal (simbolizados por pares de *Simbolo*) a produções, que são representadas por vetores de *Simbolo*. Essa tabela é essencial para a análise sintática, indicando quais produções aplicar durante o processo.
- **Pilha pilhaDeAnalise**: Uma pilha usada para armazenar símbolos durante a análise da cadeia de entrada. A pilha facilita o controle do processo de análise, permitindo operações de inserção e remoção de elementos conforme as regras da gramática são aplicadas.
- **Vetor (vector):** Vetores são usados para armazenar as produções na tabela de análise sintática. Cada entrada no mapa é um vetor que representa os símbolos a serem empilhados na pilha de análise quando uma regra de produção é aplicada.

2.2 Algoritmos Implementados

O código baseia-se em dois algoritmos principais:

1. **Inicialização da Tabela de Análise Sintática**: Este algoritmo preenche a tabela de análise sintática com as produções da gramática. Utiliza-se um mapeamento direto entre os pares de não-terminal e terminal para suas respectivas produções, facilitando a consulta durante a análise da cadeia de entrada.
2. **Processamento da Cadeia de Entrada**: Algoritmo principal que realiza a análise sintática da cadeia de entrada. A estratégia empregada é a seguinte:
   (a) Inicializar a pilha de análise com o símbolo de fim de cadeia e o símbolo inicial da gramática.
   (b) Ler os símbolos da entrada, um a um, e compará-los com o elemento no topo da pilha.
   (c) Se o topo da pilha é um terminal, verifica-se a correspondência com o símbolo atual da entrada. Em caso positivo, ambos são removidos; caso contrário, reporta-se um erro.
   (d) Se o topo da pilha é um não-terminal, consulta-se a tabela de análise para encontrar a produção correspondente. A produção encontrada é então empilhada em ordem inversa (exceto por produções vazias).

(e) O processo continua até que a pilha esteja vazia e a entrada tenha sido completamente processada, indicando sucesso na análise.

Este método permite determinar se uma cadeia de entrada pertence à linguagem definida pela gramática.

3 Dificuldades Encontradas

As dificuldades encontradas foi criar as condições para implementar um analisador sintático LL(1).

- **Determinação de Terminais e Não-terminais:** O algoritmo precisar distinguir claramente entre terminais e não-terminais ao processar o topo da pilha. Essa distinção é crucial para decidir se deve-se avançar na entrada ou consultar a tabela de análise para uma produção.
- **Consultas à Tabela de Análise:** A necessidade de consultar a tabela de análise sintática e lidar com a ausência de uma regra de produção aplicável (indicando um erro) apresenta um desafio lógico no design do algoritmo.
- **Manuseio do Símbolo Epsilon ():** A introdução do símbolo epsilon para representar produções vazias adiciona complexidade, pois requer tratamento especial para remover o não-terminal da pilha sem avançar na entrada.
- **Condições de Erro:** Implementar as condições corretas para declarar uma entrada como não pertencente à linguagem (por exemplo, quando o terminal atual não corresponde ao topo da pilha ou quando não há produção aplicável para um não-terminal dado) requer atenção cuidadosa às condições de erro.

4 Aprendizados Adquiridos

- **Uso Avançado de Pilha:** Um uso mais complexo da estrutura de dados de pilha, mostrando como ela pode ser empregada para implementar algoritmos complexos de análise sintática. A manipulação da pilha, incluindo empilhamento e desempilhamento de símbolos de acordo com as regras de produção da gramática, é um ponto de aprendizado importante.
- **Manipulação de Estruturas de Dados Complexas:** O uso combinado de mapas e vetores para implementar a tabela de análise sintática oferece uma visão valiosa sobre como estruturas de dados podem ser usadas em conjunto para resolver problemas específicos de computação.

5 Conclusão

Concluir este projeto de análise sintática utilizando C++ foi desafiadora, porém gratificante. Foi uma oportunidade para não apenas desenvolver um analisador sintático funcional, mas também para aprofundar nosso entendimento sobre conceitos de análise sintática e aprimorar habilidades de programação, como pilha e mapa.

Em resumo, o algoritmo foi uma excelente lição para aprimorar nossos conhecimentos e evoluir como programadores, valeu professor.