Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Processsamento de Linguagens

Ano Letivo de 2023/2024

Compilador de Forth

António Filipe Castro Silva(a100533)
Flávio David Rodrigues Sousa(a100715)
Pedro Emanuel Organista Silva(a100745)

12 de maio de 2024

PL

1 Introdução

Este relatório documenta o processo de desenvolvimento de um compilador para a linguagem de programação Forth. O objetivo deste projeto é criar um compilador funcional capaz de traduzir programas escritos em Forth para código de máquina executável.

Forth é uma linguagem de programação de nível baixo e de pilha, conhecida pela sua simplicidade e eficiência.

O compilador Forth a ser desenvolvido neste projeto envolve várias etapas, incluindo análise léxica e sintática, geração de código intermediário e otimização de código. O processo de compilação será dividido em fases distintas, cada uma com seus próprios desafios e requisitos técnicos.

Este relatório irá abordar detalhadamente os seguintes aspectos do projeto:

1. **Análise Lexica e Sintática:** Identificação e análise dos tokens e estruturas de controle da linguagem Forth.
2. **Geração de Código Intermediário:** Transformação das estruturas de controle em uma forma intermediária de representação de código.
3. **Otimização de Código:** Aplicação de técnicas de otimização para melhorar a eficiência e desempenho do código gerado.
4. **Geração de Código de Máquina:** Tradução do código intermediário em código de máquina adequado à arquitetura de destino.

Ao longo deste relatório, serão apresentados exemplos de código, resultados de testes e discussões sobre as decisões de design e implementação tomadas durante o desenvolvimento do compilador.

Espera-se que este projeto proporcione uma compreensão aprofundada dos princípios de compilação e da linguagem Forth, além de oferecer uma oportunidade para aplicar os conceitos teóricos aprendidos em um contexto prático.

2 Análise Léxica e Sintática

2.1 Análise Léxica

De forma a dar inicio ao projeto, foi necessário disponibilizar algum tempo para a compreensão da linguagem FORTH. O objetivo principal com isso, centrava-se na capacidade de criar, de forma eficiente e coerente, um analizador léxico conciso capaz de analisar todo o léxico presente nesta linguagem e, detetar, desde já, algum erro de digitação de código. O analisador léxico desenvolvido é parte fundamental do processo de compilação para a linguagem Forth. A sua função é ler o código-fonte Forth fornecido e transformá-lo em uma sequência de tokens, que são os blocos básicos de construção da linguagem. Cada token representa uma unidade léxica distinta, como identificadores, números, operadores, etc.

2.1.1 Lista de Tokens

O analisador reconhece uma variedade de tokens comuns na linguagem Forth, incluindo identificadores (`ID`), números (`NUMBER`), operadores aritméticos (`ADD`, `MINUS`, `TIMES`, `DIVIDE`, `MOD`), operadores de comparação (`INF`, `INFEQ`, `SUP`, `SUPEQ`), e tokens especiais como `FUNCTION`, `STRING`, `EMIT`, entre outros.

2.1.2 Expressões Regulares

Cada token é associado a uma expressão regular que define a sua estrutura. Por exemplo, o token `ID` é definido pela expressão regular `[a-zA-Z]+`, que corresponde a uma ou mais letras do alfabeto. Da mesma forma, o token `NUMBER` é definido pela expressão regular `\d+(\.\d+)?`, que corresponde a um número inteiro ou decimal.

2.1.3 Tratamento de Erros

O analisador também inclui tratamento de erros para lidar com caracteres ilegais encontrados no código-fonte. Quando um caractere ilegal é detectado, o analisador imprime uma mensagem de erro indicando o caractere e a linha onde foi encontrado.

2.1.4 Construção do Analisador

Após definir os tokens e as suas expressões regulares, o analisador é construído usando a biblioteca Ply, que fornece ferramentas para a construção de analisadores léxicos em Python. O analisador resultante é capaz de processar o código-fonte Forth e produzir uma sequência estruturada de tokens, que é então utilizada pelo próximo estágio do processo de compilação.

Este analisador léxico desempenha um papel crucial no processo de compilação, fornecendo a base para a análise sintática e geração de código posterior. A sua implementação cuidadosa e eficiente é essencial para garantir a correta interpretação do código-fonte Forth e a produção do código final pretendido.

2.2 Análise Sintática

O desenvolvimento do analisador sintático para a linguagem de máquina virtual foi uma etapa crucial para possibilitar a interpretação e execução de programas nesta linguagem. Este analisador foi implementado em Python, utilizando a biblioteca PLY (Python Lex-Yacc) para a definição e análise da gramática formal da linguagem.

Antes de iniciar o desenvolvimento do analisador sintático, foi necessário um estudo aprofundado da linguagem de máquina virtual. Isso incluiu a análise das instruções disponíveis na linguagem, as suas operações subjacentes e a estrutura geral do código que ela pode processar. Compreender esses aspectos foi essencial para definir as regras gramaticais necessárias para o analisador.

2.3 Implementação do Analisador Sintático

O analisador sintático foi implementado em várias etapas, cada uma correspondendo a diferentes construções sintáticas da linguagem. Aqui estão algumas das funcionalidades principais:

1. **Declaração de Funções:** O analisador é capaz de reconhecer e processar declarações de funções, incluindo os seus nomes, parâmetros, tipo de retorno e corpo.

2. **Expressões Aritméticas e Condicionais:** Ele é capaz de analisar expressões aritméticas e condicionais, permitindo operações matemáticas e a implementação de condições.

3. **Geração de Código Intermediário:** Durante a análise sintática, o analisador gera código intermediário que representa a estrutura e o fluxo do programa de uma maneira mais abstrata. Isso simplifica a tradução subsequente para o código final executável pela máquina virtual.

4. **Deteção de Erros Sintáticos:** O analisador é capaz de identificar e relatar erros sintáticos no código fonte, como tokens inválidos ou construções sintáticas incorretas.

5. **Conversão para Código Final:** Após a análise sintática e a geração do código intermediário, o analisador converte esse código intermediário para o código final que pode ser interpretado e executado pela máquina virtual.

O analisador sintático é uma parte fundamental do processo de interpretação e execução de programas na linguagem de máquina virtual. A sua implementação exigiu um entendimento detalhado da linguagem alvo, bem como habilidades de programação para desenvolver as regras gramaticais e as funcionalidades necessárias.

3 Otimização de Código

Para otimizar o tempo de deteção de erros e garantir uma análise mais precisa do código fonte, implementamos uma série de medidas no analisador sintático de Forth. Essas medidas visam identificar e relatar erros de forma rápida e eficiente, permitindo uma depuração mais ágil e precisa do código.

Uma das estratégias adotadas foi realizar uma contagem preliminar de certos tokens-chave antes mesmo de iniciar a análise do código fonte. Por exemplo, o interpretador conta todas as ocorrências dos caracteres *:* e *;* para determinar se existe o mesmo número de ambos. Caso contrário, o sistema emite uma mensagem de erro indicando a última linha onde encontrou *;*.

Da mesma forma, o analisador realiza uma contagem de parênteses *(* e *)*, garantindo que o número de aberturas e fechamentos seja consistente. Se essa contagem revelar uma disparidade, o sistema também interrompe a análise e informa o erro com a última linha onde encontrou um *)*.

No analisador léxico, implementamos uma abordagem proativa para detecção de erros. Se o analisador encontrar qualquer irregularidade ou token inválido durante a análise do código fonte, ele interrompe imediatamente o processo e relata qual foi o erro e em qual linha ocorreu, permitindo uma correção rápida por parte do programador.

Além disso, no analisador sintático, adotamos uma abordagem semelhante em caso de erros. Se o sistema detetar qualquer erro durante a análise do código intermédio ou durante a análise gramatical, ele para imediatamente a análise e retorna uma mensagem de erro, indicando o tipo de erro encontrado e em qual ponto do código ele ocorreu.

Essas medidas foram implementadas para melhorar significativamente a eficiência da deteção de erros e oferecer uma experiência de depuração mais ágil e eficaz para os utilizadores deste compilador. Através dessas melhorias, procuramos garantir a robustez e a confiabilidade do sistema, facilitando o processo de desenvolvimento e depuração de código em Forth.

4 Dificuldades na Conversão para o Código Alvo e Limitações do Analisador Sintático Forth

Durante o processo de desenvolvimento do analisador sintático para a linguagem Forth, deparamo-nos com desafios significativos na conversão eficaz do código fonte para o código alvo, bem como na implementação de recursos avançados da linguagem.

Uma das principais dificuldades encontradas foi garantir que o código alvo gerado pelo analisador sintático interpretasse corretamente as funções definidas no código fonte. Descobrimos que o analisador atual é capaz de interpretar corretamente funções que possuem exatamente dois parâmetros de entrada. No entanto, funções com um número diferente de parâmetros não são interpretadas corretamente, o que limita a capacidade do sistema de processar corretamente certas estruturas de código.

Além disso, também enfrentamos desafios na implementação de loops na linguagem Forth. Até o momento, os loops não foram adequadamente interpretados pelo analisador sintático, o que significa que certas construções de código que dependem de loops não são processadas corretamente pelo sistema.

Essas limitações atuais do analisador sintático representam áreas de melhoria significativas para o desenvolvimento futuro do sistema.

5 Objetivos alcançados

Durante o desenvolvimento do analisador sintático de Forth, alcançamos uma série de objetivos que contribuíram significativamente para a robustez e funcionalidade do sistema. Abaixo estão os principais objetivos alcançados:

1. **Interpretação Correta de Expressões Aritméticas:** Implementamos uma lógica sólida para interpretar todas as expressões aritméticas presentes no código fonte. Isso inclui operações como adição, subtração, multiplicação, divisão e módulo, garantindo que os cálculos sejam realizados de maneira precisa e eficiente.

2. **Interpretação Correta de Funções com 2 Parâmetros:** Garantimos a interpretação adequada de todas as funções que possuem dois parâmetros de entrada. Isso envolve a correta manipulação dos argumentos passados para a função, bem como a execução correta das instruções contidas dentro da função.

3. **Interpretação Correta de Formas de Impressão (PRINT):** Implementamos funcionalidades para interpretar corretamente todas as formas de impressão presentes no código fonte. Isso inclui a impressão de valores numéricos, strings e caracteres individuais, garantindo que a saída do programa seja formatada de acordo com as especificações.

4. **Interpretação Correta de Manipulações de STRINGS:** Desenvolvemos mecanismos para manipular corretamente strings no código Forth. Isso inclui a interpretação adequada de funções como CHAR, SPACE e EMIT, permitindo a manipulação eficaz de strings dentro do programa.

5. **Interpretação Correta de Condicionais:** Implementamos uma lógica robusta para interpretar todas as estruturas condicionais presentes no código fonte. Isso inclui condicionais simples (IF-ELSE-THEN), garantindo que o fluxo de controle seja gerenciado corretamente durante a execução do programa.

Através da implementação bem-sucedida desses objetivos, alcançamos uma maior robustez e funcionalidade no analisador sintático de Forth, tornando-o uma ferramenta eficaz para desenvolvedores que trabalham com essa linguagem de programação. Esses recursos

garantem uma interpretação precisa e eficiente do código fonte, facilitando o processo de desenvolvimento e depuração de programas em Forth.

6 Testes de utilização

Para confirmarmos que o nosso compilador funciona fizemos alguns testes um pouco mais complexos.

Começamos por testar uma função que imprimia uma string:

```
: STRING ."Hello World" ;
STRING
```

Figura 6.1: Função para imprimir string em Forth

```
PUSHN 0
START
PUSHS "Hello World"
WRITES
STOP
```

Figura 6.2: Função para imprimir string em VM Code

Depois testamos uma função para testar os condicionais e as relacionais:

```
: MAIORDOIS 2DUP > IF SWAP . ."MAIOR" ELSE . ."MENOR" THEN ;
2 5 MAIORDOIS
```

Figura 6.3: Função para testar condicional em Forth

```
PUSHN 0
START
PUSHI 2
PUSHI 5
PUSHI 2
PUSHI 5
SUP
JZ else1
SWAP
WRITEI
WRITELN
PUSHS "MAIOR"
WRITES
JUMP endif1
else1:
WRITEI
WRITELN
PUSHS "MENOR"
WRITES
JUMP endif1
endif1:
STOP
```

Figura 6.4: Função para testar condicional em VM Code

E terminamos com uma função para calcular média com duas variáveis e testar as aritméticas:

```
: MEDIA (a b -- media) + 2 / ;
100 200 MEDIA .
```

Figura 6.5: Função para calcular média em Forth

```
PUSHN 0
START
PUSHI 100
PUSHI 200
ADD
PUSHI 2
DIV
WRITEI
WRITELN
STOP
```

Figura 6.6: Função para calcular média em VM Code

7 Conclusão

O desenvolvimento do analisador sintático de Forth foi um desafio gratificante que nos permitiu explorar em profundidade os conceitos de linguagens de programação e construção de compiladores. Ao longo deste projeto, enfrentamos uma série de desafios e aprendemos muitas lições valiosas que contribuíram para o nosso crescimento profissional e acadêmico.

Uma das principais realizações deste trabalho foi a implementação de um analisador sintático robusto e funcional que é capaz de interpretar corretamente uma variedade de expressões e estruturas presentes na linguagem Forth. Desde a interpretação de expressões aritméticas até a manipulação de strings e a execução de condicionais, o nosso analisador sintático demonstrou uma capacidade abrangente de entender e processar o código fonte de forma precisa e eficiente.

Além disso, alcançamos os nossos objetivos de melhorar o tempo de deteção de erros e garantir uma experiência de desenvolvimento mais fluida para os utilizadores do analisador sintático. Implementamos medidas proativas para identificar erros de sintaxe e semântica durante a análise do código, fornecendo mensagens de erro claras e precisas para orientar os programadores na resolução de problemas.

Embora tenhamos alcançado muitos dos objetivos estabelecidos para este projeto, reconhecemos que ainda há espaço para melhorias e expansões futuras. Por exemplo, a implementação da interpretação de loops e aprimoramentos na deteção de erros são áreas que podem ser exploradas em trabalhos futuros para tornar o nosso compilador ainda mais robusto e versátil.

Em última análise, este projeto proporcionou-nos uma oportunidade única de aplicar os nossos conhecimentos teóricos em um contexto prático e desafiador. Estamos orgulhosos do que conquistamos e confiantes de que as habilidades e experiências adquiridas ao longo deste projeto serão bastante importantes na nossa vida profissional.