Universidade do Minho
Licenciatura em Engenharia Informática

Unidade Curricular de Processamento de Linguagens

Ano Letivo de 2023/2024

Compilador Forth

Projeto final

Grupo 41
Filipe Santos Gonçalves, a100696
João Andrade Rodrigues, a100711
Mateus Lemos Martins, a100645
Rafael Vale da Costa Peixoto, a100754

Introdução

Este relatório descreve o desenvolvimento de um compilador Forth, conforme especificado no enunciado, capaz de gerar código para a máquina virtual disponibilizada.

Metodologia

O desenvolvimento do compilador Forth seguiu as seguintes etapas:

Estudo da Linguagem Forth:

Inicialmente, foi realizada uma revisão detalhada da linguagem Forth, compreendendo suas principais características, como a notação pós-fixa, o uso de pilha de dados e sua extensibilidade.

Análise do Enunciado:

Em seguida, analisou-se cuidadosamente o enunciado do projeto, identificando os requisitos mínimos e as funcionalidades desejadas do compilador, assim como se estudou as funcionalidades e operações suportadas pela VM.

Implementação das Funcionalidades:

Com base nos requisitos especificados, procedemos com a implementação das funcionalidades essenciais do compilador. Para isso, com uso das bibliotecas ply.lex e ply.yacc abordadas nas aulas, começou-se por identificar os tokens usados em Forth e concebeu-se a seguinte lista:

Token	Expressão Regular
COMENTARY	\(.* \)
NUM	\d+
SOMA	\+
SUBTRACAO	\-
DIVISAO	\/
MULTIPLICACAO	*
RESTO	\%
POTENCIA	\^
2PONTOS	\:
PONTOVIRGULA	\;
EQUAL	=
POINT	\.
PRINTSTRING	\."[^"]*" ou \.\([^"]*\)
DIVIDE_BY_2	2\/
SWAP	swap
CR	cr
EMIT	emit
SPACES	spaces
SPACE	space
KEY	key
DUP	dup
2DUP	2dup
DROP	drop
SUP	>
SUPEQUAL	>=
INF	<
INFEQUAL	<=
IF	if
ELSE	else
THEN	then
DO	do
LOOP	loop
I_COUNTER	i
CHAR	char\s.\s
VAR_DECLARACAO	variable\s[A-Za-z0-9_]+
VAR_ATRIBUICAO	[A-Za-z0-9_]+\s+!
VAR_CHAMADA	[A-Za-z0-9_]+\s@
FUNCAO	[a-zA-Z_][a-zA-Z_0-9]*
ID	[a-zA-Z_][a-zA-Z_0-9!?-]*

Nota: Todas as regex definidas, contêm `(?i)(?<!\S)regex(?!\S)`, ocultado da tabela para tornar visualmente mais agradável, desta maneira é garantido que um token de facto é uma única palavra e não está contido noutra palavra e

case insensitive.

Após termos a lista de tokens e as suas expressões regulares, construíu-se o lexer. Passando então a uma das partes mais desafiantes deste projeto, a tal construção da grámatica.

Na construção da mesma foi utlizada uma abordagem em que se consideram múltiplas linhas e o input ou é vazio ou é linha com a chamada recursiva à esquerda do input.

Uma linha ou é um elem (um número, um operador, um Char ou uma função reservada forth como Emit, Space...), um condicional (bloco if), um ciclo (loop) ou atribuição de variáveis. É então apresentada a gramática concebida abaixo.

input : input linha
| empty
linha : elem
| 2PONTOS funcao input PONTOVIRGULA
| condicional
| ciclo
| variaveis
funcao : FUNCAO
elem : NUM
| operador
| ID
| POINT
| PRINTSTRING
| SWAP
| CR
| EMIT
| CHAR
| SPACES
| SPACE
| KEY
| DUP
| 2DUP
| DROP
| ICOUNTER
operador : SOMA
| SUBTRACAO
| DIVISAO
| MULTIPLICACAO
| RESTO
| POTENCIA
| DIVIDE_BY_2
| EQUAL
| SUP
| SUPEQUAL
| INF
| INFEQUAL
condicional : IF input ELSE input THEN input
| IF input THEN input
ciclo : DO input LOOP
variaveis : VAR_DECLARACAO
| VAR_ATRIBUICAO
| VAR_CHAMADA
empty :

Suporte às expressões aritméticas:

Sendo Forth uma linguagem de notação pós fixa e a VM funcionando da mesma maneira, a geração de código desta parte é direta. Assim a única verificação que precisamos fazer é se há números na stack para poder realizar a operação introduzida e posteriormente gerar o código necessário. Para isso foi definida uma lista (stack) variável interna do nosso analisador sintático que sempre que capta um operador verifica se existem elementos na variável stack definida para realizar a operação, da mesma maneira sempre que é captado um número o mesmo é adicionado à stack e quando aplicada uma operação os valores da stack são atualizados. Posto isto a geração de código para a VM dum número foi feita através do "pushi NUM"e a geração de código do operador traduzida para o correspondente da VM.

Criação de funções:

Neste segundo ponto já foi necessário uma abordagem diferente, como tal a estratégia definida consistiu em sempre que uma função era definida, o ID da função é adicionado a um dicionário interno em que para um ID o código produzido por essa função é lá armazenado, e sempre que um ID é captado é verificado se essa função existe no dicionário e caso exista o código correspondente é injetado no output por cada chamada. Não foi utilizado qualquer tipo de labels e chamadas da função na VM através de call e pusha, uma vez que desta maneira o conteúdo da stack na VM é mais facilmente gerido e é menos propício a erros. De notar que tudo o que é definido dentro de uma função não é armazenado na nossa variável interna stack porque o momento em que a função é definida e chamada pode ser totalmente diferente então o tratamento de erros deste lado é passado para o compilador da VM, por exemplo caso uma função some os últimos dois elementos e no momento em que é chamada não há elementos na stack.

Print de carateres e strings:

Uma vez termos a nossa variável interna da stack, sempre que um ponto(".") é capturado é verificado o tipo do último elemento na stack, caso seja um INT é gerado o comando "writei"ou caso seja uma STRING é gerado o comando "writes"para a VM. Caso seja capturado o PRINTSTRING ("string") em nada é alterado o conteúdo da stack, uma vez este comando apenas imprimir uma

string no terminal, é por isso gerado um pushs com a string capturada entre as e depois realizado um "writes". No caso do EMIT é automaticamente passado o comando "writechr"que faz o mesmo que o EMIT na VM. Por fim quando um CHAR é reconhecido pelo nosso lexer, que reconhece na mesma regex a palavra CHAR seguida do carater, é gerado o comando "pushi"juntamente com o ord(carater) para a VM.

Condicionais:

No caso dos condicionais, "ifs"(mais propriamente ditos), é utilizado um contador por cada "if"e criado uma label na VM, "if"com o contador e todos os elementos dentro do bloco if têm o mesmo contador associado à sua label tais como o "jz endif"que verifica a condição do "if", o "jumpendif para caso entre no bloco "if"salte por cima do "else"e desta maneira é utilizada uma recursividade entre os tokens "IF"e "ELSE"para suportar "ifs"dentro de "ifs"e tudo mais, sendo a chamada recursiva da definição mais superior da gramática.

Ciclos:

Esta funcionalidade foi a que mais pensamento requeriu e estudo, uma vez que numa fase inicial não se estava a conseguir chegar a nenhuma solução. Foi então que se decidiu criar um contador para bloco de ciclo, desta maneira é lido o código forth, antes do parser começar, dado como input e então por cada "DO"o contador é incrementado numa unidade, após isso para cada unidade do contador é criado um dicionário interno que para cada unidade do contador (chave) origina um tuplo com os índices das váriaveis dos limites inferiores e superiores do ciclo, é reservado então um espaço para cada delimitador antes de a VM começar (START). Desta maneira como fazemos um tratamento dos dados anterior ao parser começar, conseguimos prever e reservar o que o parser necessitará quando ocorrer o geramento de código para a VM.

Assim no geramento de cada bloco "do"os útlimos dois valores da stack são armazenados nas variavéis correspondentes e carregados outra vez para a stack, o limite inferior é então subtraído ao limite superior e executado o "jz endDo"caso seja o final das iterações, caso contrário no final de cada iteração é lido o limite inferior e este é incrementado e voltado a ser guardado no seu espaço reservado garantindo assim o funcionamento correto na VM.

Variáveis:

Uma vez termos usado variavéis para o tratamento de ciclos, foi utilizada a mesma estratégia para esta parte. É então lido o ficheiro previamente e por cada declaração das variáveis é utilizado um contador das mesmas e alocado um espaço para cada antes da execução da VM. Quando o parser começa a sua execução por cada declaração captada é adicionado a um dicionario com id da variável ao indice do seu espaço reservado e quando esta é atualizada ou chamada o mesmo indice é utilizado para a manipulação do seu conteúdo.

Casos adicionais:

2DUP - a abordagem do tratamento desta função predefinida pela linguagem forth exigiu também uma abordagem relativamente diferente quando comparada às outras funções. Pelo facto de poder ser usada na definição de uma função, com uso da stack inerna do analisador sintático, não é possível saber o tipo dos elementos que estão na stack no momento que a função é chamada pois ela pode ser definida numa parte do código onde a stack está vazia e depois chamada quando elementos foram adicionados à stack. Por isso são usados dois espaços na tabela de variáveis que servem para quando a função "2dup"é chamada através do comando "storeg"guardar os últimos dois valores da stack nessas posições e logo de seguida executar os "pushg"dessas duas variavéis desta função duas vezes para cada de forma a simular o funcionamento da "2dup"em forth.

i - esta pequena função que tem como resultado o número de cada iteração dentro de um bloco do é feita sempre utilizando a subtração dos delimitadores definidos para esse bloco do ciclo, descrito anteriormente.

Conclusão:

O desenvolvimento do compilador Forth foi uma experiência enriquecedora, que permitiu aprofundar conhecimentos concretos em engenharia de linguagens e programação generativa. Este projeto intrudoziu-nos uma nova linguagem que segue uma metodologia e métodos de programar completamente diferente daqueles a que estamos habituados o que nos desenvolveu novas maneiras e abordagens para problemas futuros. Permitiu também o conhecimento de duas bibliotecas python (lex e yacc) que auxiliaram e melhoraram o nosso desenvolvimento de gramáticas e também um aprofundamentos sobre os autómatos

utilizados. De uma maneira geral, concluímos que o projeto foi finalizado satisfatóriamente cumprindo todos os requisitos inicialmente propostos.

Referências:

Documentação de um manual online da linguagem Forth - https://www.forth.com/starting-forth/.

IDE de desenvolvimento e testes - https://www.jdoodle.com/execute-forth-online.

Máquina Virtual (VM) - https://ewvm.epl.di.uminho.pt/.