Gustavo P. Gouveia (6482819), Victor Lassance (6431325)

Relatório de Compiladores

Segunda Etapa
Definição formal da sintaxe da linguagem de programação CZAR

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a aprovação na disciplina Linguagens e Compiladores no quinto módulo acadêmico do curso de graduação em Engenharia de Computação, junto ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

Universidade de São Paulo

Escola Politécnica

Engenharia de Computação - Curso Cooperativo

Professor: Ricardo Luis de Azevedo da Rocha

São Paulo

2013

Resumo

Este trabalho descreve a concepção e o desenvolvimento de um compilador utilizando a linguagem C. O escopo do compilador se limita a casos mais simples, porém simbólicos, e que servem ao aprendizado do processo de criação e teste de um compilador completo. A estrutura da linguagem escolhida para ser implementada se assemelha a própria estrutura do C, por facilidade de compreensão, porém com algumas peculiaridades trazidas de outras linguagens.

Palavras-chaves: Linguagens, Compiladores, Definição formal da Sintaxe.

Sumário

1 Introdução

Este projeto tem como objetivo a construção de um compilador de um só passo, dirigido por sintaxe, com analisador e reconhecedor sintático baseado em autômato de pilha estruturado.

Em um primeiro momento, foi definida uma linguagem de programação e identificados os tipos de átomos. Para cada átomo foi escrito uma gramática linear representativa da sua lei de formação e um reconhecedor para o átomo. Desse modo, as gramáticas assim escritas foram unidas e convertidas em um autômato finito, o qual foi transformado em um transdutor e implementado como sub-rotina, dando origem ao analisador léxico propriamente dito. Também foi criada uma função principal para chamar o analisador léxico e possibilitar o seu teste.

Nessa etapa, **TODO:Victor**

Como material de consulta, além de sites sobre o assunto, como por exemplo um que permite verificar a definição em Wirth e criar os diagramas de sintaxe[1], foi utilizado o livro indicado pelo professor no começo das aulas (NETO, 1987), para pesquisa de conceitos e possíveis implementações.

O documento apresenta a seguir as respostas às questões propostas para a segunda etapa, assim como uma conclusão e apêndices relacionados à atividade.

[1] Site: http://karmin.ch/ebnf/index

2 Descrição Informal da Linguagem

TODO

3 Exemplo de Programas na Linguagem

TODO

4 Descrição da Linguagem em BNF

TODO

5 Descrição da Linguagem em Wirth

```
1  PROGRAM          = HEADERS DECLARATIONS PROGRAM_BLOCS MAIN_BLOCK.
2  HEADERS          = { "<" IDENT ">" }.
3  DECLARATIONS     = { TYPE IDENT [ "=" EXPR ] ";"}.
4  TYPE             = IDENT {'*'} {"["NUMBER"]"}.
5  PROGRAM_BLOCS    = { FUNCTION_BLOC | STACK_BLOC }.
6  FUNCTION_BLOC    = "function" IDENT STACK_BLOC.
7  MAIN_BLOC        = "main" STACK_BLOC.
8  STACK_BLOC       = "(" [ ARG { "," ARG } [","] ] ")" "{"
       DECLARATIONS { INSTR } "}".
9  ARG              = TYPE [OUT_MODIF] VARSPEC.
10 INSTR            = ATRIB | FUNCTION_CALL | FLOW_CONTROL.
11 IN_MODIF         = ">".
12 OUT_MODIF        = "*".
13 FLOW_CONTROL     = WHILE_CONTROL | IF_CONTROL | IFELSE_CONTROL .
14 WHILE_CONTROL    = "while" "(" CONDITION ")" STACK_BLOC .
15 IF_CONTROL       = "if" "(" CONDITION ")" STACK_BLOC .
16 IFELSE_CONTROL   = "ifelse" "(" CONDITION ")" STACK_BLOC .
17 ATRIB            = VARSPEC "=" EXPR ";".
18 FUNCTION_CALL    = "^" IDENT [ ":" IDENT ] "(" [ VARSPEC { ","
       VARSPEC } [","] ] ")" ";".
19 INTEG_EXPR       = INTEG_TERM { ("+"|"-") INTEG_TERM }.
20 INTEG_TERM       = ( "(" INTEG_EXPR ")" | VARSPEC | NUMBER ) {
       ("*"|"/") INTEG_TERM }.
21 VARSPEC          = IDENT | IDENT { "[" NUMBER "]" } .
22 CONDITION        = "????".
23 EXPR             = "????".
```

6 Diagrama de Sintaxe da Linguagem

TODO

7 Conjunto das Palavras Reservadas

```
1  int
2  float
3  string
4  char
5  bool
6  for
7  do
8  while
9  if
10 else
11 read
12 write
13 and
14 or
15 main
16 return
```

8 Considerações Finais

O projeto do compilador é um projeto muito interessante, porém complexo. Desta forma, a divisão em etapas bem estruturadas permite o aprendizado e teste de cada uma das etapas. Em um primeiro momento, o foco foi no analisador léxico, o que permitiu realizar o *parse* do código e transformá-lo em tokens. Para a realização do analisador, tentamos pensar em permitir o processamento das principais classes de tokens, com o intuito de entender o funcionamento de um compilador de forma prática e didática.

Já na segunda etapa, **TODO:Victor**

Para as próximas etapas, espera-se atualizar o analisador léxico quando for necessário, visando agregar os ensinamentos das próximas aulas.

Referências

NETO, J. J. *Introdução à Compilação*. [S.l.]: LTC, 1987. (ENGENHARIA DE COMPUTAÇÃO).