ESCOLA POLITÉNICA □DA UNIVERSIDADE DE SÃO PAULO

Linguagens e Compiladores

P2 - Final

Alunos: Rafael Camargo Leite – 7629953
Vinicius Correa - 7631553

Índice

1. Analisador léxico

Como a linguagem LazyComp não possui identificadores, números ou literais, o analisador léxico é extremamente simples:

- Palavras reservadas:
  - i
  - K
  - k
  - S
  - s
  - 0
  - 1
  - *
  - `
- Seperadores:
  - (
  -)

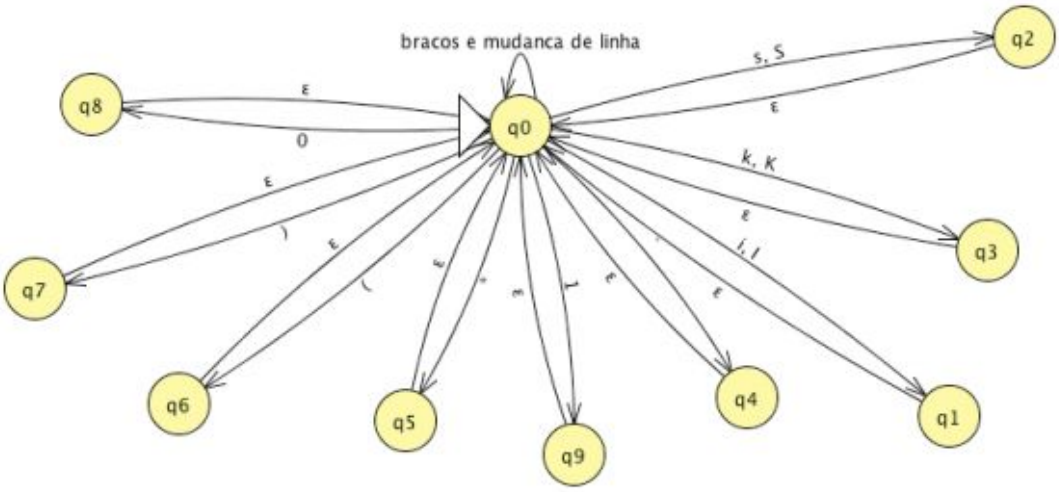

Figura 1 - Autômato finito completo para análise léxica

2. *Analisador Sintático*

Para a construção do autômato de pilha estruturado, o BNF da linguagem foi transformado no seguinte WIRTH:

```
Program = { Expr }.
Expr = "i"
    | "I"
    | "K" | "k"
    | "S" | "s"
    | NonEmptyJotExpr
    | "`" Expr Expr
    | "*" Expr Expr
    | "(" { Expr } ")".
NonEmptyJotExpr = { NonEmptyJotExpr } ( "0" | "1" ).
```

Neste WIRTH, foi aplicado o algorítmo para se chegar aos autômatos de pilha estruturado:

```
Program = 0 { 1 Expr 2 } 1 .
---
Expr = 0 "i" 1 | 0 "I" 2 | 0 "K" 3 | 0 "k" 4 | 0 "S" 5 | 0 "s" 6 | 0 NonEmptyJotExpr 7 | 0 "`" 8 Expr 9 Expr 10 | 0 "*" 11 Expr
12 Expr 13 | 0 "(" 14 { 15 Expr 16 } 15 ")" 17 .
---
NonEmptyJotExpr = 0 { 1 NonEmptyJotExpr 2 } 1 ( 1 "0" 4 | 1 "1" 5 ) 3 .
```

Figura 2 - notação wirth

Finalmente, foram construídos os autômatos correspondentes:

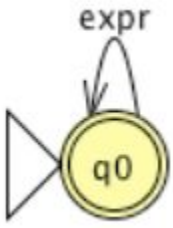

Figura 3 - autômato finito *program*

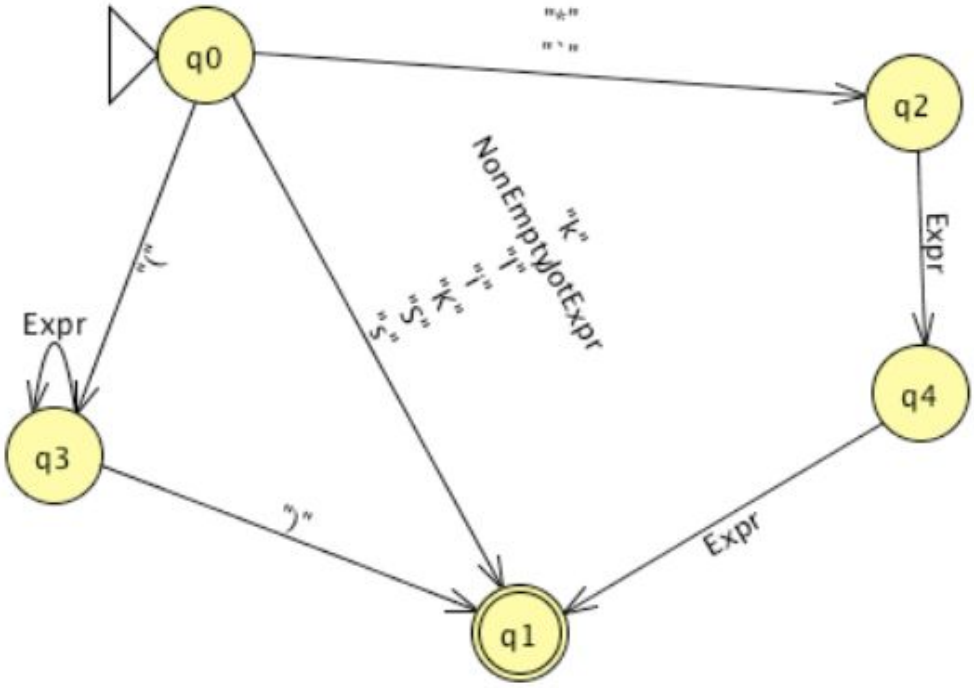

Figura 4 - Autômato finito *expr*

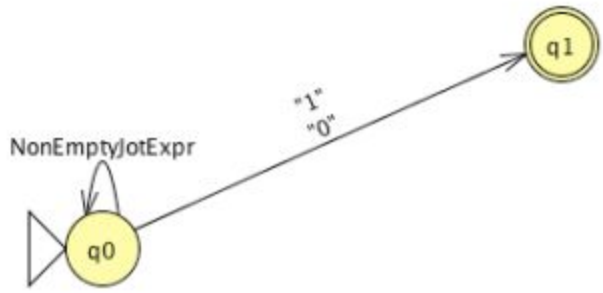

Figura 5 - Autômato finito *NonEmptyJotExpr*

3. Ambiente de execução e análise semântica

Funções semânticas

Funções semânticas são executadas de acordo com o token lido. Com base no significado do token existem 3 ações possíveis: abrir um novo escopo, fechar um escopo ou adicionar um parametro novo em seguida tentar reduzi-lo As funções semanticas são escritas na linguagem do compilador

Funções do Ambiente de execução

Função de ambiente de execução: (escritas em C)

- Função AbreEscopo: responsável por adicionar um escopo novo na lista de escopo
- Função FechaEscopo: responsável por copiar todos parametros do escopo a ser fechado para o escopo pai
- Função TentaReduzir: verifica se o parametro atual tem todas suas dependencias para ser reduzido
- Função InsereParametro: insere o novo token no escopo atual
- Função I: implementa a função identidade onde *f(x)=x*
- Função K: implementa a função K onde *f(xy)=x*
- Função S: implementa a função S onde *f(xyz)=xz(yz)*

4. Implementação

1. Analisadores

Para a implementação dos analisadores léxico, sintático e semântico, foi utilizada a linguagem go. Como a linguagem para a qual os arquivos escritos em lazyComb são compilados é C, a linguagem utilizada para as funções do ambiente de execução foi C.

Todos os autômatos (tanto finitos quanto de pilha estruturado) foram implementados em tabelas com ponteiros de função.

Como exemplo, o código abaixo mostra a tabela de submáquinas do automato de pilha estruturado do analisador sintático

```
 submachines = []func(t *Token) int{
        fsmProg,
        fsmExpr,
 }
```

Neste trecho de código pode-se observar ser criado um array de pointeiros para função sendo que essas funções recebem um parâmetro, o token atual, e retornam um inteiro, que representa o estado da submaquina após a execução da função.

Abaixo está copiada a função principal do compilador

```
func Analyze(file *os.File) int {
        for {
                if globalState.getTokenFlag {
                        globalState.currToken = GetToken(file)
                        if globalState.currToken == nil {
                                break
                        }
                }
                globalState.getTokenFlag = true
                globalState.currSubmachineState                                         =
submachines[globalState.submachine](globalState.currToken)
                if globalState.currSubmachineState == ERROR {
                        fmt.Println("[ERROR] Compilation error")
                        return 1
                }
        }

        fmt.Println("[INFO] Compilation Successful")
        semanticFlushCode()
        return 0
}
```

O código acima é um loop simples:
Até que o token pedido ao léxico seja nulo, utilize a matriz de funções para calcular qual estádo deve ser alcançado dependendo do token lido.

Vale mencionar o modo como o código foi estruturado foi o mesmo utilizado no projeto da disciplina. Só foi utilizada outra linguagem de programação para termos as facilidades de uma linguagem um pouco mais alto nível.

2. *Ambiente de execução*

A principal estrutura do ambiente de execução é uma pilha de escopos para que cada função possa ser executada corretamente utilizando os parâmetros fornecidos.
Fora isso, dependendo do parâmtro recebido(entre S,s, K, k, I, i, (,)) alguma ação correspondente é executada.

Caso seja abertura de escopo ‘(’, um novo escopo é adicionado à pilha de escopos.
Caso seja fechamento de escopo ‘)’ os parâmetros deste escopo a ser fechado são copiados ao escopo ‘pai’ e o escopo atual é deletado.

5. Testes

Os dois casos de teste foram:

1. example1.lazy
   I(KKI)(IIKSI)
   que deve ser reduzido a K(S)

2. example2.lazy
   SII(III)
   que deve ser reduzido a I

Abaixo estão prints da tela na execução dos dois teste

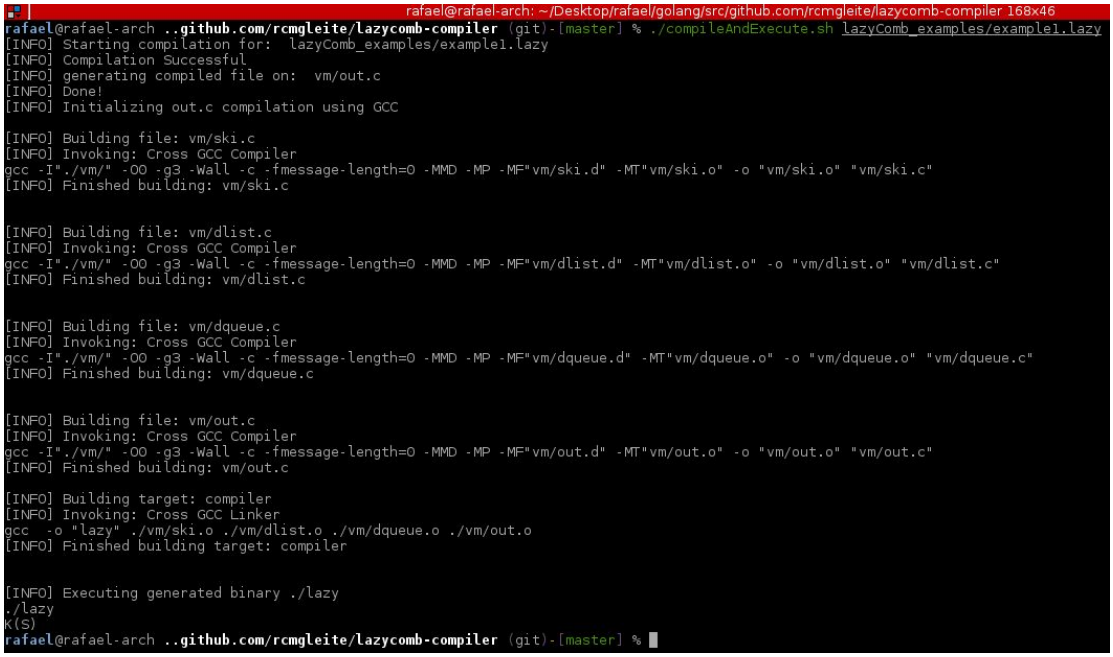

Figura 6 - execução example1.lazy

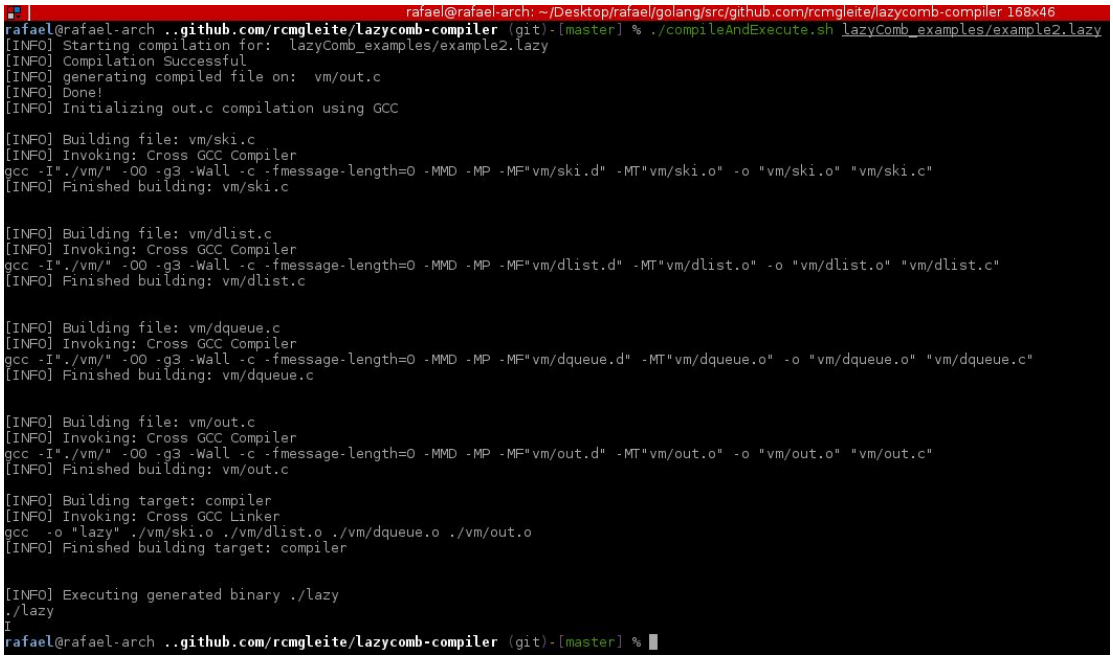

Figura 7 - execução example2.lazy