Implementação de Gerador de Autômato Finito Determinístico

Diogo M. Cunha[1],

[1]Universidade Federal da Fronteira Sul (UFFS)
Caixa Postal 181 – 89.802-112 – Chapecó – SC – Brazil

{diogo.cunha}@estudante.uffs.edu.br

Abstract. *The study of formal languages and their structures is a fundamental part of the construction of compilers in computing, understanding their concepts it is possible to optimize the process of compilation and/or interpretation of a source code in orders of magnitude. This work aims to describe the operation of a finite automaton generation code implemented in the Go language and with tabular output. In the end, the application was able to generate a deterministic finite automaton from the input but was not able to minimize it.*

Resumo. *O estudo das linguagens formais e suas estruturas é parte fundamental da construção de compiladores dentro da computação, entendendo seus conceitos é possível otimizar o processo de compilação e/ou interpretação de um código fonte em ordens de grandeza. Esse trabalho tem como o objetivo descrever o funcionamento de um código de geração de autômato finito implementado na linguagem Go e com saída tabelar. Ao fim, o aplicativo conseguiu gerar um autômato finito deterministico a partir da entrada mas não foi capaz de minizá-lo.*

1. Introdução

O estudo da teoria de linguagens formais é um ramo da Teoria da Computação focado na construção e análise de diferentes tipos de gramáticas, seus codificadores e decodificadores. Um desses tipos são as gramáticas regulares e seus decodificadores, os autômatos finitos.[Scheffel 2011]

Neste trabalho, será explicado o funcionamento de um aplicativo que recebe uma gramática regular e um grupo de tokens e gera um autômato finito determinístico delas em formato tabelar.

2. Referencial Teórico

Em suma, os principais conceitos relacionados a esse trabalho é a definição da área de linguagens formais, definição e explicação das gramáticas e a explicação do funcionamento bloco léxico, a unidade do compilador responsável por utilizar o autômato finito.

2.1. Linguagens Formais

O começo das discurssões sobre algoritmos datam dês dos primódios da humanidade mas a ideia de um algoritmo computável é bem mais recente, do século 19 com Charles Babbage que buscava automatizar a forma como os cálculos eram feitos na Royal Astronomical Society. Por meados do século 20, houve uma gradual separação do que viria a

ser a Computação, como conhecemos hoje, da matemática graças aos trabalhos como a Máquina de Turing e as Funções Recursivas de Church.[Scheffel 2011]

Ao perceber que as máquinas poderiam gerar novas saídas diferentes de números, além de analizar sequências diferentes de tokens diferentes de números, foi necessário formalizar essas descobertas e nisso, foi criada a Teoria das Linguagens Formais que é responsável por analizar o formato e o significado de uma sequência de entradas qualquer.[Scheffel 2011]

2.2. Gramáticas

Uma gramática é uma representação de uma linguagem através de simbolos terminais, que não podem ser substituido por outros e pertencem ao alfabeto da linguageme, e não terminais, que podem ser substituídos por outros não terminais ou terminais. Formalmente, ela é descrita como possuidora de um ”conjunto finito de variáveis (também chamadas de não-terminais). A linguagem gerada por uma gramática é definida recursivamente em termos das variáveis e de símbolos primitivos chamados terminais, que pertencem ao alfabeto da linguagem.”[Scheffel 2011]

Portanto, as suas diferenças estarão na forma como permitem organizar o conjunto de terminais e não terminais e no conjunto de problemas que conseguem resolver. Uma gramática regular não permite produções de não terminais com não terminais e seu identificador é o autômato finito, por outro lado uma gramática livre de contexto consegue resolver todos os problemas de gramática regular por permite produções de apenas não terminais mas demanda mais recurso computacional.[Scheffel 2011]

2.3. Bloco Léxico

O bloco léxico é responsável por quebrar uma cadeia de caracteres em tokens, que são duplas de tipo CLASSE:VALOR, que será tratada pelo parser no bloco sintático. Para realizar essa tarefa ele utiliza uma tabela de simbólos e regras de mapeamento que também são acessadas pelo parser na etapa posterior. É também a função dele tratar espaços em branco, remover comentários e avisar caso haja erros léxicos[Scheffel 2011]

3. Implementação

3.1. Entrada de Dados

A entrada de dados do projeto foi feita através de um arquivo chamado rules.in que é responsável por armazenar as regras da gramática, palavras chave e caracteres especiais da linguagem. Sua estrutura é como no exemplo abaixo

```
se
senao
entao
--
<S> ::= a<A>|b<B>|b|c<S>|c|epsi
<A> ::= a<S>|a|b<C>|c<A>
<B> ::= a<A>|c<B>|c<S>|c
<C> ::= a<S>|a<D>|c<A>|c<C>
<D> ::= epsi
```

Após a leitura dos dados, as regras e os tokens são tratados separadamente. Primeiro, é carregada as regras que são mapeadas para uma struct Rule que será utilizada pelo af para gerar os estados e produções. Por fim, é feito a carga de regras baseadas na geração dos tokens o array de regras é retornando para ser utilizado em outras partes do projeto. Os possíveis nomes, do estados são mapeados usando um array de strings chamado Names.

```
type Rule struct {
   Name        string
   Productions []string
}
...
var Names []string = [...]
...
func ReadRules(filename string) []Rule {
   data, err := os.ReadFile("rules.in")

   unames := Names

   if err != nil {
      log.Fatal(err.Error())
   }
   temp := strings.Split(strings.ReplaceAll(string(data), " ", ""), "--
   rtokens,gr := temp[0],temp[1]

   rules := readGR(&unames, gr)
   rules = readTokens(rules, &unames, rtokens)
   Names = nil

   return rules
}
```

3.2. Geração do Autômato Finito

Para mapear os estados, foi utilizada uma lógica parecida com a representação de uma lista de adjacência, há um array principal, que represeta o autômato finito, e cada parte dele possui uma identificação e um outro array de produções que é composto por duplas de simbolos terminais e o estado que por eles é alcançado.

```
type Beam struct {
Simbol string
State  string
}

type State struct {
Name       string
...
Production []Beam
```

```
}

...

type AF []State
```

Graças a semelhança entre a estrutura que armazanea as regras e a que armazena os estados, a criação do autômato é um mapeamento direto um pra um com o detalhe de separar terminais de não terminais no momento em que se é mapeamedo o feixo.

3.3. Determinização do Autômato

Por outro lado, durante a determinização foi necessário a criação de funções auxiliares de navegação dentro do autômato em decorrência da estrutura pouco usual dele. Nesse momemnto, foi visto a necessidade de sanitizar a entrada para remover caracteres como

`\n`

que geravam comportamentos inexperados durante a execução e a inserção de um campo de indentificação nos estados que foram gerados para resolver indertemização, evitando a criação de estados desnecessários.

```
    type State struct {
...
Ind     string
...
}

type Indetermination struct {
Simbol      string
States      string
Parent      *State
Productions []Beam
}
func removeUnterminals(production string) string {
...
return sanitaze(production[:start] + production[end+1:])
}

func removeTerminals(production string) string {
...
return sanitaze(production[start : end+1])
}

func sanitaze(s string) string {
...
}
```

4. Resultados

Foi testado 5 gramáticas diferentes e 4 palavras reservadas no autômato para garantir que estava funcionando corretamenta, em decorrência de problemas de tempo e

implementação. O código funciona corretamente, embora apresente alguns bugs visuais que não puderam ser resolvos e outros problemas desconhecidos no momento. Uma de suas deficiências é a não minimização do autômato final e também a não remoção de epson produções que não foram implementadas pelos problemas citados anteriormente, o que gera ”AF”s maiores do que deveriam por possuir estados inalcançáveis e mortos como no exemplo abaixo. Entrada:

```
se
senao
entao
--
<S> ::= a<A>|b<B>|b|c<S>|c|epsi
<A> ::= a<S>|a|b<C>|c<A>
<B> ::= a<A>|c<B>|c<S>|c
<C> ::= a<S>|a<D>|c<A>|c<C>
<D> ::= epsi
```

Autômato Finito Não Determinístico

```
    ...
|  <C>  |  <S><D> | ...
| *<D>  |  ...
    ...
```

Autômato Finito Determinístico

```
No exemplo D se torna um estado morto
    ...
|  <C>  |  <3> | ...
| *<D>  |  ...
    ...
```

5. Conclusão

Neste trabalho, foi feito um breve levamtamento bibliográfico sobre as principais linhas de pesquisas dentro do smart farming e seus principais desafios. Nisso constasse uma grande quantidade de pesquisas direcionadas a redes IoT, focadas em tratar questões como sensoriamento, infraestrutura da rede, protocolos e segurança onde seus principáis desafios são provenientes da distribuição heterogênica dos dispositivos, formas de comunicação entre eles e garantia de segurança dentro da rede.

Desta forma, foram avaliadas duas possibilidades de futuros trabalhos: uma pesquisa experimental vizando corrigir e/ou atenuar um dos problemas acima e uma pesquisa bibliográfica focada em uma das linhas de pesquisa referênciadas para aprofundar em um dos telas.

Referências

Scheffel, R. M. (2011). *Apostila Linguagens Formais e Autômatos*. Departamento de Ciencias Tecnologicas, UNISUL, 1 edition.