UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GIOVANNI VERARDI TACCHINI

IMPLEMENTAÇÃO EM C PARA CONVERSÃO DE AUTÔMATOS FINITOS EM GRAMÁTICAS LINEARES UNITÁRIAS À DIREITA

PONTA GROSSA - PR

26/06/2024

GIOVANNI VERARDI TACCHINI

IMPLEMENTAÇÃO EM C PARA CONVERSÃO DE AUTÔMATOS FINITOS EM GRAMÁTICAS LINEARES UNITÁRIAS À DIREITA

Implementation in C for converting finite automata into right unit linear grammar

Trabalho de Linguagem Formais e Autômatos, do curso de Bacharel em Ciências da computação pelo Departamento de Informática (DAINF) da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador(a): Gleifer Vaz Alvez.

PONTA GROSSA - PR

26/06/2024

RESUMO

Este artigo detalha o desenvolvimento de um programa em linguagem C destinado a converter autômatos finitos, tanto determinísticos quanto não determinísticos, em gramáticas lineares unitárias à direita. Exploramos a teoria referente a tradução de autômatos para gramáticas e demonstramos sua aplicação prática, baseado nisso aplicamos o desenvolvimento de um software que automatiza esse processo. O código implementado respeita rigorosamente os teoremas discutidos, facilitando um entendimento mais profundo tanto da teoria quanto de suas implementações práticas.

Palavras-chave: linguagem; autômato; conversão; gramática.

ABSTRACT

This article details the development of a C language program designed to convert finite automata, both deterministic and non-deterministic, into right-hand unitary linear grammars. We explore the theory regarding the translation of automatons into grammars and demonstrate its practical application, based on this we apply the development of software that automates this process. The implemented code strictly respects the theorems discussed, facilitating a deeper understanding of both the theory and its practical implementations.

Keywords: language; automata; convert; grammar.

SUMÁRIO

1 INTRODUÇÃO

A teoria dos autômatos finitos e das gramáticas formais constitui a espinha dorsal da ciência da computação teórica, desempenhando um papel fundamental no desenvolvimento de linguagens de programação, compiladores e na análise de complexidade de algoritmos. Autômatos finitos, seja em sua forma determinística ou não determinística, são amplamente utilizados para modelar processos que envolvem um número finito de estados. Por outro lado, as gramáticas lineares unitárias à direita (GLUD) são uma classe de gramáticas formais que geram linguagens regulares, sendo essenciais para a compreensão de construções linguísticas e padrões de reconhecimento.

Este artigo apresenta um software desenvolvido em C que automatiza a conversão de autômatos finitos, tanto determinísticos quanto não determinísticos, em GLUDs. Abordaremos a teoria que fundamenta essa conversão e detalharemos a implementação prática do sistema, ilustrando como a teoria pode ser aplicada para resolver problemas práticos na ciência da computação.

2 DESENVOLVIMENTO

Neste tópico, exploramos detalhadamente os aspectos teóricos e práticos envolvidos na conversão de autômatos finitos em gramáticas lineares unitárias à direita. O processo de desenvolvimento é dividido em três partes principais: a fundamentação teórica, que fornece a base conceitual necessária; a metodologia, que descreve o planejamento e as estratégias adotadas para a implementação; e a implementação em si, que detalha e apresenta a construção e a operação do software desenvolvido. Cada uma dessas etapas é crucial para entender tanto os desafios teóricos quanto as soluções práticas correspondentes a tradução de modelos computacionais de autômatos finitos em linguagens formais.

2.1 Fundamentação Teórica

A teoria dos autômatos finitos e das gramáticas formais é uma área de estudo fundamental na teoria da computação, fornecendo as bases para o entendimento de linguagens de programação e sistemas de computação. Nesta seção, discutiremos os conceitos fundamentais de autômatos finitos (AFs) e gramáticas lineares unitárias à

direita (GLUDs), bem como a teoria aplicada que possibilita a conversão de um AF em uma GLUD.

Autômatos finitos são modelos matemáticos de máquinas computacionais com um número finito de estados. Existem duas variantes principais: autômatos finitos determinísticos (AFD) e autômatos finitos não determinísticos (AFND). Enquanto um AFD possui uma função de transição que retorna um único estado para cada par de estado e símbolo de entrada, um AFND pode retornar múltiplos estados, proporcionando uma flexibilidade maior na modelagem de processos.

As GLDs são um tipo de gramática formal onde todas as produções possuem a forma $A \rightarrow wB$ ou $A \rightarrow w$, com A e B sendo variáveis e w um símbolo terminal, já as GLUDs adicionam um teorema a mais, onde $|w| =< 1$, essa regra implica que a gramática será unitária, uma vez que, w sempre será 1 ou 0. Essas gramáticas são especialmente úteis para gerar linguagens regulares e possuem uma forte correlação com os AFs, pois cada produção de uma GLUD pode ser mapeada diretamente para uma transição em um AF.

O teorema fundamental que da conversão de AFs em GLUDs estabelece que para cada AF, existe uma GLUD correspondente que gera a mesma linguagem. Além disso para realizar a conversão primeiramente será necessário comparar os elementos da forma descritiva de cada um. Autômatos finitos: $M = (\Sigma, Q, \delta, q0, F)$ e gramatica: $G = (V, T, P, S)$, podemos dizer que $T = \Sigma$, $V = Q \cup \{S\}$. A última parte da tradução está relacionado em transformar a função programa em produção da gramática, para realizar isso precisamos entender que, se em um AF temos uma transição do estado Qi para o estado Qj com o símbolo a, na GLUD correspondente teríamos uma produção $Qi \rightarrow aQj$. Além disso, para realizar a transição será necessário adicionar uma variável S que aponta para o estado inicial e fazer com que todos os estados finais gerem palavras vazias.

Esta seção teórica forma a base sobre a qual o software de conversão é construído. Compreender essas teorias não apenas assegura que as transformações sejam realizadas corretamente, mas também facilita a validação e a verificação dos resultados gerados pelo software, conforme será visto nas seções de metodologia e implementação.

2.2 Metodologia

Entender a teoria da tradução foi com certeza a parte mais importante antes de decidir iniciar o projeto, analisar todos os teoremas e fazer testes simples em códigos com entradas predefinidas foi essencial para a formatação do código. O código foi escrito em C pois apresento mais conhecimento nessa linguagem, mesmo que sua parte gráfica seja mais simples a manipulação de ponteiros e matrizes foi essencial para a criação do código. Usei o programa Codeblocks no Windows para realizar a escrita e compilação do código.

Para iniciar o projeto decidi criar um código que conseguisse ao menos realizar o básico, de transformar um autômato finito determinístico em uma gramatica, assim estaria trabalhando com apenas um resultado prévio de uma entrada conjunta de um estado e símbolo especifico, para realizar essa tarefa decidi usar uma matriz quadrada que armazenava o resultado da transição e recebia o estado e o símbolo da mesma, logo após tudo correr como o planejado decidi transformar o código para receber a leitura de um autômato finito não determinístico, para isso a matriz foi incrementada com mais um elemento, transformando-a em uma matriz tridimensional, nesse caso agora as entradas correspondem ao estado, símbolo e número de transição, resultando na resposta da transição.

O design do software reflete a estrutura teórica dos autômatos e gramáticas, com módulos separados para cada funcionalidade principal, facilitando manutenção e expansões futuras. Seguimos um modelo iterativo, desenvolvendo e refinando cada módulo em ciclos. Testes unitários e de integração foram realizados para cada componente, com testes de validação baseados em casos conhecidos para assegurar a precisão das conversões.

2.3 Implementação

A implementação do software que traduz autômatos finitos em gramáticas lineares unitárias à direita utilizou a linguagem C, enfocando eficiência e clareza. As principais estruturas de dados, como matrizes tridimensionais para as transições e listas encadeadas para as produções das gramáticas, foram escolhidas para suportar operações dinâmicas e eficientes. A seguir irei apresentar a base do código como as structs e as funções criadas, destacando o principal ponto entre elas:

Struct AF_NO: Esta estrutura armazena todas as informações necessárias sobre o autômato, incluindo o número de estados, alfabeto de entrada, tabela de transições, estados finais e o estado inicial. A representação da função programa no código seria a tabela de transições, que recebe 3 entradas, sendo elas o estado, o símbolo e a transição, dessa forma nessa matriz tridimensional podemos armazenar mais de uma transição para a mesma entrada de estado e símbolo e assim tratar autômatos não determinísticos. A figura 1 apresenta o trecho do código da definição dessa struct.

Figura 1 - Definição da Struct AF_NO

```
18 typedef struct {
19     int numEstados;
20     int numSimbolos;
21     char alfabeto[MAX_SIMBOLOS];
22     int transicao[MAX_ESTADOS][MAX_SIMBOLOS][MAX_TRANSICOES];
23     int numTransicoes[MAX_ESTADOS][MAX_SIMBOLOS];
24     int estadoFinal[MAX_ESTADOS];
25     int estadoInicial;
26 } AF_NO;
```

Função iniciaAF: Esta função prepara o ambiente inicial do autômato, definindo todos os estados como não finais, manipulando a variável de estado final e definindo todos os valores para 0, em seguida ela define o valor 0 para o número de transações em todas as combinações de estados e símbolos, por fim as transições de todos os possíveis estados, símbolos e número de estados é dado como indefinido (-1 = indefinido). Essa inicialização é crucial para evitar inconsistências durante a entrada de dados, garantindo que o autômato comece sem resíduos de execuções anteriores. Na figura 2 podemos analisar o trecho do código responsável por iniciar o autômato:

Figura 2 - Função iniciaAF

```
52 void iniciaAF(AF_NO *af) {
53     for (int i = 0; i < MAX_ESTADOS; i++) {
54         af->estadoFinal[i] = 0;
55         for (int j = 0; j < MAX_SIMBOLOS; j++) {
56             af->numTransicoes[i][j] = 0;
57             for (int k = 0; k < MAX_TRANSICOES; k++) {
58                 af->transicao[i][j][k] = -1;
59             }
60         }
61     }
62 }
```

Função leituraAF: Esta função é responsável por interagir com o usuário para coletar todas as informações necessárias para definir um autômato. A interação ocorre por meio de prompts que solicitam o número de estados, os símbolos de entrada e as transições entre estados, além de quais estados são finais.

Inicialmente, o usuário é solicitado a inserir o número total de estados e o conjunto de símbolos do alfabeto. Essas informações definem as dimensões básicas do autômato e permitem a configuração inicial das estruturas de dados. O coração da função leituraAF está na captura das transições do autômato. Durante este processo, são utilizadas três variáveis auxiliares: est (estado): indica o estado atual de onde a transição se origina. proxEst (próximo estado): indica o estado de destino da transição. simb (símbolo): representa o símbolo que desencadeia a transição. O usuário insere as transições no formato (estado, símbolo, próximo estado). Após cada entrada, o programa verifica se foram fornecidos exatamente três elementos e se o valor para a interrupção da leitura (geralmente -1) não foi inserido.

Validação e Armazenamento: Uma vez recebida uma entrada válida, a função realiza verificações para assegurar que o estado e o símbolo correspondem aos limites definidos anteriormente (por exemplo, que o número do estado não exceda o número máximo de estados e que o símbolo esteja dentro do conjunto definido). O símbolo é então convertido para um índice inteiro baseado na sua posição no alfabeto para facilitar o acesso às transições no array tridimensional. As transições válidas são armazenadas no array de transições na estrutura do autômato. Este registro é essencial para construir a matriz de transições que será usada posteriormente na conversão para a gramática.

Na figura 3 podemos ver o trecho de código capaz de receptar a entrada de uma transição e armazena na matriz tridimensional.

Figura 3 - Trecho para leitura de transações

```
87      printf("Informe as transacoes (ex: estado simbolo proximoEstado, -1 para parar): \n");
88      int est, proxEst;
89      char simb;
90      while (scanf("%d %c %d", &est, &simb, &proxEst) == 3 && est != -1) {
91      if ((est >= 0 && est < af->numEstados) && (simb >= 'a' && simb < 'a' + af->numSimbolos))
    {
92          int simbIndex = simb - 'a';
93          int t = af->numTransicoes[est][simbIndex];
94          if (simbIndex < af->numSimbolos && t < MAX_TRANSICOES) {
95              af->transicao[est][simbIndex][t] = proxEst;
96              af->numTransicoes[est][simbIndex]++;
97              printf("Transicao adicionada: De %d, com %c, para %d\n", est, simb, proxEst);
98              }
99          } else {
100             printf("Erro ao declarar transicao, por favor, verifique o estado e simbolo\n");
101         }
102     }
```

Função converteGLUD: Esta função é o coração do processo de conversão, aplicando as regras teóricas para transformar o autômato especificado em uma gramática linear unitária à direita. Antes de começar a conversão a função inicializa a estrutura GLUD_NO que armazena as produções da gramática, do mesmo jeito que fizemos para a função iniciaAF.

Para cada estado do autômato, e para cada símbolo do alfabeto a função verifica se existe uma transição definida. Se uma transição existir a função cria uma regra de produção que reflete essa transição, baseado nos teoremas já discutidos anteriormente nesse artigo, após a conversão essa produção é adicionada no array de produções da estrutura, o ponto essencial aqui é entender que essa verificação de existência de uma transição serve para o código compreender autômatos não determinísticos, pois eles podem apresentar mais de uma função para o mesmo estado e símbolo.

Ao final da verificação de todas as transições o código parte para detectar os estados finais e assim para cada estado final, uma produção adicional é criada indicando que esse estado pode produzir a palavra vazia (símbolo ε).

A figura 4 demonstra o trecho do código que converte o autômato finito em gramática.

Figura 4 - Código para converter AF em GLUD

```
162 void converteGLUD(AF_NO *af, GLUD_NO *glud) {
163     printf("\nConvertendo AF em GLUD....\n");
164     glud->numProducao = 0;
165
166     snprintf(glud->producao[glud->numProducao++], sizeof(glud->producao[0]), "S -> Q%d", af-
>estadoInicial);
167
168     for (int i = 0; i < af->numEstados; i++) {
169         for (int j = 0; j < af->numSimbolos; j++) {
170             for (int k = 0; k < af->numTransicoes[i][j]; k++) {
171                 int proxEstado = af->transicao[i][j][k];
172                 if (proxEstado != -1) {
173                     snprintf(glud->producao[glud->numProducao++], sizeof(glud->producao[0]),
"Q%d -> %cQ%d ", i, af->alfabeto[j], proxEstado);
174                 }
175             }
176         }
177         if (af->estadoFinal[i]) {
178             snprintf(glud->producao[glud->numProducao++], sizeof(glud->producao[0]), "Q%d ->
e", i);
179         }
180     }
181 }
```

Funções Print: As funções printAF e printGLUD são utilizadas para apresentar visualmente o autômato e a gramática resultante. Elas são projetadas para formatar e exibir a definição formal idêntica a apresentada em aula, facilitando a revisão e a validação das estruturas pelo usuário.

A implementação seguiu um ciclo iterativo de desenvolvimento, incluindo testes contínuos para garantir que cada função desempenhasse corretamente suas tarefas. O software final é capaz de converter de forma eficiente e precisa, proporcionando uma ferramenta robusta para a análise e transformação de autômatos em gramáticas.

.

3 RESULTADOS E DISCUSSÃO

Os resultados obtidos com o desenvolvimento e a implementação do software que converte autômatos finitos em gramáticas lineares unitárias à direita demonstraram a eficácia da abordagem escolhida. O software foi capaz de processar uma variedade de autômatos, tanto determinísticos quanto não determinísticos, e gerar as correspondentes gramáticas com alta precisão.

Para demonstrar as respostas geradas pelo código irei pegar um exemplo passado em sala de aula, rodar ele no meu código e apresentar os resultados gerados. Figura 5 representa o exercício e a Figura 6 a resposta gerada pelo código.

Figura 5 - Exercicio de conversão

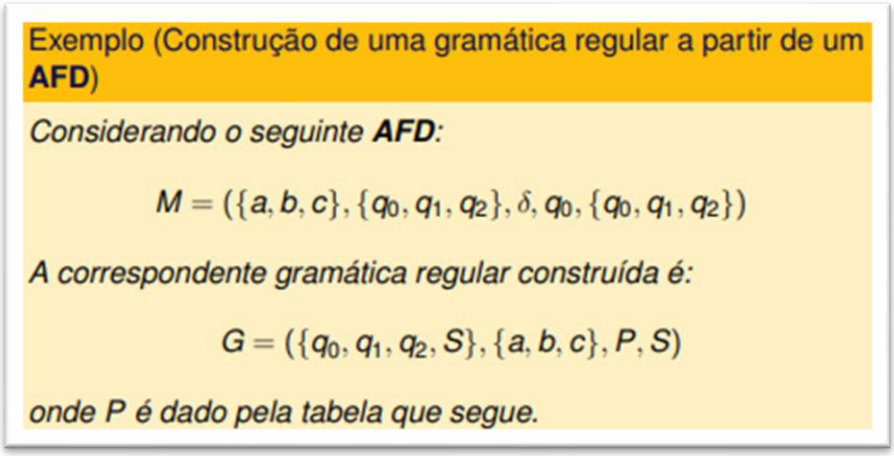

Figura 6 - Resultado gerado pelo Código

```
Definicao formal do Automato Finito a ser traduzido:
M = (E, Q, y, q0, F)
E = {a, b, c}
Q = {Q0, Q1, Q2}
y = {
    y(Q0, a) -> {Q0}
    y(Q0, b) -> {Q1}
    y(Q1, b) -> {Q1}
    y(Q1, c) -> {Q2}
    y(Q2, c) -> {Q2}
q0 = Q0
F = {Q0, Q1, Q2}

Convertendo AF em GLUD....

Definicao formal da Gramatica Linear Unitaria a Direita:
G = (V, T, P, S)
V = {S, Q0, Q1, Q2}
T = {a, b, c}
P = {
   S -> Q0
   Q0 -> aQ0
   Q0 -> bQ1
   Q0 -> e
   Q1 -> bQ1
   Q1 -> cQ2
   Q1 -> e
   Q2 -> cQ2
   Q2 -> e
}
S = S
```

Podemos analisar que o código printa a definição formar do autômato e da gramática, esse fator foi escolhido para facilitar a identificação da tradução correta. Além disso verificando os resultados podemos perceber que ele está correto, uma vez que todas as transições foram convertidas, a transição inicial S foi adicionada e as transições dos estados finais para palavra vazia também foram adicionados.

Agora após apresentar um exemplo de um autômato finito determinístico irei apresentar o código trabalhando com um autômato não determinístico e apresentar

seus resultados. Figura 7 representa o exercício de conversão de autômato não determinístico e Figura 8 o resultado gerado com essa entrada.

Figura 7 - Diagrama não determinístico a ser transformado

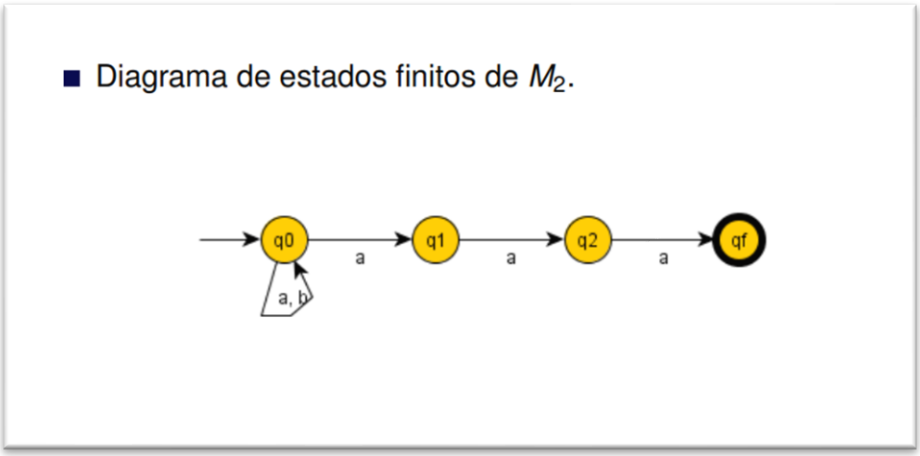

Figura 8 - Resultado do não deterministico

```
Definicao formal do Automato Finito a ser traduzido:
M = (E, Q, y, q0, F)
E = {a, b}
Q = {Q0, Q1, Q2, Q3}
y = {
    y(Q0, a) -> {Q0, Q1}
    y(Q0, b) -> {Q0}
    y(Q1, a) -> {Q2}
    y(Q2, a) -> {Q3}
q0 = Q0
F = {Q3}

Convertendo AF em GLUD....

Definicao formal da Gramatica Linear Unitaria a Direita:
G = (V, T, P, S)
V = {S, Q0, Q1, Q2, Q3}
T = {a, b}
P = {
   S -> Q0
   Q0 -> aQ0
   Q0 -> aQ1
   Q0 -> bQ0
   Q1 -> aQ2
   Q2 -> aQ3
   Q3 -> e
}
S = S
```

O que podemos analisar desse segundo teste está referente a definição da forma normal do autômato, aonde Q0 lendo a entrada “a” pode levar a dois resultados distintos, na definição formal da gramática isso foi traduzido para duas transições distintas, além de claro, adicionar o teorema de S apontando para o estado inicial e os estados finais gerando palavra vazia.

Analisando os resultados podemos concluir que as gramáticas geradas estão de acordo com os teoremas e implicações teóricas previamente estabelecidas,

validando a correção das conversões realizadas. Devemos destacar que a interface do usuário provou ser intuitiva e eficaz, facilitando a interação do usuário com o sistema e permitindo a entrada e visualização de dados de forma simples e rápida.

Após a criação do código podemos chegar em alguns feedbacks, o primeiro que gostaria de destacar está relacionado a criação de uma matriz tridimensional e o preenchimento dela, essa metodologia foi eficaz, mas apresenta um gasto grande e muitas vezes inútil de dados do sistema, uma possibilidade seria tratar os autômatos não determinísticos com um ponteiro de resultados. Outro ponto principal está relacionado as variáveis globais que não podem ser alteradas pelo usuário, então o código tem uma limitação máxima dependendo do valor dado pelo programador (para teste foi definido o valor 20).

4 CONCLUSÃO

Este estudo demonstrou com sucesso em converter autômatos finitos, tanto determinísticos quanto não determinísticos, em gramáticas lineares unitárias à direita por meio de um software desenvolvido em linguagem C. Através da compreensão da teoria, foi possível aplicar uma metodologia sólida e uma implementação cuidadosa, além da realização da tradução do autômato para gramatico, esse artigo foi capaz de traduzir uma teoria em pratica, proporcionando uma ferramenta útil e eficaz para a análise e transformação de modelos computacionais em estruturas formais de linguagem.

Os resultados obtidos reforçam a eficácia do software em realizar conversões precisas e confiáveis, alinhadas com os princípios teóricos estabelecidos. Além disso, nesse estudo foi definidos feedbacks para uma possível melhoria futuramente, tanto por mim quanto para um outro leitor. Encorajamos a continuação da pesquisa nesta área, com foco no aprimoramento das capacidades do software e na exploração de novas aplicações práticas para a conversão de autômatos em gramáticas. Espero que este trabalho inspire desenvolvimentos adicionais e contribua para o avanço da teoria da computação e suas aplicações práticas.

REFERÊNCIAS

AUTOMATA, P.--REQUISITE K. D. et al. **Converting DFA to regular grammar**. Disponível em: <https://www.jflap.org/modules/ConvertedFiles/DFA%20to%20Regular%20Grammar%20Conversion%20Module.pdf>. Acesso em: 25 jun. 2024.

DA COMPUTACAO, A. DE C. EM A. E. T.-D. F. DE T. **Autômatos Finitos e Não-determinismo**. Disponível em: <https://homepages.dcc.ufmg.br/~msalvim/courses/ftc/Aula1.1_AFDs-AFNs[still].pdf>. Acesso em: 25 jun. 2024.

Disponível em: <https://moodle.utfpr.edu.br/pluginfile.php/374989/mod_resource/content/4/LFA-Aula-03.pdf>. Acesso em: 25 jun. 2024a.

Disponível em: <https://moodle.utfpr.edu.br/pluginfile.php/375002/mod_resource/content/2/Handout-Aula-07-Gramatica-Regular.pdf>. Acesso em: 25 jun. 2024b.