IVONE PENQUE MATSUNO

Um Estudo dos Processos de Inferência de Gramáticas Regulares e Livres de Contexto Baseados em Modelos Adaptativos

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica

SÃO PAULO
2006

IVONE PENQUE MATSUNO

Um Estudo dos Processos de Inferência de Gramáticas Regulares e Livres de Contexto Baseados em Modelos Adaptativos

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Ricardo Luis de Azevedo da Rocha

SÃO PAULO

2006

Ficha Catalográfica

MATSUNO, IVONE PENQUE

UM ESTUDO DOS PROCESSOS DE INFERÊNCIA DE GRAMÁTICAS REGULARES E LIVRES DE CONTEXTO BASEADOS EM MODELOS ADAPTATIVOS, São Paulo, 2006. 107p.

Dissertação de Mestrado – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais

1. Inferência Gramatical. 2. Modelo Adaptativo. 3.Aprendizagem de Máquina 3. I Universidade de São Paulo. Escola Politécnica de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais. II. Título.

À minha família.

AGRADECIMENTOS

Ao meu orientador Professor Dr. Ricardo Luis de Azevedo da Rocha pela orientação, dedicação, compreensão e incentivo para realização do trabalho e das publicações de artigos realizados.

Aos professores que participaram da banca do exame de qualificação, Prof. Dr. João José Neto e Prof. Dr. César Bravo Pariente que muito contribuiram com suas revisões e sugestões na avaliação da proposta inicial desta dissertação.

A toda minha família, em especial à minha filha Natália, ao meu marido Juliano Yugoshi, à minha mãe Ordylette Penque, às minhas irmãs Elisa Penque Matsuno e Isabella Penque Matsuno, que estiveram presentes em todos os momentos e compreenderam minha ausência.

Aos amigos que estiveram comigo nestes anos em que cursei o mestrado aqui em São Paulo: Ana Cristina dos Santos, Denise Goya, Thiago Pedrazzi, Valgney Ishimi, Elisa Yamauchi, Ariane Lima, Tatiana Takei, Márcia Sayama, Valguima Odakura e Fábio Viduani.

Aos professores Ricardo Rocha e João José Neto, que propiciaram durante a realização deste trabalho também uma grande amizade com contribuições que levo para minha vida.

A todos que direta ou indiretamente contribuíram para a realização deste trabalho.

RESUMO

Este trabalho estuda o problema da inferência de gramáticas regulares e livres de contexto, através de conjunto de exemplos de cadeias que pertencem ou não à linguagem, tendo como principal objetivo a utilização do modelo adaptativo como uma possibilidade de solução para o aprendizado de gramáticas. Como principal base teórica foram utilizadas as propostas de José Neto (1993 e 1998), no que se refere aos dispositivos adaptativos e de construção de autômatos regulares com base em exemplos. Utiliza-se também das propostas já divulgadas na literatura para o problema da inferência de gramáticas, que utilizam outros modelos. Nesta pesquisa são apresentadas as aplicações, um resumo das principais proposta de soluções, bem como a implementação do uso de modelos adaptativos no processo de inferência gramatical. Através deste estudo, avaliou-se o modelo adaptativo no processo de inferência de gramáticas regulares, como também, apresentaram-se propostas de melhorias que vêm contribuir como solução para inferência de gramáticas livres de contexto. Os resultados obtidos no presente estudo demonstram que o modelo adaptativo é propício para o aprendizado de gramáticas regulares e livres de contexto, possibilitando novos caminhos na busca de solução para o problema em questão.

ABSTRACT

This study addresses the issue of inference of regular and context-free grammars by means of a set of examples of chains belonging or not to the language, having as primary purpose the use of the adaptive model as a potential solution for the learning of grammars. The key theoretical basis consisted of the proposals formulated by José Neto (1993, 1998), regarding adaptive devices and example-based regular automaton–generating devices. Proposals found in the literature have also been incorporated, which address the problem of grammar inference by using other models. The study describes the applications, with a summary of the principal solutions proposed, and the implementation of adaptive models in the grammar inference process. It also evaluates the adaptive model in the process of inference of regular grammars and puts forward proposals for improvement, as a contribution to the solution of inference in context-free grammars. The results obtained demonstrate that the adaptive model is suitable for the learning of regular and context-free grammars, widening the array of possible solutions for this problem.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS

.

AA – Autômato Adaptativo
AF – Autômato Finito
AFA - Autômato Finito Adaptativo
AFD – Autômato Finito Determinístico
AFND – Autômato Finito Não Determinístico
AInf – Autômato Inferido Positivo
AInf–- Autômato Inferido Negativo
APE – Autômato de Pilha Estruturado
APref – Autômato de Prefixo
APTA – *Augmented Prefix Tree Accept*
ASuf – Autômato de Sufixo
CYK – Cocke-Younger-Kasami
DNA – Ácido Desoxirribonucléico (ADN)
EDH - *Evidence-Driven Heuristic*
EDSM – *Evidence Driven State Merging*
$\mathbf{G}_{bb}$ – Gramática Binária Balanceada
$GMin_i$ - Gramática Mínima
JJ98 – Algoritmo Prospoto por José Neto (1998)
LAPFA - *Learn Acyclic Probabilistic Finite Automata*
LZ77 – Algoritmo de Liv and Zempel (1977)
NPO – Otimização em NP
RNA – Ácido Ribonucléico
RPNI – *Regular Positive and Negative Inference*
SAGE – *Self-Adaptive Greedy Estimate*
SYNAPSE – *Synthesis by Analyzing Positive String Examples*

1. INTRODUÇÃO

Os programas computacionais convencionais seguem usualmente um conjunto de regras bem definidas por um agente externo. As áreas de pesquisa que mais estão em ampliação são aquelas relacionadas com a possibilidade de que as máquinas possam adquirir conhecimento com base na experiência adquirida e, desta maneira, tomar decisões e executar as ações mais adequadas sem que haja necessidade de que um programador (agente externo) interfira. O aprendizado computacional tem sido pesquisado em diversas áreas e aplicações, como pode ser observado em Mitchell (1997).

Para que seja possível que um dispositivo computacional adquira algum conhecimento, é necessário ensiná-lo. Para tanto, as informações a serem reconhecidas devem ser representadas em um modelo que o computador reconheça, para que ele processe a informação e a transforme em conhecimento. As pesquisas nesta área buscam métodos para modelar matematicamente os processos de aprendizado.

A automatização do aprendizado por computador tem sido alvo de estudo na literatura em várias áreas de pesquisa e para as diversas aplicações, contribuindo para o surgimento de modelos computacionais que propiciam a aprendizagem de máquina. Em cada modelo proposto, o conhecimento, ou a informação, é adquirido e representado de determinada maneira.

O formalismo escolhido para representar o conhecimento neste trabalho, são as gramáticas. Em uma definição simples, uma gramática é um formalismo que especifica a regra de formação das sentenças de uma linguagem. Os sistemas para aprendizado computacional que utilizam o modelo de gramáticas para representação das informações do mundo real são algoritmos de inferência gramatical. Este trabalho tem como linha de pesquisa o aprendizado de gramáticas de linguagem, com base em exemplos de cadeias que pertencem ou não à linguagem.

A inferência gramatical é pesquisada por diversas linhas de pesquisa: aprendizagem de máquina, teoria e fundamentos de linguagens formais, reconhecimento de padrões sintático e

estrutural, biologia computacional e reconhecimento de voz, som e imagem. Esta diversidade de linhas de pesquisa tem apresentado diferentes modelos e soluções para este problema.

1.1. MOTIVAÇÃO

Ao definir um método que converte uma seqüência de caracteres em um formalismo denotacional, como as gramáticas que já apresenta técnicas e implementações definidas, podem-se vislumbrar outras aplicações dentro da Teoria de Linguagens, como, por exemplo, o estudo de linguagens naturais, processamento de imagens, reconhecimento de voz e de padrões, além de processamento de cadeias de genéticas, entre outras citadas no Capítulo 3.

O modelo formal de Autômato Adaptativo proposto por José Neto (1993) é propício para o aprendizado computacional e tem sido utilizado em diversas áreas de aprendizado computacional e que não fora utilizado e avaliado no processo de inferência de gramáticas.

1.2. OBJETIVO

O objetivo geral do presente trabalho é o de propor algoritmos que utilizem o formalismo de Autômatos Adaptativos no processo de inferência de gramáticas, implementá-los e apresentar os resultados obtidos.

Os objetivos específicos foram:

→ Pesquisar sobre os algoritmos e modelos utilizados no processo de inferência gramatical através de exemplos de sentenças de linguagem, ou seja, realizar um levantamento do estado da arte dos métodos de inferência de gramáticas.

→ Classificar, de acordo com a literatura, os métodos de inferência segundo a Hierarquia de Chomsky (Chomsky, 1959); portanto, foram estudados, dentro da hierarquia, os métodos regulares e livres de contexto.

→ Implementar e avaliar o algoritmo para inferência de gramáticas regulares proposto em José Neto (1998). Com base nos resultados obtidos na inferência, apresentar uma proposta de algoritmo para inferência gramatical livre de contexto e avaliá-lo.

1.3. JUSTIFICATIVA

O processo de inferência de gramáticas, conforme mencionado na seção anterior sobre motivação, tem extrema relevância e aplicação em diversas áreas de pesquisa, como, por exemplo, no processamento de dados genéticos. Entretanto, não há um algoritmo genérico que possa ser considerado como o melhor; o que há são diversas propostas [(Julié, 1998), (Sakakibara 1992, 1994, 1995, 1997), (Nevill-Manning, 1997), (Carrasco, 1999)] de algoritmos e alguns resultados teóricos, que mostram que a aproximação da gramática por alguma outra é um problema computacionalmente difícil dentro da classe de problemas NP (Ausiello et al, 1999).

Por conta disso, implementar e testar um modelo de inferência recente contribui para ampliar o conhecimento na área de inferência de gramáticas. Além disso, apresentar propostas de melhoria em algoritmos já implementados também contribui para ampliar o conhecimento na área.

O processo de aprendizagem computacional utiliza normalmente um conjunto de casos de exemplo positivos, cujos elementos devem ser gerados pela gramática que representa a linguagem, e negativos, cujos elementos não devem ser gerados pela gramática inferida. Este processo apresenta muitas limitações e deficiências (Michell, 1997). Diante disso, a busca por melhorias, demonstrações ou novas propostas de algoritmos, modelos ou formalismos que apresentem melhores resultados no processo aprendizagem podem trazer contribuições.

No compêndio relativo a problemas de aproximação, o problema do "Autômato finito mínimo consistente" é dito não-aproximável dentro de 2 - ε para qualquer ε (Crescenzi, 2006), ou seja, não é possível encontrar uma aproximação para o autômato mínimo para cadeias binárias através de conjuntos de exemplos positivos e negativos com o dobro ou menos de estados – e o problema é da classe NPO (problema de otimização em NP).

1.4. ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em cinco capítulos.

O Capítulo 1 trata da introdução geral ao tema proposto, motivação, definição do objetivo e a descrição da organização do texto.

No Capítulo 2 são apresentados os conceitos e as terminologias utilizadas ao longo do texto e como fundamento teórico deste trabalho. Aqui são apresentados os conceitos de linguagens formais e seus respectivos formalismos para tratar as linguagens formais como os reconhecedores e as gramáticas clássicos e as variações como autômatos de prefixo e sufixo, os autômatos adaptativos, as gramáticas mínimas e as estocásticas.

No Capítulo 3, é abordado o problema da inferência gramatical, cujo foco é apresentar o estado da arte em relação à pesquisa de inferência gramatical e das dificuldades intrínsecas neste processo. São apresentados também: a definição, a formalização, as principais técnicas e algoritmos utilizados no processo de inferência de gramáticas regulares e livres de contexto.

No Capítulo 4, considerando-se os fundamentos, a tecnologia adaptativa e o estado da arte, são apresentadas as propostas de algoritmos para inferência gramatical regular e livre de contexto, utilizando-se o modelo de autômatos adaptativos, os testes e resultados obtidos por meio da aplicação dos algoritmos propostos no conjunto de exemplos utilizados nos métodos referenciados no Capítulo 3 disponíveis na Internet.

No Capítulo 5, são apresentadas as conclusões, contribuições e considerações sobre o trabalho realizado, as melhorias necessárias e perspectivas de trabalhos futuros.

2. CONCEITOS

Neste capítulo apresenta-se uma visão geral dos conceitos, das nomenclaturas e terminologias dos formalismos empregados como fundamento teórico e formal utilizados para o desenvolvimento do algoritmo de inferência proposto neste trabalho..

2.1. LINGUAGENS FORMAIS

Uma linguagem formal é definida por um conjunto de cadeias que podem ser formadas sobre um conjunto de símbolos, alfabeto (Σ). A sua expressividade e os respectivos modelos para tratá-las dependem do tipo da linguagem, ou seja, quanto maior a expressividade, mais complexo serão os formalismos para tratá-las computacionalmente, como pode ser observado na Figura 2.1, a qual apresenta a classificação das linguagens por Chomsky e os respectivos formalismos para tratá-las.

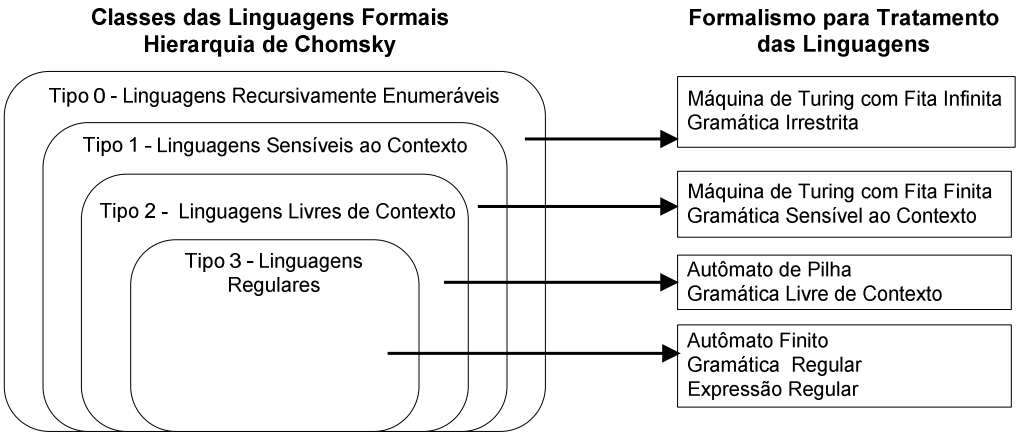


Figura 2.1 - Hierarquia de Chomsky e Formalismos.

Esta seção baseou-se em Hopcroft (1979), José Neto (1987), Lewis (1998) e Menezes (2002) para apresentação dos conceitos apresentados.

2.2. RECONHECEDORES

São dispositivos formais que servem para verificar se uma sentença pertence ou não a determinada linguagem. Aqui serão descritas as nomenclaturas dos respectivos reconhecedores utilizados para o tratamento das linguagens formais da Hierarquia de Chomsky.

2.2.1. AUTÔMATO FINITO – AF

São dispositivos reconhecedores para Linguagens Regulares e constituídos de máquinas de transições de estados sem memória. Um Autômato Finito **M** pode ser definido pela quíntupla $\mathbf{M} = (\Sigma, Q, \delta, q_0, F)$, em que:

- Σ: Conjunto de símbolos que compõem o alfabeto da linguagem.
- Q: Conjunto de estados do autômato **M**.
- δ: Conjunto de transições do autômato.
- q_0: Estado inicial do autômato **M**.
- F: Conjunto de estados finais do autômato **M**.

Cada transição do conjunto δ: $Q \times \Sigma \rightarrow Q$ descreve que um estado e um símbolo de entrada são mapeados em um estado. Um AF é determinístico (AFD) se a o conjunto de transição é definido por uma função de transição δ completa, ou seja, para todos os estados em Q existe uma transição prevista para todos os símbolos do alfabeto (Σ). Um AF é não-determínistico (AFND), se existem transições múltiplas com o mesmo símbolo partindo de um mesmo estado, ou transições sem consumo de símbolo, ou seja, transições com o símbolo que indica cadeia vazia (ε).

Nas figuras deste trabalho, a representação gráfica do estado inicial, será indicada por um círculo prenchido com a cor cinza.

2.2.2. AUTÔMATO DE PILHA ESTRUTURADO

Dentre as diversas formalizações existentes para o autômato de pilha, adota-se o uso do Autômato de Pilha Estruturado (APE) proposto em José Neto (1987). Nesta formalização, o autômato é uma máquina de estados composta por um conjunto de submáquinas com transições que consomem símbolos de entrada ou fazem chamada ou retorno de outra submáquina. Estas submáquinas comportam-se na verdade, como autômatos finitos, diferenciando-se dos mesmos apenas pelas transições de chamada e retorno de submáquinas.

Uma chamada de submáquina implica em guardar o estado de retorno em uma pilha e continuar o reconhecimento da cadeia a partir do estado inicial da submáquina em questão. Ao término do processamento dessa cadeia, sendo o estado corrente um estado final, a análise da cadeia continuará, então, a partir do estado indicado no topo da pilha, o qual será desempilhado. Este procedimento é chamado transição de retorno de submáquina.

Um APE **N** pode ser representado pela óctupla $\mathbf{N} = (Q, \Sigma, \Gamma, P, Z_0, q_0, F, A)$ em que:

- Q : Conjunto de estados do APE (todos os estados de todas as submáquinas).
- Σ : Conjunto de símbolos, que compõem o alfabeto da linguagem.
- Γ : Conjunto de símbolos, que compõem o alfabeto da pilha.
- P : Conjunto de regra de produções do APE **N**, e $P = \{ P_1 \cup P_2 \cup . .. \cup P_n \}$, em que n é o número de submáquinas do conjunto A
- Z_0: Símbolo inicial da pilha.
- q_0 : Estado inicial do APE.
- F : Conjunto de estados finais do APE.
- A : Conjunto de submáquinas do formato $a_i = (Q_i, \Sigma_i, P_i, E_i, S_i)$.

em que:

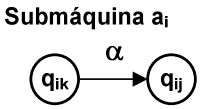
- $Q_i \subseteq Q$ é o conjunto dos estados de q_{ij} da submáquina a_i .
- $\Sigma_i \subseteq \Sigma$ é o conjunto de símbolos de entrada da submáquina a_i .
- $E_i \subseteq Q_i$ é o estado inicial da submáquina a_i (contém o único estado de entrada da submáquina a_i ($E_i = \{q_{i0}\}$).

- $S_i \subseteq Q_i$ é o conjunto de estados finais da submáquina a_i .
- $P_i \subseteq P$ é o conjunto de produções da submáquina a_i, que podem ser representadas da seguinte forma:

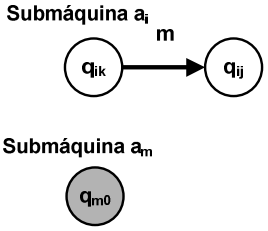
  - Transições internas da submáquina a_i : $(\gamma, q_{ik}, \alpha\beta) \rightarrow (\gamma, q_{ij}, \beta)$.
  - Transições de chamada da submáquina a_m : $(\gamma, q_{ik}, \alpha\beta) \rightarrow (\gamma(i, j), q_{m0}, \beta)$.
  - Transições de retorno da submáquina a_m : $(\gamma (i, j), q_{mn}, \alpha\beta) \rightarrow (\gamma , q_{ij} , \beta)$

Representação gráfica das transições em um autômato de pilha estruturado:

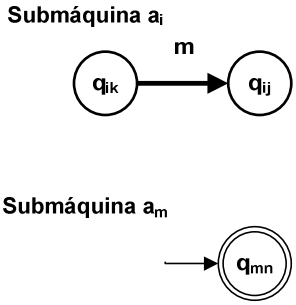
- Transições internas de submáquina $\mathbf{a_i : (\gamma , q_{ik} , \alpha\beta) \rightarrow (\gamma , q_{ij} , \beta)}$



- Transições de chamada de submáquina $\mathbf{a_m :(\gamma , q_{ik} , \alpha) \rightarrow (\gamma(i, j) , q_{m0} , \alpha)}$



- Transições de retorno de submáquina a_m :$\mathbf{(\gamma (i, j), q_{mn}, \alpha\beta) \rightarrow (\gamma , q_{ij} , \beta)}$



Nessa proposta a pilha é utilizada somente para o controle do aninhamento sintático, que é a característica que difere uma construção livre de contexto não regular (José Neto, 1987). A cada nova instância do aninhamento, uma submáquina será chamada, sendo esta chamada a responsável pelo tratamento da parte recursiva do aninhamento. Detalhes e exemplos ilustrativos de APE's podem ser encontrados em (José Neto, 1987, 1993) e (Pedrazzi, 2004).

2.2.3. AUTÔMATOS ADAPTATIVOS (AA)

Dispositivos Adaptativos são dispositivos formais convencionais dirigidos por regras. Já os não-adaptativos são imperativos, apenas executam uma seqüência de regras para todos os casos que aparecem na entrada, apresentando a mesma estrutura definida no início até o fim. Já os dispositivos formais adaptativos são definidos como aqueles cujo comportamento varia dinamicamente ao longo de sua operação como reação à recepção de determinados estímulos em sua entrada. Em dispositivos adaptativos, essa reação faz parte das características do mesmo e deve ocorrer espontaneamente, sem a interferência do operador ou de outros agentes externos.

Para incorporar tais características em um dispositivo adaptativo, cujo comportamento seja completamente descrito por um conjunto de regras, deve-se modificar, correspondentemente, o conjunto de regras que o definem. Para tornar essas modificações viáveis, os dispositivos têm de antecipar todas as possíveis alterações que seu comportamento possa sofrer. Assim, sempre que ocorrer alguma situação que exija modificação do conjunto de regras de operação, o dispositivo estará apto a fazê-la. Note-se que não há necessidade de que todas essas alterações estejam explicitamente definidas a priori, embora se exija que o dispositivo esteja apto a determiná-las em qualquer situação em que as modificações possam se tornar necessárias. Os dispositivos adaptativos devem ser criados de tal modo que sejam capazes de detectar situações nas quais não lhes seja possível dar continuidade à sua operação sem antes alterar seu comportamento e, em resposta a tal necessidade, promover todas as alterações cabíveis antes de prosseguir.

Em José Neto (2001) encontra-se um estudo sobre os conceitos que permeiam a estrutura e o funcionamento de um dispositivo adaptativo genérico. Um exemplo de aplicação, de dupla finalidade no estudo de dispositivos adaptativos, ilustra a visão conceitual apresentada nesse artigo, aplicando-a a um caso particular, mostrando como se formam as tabelas de decisão adaptativas a partir de tabelas de decisão convencionais e, em seguida, exemplificam o uso de tal dispositivo adaptativo na simulação de um outro: o autômato adaptativo. Outros modelos automodificáveis são encontrados na literatura, como em Shutt (1993, 1999), Rubinstein (1993), Klein (2002) e Jackson (2001).

Um Autômato Adaptativo (AA) proposto por José Neto (1993) é um modelo adaptativo que estende o poder dos modelos operacionais de Autômatos Finitos (AF). Em uma visão geral, um AA é um autômato de pilha estruturado capaz de alterar sua estrutura (conjunto de transições e estados) durante o reconhecimento de uma cadeia. Essa possibilidade de automodificação durante o processamento da cadeia, sem a intervenção externa, faz deste um modelo apropriado para aplicações de aprendizado computacional.

A proposta deste estudo usa o modelo de AA como base para sua implementação em suas etapas, uma vez que este pode ser trabalhado como um AA sem o uso de suas ações adaptativas, representando, dessa forma, um Autômato de Pilha Estruturado (APE); sem o uso de transições de chamadas de submáquinas, pode trabalhar como um autômato finito (AF). Isso é relevante por questões de reutilização e, principalmente, na portabilidade e adequação no uso dos formalismos necessários em cada etapa, sem a necessidade de possuir um interpretador para cada tipo de reconhecedor.

Há outras aplicações dos formalismos baseados Modelos Adaptativos em AA, tais como: Tabela de Decisão Adaptativa (José Neto, 2001), Gramáticas Adaptativas (Iwai, 2000 e Pariente, 2003) e Árvore de Decisão Adaptativa (Pistori, 2002) e suas respectivas aplicações.

2.2.3.1. Definição de Autômato Finito Adaptativo (AFA)

Formalmente um Autômato Finito Adaptativo (AFA) pode ser definido pela óctupla $M=(Q, \Sigma, q_0, F, \delta, Q_\infty, \Gamma, \Pi)$ em que os primeiros cinco elementos referem-se ao autômato finito inicial utilizando notação especificada em (Pistori, 2003).

- $Q \subseteq Q_\infty$ é o subconjunto finito de um conjunto de estados, possivelmente infinito.
- Σ é o alfabeto de entrada finito e não vazio.
- $q_0 \in Q$ é o estado inicial do autômato
- $F \subseteq Q_\infty$, é o conjunto de estados finais
- $\delta \subseteq (Q_\infty \times (\Sigma \cup \{ \varepsilon \}) \times Q_\infty)$ é a relação de transição

Os três últimos elementos de M são responsáveis pela adaptatividade do modelo:

- Q_∞ é um conjunto de estados possivelmente infinito
- $\Gamma \subseteq (\{+, -, ?\} \times (Q_\infty \times (\Sigma \cup \{ \varepsilon \}) \times Q_\infty)$ é o conjunto de ações adaptativas elementares que podem ser executadas peloAutômato Adapatativo:
  (+) inserção
  (–) exclusão
  (?) pesquisa de uma relação.
- $\Pi: (Q_\infty \times (\Sigma \cup \{ \varepsilon \}) \times Q_\infty) \rightarrow \Gamma^*$ é uma função que associa a cada transição em δ uma seqüência de ações adaptativas elementares as transições do AA. Para indicar que uma transição não possui ações adaptativas, associa-se uma seqüência vazia.

Para toda transição $\alpha \in \delta$ em que $\Pi(\alpha) = \varepsilon$, o autômato finito adaptativo opera como um autômato finito simples. Nos demais casos, antes de executar a transição o AA deverá efetuar a seqüência de ações adaptativas associadas à transição e depois operar com a sua nova estrutura. A computação, ou seja, o processamento da cadeia em questão, considera a nova configuração do AA em cada instante, podendo existir ações anteriores ou posteriores a ao consumo de símbolo.

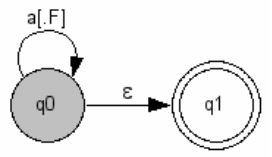
Estas características conferem a possibilidade de, a cada nova transição, poder alterar a sua própria estrutura, ou seja, adicionar ou remover estados e transições por meio das solicitações da execução das funções adaptativas.

Em Rocha (2000b) encontra-se uma demonstração da equivalência de expressividade computacional entre a Máquina de Turing e o Autômato Adaptativo. O artigo de José Neto (2001) apresenta mais detalhes do formalismo que define os dispositivos adaptativos,

baseados em regras que o presente trabalho utiliza, e um conjunto de estudos realizados, utilizando-se o modelo proposto, se encontra em LTA (2006).

2.2.3.2. Operação e Exemplo Ilustrativo

Para exemplificar a construção de operação de um AA, apresenta-se o Autômato Adaptativo M_1 para reconhecer a seguinte linguagem $L = \{w \in \{a, b, c\}^* \mid w = a^n b^n c^n, n \in \mathbf{N}\}$. , o qual é apresentado na Figura 2.2(a) e a função adaptativa **F** na transição $(q_0\, a,\, q_0)$ é definida na Figura 2.2(b)



(a) Autômato Adaptativo M1

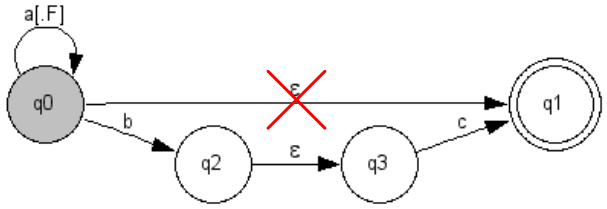
$F(q_i, a, q_j) \rightarrow \{ ?(q_i, \varepsilon, q_j),$
$-(q_i, \varepsilon, q_j),$
$?(q_j, b, q_k)$
$+(q_j, b, q_k),$
$+(q_k, \varepsilon, q_m),$
$+(q_m, c, q_n)$
$\}$

(b) Função Adaptativa F

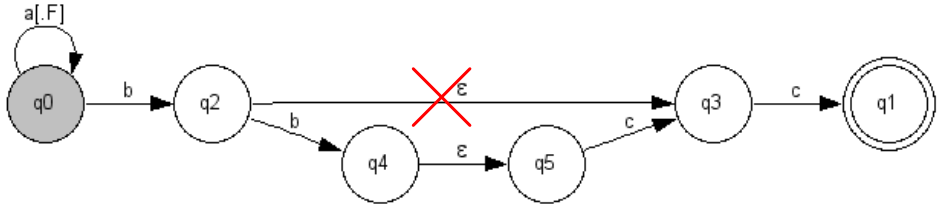
Figura 2.2 – Exemplo de Autômato Adaptativo para reconhecer $a^n b^n c^n$

Na Figura 2.3 são apresentadas as modificações do AA da figura anterior para reconhecer a cadeia *w* = "aaabbbccc".

Passo 1: Estado Corrente: q_0 ;Consumo do símbolo: 'a' ; Subcadeia de *w* = "aabbbccc"



Passo 2: Estado Corrente: q_0 ;Consumo do símbolo: 'a' ; Subcadeia de *w* = "abbbccc"



Passo 3: Estado Corrente: q_0 ;Consumo do símbolo: 'a' ; Subcadeia de *w* = "bbbccc"

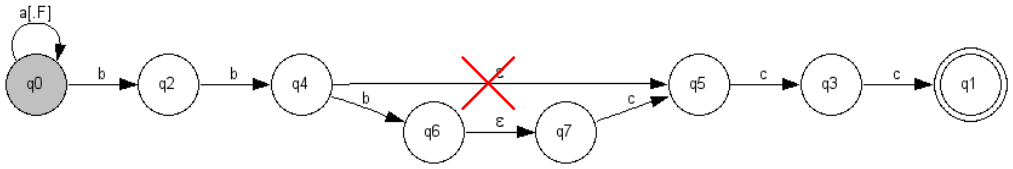


Figura 2.3 – Modificações do AA durante o reconhecimento de uma cadeia

Após a criação de transições o reconhecimento da subcadeia "bbbccc", continua como nos autômatos finitos, e o autômato adapativo M1 reconhece *w*.

2.2.4. AUTÔMATO DE PREFIXO

Um Autômato de Prefixo (A_{Pref}) é um Autômato Finito **M** com determinadas particularidades e que representa um conjunto de *n* sentenças, sem transições que representem ciclos. A partir do estado inicial *q* as transições que identifiquem prefixos de cadeias comuns são representadas por uma única transição. Cada caminho possível a partir do estado inicial até um estado final representa a sentença i, em que $\boldsymbol{0 < i < n}$.

2.2.5. AUTÔMATO DE SUFIXO

Um Autômato de Sufixo (A_{Suf}) é um Autômato Finito **M** com determinadas particularidades e que representa um conjunto de *n* sentenças, sem transições que representem ciclos. Alcançando um estado final *q* as transições que identifiquem sufixos de cadeias comuns são representadas por uma única transição. Cada caminho possível a partir de um estado inicial até o estado final representa a sentença i, em que $\boldsymbol{0 < i < n}$.

2.2.6. MÁQUINA DE TURING

Uma Máquina de Turing é o dispositivo conceitual utilizado para representar processos computacionais, conforme a Tese de Church (Lewis, 1998). Uma máquina de Turing é uma quádrupla (Σ, Q, δ, q_0), onde Σ, Q, q_0 representam respectivamente o alfabeto (inclui um símbolo especial "#" que indica que uma célula não contém símbolo), o conjunto finito de estados e o estado inicial ($q_0 \in Q$). A função δ: $Q \times \Sigma \rightarrow Q \times (\Sigma \cup \{L, R\})$ representa a ação tomada pela máquina de Turing quando se encontra em um estado q_i e encontra na fita um símbolo $a \in \Sigma$. A máquina então troca de estado, passando para um estado q_j e pode escrever um símbolo $b \in \Sigma$ na mesma célula da fita onde estava o símbolo a, como pode

movimentar o cabeçote de leitura e escrita uma posição para a direita (representado pela ação R) ou uma posição para a esquerda (representado pela ação L).

Há ainda um estado especial, o estado final (h), que é atingido pela máquina de Turing somente quando esta termina sua operação, isto é, ao finalizar sua tarefa de computação deverá ocorrer uma transição para o estado final. Somente neste estado a máquina terá efetivamente parado.

2.3. GRAMÁTICAS

De forma geral, uma gramática é um formalismo que define as regras que geram as sentenças de uma linguagem sobre um conjunto de símbolos. esta seção baseia-se na notação de gramáticas utilizada em Lewis (1998).

Definição: Uma gramática G é representada pela quádrupla $G = (N, T, P, S)$, em que

- $V = (N \cup T)$, é o vocabulário da gramática G
- N é conjunto finito de símbolos não-terminais (ou variáveis)
- T é o conjunto finito de símbolos terminais, tal que V e T são disjuntos.
- S é um símbolo não-terminal que representa o símbolo inicial da gramática
- P é o conjunto de finito de regras de produção, em que cada regra é escrita da forma:

$$\alpha \rightarrow \beta \text{ , em que } \alpha \in V^*NV^* \text{ e } \beta \in V^*$$

V^* é conjunto de todas as possíveis cadeias compostas por símbolos do vocabulário V de G, incluindo a cadeia vazia, denotada neste trabalho por ε.

2.3.1. GRAMÁTICA REGULAR

Uma **gramática é regular** se todas as regras de produção são da forma: $\alpha \rightarrow \beta$, em que $\alpha \in N$, tal que $|\alpha| = 1$ e $\beta = wA$ ou $\beta = w$, e $A \in V$ e $w \in T^*$.

As gramáticas regulares definem as regras de formação de linguagens regulares e são usualmente utilizadas para definição de cadeias mais simples de uma linguagem mais complexa. A conversão entre os formalismos de gramáticas regulares em autômatos finitos é direta.

No caso das Linguagens de Programação, essas gramáticas definem a forma em que podem ser descritos os elementos léxicos dos nomes, tais como: identificadores, palavras-chave, palavras reservadas, números e operadores.

2.3.2. GRAMÁTICA LIVRE DE CONTEXTO

Uma **gramática G é livre de contexto** se todas as regras de produções são da forma: $\alpha \rightarrow \beta$, em que $\alpha \in N$ ($|\alpha| = 1$), e tal que $\beta \in V^*$.

Com gramáticas livres de contexto, é possível formalizar regras de produção que possibilitam gerar linguagens que permitem que uma parte da cadeia seja formada por todas as regras de formação da linguagem novamente. Este caso inclui as expressões aritméticas com aninhamento de parênteses, a representação de aninhamento de blocos.

A conversão de uma gramática livre de contexto para um APE também é direta (José Neto, 1998).

2.3.3. GRAMÁTICA MÍNIMA

Uma **gramática livre de contexto** é considerada **mínima** quando gera uma única sentença. Seja x uma cadeia sobre um alfabeto, utilizar-se-á a notação G_x para representar a gramática que gera uma linguagem L, a qual é composta por uma única sentença x, $L = \{x\}$.

O tamanho da gramática considerado é dado pelo total de símbolos à direita das regras de produções de G_x. Dada uma sentença x sobre um alfabeto de comprimento n, o problema

da gramática mínima consiste em encontrar a gramática G_x, que gere somente x, tal que o tamanho de G seja menor que n, isto é, menor que a sentença original. A solução para o problema da gramática mínima é aplicada em importantes áreas, como, por exemplo, compressão de dados e reconhecimento de padrões.

2.3.4. GRAMÁTICA SENSÍVEL AO DE CONTEXTO

Uma gramática é sensível ao contexto se as regras de produção de P é da forma: $\alpha \rightarrow \beta$, onde:

- α é uma palavra de $(N \cup T)^* N (N \cup T)^*$
- β uma palavra de $(N \cup T)^*$
- $|\alpha| \leq |\beta|$ excetuando-se, eventualmente, para $S \rightarrow \varepsilon$. Neste caso, S não pode estar presente no lado direito de qualquer produção, ou seja, a cada etapa de derivação, o tamanho da palavra derivada não pode diminuir, executando-se para gerar a palavra vazia, se esta pertencer à linguagem.

2.3.5. GRAMÁTICA IRRESTRITA

Uma gramática é irrestrita quando não apresenta restrições nas regras de produção da gramática.

3. INFERÊNCIA GRAMATICAL

Inferir significa obter uma resposta generalista com base nos fatos ocorridos, analisando-se apenas uma amostra dos fatos e não todos eles. Como apresentado no Capítulo 2, as gramáticas são formalismos para definir as regras de formação das sentenças de uma linguagem. A inferência de gramáticas tem por objetivo encontrar uma gramática de uma linguagem baseada em exemplos de cadeias que pertencem ou não à linguagem em questão. A inferência gramatical, também referenciada por indução gramatical, é um subconjunto dos problemas de aprendizado computacional.

Neste capítulo serão apresentadas: uma visão geral do aprendizado computacional, a formalização do problema da inferência gramatical, as aplicações que podem utilizar inferência gramatical, e uma visão resumida dos principais métodos disponíveis na literatura.

Um sistema de aprendizado de gramáticas pode ser definido por enumeração ou por construção. O aprendizado por enumeração consiste em gerar todas as possíveis combinações de gramáticas de uma determinada classe ou, pelo menos, de um domínio considerado e, em seguida, através da análise dos exemplos, a gramática final é obtida por eliminação. Cada uma das combinações geradas que não satisfizer o conjunto de exemplos válidos será eliminada. Ao fim do processo, uma ou mais gramáticas serão consideradas como os possíveis resultados do aprendizado. O aprendizado por construção, a gramática resultante é construído diretamente dos exemplos de treinamento. Devido à inviabilidade de enumerar todas as possíveis gramáticas que poderiam representar o conceito-alvo, são estudados na literatura os métodos por construção. Os sistemas de aprendizado por construção utilizam-se de um conjunto de exemplos e de um algoritmo de inferência para obter um modelo que represente o conceito-alvo.

O processamento de um algoritmo de aprendizado é também referenciado por treinamento do sistema de aprendizado. Uma característica desejável nos algoritmos de aprendizado é a capacidade de generalização, ou seja, de associar hipóteses que consigam classificar corretamente outros exemplos, além daqueles utilizados durante o treinamento. A escolha de um conceito no espaço de hipóteses deve garantir que o conceito escolhido é

aquele que melhor representa um conceito-alvo. Dessa maneira, pode-se modelar o processo de escolha como a minimização de uma medida de erro, a qual quantifica a distância entre uma hipótese e o conceito-alvo. A avaliação de uma hipótese escolhida é, em geral, realizada sobre um conjunto de exemplos de validação. O erro da hipótese sobre os dados de validação fornece uma medida do seu desempenho.

3.1. FORMALIZAÇÃO DO MODELO DE APRENDIZADO DE GRAMÁTICAS

Devido à existência de diversas nomenclaturas nas referências encontradas, nesta dissertação será utilizada a referência adotada em Mitchell (1997), que utiliza a seguinte definição para um modelo de aprendizagem: ***MAp*** = (Σ, *C, X, E, S, F, H, T*), em que:

- Σ : Alfabeto em que os exemplos são identificados.
- ***C*** : Conjunto de todos os possíveis conceitos.
- ***X*** : Conceito-alvo que se deseja que o modelo aprenda.
- ***H*** : Espaço de hipóteses sobre o conceito-alvo que se deseja inferir.
- ***S***: Amostra do conceito ***C***. É um conjunto de fatos ou de exemplos.
    - ***S***: Subconjunto de ***C*.**
    - S^+: Conjunto de exemplos positivos, subconjunto de ***X*.**
    - S^-: Conjunto de exemplos negativos, subconjunto de Σ* - ***X*.**
    - $S=\{S^+ \cup S^-\}$.
- ***F***: Função ou algoritmo de aprendizado.
- ***H*** : Heurística para generalização do modelo inferido.
- ***T*** : Conjunto de validação ou teste.

Os modelos de aprendizado que utilizam uma linguagem formal ***L*** para a representação do conceito real, ou aplicação específica, podem utilizar-se de modelos de aprendizado de gramáticas em seus algoritmos de inferência, ou seja, obter a gramática que represente a linguagem ***L*** em questão.

As sentenças da linguagem são geradas sobre um conjunto de símbolos Σ, definido como alfabeto da linguagem. O objetivo da inferência gramatical é a de construir automaticamente uma gramática que gere uma linguagem formal ***L*** através do treinamento do

modelo de aprendizado com um conjunto de exemplos de cadeias que pertençam à linguagem, e outro conjunto que não pertencem à linguagem. Dessa forma, o modelo de aprendizado pode ser descrito como:

- Σ : Alfabeto da linguagem $\boldsymbol{L}$.
- $\boldsymbol{C}$: Aplicação do problema do mundo real.
- $\boldsymbol{X}$: Conceito-alvo, a linguagem-alvo $\boldsymbol{L}$ que se deseja aprender.
- $\boldsymbol{E}$: Características conhecidas da linguagem, como, por exemplo, a classificação da linguagem na Hierarquia de Chomsky.
- $\boldsymbol{S}$: Conjunto de cadeias que pertencem a Σ^*.
    - $\boldsymbol{S}$: Subconjunto de sentenças de $\boldsymbol{C}$.
    - $\boldsymbol{S^+}$: Conjunto de exemplos positivos, cadeias w, em que $w \in \Sigma^*$ e $w \in \boldsymbol{L}$.
    - $\boldsymbol{S^-}$: Conjunto de exemplos negativos, w, em que $w \in \Sigma^*$ e $w \notin \boldsymbol{L}$.
    - $\boldsymbol{S = \{S^+ \cup S^-\}}$.
- $\boldsymbol{F}$: Algoritmo de inferência.
- $\boldsymbol{H}$: Heurística para generalização do modelo inferido.
- $\boldsymbol{T}$: Conjunto de validação ou teste, *benchmarks* utilizados nas referências citadas e nas competições de inferência gramatical. [Tomita (1982), Duppont (1994), Abbadingo (1997), Omphalos (2004)].

O foco deste trabalho é o estudo dos algoritmos para inferência de gramáticas. De acordo com referências clássicas [Lewis (1998) e Hopcroft (1979)], dado um reconhecedor $\boldsymbol{M}$ que reconhece $\boldsymbol{L}$, pode-se obter uma gramática $\boldsymbol{G}$ que gera $\boldsymbol{L}$ e o processo inverso também é possível. Portanto, através da geração de um autômato é possível, ainda, obter uma gramática que representa o conceito-alvo. Alguns algoritmos apresentados nas subseções a seguir são baseados na construção de reconhecedores.

Os métodos de inferência de gramáticas são classificados com base na classe da linguagem $\boldsymbol{L}$ na Hierarquia de Chomsky. Inicialmente, o foco da inferência de gramáticas era na classe de linguagens regulares por tratar de linguagens de baixa complexidade. Entretanto, tal simplicidade neste tipo de linguagem, implica um conjunto de aplicações mais restrito. Recentemente, o estudo de inferência de gramáticas livres de contexto tem despertado um grande interesse devido à maior abrangência de suas aplicações. Porém, aumentando a

complexidade da classe de linguagem da gramática a ser inferida, conseqüentemente, a complexidade computacional e tempo de aprendizado também aumentam.

3.2. DIFICULDADES NO PROCESSO DE INFERÊNCIA GRAMATICAL

No processo de aprendizagem computacional em geral existem limitações e problemas relacionados, tais como: garantir o processo de aprendizagem, dependência das características dos exemplos, quantidade mínima de exemplos para o treinamento do modelo, aumento do processamento de acordo com a quantidade de exemplos e as limitações dos algoritmos de aprendizagem. Tais dificuldades também existem no processo de inferência de gramáticas, sendo a principal delas definir qual classe de linguagem ela gera.

Atualmente, a pesquisa de inferência de gramáticas lida mais especificamente com as linguagens das classes regular e livre de contexto. Muitos dos algoritmos que tratavam apenas dos aspectos regulares foram incrementados para tratar aspectos livres de contexto. Além de ser necessário distinguir os aspectos regulares dos livres de contexto, pois, em muitos casos, , é preciso distinguir os ciclos que representam aninhamentos sintáticos dos ciclos regulares lineares. Portanto, é fundamental estudar os algoritmos de inferência de linguagens regulares.

As gramáticas livres de contexto são mais expressivas; no entanto, implementações através de gramáticas regulares são freqüentemente preferidas devido à decidibilidade e eficácia nos resultados. Andrei et al (2004) apresentam uma coletânea de trabalhos relacionados aos aspectos regulares em linguagens livres de contexto e um estudo sobre as dificuldades em se distinguirem linguagens regulares de livres de contexto. Algumas subclasses de linguagens livres de contexto podem ser expressas por uma gramática livre de contexto com partes regulares.

Outras dificuldades em relação aos algoritmos de inferência gramatical são: definir algoritmos com complexidade computacional em tempo polinomial, eliminação de não-determinismos, geração de gramáticas ambíguas e dificuldade em identificar e demonstrar os problemas encontrados. Recentemente é que se tem definido padrões para os *benchmarks* e tipo de problemas que estão em aberto.

Existe um estudo teórico formal que busca responder questões como: “Em quais condições o aprendizado de máquina com sucesso é possível ou impossível?”; “Em quais condições um determinado algoritmo garante o aprendizado com sucesso?”.Em geral, essa teoria busca “leis” para restringir e avaliar o aprendizado indutivo, relacionando: a probabilidade do sucesso do aprendizado, o número de exemplos de treinamento, a complexidade do espaço de hipóteses, a precisão com a qual a função do conceito é aproximada e a forma em que os exemplos de treinamentos se apresentam, entre outras questões [Gold, (1967), Angluin (1992), Li,(1994) e Kearns, (1994)].

Todavia, na prática, após o treinamento da máquina de aprendizado, o ideal é que se possa afirmar que os exemplos positivos serão reconhecidos e os negativos, rejeitados. Inicialmente, esta é a única base; assim sendo, quando os conjuntos de exemplos são inadequados, o processo de generalização do conhecimento adquirido pela máquina apresenta respostas muito abrangentes, restringindo-se apenas aos casos negativos, ou vice-versa. Outro aspecto que desfavorece um sistema de aprendizado é o tempo de processamento e o uso de memória, que aumentam de acordo com o tamanho do conjunto de exemplos e do “conhecimento” já adquirido pela máquina.

Dado um modelo para aprendizagem de máquina, a avaliação da complexidade algorítmica segue o processo usual. Porém, os estudos teóricos para avaliar o grau de conhecimento adquirido pela máquina, ou seja, se aprendizagem foi adequada ou não, ainda não é completamente definido e aceito pela comunidade científica. Na prática, a avaliação é realizada através de uma comparação com os demais métodos já existentes executados para uma base de treinamento comum, como, por exemplo, a base de dados e resultados disponíveis em Blake (1998), Abbadingo (1997) e Omphalos (2004).

Estes aspectos de avaliação do processo de aprendizagem são importantes para definir como será avaliado um algoritmo de inferência. No presente trabalho as propostas realizadas foram testadas através da exposição dos algoritmos aqui estudados, submetendo-os ao aprendizado de linguagens conhecidas com exemplos de treinamento construídos *ad hoc* com este fim e aos *benchmarks* citados anteriormente.

3.3. APLICAÇÕES DA INFERÊNCIA GRAMATICAL

As aplicações relatadas nesta seção são também pesquisadas por outros modelos computacionais; devido ao foco deste trabalho, são apresentados os problemas do ponto de vista de gramáticas, ou seja, aplicações em que o conceito-alvo é uma linguagem formal *L*, para a qual se deseja obter uma gramática que defina suas regras de formação.

A inferência gramatical é aplicada para identificar padrões em um espaço de amostras. Considerando que tais padrões naturalmente correspondem a símbolos não-terminais em uma gramática, são exemplos de aplicações de reconhecimento de padrões: identificar regularidades em seqüências de DNA, realçar padrões em contagens musicais (Cruz-Alcazar, 1998), descobrir propriedades de linguagem de textos e reconhecimento de voz, entre outros (Miclet et al, 2004).

3.3.1. PROCESSAMENTO DE LINGUAGEM NATURAL

O estudo do processamento de linguagem natural dedica-se ao desenvolvimento de técnicas e teorias de interpretação, geração automática de frases e textos em alguma língua natural, a tradução automática entre linguagens e aprendizado de linguagens naturais.

Identificar palavras que são válidas em uma linguagem natural, como, por exemplo, na língua portuguesa, em que, de maneira simples, se pode dizer que uma palavra é formada por um conjunto de símbolos do alfabeto de S = {a,..., z} e que o conjunto das vogais é representado por *Vogais* = {a, e, i, o, u} e de consoantes por *Consoantes* = S – *Vogais* e que toda palavra, com algumas exceções, é formada por um símbolo do conjunto de consoantes concatenados com os símbolos de vogais. Mas quando não se têm as características da formação das palavras, seria interessante, tendo apenas um conjunto de exemplos de palavras desta linguagem, identificar suas regras de formação.

Quando temos o conjunto de palavras da linguagem, por exemplo, um dicionário, será que é possível verificar se um conjunto de palavras forma uma frase na linguagem? Se

possível, definir que toda frase é formada pela regra: sujeito + predicado e que sujeito é formado por um artigo + substantivo e predicado por um verbo + substantivo + complemento e assim sucessivamente.

Especificar estas regras formalmente através de uma gramática pode possibilitar a interpretação, correção e tradução. Em cada etapa do processamento de linguagens naturais, têm-se linguagens da classe regular, livre de contexto, sensível ao contexto e irrestrita.

Os exemplos de processamento de linguagem natural através de inferência de gramáticas na literatura são encontrados em Vervoort (2002) com o *Emile*, van Zaanen, (2000) e Geertzenl (2004) com o *Engine* e Parekh (2000).

3.3.2. BIOLOGIA GENÉTICA COMPUTACIONAL

A biologia genética computacional estuda as moléculas de formação do DNA, através do mapeamento dos nucleotídeos, que são estruturas menores que as formam. Uma fita de DNA pode ser considerada como uma sentença da linguagem, formada pela combinação de seus nucleotídeos, adenina, timina, guanina e citosina, representados pelos símbolos ‘a’, ‘t’, ‘g’ e ‘c’, respectivamente. A ordem com que estes nucleotídeos formam uma fita de DNA possibilita detectar propriedades existentes em determinados conjuntos de fitas com base em trechos de DNA já caracterizados.

A pesquisa de algoritmos computacionais para o processamento e para a classificação de genes tem usado modelos de diversas áreas. Os estudos que utilizam métodos de inferências de gramáticas consideram os trechos de fitas de DNA como um exemplo de sentença da linguagem da gramática a ser inferida. Além da caracterização dos genes e da sua formação básica, existem outras particularidades, como o RNA, síntese de proteínas. Propostas que estudam a biologia computacional por inferência de gramáticas são estudas por vários autores, como Wyard (1994), Dupont et al(1994), Sakakibara et al(1992), Lima (2002) e Vieira (2004).

3.3.3. COMPRESSÃO DE DADOS

A compressão de dados tem por objetivo reduzir o número de *bits* a serem armazenados ou transmitidos. Dentre os vários aspectos considerados na utilização dos algoritmos para a compactação dos dados, os principais são: a capacidade de compressão e o tempo de processamento para realizá-la. Isto é, além de reduzir a representação dos dados, os algoritmos de compressão buscam um desempenho de processamento de forma que o tempo utilizado nesse processo não comprometa a aplicação final.

Uma das técnicas de compressão de dados é baseada em gramáticas para representar os dados. Por exemplo, as sentenças de um texto que apresenta repetição podem ser representadas por um símbolo não-terminal da gramática, que especifica a sentença que deve ser gerada.

A solução da construção da gramática mínima é utilizada nesta área, pois, em vez de armazenar uma longa sentença, armazena-se a gramática que a gera. Resultados empíricos indicam que essa técnica é competitiva com as que são empregadas na prática (Kieffer, 2000a), (Nevill-Manning, 1997).

O tamanho da gramática será considerado pelo total de símbolos à direita de suas produções o qual deve ser menor que o tamanho do conjunto de dados a ser comprimido. As soluções apresentadas por Kieffer (2000b), Charikar et al(2002) e Ryterr (2002) utilizam-se de gramáticas para inferência das mesmas.

3.3.4. PROJETO DE LINGUAGENS DE PROGRAMAÇÃO E CONSTRUÇÃO DE COMPILADORES

Em uma definição simples, pode-se dizer que uma linguagem de programação é a forma padronizada para expressar instruções a serem executadas em um computador, ou seja, é um conjunto de regras léxicas, sintáticas e semânticas usadas para definir um programa. Uma linguagem permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como eles serão armazenados ou tratados e quais ações devem ser

tomadas em diversas circunstâncias. O compilador é um programa que transforma o código escrito em linguagem de programação (alto nível) em uma seqüência de instruções (programa) que será executada por um sistema computacional. O código de alto nível é também chamado por código-fonte, formado como uma seqüência de símbolos do alfabeto da linguagem. Para que os compiladores gerem códigos executáveis em uma máquina computacional, estes símbolos que formam a sentença devem ser analisados quantos aos aspectos: léxico, sintático e semântico, para que se possa, posteriormente, realizar a geração de código de máquina.

Em projetos de linguagens de programação e de compiladores, a utilização de técnicas de geração automática é pesquisada na construção automática de compiladores, em geral nos analisadores léxico e sintático, os quais podem ser gerados automaticamente com base nas especificações da gramática. Contudo, no aspecto de projeto de uma linguagem de programação, como definição formal da sintaxe não são muito estudas.. A proposta de Matsuno (2004) usa a inferência gramatical para auxiliar no processo de especificação formal das gramáticas de linguagens de programação baseadas em códigos-fonte ou trechos de exemplos. Outro exemplo pode ser encontrado na pesquisa de Nevill-Manning (1996).

Ao se definir um método de projeto de linguagens de programação que parta de um conjunto de códigos-fonte com exemplos da linguagem de programação, pode-se contribuir tanto nos aspectos de projeto de linguagem como na automatização de processos em compiladores, além de identificar estruturas sintáticas em documentos estruturados, como, por exemplo, em XML (Chiidlovski, 2001).

3.3.5. RECONHECIMENTO DE PADRÕES SINTÁTICOS EM VOZ, IMAGEM E TEXTO

Reconhecimento de padrões é uma área de pesquisa na qual se estuda a descrição ou classificação, principalmente de imagens e voz. Técnicas para reconhecê-los são importantes componentes de sistemas inteligentes.

Ao modelar o problema como gramática, tem-se que padrões podem ser representados por símbolos não-terminais, os quais, por sua vez, definem outros padrões. Exemplos de estudos de reconhecimentos de padrões por inferência gramatical são os trabalhos de Oncina (1992a), Gregor (1994) e Graña et al. (2003).

3.4. ALGORITMOS PARA INFERÊNCIA DE GRAMÁTICAS REGULARES

Métodos para a inferência de gramáticas regulares utilizam-se de modelos de Autômatos Finitos Determinísticos, uma vez que o mapeamento entre gramática regular e AFD é automático. Na literatura clássica de Teoria da Computação e Linguagens Formais, são apresentados os algoritmos e demonstrações do mapeamento entre estes modelos (Lewis, 1998, e Hopcroft, 1979).

A utilização de exemplos positivos e negativos é importante no processo de generalização de inferência. Entretanto, na prática, nem sempre se tem amostra de exemplos negativos; portanto, existe uma necessidade de pesquisar algoritmos que considerem somente amostras positivas. Quando se trata de linguagens regulares, existe a possibilidade de inferir a gramáticas apenas através de exemplos positivos (Gold, 1967); porém, ao tratar de linguagens livres de contexto, não é válida tal afirmação (de la Higuera, 2005).

Cichello e Kremer (2003) apresentam uma visão geral dos algoritmos que participaram da primeira competição na *web* de inferência gramatical regular (Abbadingo, 1997). Após esta competição, foi definido um padrão das especificações para os arquivos de treinamento e de teste para se avaliarem novos algoritmos de inferência de gramáticas regulares. Os *benchmarks,* considerados para teste em geral, são os utilizados por Tomita (1982) e Duppont (1994) em seus trabalhos e nos dados de treinamento das gramáticas regulares foi a Competição de Gowachin (2004). Os dados de treinamento e testes disponibilizados nestas competições na Internet, para uma avaliação, devem ser submetidos aos respectivos oráculos para avaliar o desempenho do algoritmo. Não é divulgada a linguagem de cada conjunto de treinamento mesmo após o término das competições.

A pesquisa realizada por Vieira (2004) apresenta uma abordagem detalhada dos principais algoritmos de inferência de gramáticas regulares e as devidas modificações para os métodos que não consideram a probabilidade estatística no processo de inferência. Neste trabalho, apresenta resumidamente apenas as linhas gerais dos principais métodos de inferência de gramáticas regulares. *Alergia* (Carrasco, 1994a), RLIPS (Carrasco, 1999), GIG (Duppont, 1994), RPNI (Duppont, 1996), *Exbar* (Lang, 1999), *Ed-Beam* (Lang, 1999), LAPFA (Ron, 1995), *Amnesia* (Ron, 1995), EDMS (Lang et al, 1998) e SAGE (Juilié, 1998).

3.4.1. RPNI – *REGULAR POSITIVE AND NEGATIVE INFERENCE*

Este é um dos mais antigos algoritmos de inferência gramatical e foi proposto por Oncina (1992b) e Lang (1992). Baseado no algoritmo de Trakhtenbrot e Barzdin (Trakhtenbrot, 1973).

Este algoritmo utiliza uma amostra positiva e outra negativa. Os prefixos são ordenados de acordo com a ordem-padrão (ordem lexicográfica). O algoritmo inicia-se construindo o Autômato de Prefixo para os exemplos positivos (*APref*+) e com os exemplos da amostra negativa, será construído o Autômato de Prefixo Negativo (*APref*-). Um autômato A_i é compatível se rejeitar todas as cadeias dos exemplos positivos e aceitar todas as dos exemplos negativos. Para cada cadeia *w* do exemplo negativo é construído um autômato A_i, para $0 < i < n$, em que *n* representa o número de exemplos negativos. Verifica-se se este é compatível com o autômato inferido reticulado (*Ret(APref+)*) através de uma busca ordenada no Autômato de Prefixo Positivo Reticulado (*Ret(APref+)*). Se for, realiza-se a junção deste autômato A_i com o *Ret(APref+)*; caso contrário, este autômato A_i será descartado..

O RPNI possui resultados interessantes tanto do ponto de vista teórico quanto do prático, pois é um dos poucos algoritmos que utilizam amostras positiva e negativa e nenhum modelo estatístico.

3.4.2. GIG

Nesta proposta, Dupont (1994) considera que os exemplos de treinamento estão em ordem lexicográfica. Inicialmente, o *AInf+* cria transições para representar o primeiro exemplo do conjunto positivo; para os demais exemplos, podem ocorrer dois casos: ou ele já está representado em *AInf+* ou não. No segundo caso, o *AInf+* é estendido para reconhecer tal exemplo. Para realizar essas extensões, usa operadores como: *structural crossover* e *structural mutation*, utilizados em algoritmos genéticos (Goldberg, 1989) para auxiliar no processo de junções e criação de novos estados. O autor compara este método como o algoritmo RPNI; os resultados obtidos com este algoritmo são melhores que o RPNI em alguns casos.

3.4.3. ALERGIA

O algoritmo Alergia, criado por Carrasco e Oncina (Carrasco, 1994a), segue as mesmas diretrizes do RPNI. Após construir o *Ret(APref+)*, o algoritmo segue um caminho no autômato *Ret(APref+),* por meio do processo de junção de estados. A principal diferença é que este algoritmo utiliza apenas conjuntos de treinamento de exemplos positivos e considera uma distribuição de probabilidade nas amostras de treinamento de ***L***. A distribuição é considerada de acordo com o aparecimento em exemplos positivos que se repetem.

A partir do conjunto de exemplos positivos, o algoritmo gera um autômato de prefixo *APref+*; ao construí-lo, também são armazenadas as freqüências em que as sentenças da linguagem passam pelas transições. Em seguida, o algoritmo compara todos os estados dois a dois para verificar se eles são compatíveis (equivalentes) e quando o são, estes dois estados são representados em um único estado e as freqüências das transições com os mesmos símbolos de cada um deles são somadas.

Os resultados obtidos mediante a avaliação deste algoritmo são de que o tempo do algoritmo cresce linearmente em função do tamanho do conjunto de exemplos de treinamento

e dado um conjunto de exemplos suficientemente grande, obtêm-se as gramáticas inferidas corretas.

3.4.4. RLIPS - *REGULAR LANGUAGE INFERENCE FROM PROBABILISTIC SAMPLES*

Algoritmo proposto por Carrasco e Oncina (Carrasco, 1999), ele apresenta o diferencial de acrescentar a probabilidade de ocorrência das transições, inferindo autômatos estocásticos.

3.4.5. LAPFA – *LEARN ACYCLIC PROBABILISTIC FINITE AUTOMATA*

A motivação para o desenvolvimento deste algoritmo é a dificuldade de determinação de Autômatos Estocásticos e a necessidade de tratamento de seqüências de linguagem consideradas curtas. O LAPFA foi proposto na tese de Ron (1995) e, como os demais, inicialmente constrói uma árvore de prefixo positiva a partir de um conjunto de exemplos positivos. Para cada estado da árvore de prefixo são armazenados o número de identificação do estado e o número de vezes de cadeias/sentenças, que passaram por este estado com transição com símbolo α. A cada iteração, testa se o número de cada transição (q_i, α, q_j) são maiores ou iguais ao da instância anterior. É realizada a instância de um único estado final q_f e depois utilizada a probabilidade em manter ou não a transição.

3.4.6. AMNESIA

Este algoritmo, que inclui melhorias no LAPFA, citado anteriormente, diferencia-se dos demais por utilizar árvores de sufixo preditoras, que depois são convertidas em um autômato estocástico e demonstram que este modelo restrito pode ser aprendido considerando o modelo de aprendizado tipo PAC (Ron, 1995).

3.4.7. EXBAR E ED-BEAM

Ambos dos algoritmos propostos por Lang (1999) utilizam uma estrutura de árvores *black-red* em que a cor dos nós (estados), é utilizada nas operações para unir ou não os estados, controlando, assim, os novos estados do autômato inferido. Inicialmente, estes algoritmos constróem um APTA (*Augmented Prefix Tree Accept*) detalhado por Cicchelo (2002, 2003). *Exbar* utiliza-se de uma busca exaustiva e da técnica de *backtraking* para obter a árvore de derivação. Já o *ed-beam* baseia-se em uma heurística de busca e usa uma função que define se uma operação de junção deve ser desfeita ou não.

3.4.8. EDSM – *EVIDENCE DRIVEN STATE MERGING*

Este algoritmo, proposto por Lang et al (1998), é um dos algoritmos utilizado no critério de junção dos estados durante o processo de inferência. Variações adicionais neste algoritmo e comparações são realizadas por Lucas (2002). Outras propostas com base no mesmo algoritmo podem ser observadas nos estudos de Cicchelo (2003), como por exemplo, o W-EDMS (Windowed-EDMS). Este algoritmo tem sido empregado como base para outros, novos, pelos seus bons resultados práticos.

3.4.9. SAGE – *SELF-ADAPTIVE GREEDY ESTIMATE*

Este algoritmo foi proposto por Juillé (1998) e conseguiu inferir gramáticas para as linguagens propostas na Competição de Gramáticas Regulares (Abbadingo, 1997).

Como os demais, inicialmente, gera uma árvore de prefixos, com estados ainda não analisados. Aleatoriamente, um estado q é selecionado e, então, todas as transições t são analisadas, buscando estados compatíveis na árvore de prefixos. Quando existe tal estado, a

junção dos estados é realizada. Caso contrário, um novo estado é criado e adicionado à Lista *Ls* de estados a serem analisados. O processo é repetido até que todos os estados da Lista *Ls* sejam analisados. Ao final, obtém-se um AFD válido para o conjunto de treinamento. Para generalizar este AFD, seu conjunto de transições é analisado, gerando um segundo AFD, analisando-se todas as transições ***t*** da forma (q_{origem}, α, $q_{destino}$), é criado um conjunto de estados de origem possíveis e um outro com os possíveis estados destino para o símbolo α em questão. Para definir quais são os estados que devem ser considerados é verificada a compatibilidade com o AFD de treinamento e utilizando EDH (*Evidence-Driven Heuristic*), que força a criação de novos estados.

3.5. ALGORITMOS PARA INFERÊNCIA DE GRAMÁTICAS LIVRES DE CONTEXTO

A grande dificuldade da inferência é o processo de generalizar para todo o conjunto uma gramática construída para um número finito de exemplos.. Como observado nos algoritmos para inferência de gramáticas regulares, há grande variedade de estratégias nas operações de generalização, ocorrendo o mesmo nos algoritmos de inferência de gramáticas livres de contexto. Embora existam operações comuns, como, por exemplo, criar um símbolo não-terminal, a heurística adotada em cada proposta para realizar esta operação é que distingue os resultados. A seguir, serão apresentados, em linhas gerais, os principais algoritmos encontrados na literatura, tais como a inferência por exemplos estruturados (Sakakibara, 1992), Sequitur (Nevill-Manning, 1996), Eg-GRIDS (Petasis et al, 2000) e SYNAPSE (Nakamura, 2002)

3.5.1. APRENDIZADO A PARTIR DE EXEMPLOS ESTRUTURADOS

Este algoritmo, proposto por Sakakibara (1992), processa o aprendizado de gramáticas livres de contexto combinando estratégias para considerar uma subclasse das gramáticas livres de contexto, as gramáticas reversíveis, e da inferência de autômatos a partir de conjunto de

exemplos estruturados, que através de parêntenses identifica o esqueleto da cadeia, como proposto por Levy (1978).

O algoritmo usa como base as seguintes propriedades para caracterizar os autômatos à árvore reversíveis, para que ao fim do processo o autômato seja equivalente a uma gramática livre de contexto reversível: um autômato é uma árvore em que os nós internos são denomidados estados e as folhas são os símbolos do alfabeto; a raiz de uma árvore é um estado final; e o conjunto de estados é particionado em blocos.

Considerando tais características, para cada cadeia do conjunto de exemplos é gerada uma árvore. Para cada autômato é criada uma partição em que cada estado seja um bloco. Estes blocos serão agrupados até que se obtenha um único bloco, ou seja, as árvores geradas serão unificadas em uma única árvore, onde cada partição será um símbolo não-terminal da gramática que tem como produção a expansão das folhas das subárvores.

Como base neste algortimo, Sakakibara (1994, 1997) apresenta uma abordagem que utiliza gramáticas estocásticas para inferir gramáticas livres de contexto para conjunto de exemplos para problemas da biologia computacional.

3.5.2. SEQUITUR

O algoritmo Sequitur, também proposto por Nevill-Manning (1996), gera uma gramática livre de contexto mínima e uma outra, com base nas repetições encontradas na sentença e na seqüência dos símbolos terminais e não-terminais nas regras de produção que são geradas. Ao detectar uma subcadeia **x** repetida nas regras de produção de um símbolo não-terminal **x**, uma nova regra de produção Z é criada para derivar **x** e a subcadeia **x** é substituída pelo novo não-terminal. Duas restrições são utilizadas para generalizar a gramática:

- Restrição 1: Não-existência de pares adjacentes de símbolos não-terminais, por exemplo, S→aAAb.

- Restrição 2: Um símbolo não-terminal deve existir apenas em uma produção de outro símbolo não-terminal; por exemplo: considere as seguintes regras de produção P = {S→CC, A→bc , B→aA e C→BdA}. Este conjunto não respeita esta restrição, pois o símbolo A é derivado pelo símbolo não-terminal B e C e viola a primeira restrição.

O algoritmo é descrito de tal forma que garante que as operações de criação, substituição e remoção de não-terminais não violem as restrições definidas. O Sequitur-R é uma proposta para generalizar a gramática por meio da detecção dos não-terminais auto-recursivos centrais, que são definidos por similaridade mediante a unificação das derivações em novos símbolos não-terminais (Nevill-Manning, 1997).

3.5.3. EG-GRIDS

Proposto por Petasis et al (2004b), sua característica principal é que utiliza somente exemplos positivos para inferir a gramática livre de contexto. Este algoritmo é uma melhoria no e-GRIDS (Petasis et al., 2004a), o qual é baseado no algoritmo GRID apresentado por (Langley, 2000).

A partir do conjunto de treinamento, o algoritmo gera uma gramática trivial $G_{inicial}$, que contém $n + 1$, símbolos não-terminais, em que n é o número de exemplos e cada não-terminal gera uma sentença do conjunto de treinamentos. Este algoritmo usa cinco operações básicas na análise das produções para generalizar $G_{inicial}$.

- Criar não-terminal.
- Juntar não-terminal.
- Otimizar não-terminal.
- Detectar aninhamento central embutido.
- Substituir as produções já existentes por outras.

Nas próprias operações são consideradas hipóteses e heurísticas definidas pelo algoritmo e utilizadas estratégias de busca baseadas em *beam search* e algoritmos genéticos para realizar ou não algumas operações.

3.5.4. SYNAPSE – *SYNTHESIS BY ANALYZING POSITIVE STRING EXAMPLES*

Com base nos algoritmos anteriores de Nakamura (2000 e 2002), na abordagem apresentada em 2004, utiliza o Synapse com a interação do algoritmo de CYK (Cocke-Younger-Kasami), encontrado na literatura clássica (Hopcrof, 1979), com extensões (CYK-X) para o aprendizado de gramáticas para a geração de novas regras de produção da gramática a ser inferida.

A cada exemplo, este algoritmo utiliza o CYK-X para verificar se existem regras que geram o exemplo, caso positivo, outras novas são criadas. Posteriormente, observa-se se esta nova regra gera inconsistência com os exemplos positivos já avaliados e se a nova extensão na gramática não a torna ambígua. Este algoritmo precisa de uma base de treinamento suficientemente grande e gera muitas regras de produção até obter a gramática inferida.

3.6. CONSIDERAÇÕES SOBRE O ESTADO DA ARTE

Ao estudar os métodos apresentados, nota-se a existência de diversas dificuldades e que, em cada proposta, os autores precisam propor critérios próprios durante o processo de inferência de gramáticas, tais como: características de exemplos, critérios de junção de estados, critérios para estimar a probabilidade, para realizar o tratamento de casos não-determinísticos e ambigüidades, além de utilização de geradores de *parsers* para validação, entre outros.

Cada proposta apresentada possui particularidades, pois considera as características das linguagens a serem inferidas; por esse motivo, nem sempre os critérios que auxiliam em um método podem ser úteis para determinados casos e podem prejudicar o desempenho em outros.

Em relação ao processo de inferência de gramáticas livres de contexto, as características das linguagens-alvo influenciam nas heurísticas consideradas nos algoritmos propostos. Já nos algoritmos de inferência de gramáticas regulares, as características da linguagem-alvo não são consideradas como princípios para inferir a gramática.

No capítulo a seguir, é apresentada a utilização dos modelos adaptativos no processo de inferência regular e livre de contexto. O ideal seria que as características da linguagem pudessem ser definidas antes de se iniciar o processo de inferência. Desta forma, as gramáticas inferidas poderiam ser classificadas de acordo com a Hierarquia de Chomsky. Para avaliar este aspecto, os algoritmos com o uso de modelos adaptativos e propostas de extensões foram aplicados a um conjunto de exemplos regulares e livres de contexto.

4. ALGORITMOS PARA INFERÊNCIA GRAMATICAL UTILIZANDO AUTÔMATOS ADAPTATIVOS

Neste capítulo apresenta-se o uso de autômatos adaptativos no processo de inferência gramatical tanto para linguagens regulares quanto para aquelas livres de contexto. O capítulo divide-se em três partes distintas:

- Estudo e aplicações de autômatos adaptativos para inferência de linguagens regulares;
- Proposta e estudo de autômatos adaptativos para inferência de linguagens livres de contexto e investigações;
- Melhorias nos algortimos estudados visando à inferência de linguagens livres de contexto.

4.1. ALGORITMO PARA INFERÊNCIA GRAMATICAL REGULAR

Neste trabalho utilizou-se a proposta de inferência de autômatos finitos regulares proposta por José Neto (1998), sendo que este algoritmo foi implementado e testado usando os principais *benchmarks* que tratam o problema da inferência gramatical regular, levantado no Capítulo 3. Nesta seção, apresentam-se também os resultados obtidos, os problemas encontrados e os levantamentos de questões relevantes para a melhoria da proposta.

A Figura 4.1 apresenta o esquema geral proposto para inferência de gramáticas regulares. Este processo é realizado separadamente para o conjunto de exemplos positivos e negativos, obtendo-se dois autômatos *AInf*+ e *AInf*–. Posteriormente, as cadeias dos exemplos positivos e negativos são analisadas em ambos os autômatos. O esperado é que as cadeias do conjunto de exemplos sejam reconhecidas somente no *AInf*+ e rejeitadas no *AInf*– e o contrário para as cadeias do conjunto de exemplos negativos. Quando isso não ocorre, significa que é necessário realizar modificações nos Autômatos Inferidos. Tais modificações não foram completamente definidas na proposta de José Neto (1998) e, portanto, não são realizadas nesta seção.

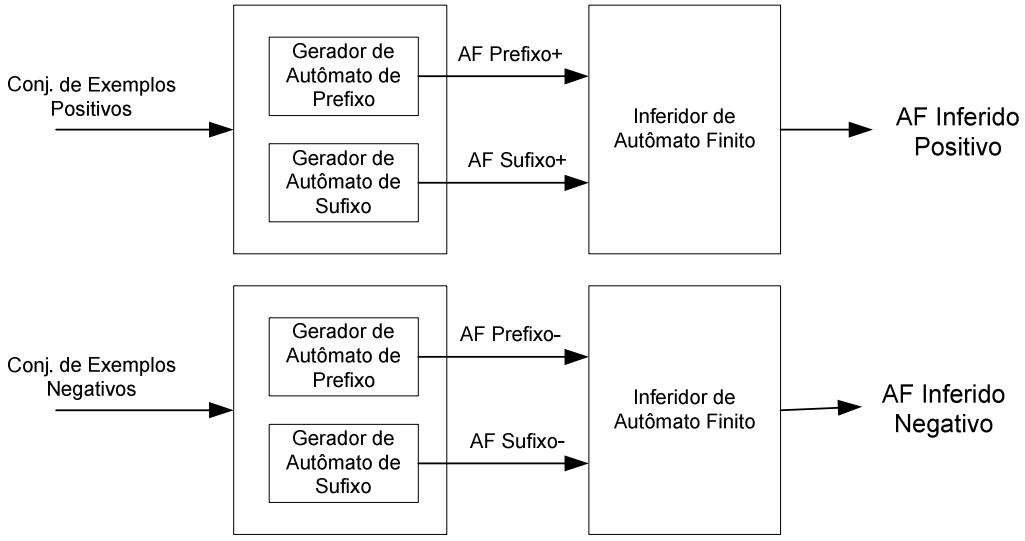


Figura 4.1 – Esquema Geral do Inferidor de Gramáticas Regulares Utilizando o Modelo Adaptativo

4.1.1. ALGORITMO PROPOSTO

O problema de inferência de uma gramática regular é baseado na existência de dois conjuntos de dados, um conjunto de exemplos positivos (S^+) e um conjunto de exemplos negativos (S^-) de uma linguagem L, para a qual se deseja inferir sua gramática regular. Usando estes exemplos um algoritmo de inferência deve produzir ao final um modelo de gramática regular (ou de autômato finito) para a linguagem L. A proposta apresentada por José Neto (1998) consiste em inferir um autômato finito que reconheça L através do percurso dos caminhos existentes nos Autômatos de Prefixo e de Sufixo construindo dois Autômatos Inferidos: o Autômato Inferido Positivo (*AInf*+) e o Negativo (*AInf*–). Após esta obtenção, pode-se convertê-lo em gramáticas.

Dado um conjunto de exemplos positivos, a primeira etapa consiste em determinar um autômato que tenha todos os prefixos em comum das cadeias do conjunto de exemplos, partindo do estado inicial, referenciado por Autômato de Prefixo (*APref*), e determinar o autômato que tenha os sufixos em comum, chegando ao estado final, referenciado por Autômato de Sufixo (*ASuf*). Esses autômatos são construídos através do reconhecimento das cadeias de exemplos pelo autômato adapativo da Figura 4.2, em que as transições para

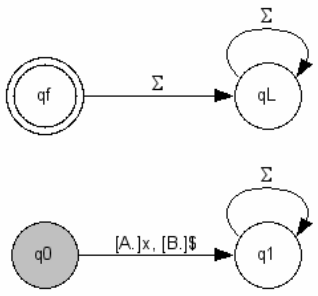
reconhecer a cadeia de entrada a partir do estado inicial são criadas e reutilizando as transições em comum. A cada nova cadeia do conjunto de exemplos, o reconehcimento inicia-se pelo estado inicial. Este autômato é apresentado no em 4.1.2.

Os autômatos *APref* e *ASuf* gerados através do conjunto de exemplos positivos são submetidos a um processo que realiza a composição dos Autômatos de Prefixo e de Sufixo gerando o Autômato Finito Inferido, de acordo com o algoritmo que será apresentado no subitem 4.1.3.

4.1.2. ALGORITMO PARA OBTENÇÃO DOS AUTÔMATOS DE PREFIXO E DE SUFIXO UTILIZANDO UM AUTÔMATO ADAPTATIVO

O Autômato de Prefixo (*APref*) é obtido pelo treinamento do Autômato Adaptativo M2 da Figura 4.2 para o conjunto de exemplos. Este autômato adaptativo realiza basicamente, a partir do estado corrente q, a verificação de existência de uma transição para consumir o símbolo de entrada α. Caso ela exista com essa característica, então ocorrerá a mudança de estado; caso contrário, ela é criada para consumir o símbolo α (q_i, α, q_j) e todas as transições a partir de q são inseridas a partir de q' para um outro novo estado q''. Ao detectar o fim da cadeia, indicado pelo símbolo '**$**' no exemplo dos autômato adaptativo da Figura 4.2(a) o estado corrente torna-se um estado final.

$\nabla x \in \Sigma$



(a) Autômato Adaptativo M2 para geração de Autômato de Prefixo

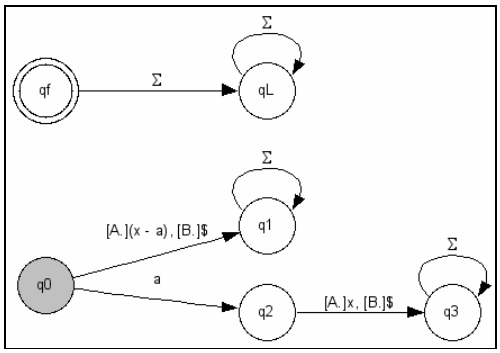
$A(q_i, \alpha, q_j) \rightarrow \{$
$\quad - (q_i, \alpha, q_j), \quad +, (q_i, \alpha, q'),$
$\quad + A(q', x\text{-}\alpha, q''), + (q', x\text{-}\alpha, q''),$
$\quad + [A].(q', \Sigma, q''), + [B].(q'', \$, q''),$
$\}$
$B(q_i, \$, q_j) \rightarrow \{$
$\quad - (q_i, \$, q_j), \quad + (q_i, \$, qL'),$
$\}$
Em que, $\alpha \in \Sigma$ e $\nabla x \in \{\Sigma - \alpha\}$ e q' e q'' são estados gerados

(b) Funções Adaptativas A e B

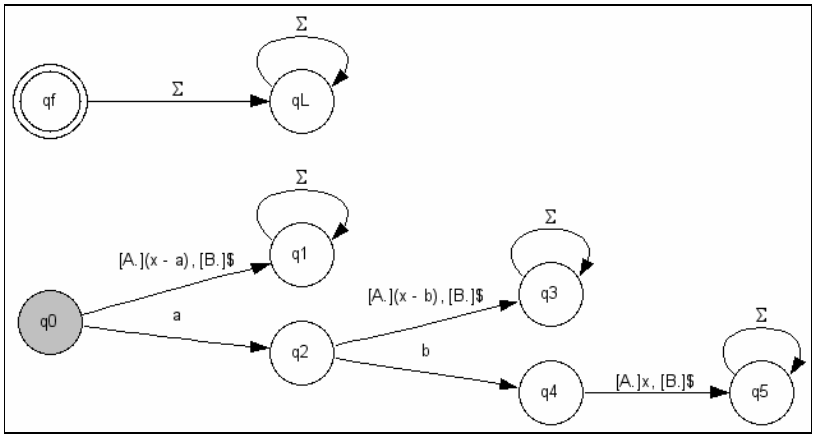
Figura 4.2 – Autômato Adaptativo para Geração do Autômato de Prefixo

A Figura 4.3 apresenta um exemplo de geração de Autômato de Prefixo e as modificações realizadas durante o reconhecimento da cadeia *w* =“ab$”.

Passo 1: Estado Corrente: q_0; Consumo do símbolo de entrada: ‘a’ ; *w* = “b$”



Passo 2: Estado Corrente: q_2; Consumo do símbolo de entrada: ‘b’ ; *w* = “$”



Passo 3: Estado Corrente: q_4; Consumo do símbolo de entrada: '$' ; w = ""

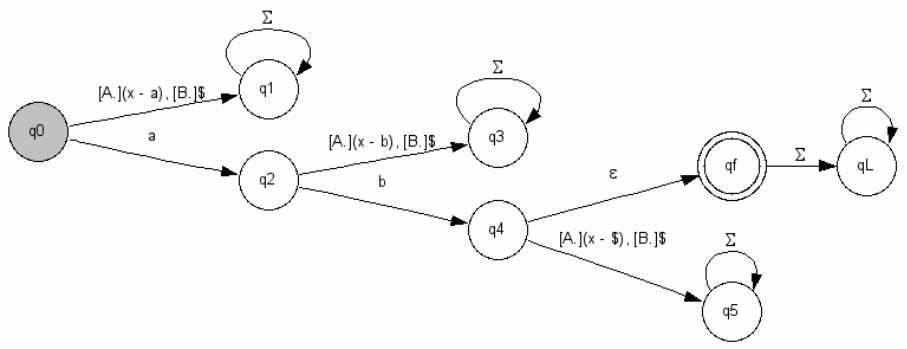
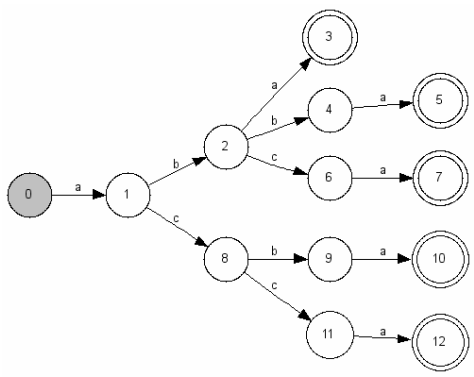


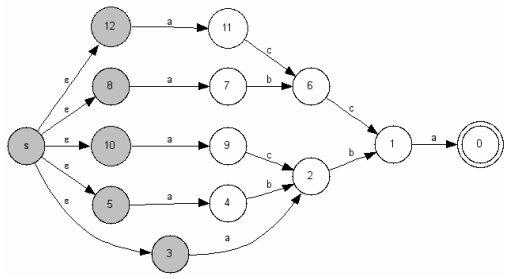
Figura 4.3 – Exemplo de Geração do Autômato de Prefixo para uma Sentença

O Autômato de Sufixo (*ASuf*) pode ser obtido de maneira semelhante através do treinamento das cadeias do conjunto de exemplos positivos em ordem inversa. Ao fim do processo tem-se um Autômato de Prefixo (*APref*) para as palavras invertidas (cadeia reversa). Para que este represente o Autômato de Sufixo desejado, a partir de cada estado final cria-se uma transição em vazio para um novo estado q_S, o qual será o novo estado inicial. O estado inicial anterior é o novo estado final e todas as transições são invertidas, ou seja, cada transição (q_j, α, q_k) é substituída por (q_k, α, q_j).

Para exemplificar a geração dos AA e AP, serão considerados exemplos de cadeias da linguagem $\boldsymbol{L_1} = a\bullet(b \cup c)^*\bullet a$, ou seja, $\boldsymbol{L_1}=\{w \in \{a,b,c\}^* \mid w$ é da forma $a\boldsymbol{u}a$ e $\boldsymbol{u} \in \{b,c\}^*$, tenham como prefixo e sufixo o símbolo '*a*'.} e o conjunto de exemplos S_1 = {*aba, abba, abca, acba, acca*}. Os Autômatos *APref* e *ASuf* gerados para esse conjunto são apresentados na Figura 4.4 (a) e (b), respectivamente.



(a) Autômato de Prefixo para **S**



(b) Autômato de Sufixo para **S** com cadeias reversas

Figura 4.4 – Autômatos de Prefixo e Sufixo para o Conjunto de Exemplos S_1

4.1.3. ALGORITMO PARA INFERIR O AUTÔMATO FINITO

A inferência do autômato para reconhecer uma linguagem regular pode ser obtida pela composição dos Autômatos de Prefixo e de Sufixo, sendo que o Autômato Inferido deve reconhecer todas as cadeias do exemplo positivo e, eventualmente, algumas cadeias adicionais que podem ser obtidas pelas mesmas regras de formação.

A idéia principal do algoritmo é percorrer, a partir do estado inicial, os Autômatos de Prefixo e de Sufixo, marcando uma nova numeração para os estados. Durante o processo de marcação da nova numeração, os caminhos analisados podem encontrar estados já com novas numerações, o que implica em novas transições ou ciclos que devem se propagar nos autômatos. A cada nova numeração nos autômatos de entrada, as transições vão compondo o autômato resultante. O processo continua até que todos os estados do Autômato de Prefixo e Sufixo tenham uma nova numeração de todos os seus estados. Ao fim desse processo, tem-se o Autômato Inferido, apresentado no Algoritmo 4.1. Na descrição deste algoritmo consideraram-se as seguintes informações:

- **Ap** é o Autômato de Prefixo para o conjunto de cadeias do exemplo.
- **As** é o Autômato de Sufixo para o conjunto de cadeias do exemplo.
- **Ai** é o Autômato resultante da composição do Autômato de Prefixo e de Sufixo.
- A função **BuscarTransição** exige a especificação do autômato e de parâmetros adicionais, como estado origem, símbolo, estado destino, estado inicial da busca. Nem todos os quatro últimos parâmetros precisam ser especificados. Esta função retorna uma lista de transições.

```
Função tipo_aa GeraAc (tipo_aa Ap, tipo_aa As)
{ i aponta para o estado inicial do Ap,  j controla a numeração dos estados em Ac
 po (estado origem), pd (estado destino) e σ (símbolo da transição)
 continua ← VERDADEIRO

 faça
    transição_Ap ← BuscaTransição (Ap, po, σ, pd, i)
        se transição_Ap é NULA então
            continua←FALSO
        senão se transição->pd tem número então
            po ← pd
        senão
            i←i+1 ,  po ← pd, pd ←NULO
            enquanto transição_Ap<> NULO faça
                transição_As ← SelecionarTransições (As, σ, i)
                se transição_As1 <> NULO então
                    j←j+1; MarcarEstadoDestino (transição_As, j)
                    CriarTransição( Ac, j-1, σ, j) // Em Ac (+, j-1, σ, j)
                    BuscarNovasTransições (transição_As)
                    Se encontrou ciclos
                        Criar ciclo em Ai
                        Propagar em As
                        Propagar em Ap
                    fim-se
                fim-se
                transição_Ap ← transição_Ap->próximo
            fim-enquanto
        fim-se
enquanto continua = VERDADEIRO
Retorna Ac
}
```

```
BuscarNovasTransições(transição_As)
{
    enquanto transição_As <> NULO faça
        x ← transição_As->po,    y ← transição_As->pd,  ρ ← transição_As->σ
        transições_As2 ← SelecionarTransições( As, ρ, 0)
        enquanto transições_As2 <> NULO faça
            se transições_As2->po = transições_As2 -> pd então
                t1 ← BuscarTransição (Ap, estado x)
                CriarTransição (Ac, x, ρ, x) // Em Ac (+, x, ρ, x)
                t2 ← BuscarTransição (Ap, estado x, ρ)
                MarcarEstadoDestino (t2, y)
            senão se transições_As2->po >transições_As2 -> pd então
                px_Ac ←BuscarEstado (Ac, x),      py_Ac ←BuscarEstado (Ac, y)
                CriarTransição (Ac, px, ρ, py) // Em Ac (+, px, ρ, py)
                t2 ← BuscarTransição (Ap, estado x, ρ, y)
                MarcarEstadoDestino (t2, y)
            fim-se
            transições_As2 ← transições_As2->próximo
        fim-enquanto
    fim-enquanto
}
```

Algoritmo 4.1 – Algoritmo para Gerar o Autômato Inferido

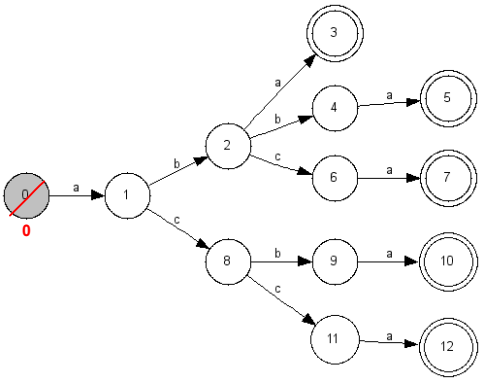
Este algoritmo proposto por José Neto(1998) será referenciado no texto como JJ98. Nas próximas sub-seções serão apresentados um exemplo do uso do algoritmo e os resultados obtidos com o uso do mesmo nos conjuntos de treinamento de Tomita(1982) e Duppont (1994).

4.1.4. EXEMPLO ILUSTRATIVO DO ALGORITMO

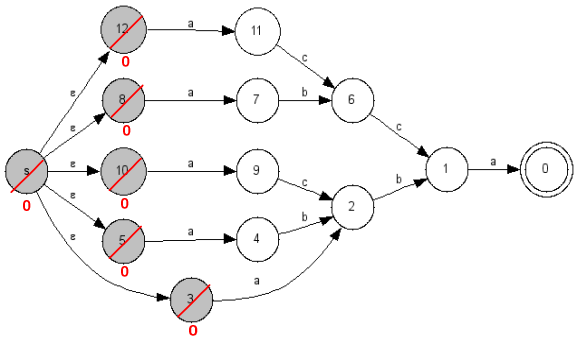
Aplicando o Algoritmo 4.1, tendo como entrada os Autômatos de Prefixo e de Sufixo da Figura 4.4, obtém-se o autômato da Figura 4.5. Nesta seção apresenta-se um exemplo ilustrativo da inferência do autômato, apresentando, resumidamente, os passos executados pelo algoritmo.

- **Passo 1:** Marcação dos estados iniciais.

Autômato de Prefixo



Autômato de Sufixo



Construção do Autômato Inferido

(a) Passo 1

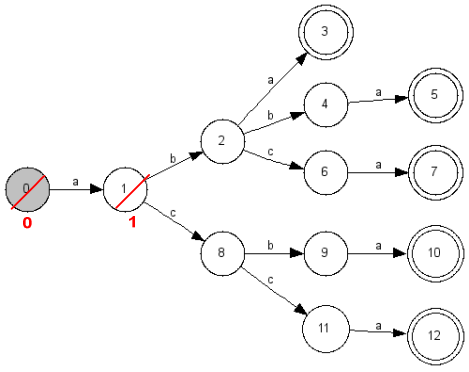
- **Passo 2:** Verificar a existência de ciclos ou novas transições a partir do estado marcado. Neste caso, não houve ocorrências de ciclos nem novas transições.

(b) Passo 2

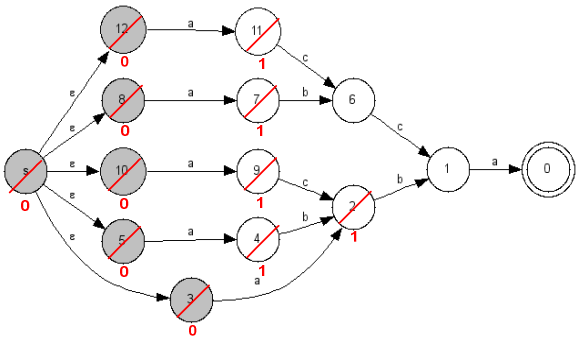
- **Passo 3:**
  - Marcação dos estados dos quais possuem transição a partir do estado 0.
  - Temos a partir de 0 a transição com o símbolo ‘a’ para o novo estado 1.
  - Inclui esta nova transição em AI.

  Propaga esta nova transição em As.

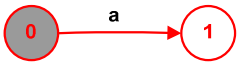
Autômato de Prefixo



Autômato de Sufixo



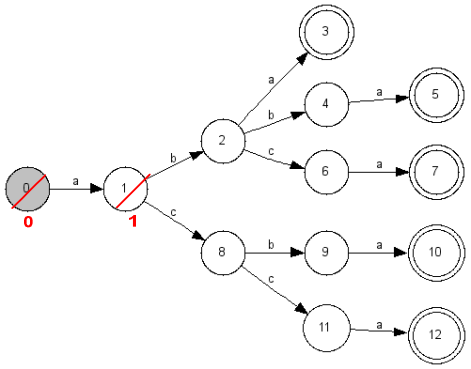
Construção do Autômato Inferido



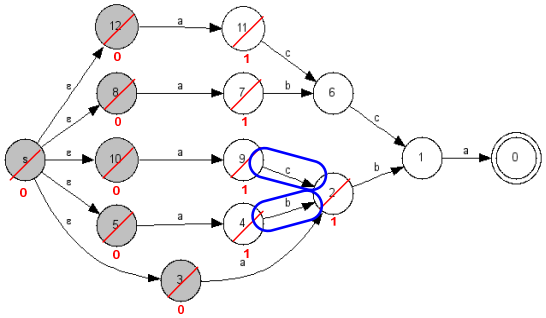
(c) Passo 3

- **Passo 4:** Ao verificar a existência de ciclos ou novas transições a partir do estado marcado, marcar-se este ciclo no *AInf*

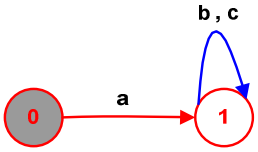
Autômato de Prefixo



Autômato de Sufixo



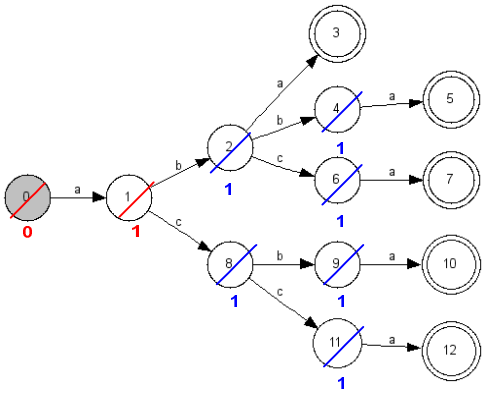
Construção do Autômato Inferido



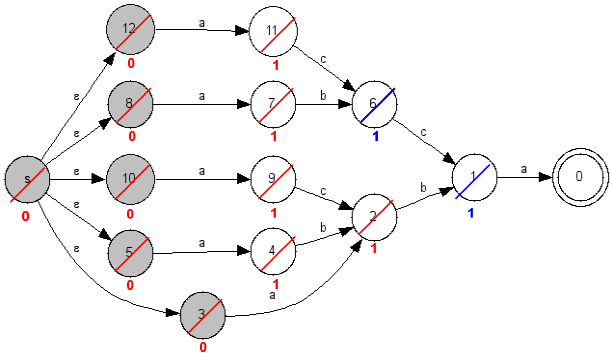
(d) Passo 4

- **Passo 5:** Propagar ciclo em As e Ap.

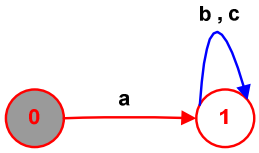
Autômato de Prefixo



Autômato de Sufixo



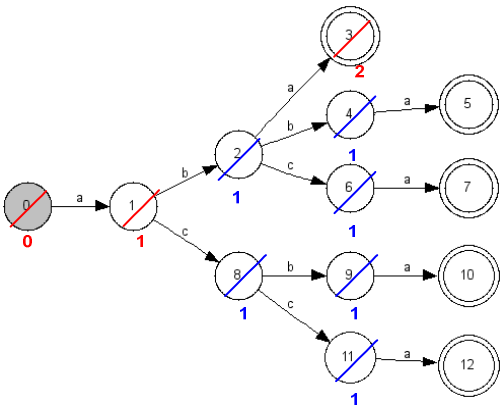
Construção do Autômato Inferido



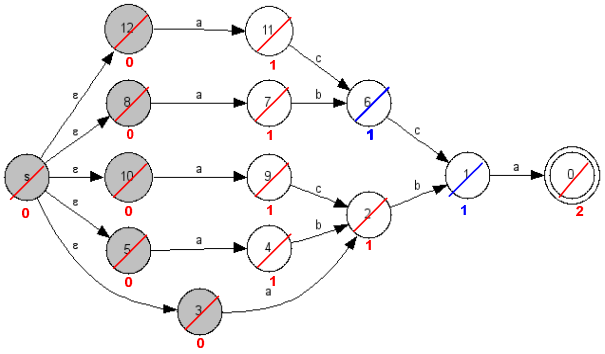
(e) Passo 5

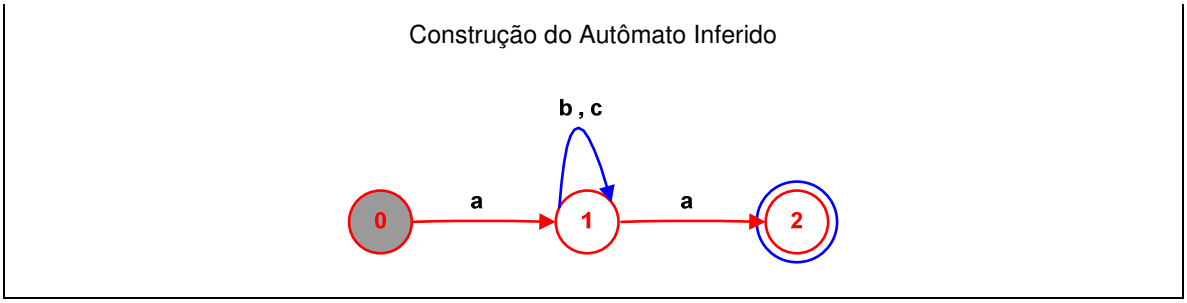
- **Passo 6:** Continuar análise a partir do estado 1.

Autômato de Prefixo



Autômato de Sufixo





(e) Passo 6

- **Passo 7:** Continuar análise a partir do estado 1.

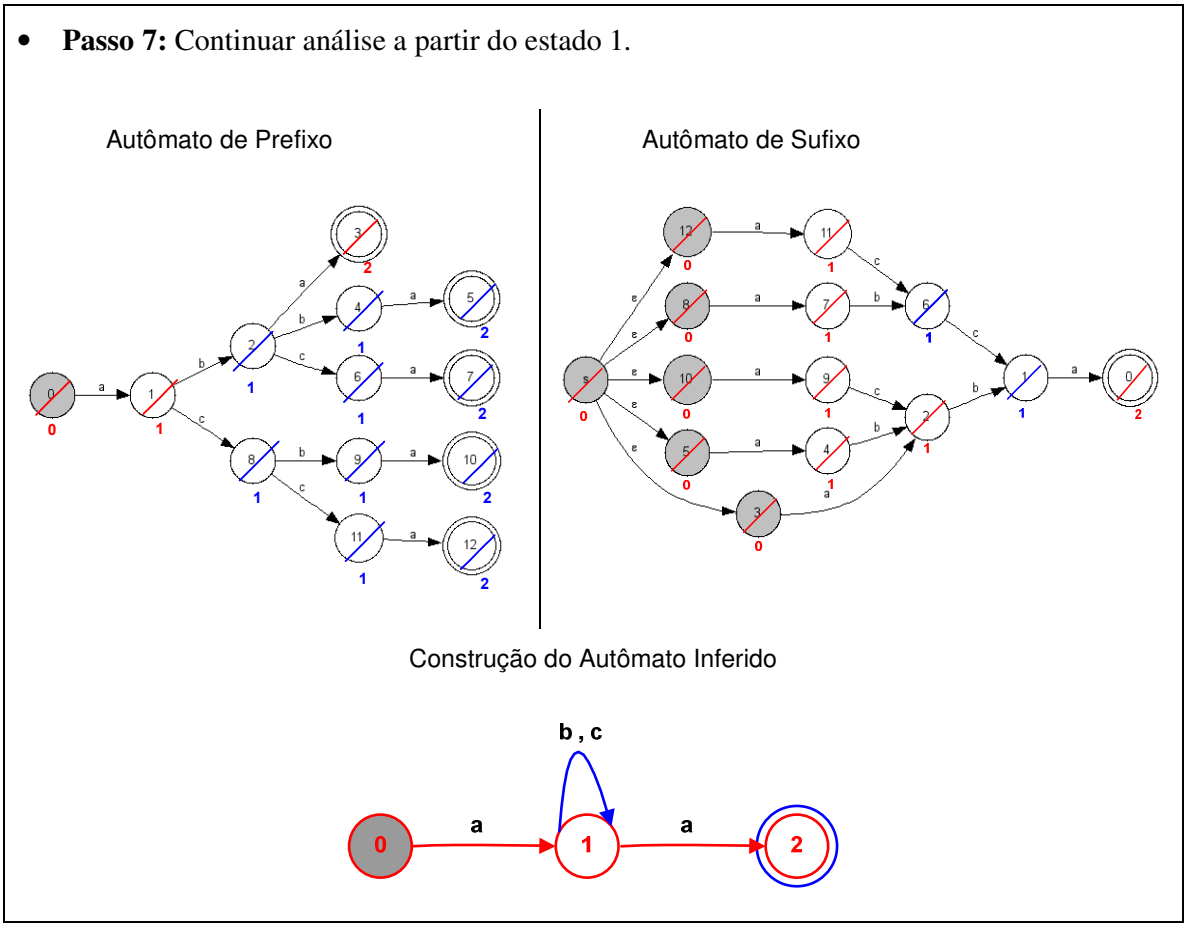


Figura 4.5 – Autômato Inferido pela Composição dos Autômatos de Prefixo e de Sufixo

Como observado, a execução do algoritmo JJ98, treinado com uma pequena quantidade de exemplos de sentenças da linguagem da linguagem *L*, o algoritmo inferiu o autômato correto.

4.1.5. TESTES DO ALGORITMO PROPOSTO

Como relatado no Capítulo 3, os conjuntos de dados de treinamento e dados de testes clássicos da literatura disponíveis na Internet são de Tomita (1982), Duppont (1994) e das competições Abbadingo (1997) e Gowachin (2004). O tipo de formato dos arquivos para treinamento e teste está disponível nos sites da própria competição.

Na Tabela 4.1 apresentam-se os resultados obtidos através do Algoritmo 4.1 para os dados de treinamento de Tomita (1982).

Tabela 4.1 – Resultados com Dados de Treinamento de Tomita (1982)

Linguagem	Resultados
$L_1 = a^*$	Gerou AFD mínimo correto somente com exemplos positivos.
$L_2 = (ab)^*$	Gerou AFD mínimo correto somente com exemplos positivos.
L_3 = qualquer sentença sem número par de as consecutivos deois de um número par de b's consecutivos	Não conseguiu, inferiu: $L_{3'} = \{w \in \{a, b\}^*\}$.
L_4 = qualquer sentença de $\{a, b\}^*$ sem mais dois a's consecutivos.	Não conseguiu, inferiu: $L_{4'} = \{w \in \{a, b\}^*\}$.
L_5 = número par de a's e número par de b's	Gerou AFD mínimo correto somente com exemplos positivos.
L_6 = qualquer sentença que o número de a's difere-se do número de b's	Gerou AFD mínimo correto somente com exemplos positivos.
$L_7 = a^*b^*a^*b^*$	Não conseguiu, inferiu: $L_{7'} = \{w \in \{a, b\}^*\}$.

Todos os Autômatos Negativos Inferidos por JJ98 com os exemplos de Tomita geraram gramáticas para $\{a,b\}^*$.

Na Tabela 4.2 apresentam-se os resultados obtidos através do Algoritmo 4.1 para os dados de treinamento de Duppont (1994)

Tabela 4.2 – Resultados com Dados de Treinamento de Duppont (1994)

Linguagem	Resultados
L_8 = a*b	Gerou AFD mínimo correto somente com exemplos positivos.
L_9 = (a* ∪ c*) b	Gerou AFD mínimo correto somente com exemplos positivos..
L_{10} = (aa)*• (bbb)*	Não conseguiu, inferiu: L_4, = {$w \in \{a, b\}^*$}
L_{11} = número par de a's e ímpar de b's	Gerou AFD mínimo correto somente com exemplos positivos.
L_{12} = a(aa)*b	Gerou AFD mínimo correto somente com exemplos positivos.
L_{13} = numero par de a´s	Gerou AFD correto somente com exemplos positivos. O AFD não é o mínimo, apresentou apenas dois estados a mais.
L_{14} = (aa)* •ba*	Não conseguiu detectar a possibilidade de palavra vazia no início da sentença. Gerou um AFD para L = (aa)(aa)*• ba*
L_{15} = (bc*b) ∪ (ac*a)	Gerou AFD mínimo correto somente com exemplos positivos.

Todos os Autômatos Negativos Inferidos por JJ98 com os exemplos de Duppont geraram gramáticas para {a,b}* ou {a,b,c}*, à exceção da linguagem L_{13}, cujo Autômato Inferido para o conjunto de exemplos negativos foi o autômato mínimo para a linguagem L_{13} Negativa ={número ímpar de a's}.

4.1.6. AVALIAÇÃO DOS RESULTADOS E PROPOSTA DE MELHORIAS

Após a realização do testes observaram-se os seguintes aspectos:

- A quantidade de exemplos de treinamento não precisa ser necessariamente grande, tanto em quantidade de exemplos como em comprimento/tamanho da sentença. Apenas com um conjunto das primeiras sentenças da linguagem, obteve-se a linguagem inferida.

- Por delimitar prefixos e sufixos das sentenças das linguagens, observou-se que, com algumas modificações, a proposta desenvolvida pode inferir também autômatos de pilha estruturados e, conseqüentemente, obter as gramáticas livres de contexto. Ao realizar o teste do algoritmo em conjunto de sentenças de linguagens livres de

contexto determinísticas, notou-se que embora seja inferido um autômato finito, o aninhamento, um ciclo auto-recursivo central, pode ser detectado utilizando o autômato inferido como base para o algoritmo proposto em 4.3.2 e a comparação com os demais algortimos em 5.3.

Os aspectos não tratados foram os seguintes:

- Em alguns casos, durante o percurso nos caminhos do Autômato de Prefixo e de Sufixo, a marcação da nova numeração dos estados percorridos gerou não-determinismo. Isto ocorre quando um estado, por um caminho, deveria ter marcação de número X e, por outro, a nova numeração seria Y. Este não-determinismo não-previsto gerava problemas nas propagações dos estados ainda não-numerados, gerando estados inatingíveis e desconexos. Entretanto, ao permitir que o autômato inferido pudesse ser não-determinístico e, posteriormente, ao eliminar o não-determinismo e minimizá-lo pelos algoritmos já propostos, o autômato inferido estava correto. Foi o caso ocorrido com a execução do conjunto de treinamentos de exemplos da linguagem de $\boldsymbol{L_{10}}$ de Duppont.

- Quando o conjunto de exemplos continha cadeias de comprimento muito grande em que suas subcadeias pertenciam à linguagem, na primeira iteração do algoritmo todos os estados eram propagados com o número 0; logo, o algoritmo inferia o autômato para a linguagem $L = \Sigma^*$.

- O algoritmo não distingue as subcadeias que possuem um mesmo símbolo como parte de suas subcadeias, $w = e1 \bullet e2$, em que $e1 = (ab)^*$ e $e2 = (aa)^*$.

- Conjunto de exemplos com *n* cadeias de exemplos, cujo valor é muito grande ($n>100$), generalizaram demais o Autômato Inferido para $L = \Sigma^*$. Ocorreu em todos os casos de treinamento de Abbadingo (1997). Realizou-se o teste do algoritmo com as primeiras 100 sentenças de treinamento e se submeteu ao oráculo disponível no *site* da Competição de Inferência Gramatical (Abaddingo, 1997). Os resultados obtidos mostraram que uma quantidade muito grande de cadeias influencia negativamente no processo de inferência.

- Com a realização dos testes e observações dos resultados, algumas características no conjunto de exemplos podem auxiliar a melhorar o desempenho do algoritmo.

Como verificado no uso do algoritmo de José Neto (1998), embora detectados alguns casos que não foram tratados e particularidades próprias, foram obtidos resultados que demonstraram aspectos positivos relevantes e que destacam o algoritmo entre os já propostos na literatura, além de ser propício o seu uso com linguagens livres de contexto, como está relatado na sub-seção 4.3.

4.2. ALGORITMO PARA INFERÊNCIA GRAMATICAL LIVRE DE CONTEXTO

Para inferir uma gramática livre de contexto, baseou-se no algoritmo para inferir a gramática mínima proposta por Charikar et al (2002) e foram realizadas modificações aqui propostas para que a gramática inferida não gerasse apenas uma sentença. Em seguida, converteu-se a gramática obtida na notação de Wirth e foram gerados os reconhecedores sintáticos das gramáticas inferidas através da construção de Autômatos de Pilha Estruturado (APE) equivalentes para realizar o teste e avaliação das gramáticas geradas. A proposta realizada pode ser assim resumida:

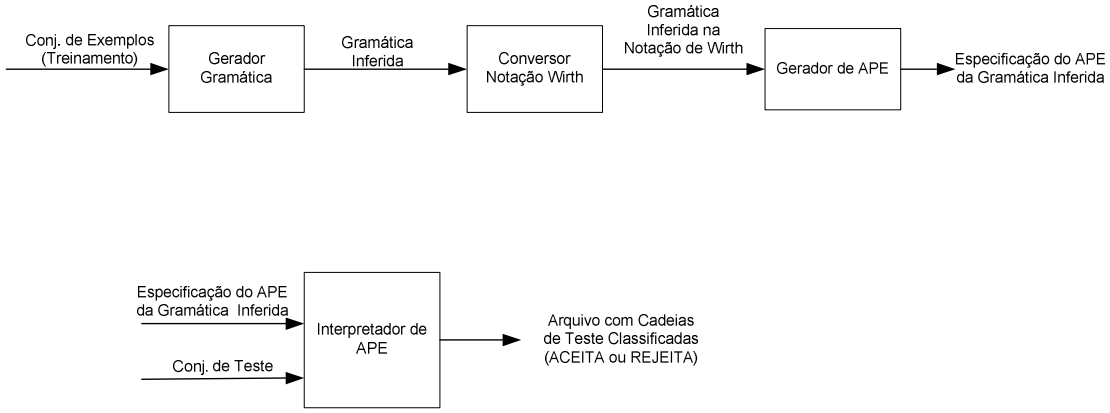


Figura 4.6 - Proposta para Inferência Gramatical Livre de Contexto

Nesta seção, será apresentada cada etapa desta proposta, que são:

- O inferidor da gramática livre de contexto, que se baseia no algoritmo da gramática mínima com as modificações que estão propostas nas sub-seções 4.2.4 e 4.3.3.
- O conversor para notação de Wirth.
- A geração dos reconhecedores de sintáticos representados por APE, baseado na proposta de José Neto (1987, 1993).
- O interpretador de APE.

Também serão apresentados os resultados e a análise dos testes realizados com exemplos usando partes de texto de linguagens de programação. Em 4.3 propõe-se o uso de exemplos clássicos determinísticos e dados de treinamento de Nakamura (2002) e disponíveis na Competição de Omphalos (Omphalos, 2004), como forma de melhor avaliar os modelos de inferência para linguagens livres de contexto.

4.2.1. ALGORITMO PROPOSTO

Neste trabalho, o inferidor de gramática livre de contexto o algoritmo de Charikar et al (2002) é utilizado para auxiliar o processo de inferência de gramáticas livres de contexto. Uma vez que este algoritmo pode inferir uma Gramática ***G*** Livre de Contexto para uma única sentença de Linguagem ***L***, nesta proposta este algoritmo é utilizado para a geração de uma gramática "mínima" para cada exemplo da linguagem livre de contexto. Porém, estas gramáticas precisam ser unificadas em uma única gramática ***G***(***L***), em que ***L*** é a linguagem livre de contexto em questão.

O problema da gramática mínima, conforme descrito no Capítulo 2, pode ser entendido como a busca pela menor gramática capaz de gerar uma sentença. Seja x uma sentença sobre um alfabeto, tal que $|x| = n$, o problema da gramática mínima é gerar uma gramática G_x, em que o tamanho de G_x seja menor que a sentença x.

Charikar et al. (2002), em um de seus artigos, estudaram esse problema e uma solução é apresentada, a qual se baseia em um algoritmo que constrói uma gramática para cada sentença, buscando a de menor complexidade. No caso, utiliza-se a medida de complexidade de Kolmogorov (Menezes, 2003). O algoritmo proposto por Charikar fundamenta-se na repetição dos símbolos da cadeia de entrada e nas propriedades de Gramáticas Binárias Balanceadas (G_{bb}.) Essa solução e respectiva implementação, realizadas por Kessey (2004), são utilizadas no Inferidor de Gramática Livre de Contexto em 4.3

4.2.2. ALGORITMO DE GERAÇÃO DA GRAMÁTICA MÍNIMA

Neste algoritmo, dada uma cadeia de entrada *x*, o objetivo é gerar a gramática G_x para serem descritos sua sintaxe e os agrupamentos dos caracteres de *x.* A repetição dos símbolos no texto de entrada *x* é a base para a geração de G_x. A primeira etapa na análise do texto é obter essa repetição; para tal, é utilizado o algoritmo de compressão LZ77 (Ziv, 1977, 1978), o qual obtém uma lista L_z de pares ordenados, (p_1, l_1), ... , (p_n, l_n), em que p_i indica a posição inicial da repetição no texto e l_i, o comprimento da subcadeia. Cada par é uma subcadeia do texto e a concatenação destes pares gera a sentença *x*.

O método para encontrar a gramática mínima estudado neste trabalho apóia-se na árvore de derivação, que será construída a partir das subcadeias de símbolos obtidas pelo algoritmo de LZ77.

Uma árvore de derivação de uma gramática *G* qualquer é construída com base nas regras de produção da gramática e como não se tem essas regras definidas, a base para a construção da árvore de derivação serão as propriedades de uma Gramática Binária Balanceada (G_{bb}). Cada símbolo da seqüência deverá ser derivado pelas produções da gramática, as quais serão construídas da forma *bottom-up* através das operações para adicionar um não-terminal, para que a árvore de derivação tenha a expansão para um par, uma seqüência ou uma subcadeia. Estas operações são detalhadas em 4.2.1.1.

Inicialmente, a gramática $G_x = (N, T, P, S_0)$ tem os conjuntos indefinidos. O conjunto de produções G_x é o único que será alterado diretamente e representado no método de geração por uma lista ativa de produções. Ao fim do processo, os demais conjuntos são obtidos a partir dessa lista.

Dada a sentença *x*, gera-se, através do algoritmo de compressão LZ77, a lista de pares $LZ = (p_1,l_1)$, ... , (p_n,l_n). Cada um deles representa uma subcadeia w_i., em particular, quando p_i é igual a zero, a subcadeia é representada por um único símbolo na posição l_i do código-fonte. A primeira subcadeia w_1, necessariamente, é um desses casos, sendo, portanto, inserida diretamente na lista ativa. Desta forma, tem-se: $P = \{S_1 \rightarrow w_1\}$, em que w_1 é uma cadeia formada por um único símbolo (p_1,l_1).

Para cada nova subcadeia w_i deverá existir uma ou mais produções para gerá-la; logo, o conjunto de produções P será modificado através da inserção de novas regras de produção, da alteração ou da remoção das regras já existentes; conseqüentemente, o conjunto de não-terminais N também será definindo. Para definir quais serão as produções necessárias para gerar w_i, serão avaliados o conjunto de produções já existente e a árvore de derivação que será construída em paralelo.

Ao analisar cada w_i, haverá dois casos:

- $\boldsymbol{w_i}$ não é gerado pelas produções de G_x.
- $\boldsymbol{w_i}$ pode ser gerado por uma ou mais produções que já existem em G_x.

No primeiro caso, uma nova regra de produção $S_k \rightarrow w_i$ é criada e w_i é representado na árvore de derivação. No segundo caso, também existem duas possibilidades:

- $\boldsymbol{w_i}$ **pode ser gerado por uma única regra de produção** $\boldsymbol{X_i}$**:** Neste caso, representa-se a subcadeia na árvore de derivação através da operação para adicionar uma subcadeia em G_{bb}. Tal operação pode alterar os símbolos terminais e acrescentar uma ou mais produções em G_{bb}; contudo, em G_x será inserida no fim da lista ativa uma única produção: $Z_i \rightarrow X_i$.
- $\boldsymbol{w_i}$ **é gerado por mais que uma produção:** Neste caso, deve-se encontrar a seqüência de não-terminais da lista ativa, que contém a expansão w_i . A seqüência de não-terminais é da forma $X_i \ldots X_t$. O primeiro e o último não-terminal da seqüência – X_i *e* X_t – são compostos por duas partes: uma que não pertence a w_i e outra que pertence.

Ao final dessas operações, G_x deverá ter seus conjuntos definidos para que gerem somente a sentença x. A árvore de derivação gerada não é a árvore a partir de G_x, mas sim as árvores de derivação auxiliares geradas pelo Algoritmo 4.2, que resume esse método de geração da gramática mínima.

```
Lz ← LZ77 (texto)
ListaAtiva ← NULO {lista de símbolos da gramática}
Inserir (ListaAtiva, 11)
i ← 2
enquanto Lz <> NULO faça
   se pi = 0 então
      Inserir(ListaAtiva,li )
   senão
      Bi ← Obter a expansão da cadeia (pi, li) através de símbolos consecutivos na
                ListaAtiva
      se Bi está contido na expansão de um único símbolo da ListaAtiva então
         Xi ←AddSubstring(G, Bi )
         InserirFim(ListaAtiva, Xi)
      senão
         X ← símbolos da lista ativa antes de Bi
         Y ← símbolos da lista ativa após Bi
         Uma nova expansão deverá ser criada com a seguinte concatenação:
         algum sufixo de X, com. Bi com algum prefixo de Y
         1 – Criar um não-terminal Mi com expansão Bi
         Mi← AddSequence(G, Bi)
         2 – Substituir a seqüência de símbolos Bi na lista ativa pelo
         único símbolo Mi
         Substituir (ListaAtiva, Bi, Mi)
         3 – Criar um não-terminal Xi com expansão igual ao prefixo X em Bi
         Criar um não-terminal Yi com expansão igual ao sufixo Y em Bi
         Xi ← AddSubstring (G, prefixo X de Bi )
         Yi ← AddSubstring (G, sufixo Yi  de Bi)
         4 – Criar um não-terminal Ni com expansão XiMiYi
         Ni ← AddSequence(G, XiMiYi)
         InserirFim(ListaAtiva, Ni)
      fim-se
   fim-se
   i←i+1
fim-enquanto
```

Algoritmo 4.2 – Geração da Gramática Mínima

Este algoritmo utiliza operações sobre gramáticas binárias balanceadas, as quais serão apresentadas a seguir, bem como um exemplos ilustrativo de geração da gramática mínima.

4.2.2.1. Gramáticas Binárias Balanceadas

Uma gramática é considerada binária somente se possuir todas as suas produções com exatamente dois símbolos do lado direito. O seu balanceamento é avaliado pelo comprimento da cadeia obtida pela expansão do símbolo não-terminal à esquerda e à direita.

Neste texto, se X é um não-terminal, será utilizado [X] para representar a cadeia que pode ser expandida por esse não-terminal. Seja uma produção da forma $Z \rightarrow XY$, tal que $[X]=\beta$ e $[Y]=\gamma$. Ela está balanceada se a equação abaixo for verdadeira para alguma constante α, tal que $0 \leq \alpha \leq ½$. Intuitivamente, α-balanceada significa que as expansões dos dois símbolos não-terminais de uma produção têm aproximadamente o mesmo comprimento. Uma gramática binária é balanceada se todas as suas produções forem balanceadas.

$$\frac{\alpha}{1-\alpha} \leq \frac{|\beta|}{|\gamma|} \leq \frac{1-\alpha}{\alpha}$$

Para construir uma árvore de derivação não violando essas propriedades, são utilizadas três operações básicas na manipulação e construção da árvore, que adicionam um não-terminal com expansão igual a um par de símbolos não-terminas, ou uma seqüência, ou uma subcadeia em uma gramática.

- **Operação para adicionar um par em uma G_{bb}**

Dada uma gramática binária balanceada G_{bb} e dois símbolos não-terminais X e Y, deseja-se incluir em G_{bb} um não-terminal Z com expansão XY, ou seja, inserir a regra de produção $Z \rightarrow XY$.

Supondo que X tem expansão $X_1X_2...X_n$ e Y tem expansão $Y_1Y_2...Y_m$ e que o comprimento de [X] é menor ou igual ao de [Y], caso contrário, o procedimento descrito a seguir é simétrico.

O primeiro passo consiste em decompor Y em uma cadeia de símbolos até que esteja balanceado com X, de acordo com a equação de balanceamento anterioriormente apresentada.

Inicialmente, a cadeia consiste do próprio símbolo Y; enquanto o primeiro símbolo da cadeia não está balanceado com X, deve-se rearranjar a cadeia segundo sua definição. Após essa verificação, tem-se Y_1 da cadeia $Y_1Y_2...Y_m$ como expansão de Y, que está balanceado com X. Note que, por construção, $Y_1Y_2...Y_i$ está balanceado com Y_{i+1}, que está balanceado para todo i, tal que $1 \leq i < m$.

A primeira produção a ser criada será $Z_1 \rightarrow XY_1$ e se define Z_1 como a raiz da árvore. Considera-se que a cada iteração, a partir da raiz, também referenciada por regra ativa, é da forma $Z_i \rightarrow A_iB_i$ .As s três afirmações a seguir devem verdadeiras:

- A expansão de Z_i é igual a $XY_1Y_2...Y_i$.
- B_i é uma subcadeia de $Y_1Y_2...Y_i$.
- Todas as regras estão balanceadas, incluindo a regra ativa.

Enquanto a regra ativa não expandir para $XY_1Y_2...Y_m$, como desejado, será incluído um símbolo Y_i a cada iteração.

Dada uma regra ativa $Z_i \rightarrow A_iB_i$ balanceada com expansão $XY_1Y_2...Y_i$, como pode ser observado na Figura 4.7, deseja-se criar um novo não-terminal Z_{i+1} com expansão $XY_1Y_2...Y_i.Y_{i+1}$, mantendo as três afirmações anteriormente citadas.

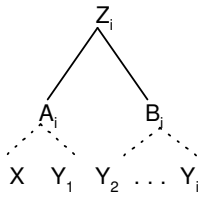


Figura 4.7 – Forma Geral das Produções Balanceadas

Existem três casos a considerar nesse processo, que poderão ser observados nas Figura 4.8 a 4.10. Nestas árvores, as linhas em negrito indicam as novas ligações entre os símbolos não-terminais.

- O primeiro caso ocorre quando Z_i está balanceado com Y_{i+1}. Aqui uma nova regra é criada: $Z_{i+1} \rightarrow Z_i\ Y_{i+1}$, como mostra a Figura 4.8.

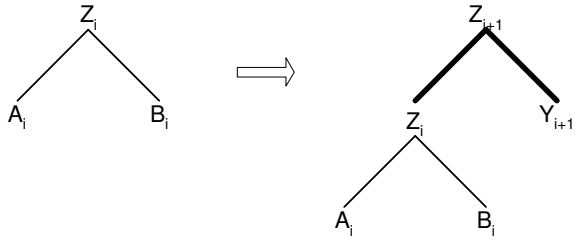


Figura 4.8 – Inserção de um Par quando a Expansão de Y_{i+1} está Balanceada com Z_i

- O segundo caso ocorre quando A_i está balanceado com B_iY_{i+1}; então, duas novas regras são criadas: $Z_{i+1} \rightarrow A_iT_i$ e $T_i \rightarrow B_iY_{i+1}$. Posteriormente, é removida a regra Z_i anterior, como pode ser observado na Figura 4.9.

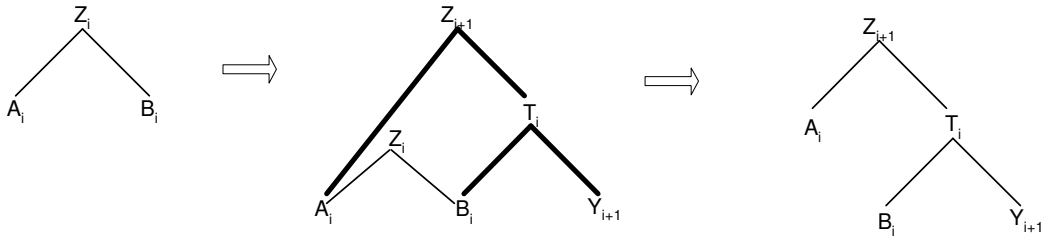


Figura 4.9 – Inserção de um Par quando a Expansão de B_iY_{i+1} Está Balanceada com A_i

- Quando ambos os casos forem falsos, então a expansão de B_i será definida pela regra $B_i \rightarrow UV$, sendo criadas três novas regras: $Z_{i+1} \rightarrow P_iQ_i$, $P_i \rightarrow A_i\ U$, $Q_i \rightarrow VY_{i+1}$. As regras Z_i e B_i serão removidas, como apresentado na Figura 4.10.

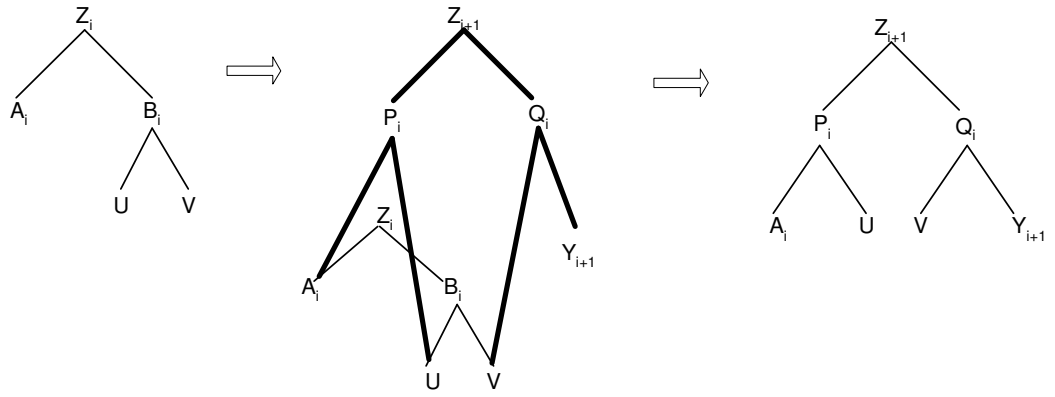


Figura 4.10 – Inserção de um Par quando a Expansão de Y_{i+1} não Está Balanceada

Após a aplicação de um desses casos, a regra Z_{i+1} criada será a nova regra ativa e esse processo será repetido até que a Z_m tenha XY como expansão.

O Algoritmo 4.3 apresenta a função da operação para adicionar um par. Essa função recebe por parâmetro os símbolos não-terminais, para o qual o novo símbolo não-terminal deverá expandir. Ele produz a árvore de derivação com essa expansão e retorna à raiz da árvore de derivação, referenciada por regra ativa. Essa função trata apenas o caso em que a expansão de X é menor ou igual a de Y. Como mencionado anteriormente, o outro caso é simétrico.

Função tipo_regra AdicionaPar (tipo_regra X, tipo_regra Y)

$A_1 \leftarrow X$
$B_1 \leftarrow Y_1Y_2...Y_m$
$t \leftarrow m \leftarrow |Y|$ <expansão de Y>

enquanto balanceamento (X, B) ≠ verdadeiro **faça**
    $B_1 \leftarrow Y_1Y_2...Y_{t-1}$
    $t \leftarrow t - 1$
fim-enquanto

$A_1 \leftarrow X$
$B_1 \leftarrow Y_1$
$Z_1 \rightarrow A_1B_1$
regra_ativa $\leftarrow Z_1$

```
i ← 1
enquanto i < m faça
   se balanceamento (Zi, Yi+1 ) = verdadeiro então
      criar a nova regra Zi+1 → ZiYi+1
      Ai+1 ← Zi
      Bi+1 ← Y1+1
   senão se balanceamento (Ai, BiYi+1 ) = verdadeiro então
      criar duas novas regras: Zi+1 → AiTi e  Ti → BiYi+1
      remover a regra Zi
      Ai+1 ← Ai
      Bi+1 ← Ti
   senão
      Bi será definida pela regra Bi →UV
      criar três novas regras: Zi+1 → PiQi ,  Pi → AiU e  Qi → VYi+1
      remover a regra Zi e Bi
      Ai+1 ← Pi
      Bi+1 ← Qi
   fim-se
   regra_ativa ← Zi+1
   i← i+1
fim-enquanto
retornar regra_ativa
fim-função
```

Algoritmo 4.3 – Operação para Adicionar um Par de Não-terminais

- **Operação para adicionar uma seqüência em uma G_{bb}**

Dada uma gramática binária balanceada G_{bb} e uma lista de símbolos não-terminais $X_1,X_2,...,X_t$ deseja-se incluir em G_{bb} um não-terminal Z com expansão $X_1X_2...X_t$, ou seja, inserir a regra de produção $Z_{i+1}\rightarrow X_1X_2,...,X_t$ na árvore de derivação que tem Z_i como raiz, como apresentado na Figura 4.11. Todavia, se deve manter o balanceamento da gramática binária.

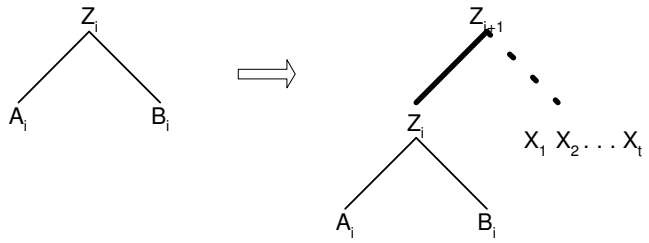


Figura 4.11 – Inserção de uma Seqüência de Símbolos sem Balanceamento

Essa operação para adicionar uma seqüência de símbolos não-terminais é uma generalização da operação para adicionar um par, utilizando-a repetidamente. A cada dois símbolos da seqüência $X_1,X_2,..., X_t$, utilizar a operação *adicionar par*, descrita anteriormente, para criar um novo não-terminal com expansão X_iX_{i+1}. Assim sendo, uma nova seqüência de não-terminais é criada e enquanto a nova lista for composta por mais de um símbolo, deve-se repetir a operação para adicionar novamente uma seqüência. A Figura 4.12 apresenta a criação da árvore de derivação para seqüência inicial com expansão $X_1X_2... X_t$ e as seqüências sucessivas que são criadas.

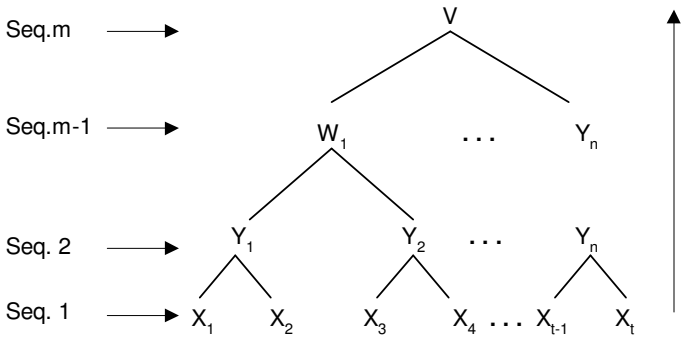


Figura 4.12 – Criação da Árvore de Derivação para a Seqüência de Símbolos

Quando a seqüência for formada por um único não-terminal V, utilizar novamente a operação para adicionar um par $Z_{i+1} \rightarrow Z_iV$. A Figura 4.13 apresenta a operação.

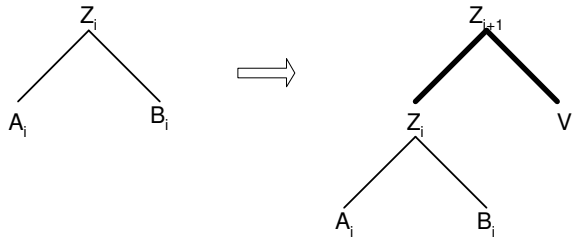


Figura 4.13 - Inserção de Produção na Árvore de Derivação

A função da operação para *adicionar uma seqüência* de não-terminais é descrita no Algoritmo 4.4. Essa função recebe por parâmetro a raiz da árvore de derivação, em que a seqüência X deve ser expandida, retornando à raiz da árvore atualizada.

```
Função tipo_regra Operação_Add_Sequence (tipo_regra raiz, lista tipo_regra X )
   Seq[1] ← X
   i ←1
   enquanto |Seq[i]| > 1 faça
      Seq[i+1] ← NULO
      enquanto Li <> NULO faça
         simb1 ← remove_inicio (Li)
         simb2 ← remove_inicio (Li)
         se simb2 <> NULO então
            simb3 ← AddPar (simb1, simb2)
            insere_fim (Seq[i+1], simb3)
         senão
            insere_fim (Seq[i+1], simb1)
         fim-se
      fim-enquanto
      i ← i+1
   fim-enquanto
   Z ← AddPar (raiz, Seq[i])
   retornar Z
fim-função
```

Algoritmo 4.4 – Operação para Adicionar uma Seqüência

- **Operação para adicionar uma subcadeia em uma G_{bb}**

Dada uma gramática binária balanceada G_{bb}, contendo um não-terminal T, que possui exatamente β como subcadeia de sua expansão, deseja-se incluir um novo não-terminal que tenha β como expansão, ou seja, inserir uma nova regra de produção que utiliza as produções já existentes, recuperando essa subcadeia.

Seja T um não-terminal com a menor expansão tal que a subcadeia β está contida da forma: $T \rightarrow XY$, em que X tem expansão $X_1X_2...X_m$ e Y tem expansão $Y_1Y_2...Y_n$.

A subcadeia $\beta = \beta_{prefixo} \bullet \beta_{sufixo}$ possui seu prefixo em X e seu sufixo em Y. A primeira etapa é encontrar a seqüência de não-terminais $X_i...X_n$, para $1 \leq i \leq n$, com expansão igual a $\beta_{prefixo}$: e $Y_1...Y_k$, $1 \leq k \leq m$, com expansão igual a β_{sufixo}, como apresentado na Figura 4.14. Considera-se que uma das seqüências pode ser vazia.

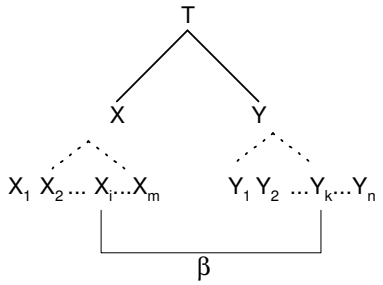


Figura 4.14 – Expansão a ser Inserida Novamente na Árvore de Derivação

Na busca pela seqüência para encontrar as expansões do prefixo e sufixo de β na árvore de derivação é construída uma árvore de derivação para gerar o sufixo com raiz $Z_{prefixo}$ e outra com expansão para o prefixo com raiz Z_{sufixo}.

A construção da árvore de derivação para gerar a expansão para $\beta_{prefixo}$ é realizada através do seguinte procedimento recursivo, que se inicia com a busca da raiz X, em que $X \rightarrow X_{esq}X_{dir}$. Através de um percurso em sua subárvore direita, encontra-se a expansão do não-terminal X_{dir}. Se essa expansão estiver totalmente contida em $\beta_{prefixo,}$ então esse não-terminal será adicionado à árvore de derivação através da operação para adicionar um par em $Z_{prefixo}$. Essa subcadeia, já representada na árvore, é removida de $\beta_{prefixo}$ e o procedimento se repete, tendo com raiz o símbolo não-terminal à esquerda X_{esq}.

Quando a expansão $[X_{dir}]$ não estiver totalmente contida em $\beta_{prefixo,}$ o procedimento se repetirá, tendo como raiz o símbolo não-terminal à direita X_{dir}. Quando toda a expansão $\beta_{prefixo}$ estiver representada na árvore de derivação, esse procedimento será encerrado.

A Figura 4.15 é um exemplo para a construção da árvore de derivação para gerar a expansão $\beta_{prefixo}$, considerando as características da árvore. As setas em negrito indicam o percurso em que há uma busca pela seqüência de não-terminais que geram $\beta_{prefixo}$.

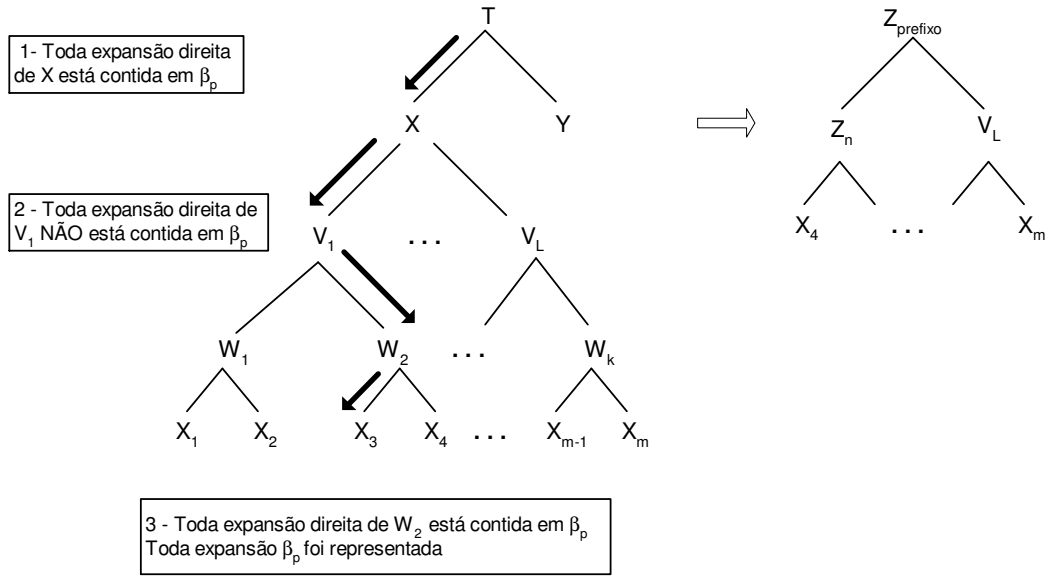


Figura 4.15 – Exemplo de Construção da Árvore de Derivação para o Prefixo da Subcadeia

Para gerar a árvore com expansão β_{sufixo}, utiliza-se este mesmo procedimento de forma simétrica. Após definir o não-terminal $Z_{prefixo}$ e Z_{sufixo}, criar um não-terminal $Z \rightarrow Z_{prefixo}Z_{sufixo}$, através da operação, para adicionar um par, como apresentado na Figura 4.16.

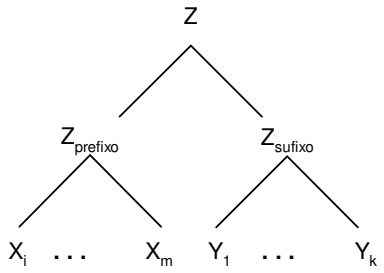


Figura 4.16 – Criação da Regra com Expansão para Subcadeia

Para adicionar uma subcadeia, o Algoritmo 4.5 apresenta uma função, a qual recebe por parâmetros: a raiz da árvore de derivação e a lista que representa a subcadeia, e então retorna a raiz da árvore de derivação atualizada.

O Algoritmo 4.6 apresenta o procedimento recursivo para gerar a árvore de derivação para o prefixo da subcadeia. Esse procedimento recebe por parâmetro: a raiz da nova árvore de derivação, a raiz da árvore já existente e a lista com a expansão desejada.

```
Função tipo_regra AdicionarSubcadeia (tipo_regra raiz, tipo_regra expansãoZ)
   Z ← nulo
   T ←BuscaNaoTerminalComExpansão (raiz, expansãoZ)
   ExpansãoA = BuscaExpansãoPrefixo (T->esq)
   ExpansãoZprefixo = SubcadeiaPrefixo (ExpansãoA, ExpansãoZ)
   EncontraNaoTerminaisSubcadeiaPrefixo (Z, T->esq, )
   ExpansãoB= BuscaExpansãoSufixo (T->dir)
   ExpansãoZsufixo = SubcadeiaSuffixo (ExpansãoB, ExpansãoZ)

   EncontraListaNaoTerminaisPrefixo (X,T->esq, ExpansãoZprefixo)
   EncontraListaNaoTerminaisSufixo (Y,T->dir, ExpansãoZsufixo)
   Z ← AdicionaPar (X, Y)
   Z1 ← AdicionaPar (raiz, Z)
   retornar Z1
fim-função
```

Algoritmo 4.5 – Operação para Adicionar uma Subcadeia

```
Procedimento EncontraListaNaoTerminaisPrefixo (tipo_regra Z, tipo_regra Raiz,
                                                tipo_regra expansão)
   se expansão ≠ vazio então
      exp_dir =Percurso (Raiz->dir)// raiz->dir->expansao
      se exp_dir está totalmente contida em expansão então
         AddPar (Z, raiz->dir)
         EncontraListaNaoTerminaisPrefixo (Z, raiz->esq, expansão – exp_dir )
      senão
         EncontraListaNaoTerminaisPrefixo (Z, raiz->dir, expansão )
      fim-se
   fim-se
fim-procedimento
```

Algoritmo 4.6 – Procedimento para Gerar Árvore com Determinada Expansão

4.2.2.2. Exemplificação de Geração da Gramática Mínima

Um exemplo do uso da solução de Charikar et al (2002) é apresentado em Matsuno (2004), que propõe a aplicação da solução do problema da gramática mínima como uma estratégia para se encontrar uma formulação adequada para a composição dos átomos de uma linguagem de programação e, a partir disso e da exposição a exemplos positivos e negativos da linguagem, encontrar uma gramática adequada para a linguagem. Nessa proposta utiliza-se as caracterísitcas da linguagem, por tratar de exemplos que são códigos fonte de uma determinada linguagem de programação, utiliza-se da análise léxica para obter um conjunto de átomos da linguagem de programação em questão. Com esta caracterização o uso da solução da gramática mínima ajuda no processo de inferência.

Para exemplificar o método de geração da gramática mínima, qualquer texto pode ser escolhido. Se ele não possuir repetições de símbolos, nenhum agrupamento ou padrão será identificado e a gramática terá uma única produção da forma $S \rightarrow x$. Para visualizar a conversão desejada em nossa proposta, o texto x será uma cadeia da linguagem $L_{20} = \{w \in \{a,b\}^* \mid w = a^n b^n,\ n > 0\}$. Enfatiza-se que, na geração da gramática mínima, segundo o algoritmo de Charikar, nada se sabe sobre as características da linguagem, não se conhece seus símbolos, nem suas regras de formação, nem a sua classificação em relação à Hierarquia de Chomsky.

Para exemplificar a geração da gramática mínima, utilizou-se a sentença x = "*aaaaaaaaabbbbbbbbb*" e nos itens e figuras que seguem são apresentadas todas as etapas do método de geração. Este exemplo ilustra didaticamente os problemas enfrentados com o modelo atual e justifica as alterações propostas em 4.3. Para os demais exemplos deste trabalho, serão apresentados e comentados apenas os resultados finais.

- **Obtenção da Lista de pares LZ77**

A primeira etapa do algoritmo de geração da gramática mínima consiste em realizar o agrupamento dos símbolos pelas repetições dos mesmos na sentença, aplicando-se o Algoritmo LZ77. Para a cadeia x citada acima,, obtêm-se os pares das cadeias, de acordo com a Tabela 4.3, ou seja, LZx = (*a,a,aa,aaaa, a, b,b,bb,bbbb, b*),

Tabela 4.3 – Agrupamentos Obtidos por LZ para a Sentença *x*

Buffer	w	p (Posição Buffer)	l (Comprimento)	i (Nº de Par)
1	a	0	1	1
2	a	0	1	2
3	aa	0	2	3
5	aaaa	0	4	4
9	a	0	1	5
10	b	10	1	6
11	b	10	1	7
12	bb	10	2	8
13	bbbb	10	4	9
14	b	18	1	10

Apenas com base na repetição de símbolos, nessa primeira etapa, os agrupamentos obtidos seriam: *aa, aaaa, bb, bbbb*.

Após obter os agrupamentos iniciais da Figura 4.17, a próxima etapa consiste na análise de cada um deles para gerar a gramática mínima G_x através do algoritmo da gramática mínima e operações em gramáticas binárias balanceadas.

- **Construção da Gramática Mínima**

Inicialmente, a gramática mínima G_x tem apenas um símbolo inicial S_0. As duas primeiras subcadeias não são repetidas; portanto, utilizando o primeiro caso de tratamento dos pares de LZ_x, (p = 0), cada símbolo é inserido no fim da lista. Para cada um deles, também é criado um não-terminal com expansão para este símbolo. Cada não-terminal deverá produzir uma árvore de derivação, garantindo as propriedades de G_{bb}; no entanto, por terem expansão para um único símbolo, a árvore ainda não é criada.

$S_0 \rightarrow S_1\ S_2$ (expansão de S_0 é $\underline{aa}$)
$S_1 \rightarrow \underline{a}$
$S_2 \rightarrow \underline{a}$

Figura 4.17 – Gramática G_x com Símbolos Não-Terminais com Expansão para um Único Símbolo

Ao analisar as próximas subcadeias $w_3 = \underline{aa}$, será utilizado o segundo caso (p ≠ 0), pois este agrupamento já está contido na expansão de um ou mais símbolos da lista ativa. Para tal, devem-se obter as seguintes listas de não-terminais:

- Lista de não-terminais com expansão para w (B).
- Lista de não-terminais antes de w (X).
- Lista de não-terminais após w (Y).

Assim, obtêm-se as seguintes listas: $B_1 = S_1S_2$, $X_1 = \varepsilon$ e $Y_1 = \varepsilon$. É necessário criar um não-terminal M_1 com expansão B_1, através da *operação para adicionar* uma seqüência de símbolos não-terminais e, então, criar a árvore de derivação para M_1, como apresentado na Figura 4.18.

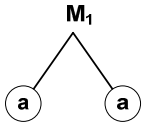


Figura 4.18 – Árvore de Derivação Gerada pela Operação Adicionar Seqüência

Após criar M_1, a seqüência de símbolos B_1 deverá ser substituída na lista ativa por M_1, gerando, pois, a gramática G_x, como se pode constatar na Figura 4.19.

$S_0 \rightarrow M_1$ (expansão de S_0 é $\underline{aa}$)
$M_1 \rightarrow S_1S_2$ (expansão $\underline{aa}$)
$S_1 \rightarrow a$
$S_2 \rightarrow a$

Figura 4.19 – Modificações na Gramática Mínima ao Encontrar uma Seqüência já Derivada por G_x por mais que um não-terminal

Após essa operação, devem-se separar o prefixo e o sufixo da expansão B, que não fazem parte de w_3. Neste caso, $B_{prefixo}$ e B_{sufixo} são nulos, criando-se, então, apenas um não-terminal N_1 com expansão M_1, que será adicionado no fim da produção inicial de G_x, resultando na seguinte produção:

$S_0 \rightarrow M_1N_1$ (expansão de S_0 é $\underline{aaaa}$)
$N_1 \rightarrow B_{1prefixo}\ M_1\ B_{1sufixo}$
$M_1 \rightarrow S_1S_2$ (expansão $\underline{aa}$)
$B_{1prefixo} \rightarrow \varepsilon$
$B_{1sufixo} \rightarrow \varepsilon$
$S_1 \rightarrow a$
$S_2 \rightarrow a$

Figura 4.20 – Modificações na Gramática Mínima ao Encontrar uma Seqüência já Derivada G_x por um Único Não-Terminal

Substituição de M_1 por N_1, tem-se, então, a seguinte simplificação:

$S_0 \rightarrow M_1M_1$ (expansão de S_0 é *aaaa*)
$M_1 \rightarrow S_1S_2$ (expansão *aa*)
$S_1 \rightarrow a$
$S_2 \rightarrow a$

Figura 4.21 – Simplificações na Gramática Mínima Apresentada na Figura 4.20

Ao analisar w_4 = "*aaaa*", tem-se de realizar o mesmo processo.

$S_0 \rightarrow M_1M_1N_1$ (expansão de S_0 é *aaaaaaaa*)
$N_1 \rightarrow M_1M_1$
$M_1 \rightarrow S_1S_2$ (expansão *aa*)
$S_1 \rightarrow a$
$S_2 \rightarrow a$

Figura 4.22 – Modificações na Gramática Mínima ao Encontrar uma Seqüência já Derivada G_x por mais de um Símbolo Não-Terminal

Ao analisar w_4 = *aaaa*, tem-se de realizar o mesmo processo.

$S_0 \rightarrow N_1N_1$ (expansão de S_0 é *aaaaaaaa*)
$N_1 \rightarrow M_1M_1$
$M_1 \rightarrow S_1S_2$ (expansão *aa*)
$S_1 \rightarrow a$
$S_2 \rightarrow a$

Figura 4.23 – Simplificações na Gramática Mínima da Figura 4.22

Ao analisar $w_5 = a$, já existe um não-terminal com expansão igual a w_5; portanto, este não-terminal é inserido no fim da lista, resultando na seguinte gramática.

$S_0 \rightarrow N_1N_1\ S_1$ (expansão de S_0 é *aaaaaaaaa*)
$N_1 \rightarrow M_1M_1$ (expansão *aaaa*)
$M_1 \rightarrow S_1S_2$ (expansão *aa*)
$S_1 \rightarrow a$
$S_2 \rightarrow a$

Figura 4.24 – Modificações na Gramática Mínima para Adicionar uma Subcadeia Derivada por um Único Símbolo Não-Terminal

Ao analisar w_6 = b, não se tem esta cadeia representada; portanto, um novo não-terminal será criado e inserido no fim da lista, resultando na seguinte gramática.

$S_0 \rightarrow N_1N_1\ S_1\ S_3$ (expansão de S_0 é *aaaaaaaab*)
$N_1 \rightarrow M_1M_1$ (expansão *aaaa*)
$M_1 \rightarrow S_1S_2$ (expansão *aa*)
$S_1 \rightarrow a$
$S_2 \rightarrow a$
$S_3 \rightarrow b$

Figura 4.25 – Modificações na Gramática Mínima para Acrescentar Nova Subcadeia

Este processo é repetido criando-se novas produções e suas respectivas árvores de derivação, o mesmo ocorrendo para as subcadeias com expansão *bb* e *bb* para obter *bbbbb.* Ao fim do processo de geração da gramática mínima para a sentença *x*, é apresentada a gramática a seguir.

$S_0 \rightarrow N_1N_1\ S_1\ N_2N_2\ S_3$ (expansão de S_0 é *aaaaaaaabbbbbbbbb)*
$N_1 \rightarrow M_1M_1$ (expansão *aaaa*)
$M_1 \rightarrow S_1S_2$ (expansão *aa*)
$N_2 \rightarrow M_2M_2$ (expansão *bbbb*)
$M_2 \rightarrow S_3S_4$ (expansão *bb*)
$S_1 \rightarrow a$
$S_2 \rightarrow a$
$S_3 \rightarrow b$
$S_4 \rightarrow b$

Figura 4.26 – Modificações na Gramática Mínima ao Fim do Processo

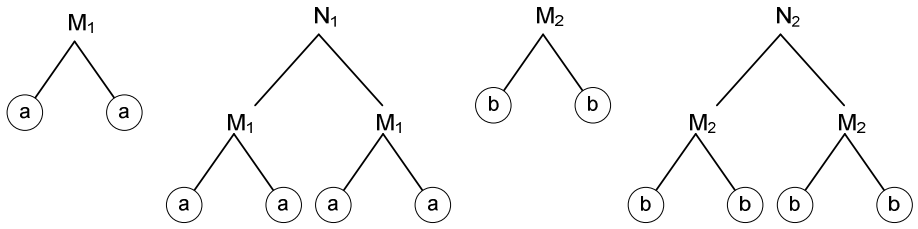


Figura 4.27 – Árvore de Derivação dos Símbolos Não-Terminais de G_x

- **Simplificação da Gramática Mínima**

Para finalizar a geração da gramática mínima G_x, ela deverá ser simplificada realizando o seguinte:

- Substituir os símbolos não-terminais, que têm como expansão um único símbolo terminal.
- Substituir os símbolos não-terminais, que têm como expansão apenas uma transição, um único símbolo não-terminal, por exemplo: A→B e B→C substituir por A→C.

Ao final do processo, tem-se G_x, que se apresenta da seguinte forma:

$S_0 \rightarrow N_1N_1a\,N_2N_2b$ (expansão de S_0 é *aaaaaaaabbbbbbbbb)*
$N_1 \rightarrow M_1M_1$ (expansão *aaaa*)
$M_1 \rightarrow aa$
$N_2 \rightarrow M_2M_2$ (expansão *bbbb*)
$M_2 \rightarrow bb$

Figura 4.28 – Gramática Mínima com Simplificações

O objetivo do algoritmo da gramática mínima é reduzir a representação da sentença *x*, como pode ser observado no exemplo desta seção, pois $|x| = 18$ e $|G_x| = 14$. Mas o objetivo

deste trabalho é encontrar os ciclos sintáticos nas sentenças de uma linguagem. Para tanto, realizou-se o uso deste algoritmo com alguns experimentos, como será apresentado no subitem a seguir, com o intuito de utilizá-lo no processo de inferência gramatical, como será apresentado no subitem 4.2.4.

4.2.3. APLICAÇÃO DO ALGORITMO DA GRAMÁTICA MÍNIMA PARA O CONJUNTO DE EXEMPLOS

Para observar os resultados dos algoritmos em relação aos exemplos, realizaram-se dois experimentos, dado um conjunto de exemplos S com n sentenças x_i, em que $0 < i \leq n.$:

1º Experimento: Gerar uma gramática mínima $GMin_i$ para cada exemplo e unificá-las posteriormente.

2º Experimento: Transformar o conjunto de sentenças x_i em uma sentença x, onde x é a concatenação de cada cadeia x_i do conjunto de exemplos. Neste caso, não será necessário unificar as gramáticas geradas, pois se teria uma única gramática $GMin_{x.}$

Na Figura 4.29 apresenta-se a aplicação da solução proposta para generalizar as regras-produção da gramática como um primeiro passo de experimentação para o processo de inferência da gramática. Para exemplificar, o uso do algoritmo com as modificações propostas será utilizado para o mesmo conjunto de exemplos $\boldsymbol{S}$ = {*ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, aaaaaabbbbbb, aaaaaaabbbbbbb*} da subseção anterior.

Em uma primeira instância, gerou-se a gramática mínima para cada sentença do conjunto $\boldsymbol{S}$, como pode ser observado na Figura 4.29. Em um segundo momento gerou-se a gramática mínima considerando que a sentença x é formada pela concatenação de todas as sentenças do conjunto de exemplos como pode ser observado na Figura 4.30.

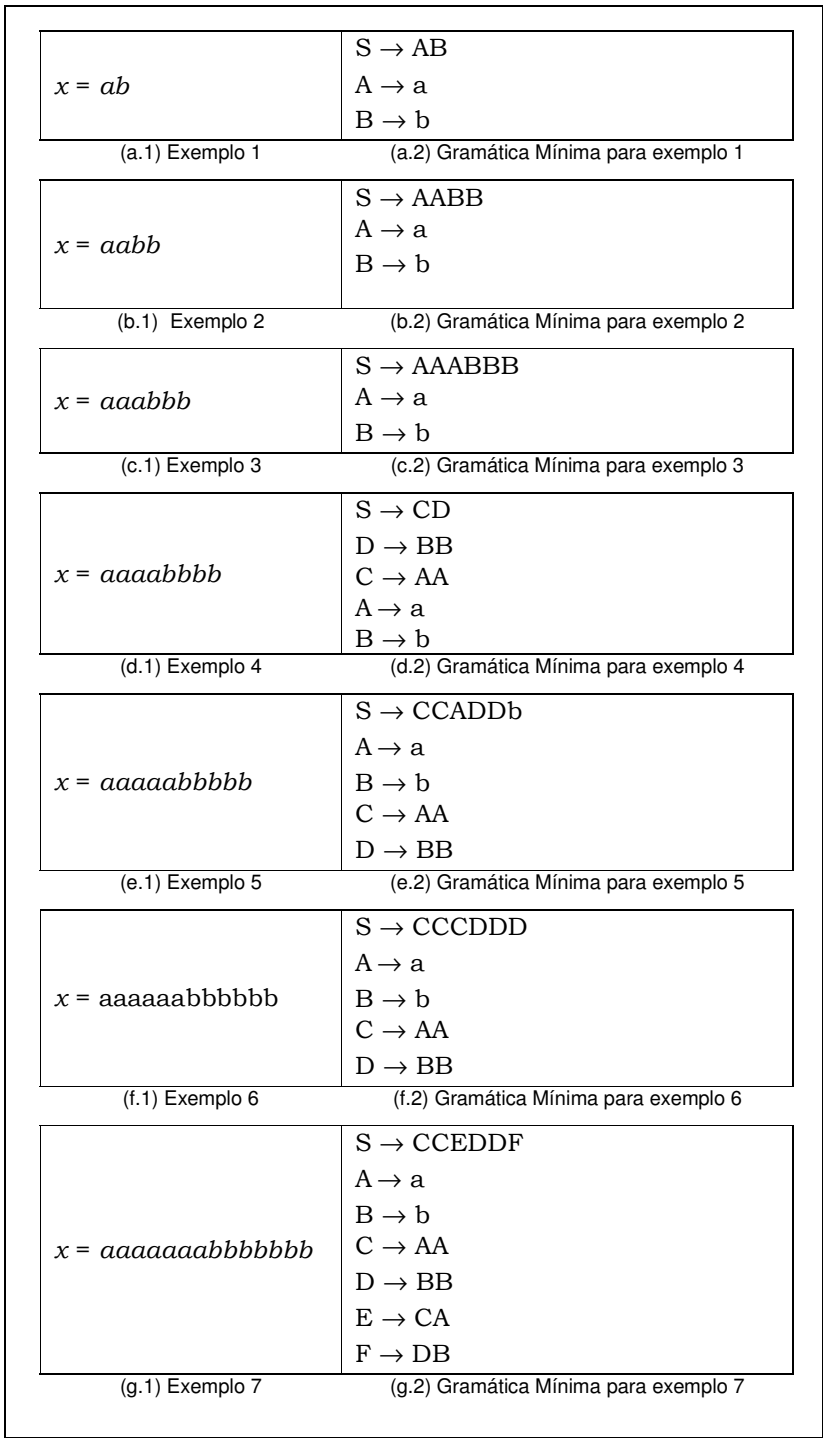


Figura 4.29 – Gramáticas Mínimas para cada Sentença do Conjunto de Exemplos S

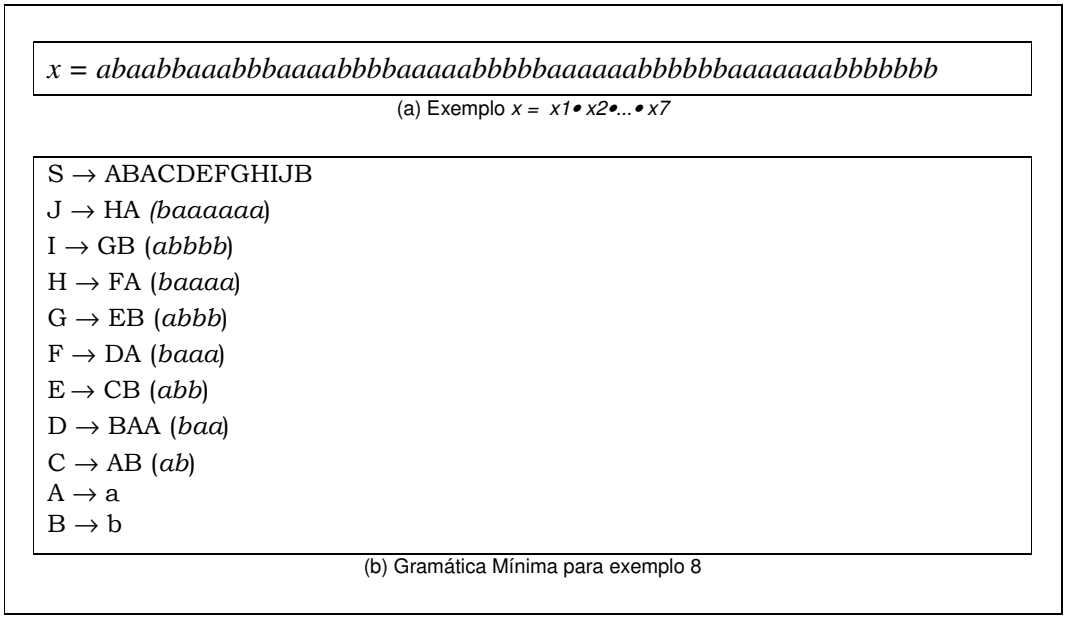


Figura 4.30 – Gramática Mínima para as Subcadeias Concatenadas do Conjunto de Exemplos S

Como pode ser observado, para o caso em questão, a gramática não foi inferida corretamente, porque o aninhamento sintático não foi percebido.

4.2.4. MODIFICAÇÕES PARA GENERALIZAÇÃO DO ALGORITMO DA GRAMÁTICA MÍNIMA

Neste trabalho, o algoritmo proposto por Charikar et al (2002) é utilizado para auxiliar o processo de inferência de gramáticas livres de contexto. Uma vez que este algoritmo pode inferir uma Gramática Livre de Contexto ***G*** para uma única sentença de Linguagem ***L***. o mesmo foi utilizado nesta proposta para a geração de uma gramática "mínima" para as *n* sentenças do conjunto com exemplos da linguagem, gerando *n* gramáticas mínimas. Entretanto, essas *n* gramáticas precisam ser unificadas em uma única gramática ***G***(***L***), em que L é a linguagem livre de contexto em questão.

Inicialmente, aplicou-se o algoritmo da gramática mínima ao conjunto de exemplos diretamente sem tratamento algum, como descrito na subseção 4.2.3.1, pois, desejava-se avaliar quais aninhamentos poderiam ser detectados. Nas subseções seguintes, apresentam-se

as modificações propostas para a utilização deste algoritmo para inferência de gramáticas livres de contexto.

Nas gramáticas mínimas geradas na seção anterior pode-se observar que os símbolos não-terminais S, A, B, C, etc, não definem, necessariamente, a mesma estrutura da linguagem em análise, apesar de apresentarem o mesmo "nome". Nesta proposta, a intenção é realizar a generalização da gramática através da análise das regras de produção equivalentes, portanto para aplicar o algoritmo de inferência os símbolos não-terminais dos diversos exemplos devem definir regras para a mesma estrutura.

Para comparar as produções será realizada a normalização dos símbolos não-terminais através da operação de união das gramáticas mínimas geradas. Para cada gramática $G_1, G_2, \ldots G_n$ gerada com os respectivos símbolos iniciais $S_1, S_2, \ldots, S_n$ será considerado um novo símbolo S_0 com a seguinte produção: $S_0 \rightarrow S_1 \mid S_2 \mid \ldots \mid S_n$, em que n é o número de exemplos. Este tratamento para os símbolos não-terminais é realizado durante a conversão do conjunto de sentenças na gramática mínima para cada exemplo, com algumas modificações no Algoritmo 4.7. O algoritmo de normalização de símbolos não-terminais é realizado no momento da busca de derivações já existentes, com intuito de reaproveitar os símbolos e as árvores de derivações já existentes, criados na geração da gramática mínima já existente. As modificações podem resumir-se no algoritmo a seguir:

1. Gerar um símbolo inicial S_0 que apresentará a união dos exemplos.
2. Para cada exemplo x gerar a lista cadeias w através do algoritmo LZ77
(referenciada por L_x)
3. Criar um símbolo S_x inicial para a geração da gramática mínima deste exemplo (G_x)
4. Para cada cadeia w em L_x realizar os seguintes passos:
Buscar na lista de S_0 uma árvore que gere a cadeia
se não existir **então**
Um novo símbolo é criado.
senão
O símbolo já existente é reutillizado na geração de G_x
As operações para gerar a gramática mínima são realizadas
Quando necessário o processo de substituição deverá ser analisado nas gramáticas já geradas.
fim-se
5. Ao fim do processo S_x é inserido como regra de S_0

Algoritmo 4.7 – Normalização dos Símbolos não-terminais

Na seção anterior, para diferenciar os símbolos não-terminais criados utilizaram-se símbolos do alfabeto com o intuito de simplificar a representação. Nesta seção além desta distinção, utilizam-se índices (que variam de 1 a ***n*** em que ***n*** é o número de exemplos) para identificar em qual dos exemplos de sentenças da linguagem o símbolo não-terminal é criado.

Com este processo de normalização, até os exemplos que não tenham repetições, que são a base para o algoritmo da gramática mínima, poderão ser considerados, pois suas cadeias poderão estar representadas em outras gramáticas já geradas e poderão ser utilizados. As modificações propostas neste trabalho mudam o foco do algoritmo da gramática mínima, portanto, o tamanho da gramática inferida e a quantidade de produções, não será necessariamente menor do que o tamanho das sentenças.

Na Figura 4.31 apresenta-se a aplicação da solução proposta para generalizar as regras de produção da gramática, como um primeiro passo de experimentação para o processo de inferência.. Para exemplificar, o uso do algoritmo com as modificações proposta será utilizado o mesmo conjunto de exemplos ***S*** ={*ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, aaaaaabbbbbb, aaaaaaabbbbbbb*} em 4.2.3.1.

No exemplo apresentado cada cadeia do conjunto do exemplo é utilizada na construção e incremento de novas produções. Esta figura apresenta a avaliação de cada cadeia x_i *de* $S = \{ x_1x_2x_3x_4x_5x_6x_7\}$ e suas respectivas modificações na gramática construída. Ao fim da análise de todas as subcadeias do conjunto *S* temos a Gramática Inferida.

x_1 =ab	$S_0 \rightarrow S_1$ $S_1 \rightarrow A_1B_1$ $A_1 \rightarrow a$ $B_1 \rightarrow b$

a) Exemplo sentença 1 (b) G1

x_2=aabb	$S_0 \rightarrow S_1 \mid S_2$ $S_1 \rightarrow A_1B_1$ $A_1 \rightarrow a$ $B_1 \rightarrow b$ $S_2 \rightarrow C_2D_2$ $D_2 \rightarrow B_1B_1$ $C_2 \rightarrow A_1A_1$

a)Exemplo adicionando sentença 2 (b) G2 = G1 + x_2

x_3 =aaabbb	$S_0 \rightarrow S_1 \mid S_2 \mid S_3$ **$S_1 \rightarrow A_1B_1$** $A_1 \rightarrow a$ $B_1 \rightarrow b$ **$S_2 \rightarrow C_2D_2$** $D_2 \rightarrow B_1B_1$ $C_2 \rightarrow A_1A_1$ **$S_3 \rightarrow C_2A_1D_2B_1$**

(a)Exemplo adicionando sentença 3 (b) G3 = G1 +x_3

x_4 = aaaabbbb	$S_0 \rightarrow S_1 \mid S_2 \mid S_3 \mid S_4$ **$S_1 \rightarrow A_1B_1$** $A_1 \rightarrow a$ $B_1 \rightarrow b$ **$S_2 \rightarrow C_2D_2$** $D_2 \rightarrow B_1B_1$ $C_2 \rightarrow A_1A_1$ **$S_3 \rightarrow C_2A_1D_2B_1$** **$S_4 \rightarrow E_4F_4$** $E_4 \rightarrow C_2C_2$ $F_4 \rightarrow D_2D_2$

(c)Exemplo adicionando sentença 4 (d) G4 = G3 +x_4

x_5 = aaaaabbbbb	$S_0 \rightarrow S_1 \mid S_2 \mid S_3 \mid S_4 \mid S_5$ **$S_1 \rightarrow A_1B_1$** $A_1 \rightarrow a$ $B_1 \rightarrow b$ **$S_2 \rightarrow C_2D_2$** $D_2 \rightarrow B_1B_1$ $C_2 \rightarrow A_1A_1$ **$S_3 \rightarrow C_2A_1D_2B_1$** **$S_4 \rightarrow E_4F_4$** $E_4 \rightarrow C_2C_2$ $F_4 \rightarrow D_2D_2$ **$S_5 \rightarrow E_4A_1F_4B_1$**

(e)Exemplo adicionando sentença 5 (f) G5 = G4 +x_5

x_6 = aaaaaabbbbbb	$S_0 \rightarrow S_1 \mid S_2 \mid S_3 \mid S_4 \mid S_5 \mid S_6$ **$S_1 \rightarrow A_1B_1$** $A_1 \rightarrow a$ $B_1 \rightarrow b$ **$S_2 \rightarrow C_2D_2$** $D_2 \rightarrow B_1B_1$ $C_2 \rightarrow A_1A_1$ **$S_3 \rightarrow C_2A_1D_2B_1$** **$S_4 \rightarrow E_4F_4$**

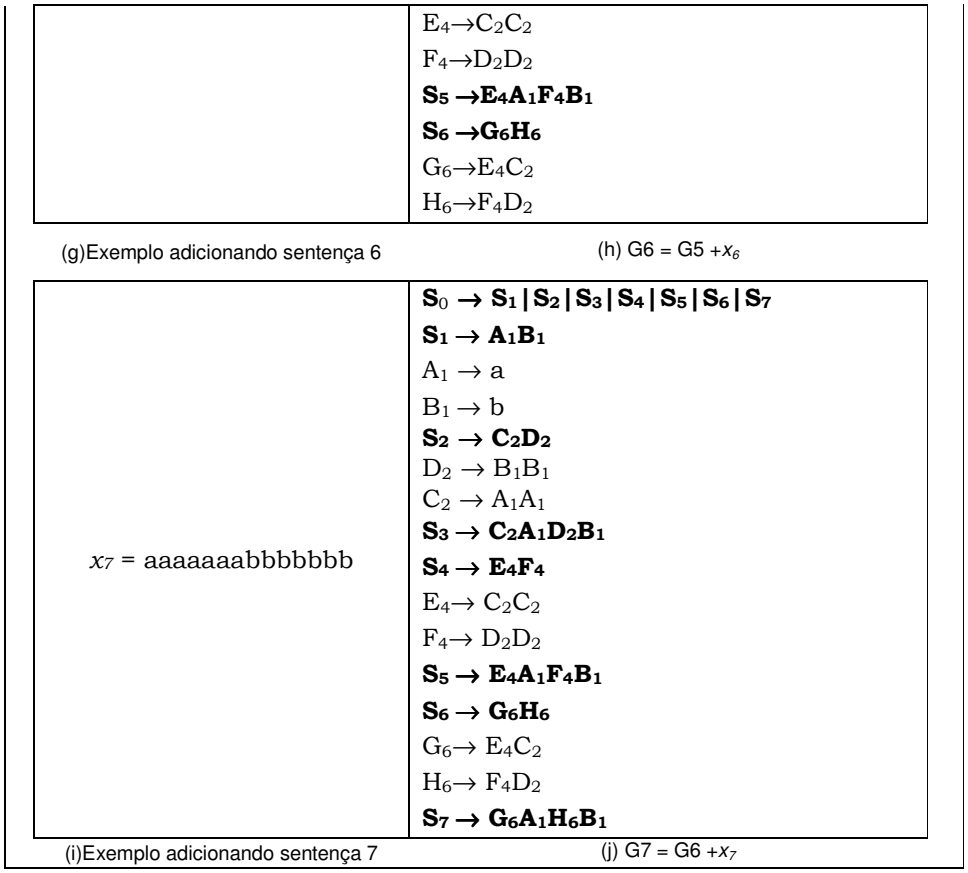


Figura 4.31 – Gramáticas Mínimas com normalização dos símbolos não-terminais

Após a normalização de "nomes" dos símbolos não-terminais, é garantido que os símbolos não-terminais representam a mesma estrutura sintática, podendo-se iniciar o processo de comparação e inferência das regras de produção. Inicialmente, analisar todos as produções dos símbolos não-terminais iniciais de cada gramática e, posteriormente, as com o mesmo nome, tem-se uma gramática inferida *Gn* para este conjunto de exemplos. Após esta etapa, a gramática é generalizada e é gerado um reconhecedor sintático para testar a gramática.

4.2.5. GERAÇÃO DE RECONHECEDORES SINTÁTICOS PARA TESTES DA GRAMÁTICA INFERIDA

A partir da gramática *G* inferida como descrito na seção anterior, esta deve ser submetida aos testes para avaliação, sendo necessário, para tanto, gerar um analisador sintático para avaliar as cadeias dos conjuntos de exemplos. Obtida a especificação de gramática, é possível realizar a geração automática de seus analisadores sintáticos, como pode ser encontrado nas referências clássicas de construção de compiladores (Aho, 1972).

Neste trabalho, utiliza-se a proposta de José Neto (1987, 1993), na qual, a partir de uma gramática *G* na notação de Wirth, se automatiza a geração de um reconhecedor sintático utilizando o simulador de Autômatos de Pilha Estruturados (APEs) apresentado na Figura 4.32. Para gerar o reconhecedor sintático e testar as gramáticas inferidas, optou-se por este método, pois o processo faz uso de um único modelo subjacente (Pedrazzi et al, 2004).

Existem diversas notações para especificar uma gramática (BNF, BNF-x, Wirth). Aquela em que a gramática inferida está representada precisa ser convertida para notação de Wirth. Isto pode ser realizado automaticamente usando-se um conversor de notações, que é um "metatradutor" baseado em APEs. Em Pistori (2002), este tradutor também pode ser automatizado da mesma forma, desde que se tenha as especificações da notação da gramática utilizada.

Para realizar a conversão, devem-se realizar os seguintes passos:

- Agrupamento das opções das regras de produção.
- Eliminar os não-terminais recursivos à esquerda.
- Encontrar os não-terminais essenciais, ou seja, aqueles auto-recursivos centrais.
- Eliminar os não-terminais que não são essenciais através da substituição dos mesmos pelas regras de produção.

Após esta conversão da gramática inferida de uma notação intermediária para notação Wirth, eliminaram-se os símbolos não-terminais que não são essenciais e se gerou o reconhecedor sintático.

O interpretador de APE é utilizado tanto no gerador de reconhecedor sintático como na realização de testes das gramáticas inferidas. Neste trabalho, não serão apresentados a geração do reconhecimento sintático e os testes, pois as gramáticas normalizadas ainda precisam ser generalizadas.

4.3. SUGESTÕES E PROPOSTAS DE MELHORIAS

Mediante as implementações e os experimentos realizados utilizando os algoritmos já existentes e com as modificações estudadas, observou-se que o aninhamento sintático pode ser detectado tanto com modificações no algoritmo de JJ98, quanto no da gramática mínima. As subseções a seguir apresentam extensões e modificações nos algoritmos estudados para detectar o aninhamento.

4.3.1. APLICAÇÃO DO ALGORITMO DE JJ98 PARA LINGUAGENS LIVRES DE CONTEXTO

Este algoritmo infere autômatos para reconhecimento de linguagens regulares; porém, ao utilizá-lo com um conjunto de exemplos de linguagens livres de contexto determinísticas, obtiveram-se resultados de AFDs, que servem de base para a detecção dos ciclos auto-recursivos centrais, como pode ser corroborado nas Figuras 4.32 a 4.36.

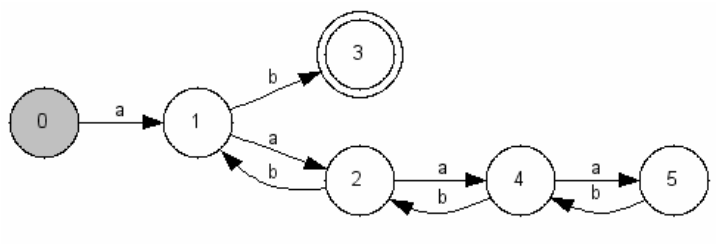


Figura 4.32 - Autômato Finito Inferido para $L_{20} = a^n b^n$

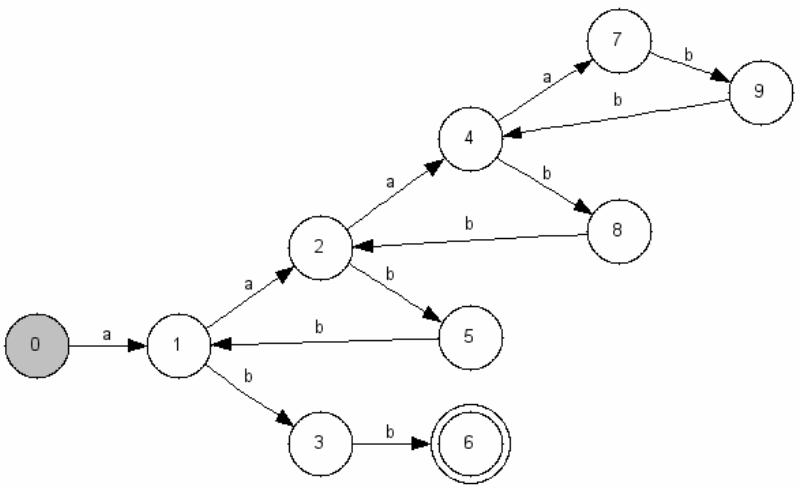


Figura 4.33- Autômato Finito Inferido para $L_{21} = a^n b^{2n}$

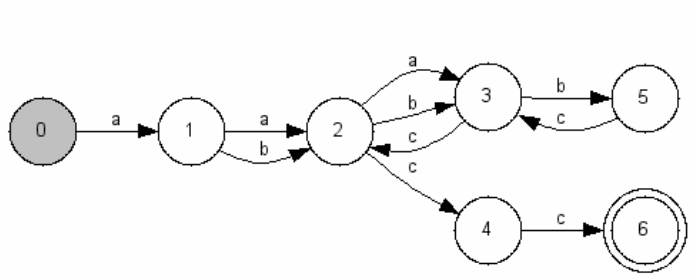


Figura 4.34- Autômato Finito Inferido para $L_{21} = a^n b^k c^{(n+k)}$

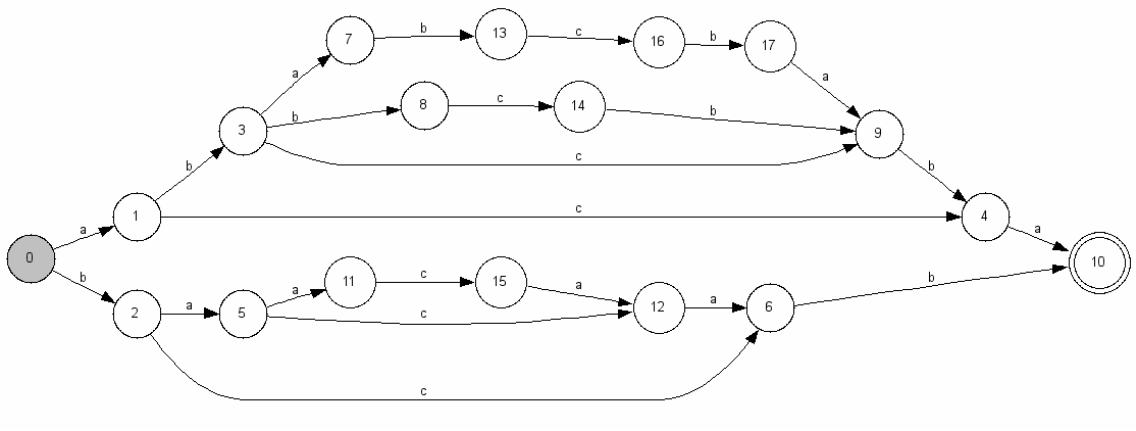


Figura 4.35– Autômato Finito Inferido para $L_{22}=wcw^R$, em que $w \in \{a,b\}^*$

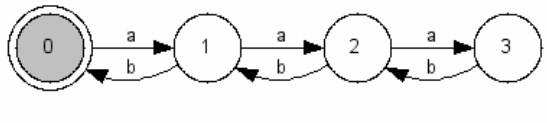


Figura 4.36– Autômato Finito Inferido para L_{24}= aninhamento de a e b

Pela observação do comportamento do algoritmo de JJ98, propõe-se o Algoritmo 4.8 para generalizar o AFD inferido em um APE para que este reconheça linguagens livres de contexto.

4.3.2. SUGESTÕES DE ALTERAÇÃO NO ALGORITMO DE JJ98 PARA INFERÊNCIA DE LINGUAGENS LIVRE DE CONTEXTO

Ao desejar inferir linguagens livres de contexto, deve-se propor um método que permita distinguir os ciclos lineares dos auto-recursivos, ou seja, detectar os aninhamentos sintáticos. O algoritmo proposto neste trabalho utiliza o Autômato Inferido pelo algoritmo de JJ98 para realizar a geração de APE e convertê-lo em uma gramática livre de contexto.

Um conjunto finito de exemplos de sentenças de linguagens livres de contexto pode ser reconhecido por um AF, pois este conjunto de exemplos é finito e, portanto, representa uma linguagem regular. Com base nesta possibilidade, utilizou-se aqui o algoritmo de JJ98 para obter um AF para o conjunto de exemplos de sentenças das linguagens livres de contexto e através do algoritmo 4.8 proposto no presente estudo, para se detectarem os aninhamentos sintáticos e transformarem o AF inferido em um APE. Aplicou-se, para tanto, o algoritmo em exemplos de linguagens regulares e livres de contexto, visto que era necessário avaliar se possibilidade de distinção entre um ciclo regular e uma chamada de submáquina através da não-existência de submáquinas auto-recursivas.

A idéia principal do algoritmo aqui proposto é percorrer o autômato regular inferido a partir do estado inicial e detectar os caminhos mínimos ($CMin_i$) sem ciclos até o estado final, em que $0< i <k$, onde k é a quantidade de caminhos mínimos encontrados. Estes $CMin_i$ são as primeiras submáquinas do APE Inferido e as primeiras regras de produção de *G*. Para cada caminho mínimo encontrado, analisa-se se existe um outro caminho ou ciclo a partir dos estados que fazem parte deste caminho. Se existir, este trecho do caminho mínimo deverá ser substituído por uma chamada de submáquina S_m, que já existe ou que deverá ser criada. O processo é repetido até que todos os estados do autômato inferido sejam percorridos. A cada nova submáquina criada, o processo de verificar novos caminhos também é repetido. O Algoritmo 4.8 apresenta a descrição deste processo e a Figura 4.37 apresenta um exemplo ilustrativo tendo como base o autômato inferido representado na Figura 4.32.

GeraAPE (AA AInferido)

n = número de submáquinas
n = 0

faça

1-Encontrar os caminhos mínimos do Estado Inicial ao Final.
ConjCMin={ CMin1, ...CMinN }

2-Estes caminhos são as primeiras produções da gramática livre de contexto.
Criar S-> CMin1| ...|CMinN

4-Para cada CMink, verificar se existe um caminho a partir dos estados do caminho mínimo que retorne ao caminho CMink.

se existir caminho mínimo C_k de qi a qj **então**
Verifica se já existe uma submáquina que represente C_k
se existir então
Será criada uma transição de chamada de submáquina Si a partir de qi para qj
senão
Uma nova submáquina (n+1) é criada
Será criada uma submáquina S(n+1)
Será criada uma transição de chamada de submáquina Si a partir de qi para qj
Na submáquina Si são criadas as seguintes transições:
ω = cadeia representada no caminho mínimo C_k
se $|C_k| = 0$ **então**
$\omega = \varepsilon$
fim-se
fim-se
Criar transição na submáquina Si do estado inicial (S_{i0}) para consumir a subcadeia ω até o estado final de S_i (S_{i1})
O caminho Ck é construído em Si desde o estado inicial até o estado final
Recursivamente ➔ VerificaCaminhoMínimo em Si
fim-se
Para continuar o percurso a partir da submáquina i
5-Para cada submáquina criada a partir do caminho o processo é repetido.

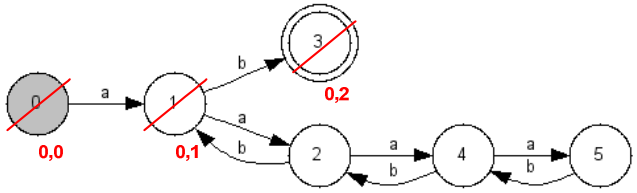
enquanto todos os estados não foram marcados

Algoritmo 4.8 – Algoritmo para Inferência de APE

Este algoritmo é apresentado como uma extensão para realização de trabalhos futuros.

Aplicando o algoritmo proposto, tendo como entrada o autômato da Figura 4.32, obtém-se o APE apresentado na Figura 4.37. A seguir, apresenta-se, passo a passo, um exemplo ilustrativo da inferência de APEs.

- **Passo 1:** Encontrar Caminho Mínimo em AF Inferido

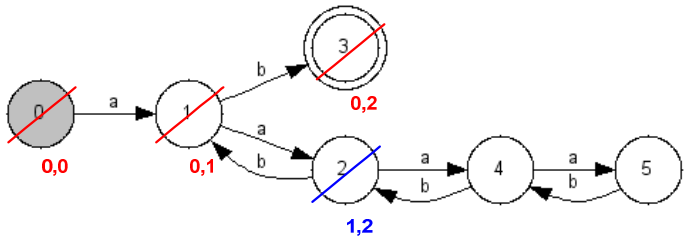


APE Inferido

S0

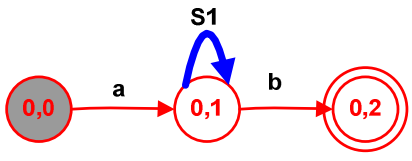
Gramática Inferida
S0 → ab

- **Passo 2:** Verificar a existência de ciclos ou novos caminhos que retornam ao caminho representado em S0
- Encontra o caminho C1=1a2b1
- Verificar a existência de uma submáquina que represente o caminho
- Como não existe, ela deve ser criada.



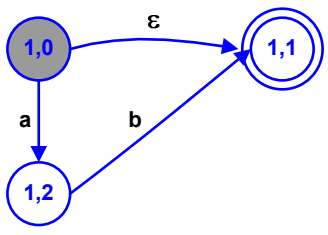
APE Inferido

S0



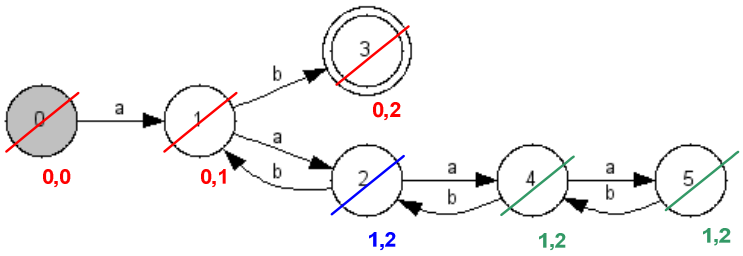
Submáquinas Geradas

S1

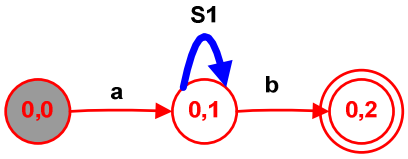


Gramática Inferida
So → a**S1**b, S1 → ab | ε

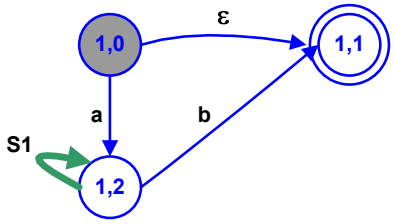
- **Passo 3:** Verificar a existência de ciclos ou novos caminhos que retornam ao caminho representado em S1
- **Encontra o caminho C2=2a4b2**
- Verificar a existência de uma submáquina que represente o caminho
- Como existe, então neste estado é realizada a chamada da submáquina S1
- **Substituir a transição por chamada de submáquina S1**
- **Propagar a chamada de submáquina**



APE Inferido
S0



Submáquinas Geradas
S1



Gramática Inferida
S0 → a**S1**b
S1 → a**S1**b | ε

Figura 4.37 – APE Inferido com base no Autômato Finito Inferido

Os demais autômatos inferidos das figuras 4.34 a 4.36 também foram testados. Em trabalhos futuros é importante avaliar o algoritmo para inferência de APE para exemplos de linguagens regulares e linguagens livres de contexto que são reconhecidos por autômatos de pilha não-determinísticos.

Já é conhecido que não existe algoritmo que pode decidir se uma gramática livre de contexto arbitrária corresponde a uma linguagem regular. Isto significa que não existe uma a técnica geral que pode transformar cada gramática livre de contexto para sua contraparte regular. Pesquisadores têm estudado as subclasses das gramáticas livres de contexto que geram linguagens regulares. Existem trabalhos que tratam gramáticas de um símbolo, gramáticas não auto-embutidas (auto-recursivas), e gramáticas que geram linguagens finitas.

Em (Andrei et al, 2004) é apresentado um sistema de equações que possibilitam a conversão de aspectos livres de contexto sem elementos auto-recursivos centrais para expressões regulares.

4.3.3. SUGESTÕES DE ALTERAÇÃO DO ALGORITMO BASEADO EM CHARIKAR

Como nos demais métodos de inferência é necessário definir alguns critérios para caracterizar um aninhamento sintático. Tais critérios podem considerar as características da linguagem, entretanto aqui seguem algumas sugestões de alterações na proposta inicial:

O algoritmo pode tentar detectar o aninhamento modificando a lista do LZ77 e não gerando uma nova lista de subcadeias a cada exemplo. Esta modificação é uma nova tentativa de detectar ciclos sintáticos uma vez que o LZ77 se baseia na repetição dos símbolos nas cadeias. Efetivamente propõe-se a modificação no algoritmo utilizado para inferência de forma a não mais gerar listas diferentes de subcadeias a cada exemplo. O resultado dessa aplicação pode aumentar a entropia da informação, mas o principal objetivo desta modificação no algoritmo é permitir que o LZ77 detecte os aninhamentos, juntamente com os prefixos e sufixos comuns. Posteriormente deve-se simplificar a gramática, eliminando símbolos não-terminais.

Uma outra forma de detectar o aninhamento é a utilização do algoritmo de JJ98 em cada regra de produção, considerando que o conjunto de produções do símbolo inicial da gramática inferida será o conjunto de sentenças de exemplos para o algoritmo. Como pode ser observado nos testes realizados em Matsuno (2004) em que se aplicou uso do algoritmo de Charikar (2002) para que atue sobre os átomos da linguagem, os resultados gerados tornam-se mais relevantes para a inferência da gramática da linguagem de programação.

Além do tratamento da representação do código-fonte, é necessário um processo que possa relacionar as diversas gramáticas geradas, identificando os ciclos sintáticos existentes e distinguindo quando ocorre um novo ciclo ou quando este já foi representado. Para que os resultados se aproximem de uma gramática mais adequada para a linguagem utiliza-se de

mais uma etapa, contendo também exemplos de código-fonte que não estão corretos. Esta etapa do método de inferência de uma gramática para uma linguagem de programação, baseia-se no treinamento de um autômato adaptativo semelhante ao proposto em José Neto (1998).

A Figura 4.38 apresenta as etapas da proposta, das quais as duas primeiras estão implementadas e a última em desenvolvimento.

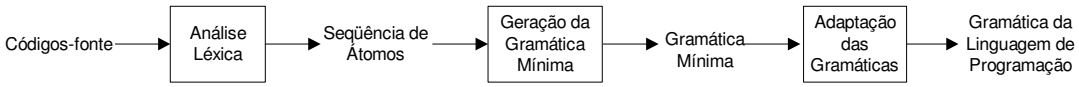
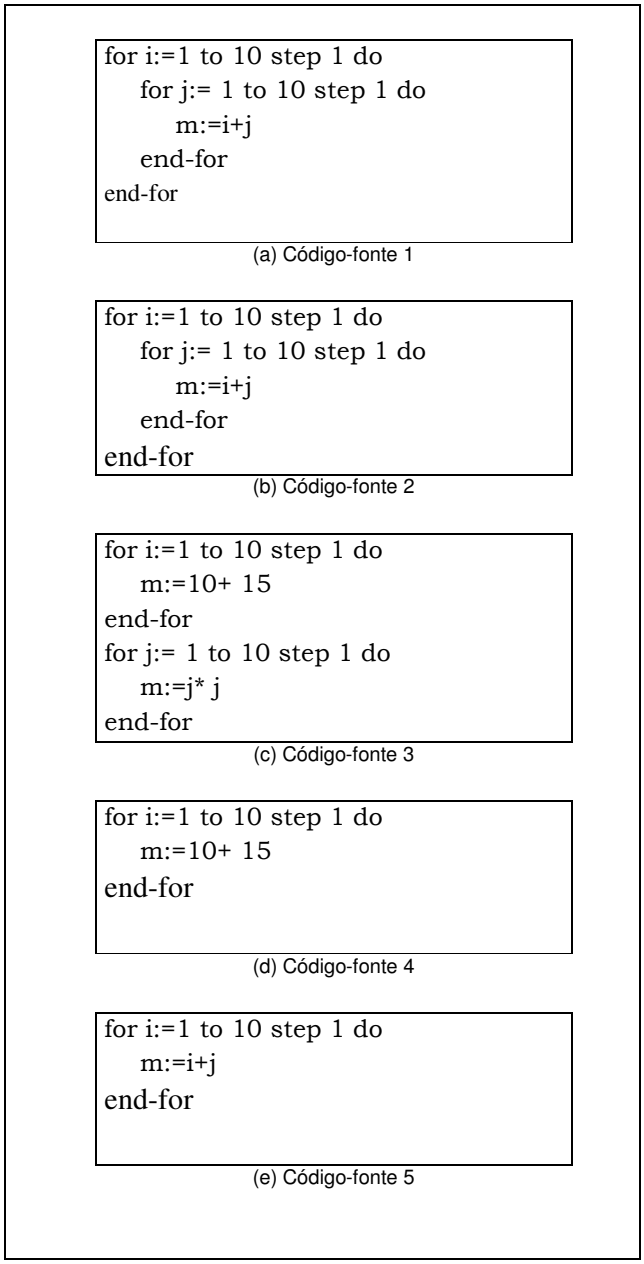


Figura 4.38 - Etapas para Inferência de Gramáticas de Linguagens de Programação

Como pode ser observado a seguir nos exemplos apresentados nas Figuras 4.39 à 4.41, para tal considere a seguinte classificação de átomos para os códigos-fonte:

Tabela 4.4 Classificação dos Átomos da Linguagem

Classe Átomo	Descrição	Sigla Átomo
identificador	letra.(letra | dígitos)*	id
constante	(dígitos)*	cte
op_atribuição	:=	op_atr
op_artmético	+ | - | * | / | ^	op_arit
for	for	for
to	to	to
step	step	step
do	do	do



```
for i:=1 to 10 step 1 do
    for j:= 1 to 10 step 1 do
        m:=i+j
    end-for
end-for
```

(a) Código-fonte 1

```
for i:=1 to 10 step 1 do
    for j:= 1 to 10 step 1 do
        m:=i+j
    end-for
end-for
```

(b) Código-fonte 2

```
for i:=1 to 10 step 1 do
    m:=10+ 15
end-for
for j:= 1 to 10 step 1 do
    m:=j* j
end-for
```

(c) Código-fonte 3

```
for i:=1 to 10 step 1 do
    m:=10+ 15
end-for
```

(d) Código-fonte 4

```
for i:=1 to 10 step 1 do
    m:=i+j
end-for
```

(e) Código-fonte 5

Figura 4.39 – Exemplos de código fontes para estrutura de repetição *for*

```
for id op_atr cte to cte step cte do
for id op_atr cte to cte step cte do
id op_atr id op_arit id
end_for
end_for
```

(a.1) Seq. Átomos p/ Código-fonte 1

S0 → **S1**
S1 → A1 A1 **B3** D1 D1
A1→ for **A2** to cte step cte do
B1→ C1 op_atr
C1→ id
D1→ end-for

A2→ B1 cte
B3→ B1 C1 op_arit C1

(a.2) Gramática Mínima Normalizada para código-fonte 1

```
for id op_atr cte to cte step cte do
for id op_atr cte to cte step cte do
id op_atr cte op_arit cte
end_for
end_for
```

(b.1) Seq. Átomos p/ Código-fonte 2

S0 → **S1|S2**

S2 → A1 A1 **A3** D1 D1
A1→ for A2 to cte step cte do
B1→ C1 op_atr
C1→ id
D1→ end-for
A2→ B1 cte

A3→ A2 op_arit cte
B3→ B1 C1 op_arit C1

(b.2) Gr. Min Normalizada para código-fonte 2

```
for id op_atr cte to cte step cte do
id op_atr cte op_arit cte
end_for
for id op_atr cte to cte step cte do
id op_atr id op_arit id
end_for
```

(c.1) Seq. Átomos p/ Código-fonte 3

S0 → **S1|S2 |S3**

S3 → A1 **A3** D1 A1 **B3** D1
A1→ for A2 to cte step cte do
B1→ C1 op_atr
C1→ id
D1→ end-for
A2→ B1 cte
A3→ A2 op_arit cte
B3→ B1 C1 op_arit C1

(c.2) Gr. Min Normalizada para código-fonte 3

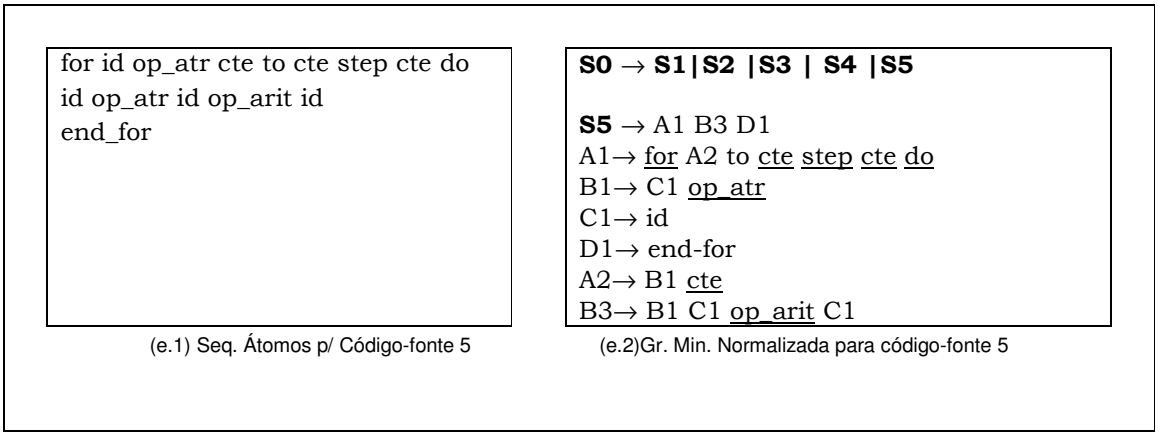
```
for id op_atr cte to cte step cte do
id op_atr cte op_arit cte
end_for
```

(d.1) Seq. Átomos p/ Código-fonte 4

S0 → **S1|S2 |S3| S4**

S4 → A1 A3 D1
A1→ for A2 to cte step cte do
B1→ C1 op_atr
C1→ id
D1→ end-for
A2→ B1 cte
A3→ A2 op_arit cte

(d.2) Gr. Min Normalizada para código-fonte 4



(e.1) Seq. Átomos p/ Código-fonte 5 (e.2)Gr. Min. Normalizada para código-fonte 5

Figura 4.40– Gramatícas geradas através dos átomos da linguagem com normalização dos símbolos não-terminais

Cada gramática é um exemplo, cada conjunto de símbolos não-terminais deverá ser analisado separadamente. Neste exemplo, será utilizado as produções do símbolo inicial de cada gramáticas mínima, ou seja, pode-se definir o conjunto de exemplos positivos da seguinte forma:

$$S^{+}= \{A_1A_1B_3D_1D_1,\ A_1A_1A_3D_1D_1,\ A_1\ A_3D_1A_1B_3D_1\ ,\ A_1A_3D_1,\ A_1B_3D_1\}$$

As Figuras 4.41, 4.42 e 4.43 apresentam respectivamente os autômatos de prefixo e sufixo para as “cadeias” do conjunto de exemplos positivos citado e o autômato inferido ao fim do processo.

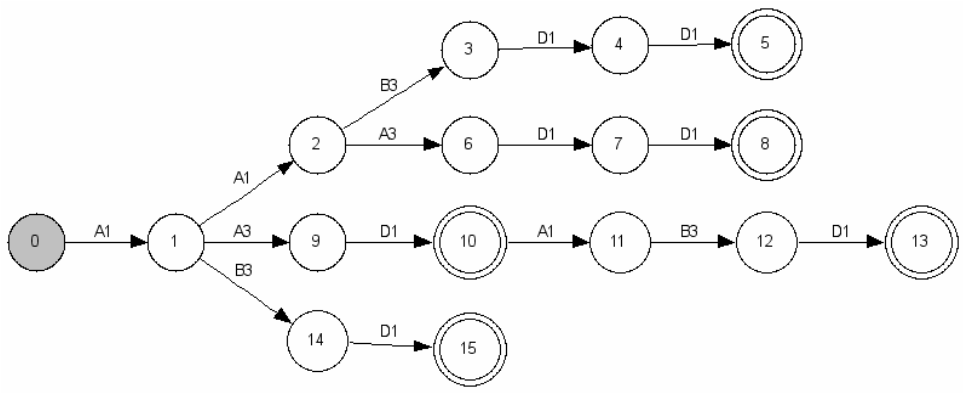


Figura 4.41– Autômatos de Prefixo para as Produções do Símbolo Inicial das Gramática

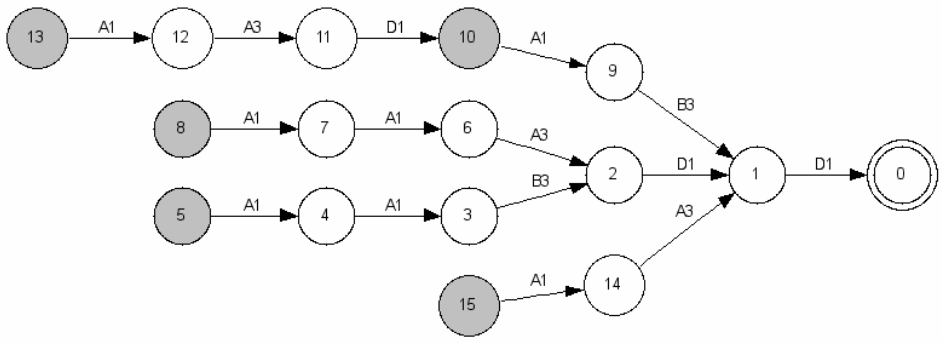


Figura 4.42 – Autômato de Sufixo para as Produções do Símbolo Inicial das Gramática

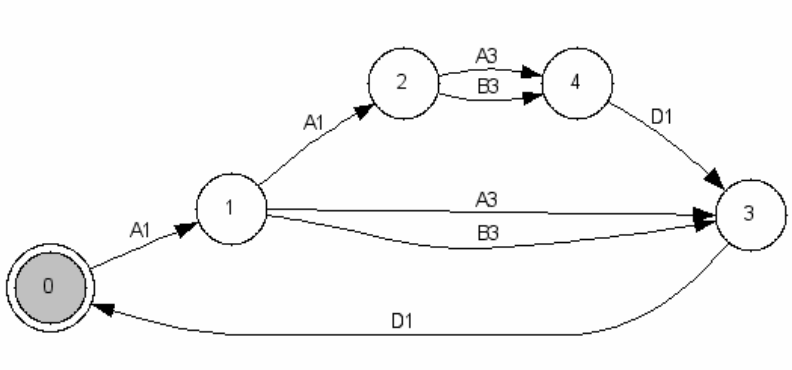


Figura 4.43 – Autômato Inferido para as Produções do Símbolo Inicial das Gramática

Como pode ser observado, o ciclo livre de contexto, para este caso poderá ser detectado utilizando-se a solução para o problema da gramática mínima com as modificações proposta e através do uso do algoritmo de JJ98 com as modificações para inferir gramáticas livres de contexto, utilizando o Algoritmo 4.8 citado em 4.3.2.

5. CONCLUSÃO

Neste trabalho realizou-se um estudo dos processos de inferências gramaticais e suas diversas aplicações. Um algoritmo adaptativo para inferência de linguagens regulares, proposto em 1998, foi implementado e testado; foram apresentadas, ainda, duas propostas de inferência de gramáticas livres de contexto. O uso de modelos adaptativos para o processo de aprendizado de gramáticas havia sido proposto em José Neto (1998), mas não implementado.

Neste capítulo apresentam-se os principais resultados, as principais contribuições e as propostas de trabalhos futuros.

5.1. RESULTADOS OBTIDOS

Como descrito no Capítulo 4, a implementação do algoritmo proposto por José Neto (1998) propiciou a realização de testes mais efetivos do algoritmo, inclusive utilizando *benchmarks* clássicos. Os testes apresentaram resultados semelhantes aos estudados na literatura para os demais métodos de inferência de gramáticas regulares. Entretanto, concluiu-se que há possibilidade de obter alguma melhoria nos resultados através da imposição de algumas modificações no algoritmo.

Com relação ao processo de inferência de gramáticas livres de contexto, utilizou-se como base o algoritmo de Charikar et al (2002). A proposta é utilizá-lo para auxiliar no processo de inferência de gramáticas livres de contexto, usando o problema da gramática mínima como heurística básica. O algoritmo proposto nesta pesquisa obteve bons resultados quando os exemplos correspondiam a trechos de código-fonte de alguma linguagem de programação (Matsuno, 2004). Mas, apesar da introdução de algumas mudanças no algoritmo, os resultados não são suficientemente bons para que seja considerado um algoritmo de inferência de gramáticas livres de contexto da forma como se encontra implementado. Contudo, os testes permitiram observar que é possível melhorar os resultados com uma alteração no algoritmo ainda nos seus estágios iniciais.

5.2. CONTRIBUIÇÕES

Como uma das principais contribuições deste trabalho, pode-se citar a implementação do algoritmo de inferência para autômatos finitos proposto por José Neto (1998), que, até o momento, não tinha sido testado com *benckmarks* clássicos da literatura. Os resultados obtidos por meio do uso do algoritmo no processo de inferência de gramáticas regulares também representam uma contribuição, porque permitem a avaliação do algoritmo frente a outros modelos.

Além disso, pôde-se realizar os experimentos com exemplos de linguagens livres de contexto, o que permitiu propor uma extensão do mesmo para tratamento de aninhamentos sintáticos.

Uma outra contribuição foi a utilização do algoritmo da gramática mínima e as modificações propostas de generalização para a inferência de linguagens livres de contexto. Os resultados conseguidos permitiram formular uma proposta de generalização do algoritmo apresentado nesta pesquisa.

Como outra contribuição, pode-se citar o experimento realizado com a junção dos algoritmos de JJ98 e as modificações propostas no algoritmo de Charikar, para sua utilização nas regras de formação da gramática livre de contexto.

Obteve-se mais uma evidência empírica da viabilidade de uso dos modelos adaptativos no processo de inferência gramatical (José Neto, 1993), o qual possui uma ampla faixa de aplicação. Além disso, o levantamento do estado da arte e de padrões de testes existentes poderá auxiliar em futuras pesquisas dos processos de inferência.

Por fim, esta pesquisa apresenta mais uma contribuição sob o ponto de vista didático-pedagógico ao mostrar o estado da arte na área de inferência gramatical e ao inserir mais um modelo como referência – o algoritmo de JJ98.

5.3. TRABALHOS FUTUROS

O trabalho realizado serve como base para outras pesquisas e trabalhos futuros:

Disponibilizar uma base integrada de construção de *software* – implementação e documentação – com base em interpretadores de autômatos (determinístico, de pilha e adaptativo) no uso do processo de inferência, como, por exemplo, a proposta de Lima (2002).

Realizar uma avaliação da complexidade computacional e expressividade das gramáticas inferidas nos diversos tipos de aplicação, como, por exemplo, em *benchmarks* de biologia computacional e outras aplicações.

Realizar a implementação do tratamento dos aspectos livres de contexto sugeridos no Capítulo 4 e estabelecer heurísticas que possibilitem distinguir os aspectos regulares dos livres de contexto, utilizando-se de critérios como os sugeridos nas propostas de Andrei et al. (2004), Adamowicza (2004), Cavaliere (2004), Nocka (2004) e Beltiukov (2004).

Analisar a possibilidade de uso dos exemplos negativos durante a inferência e não somente gerar dois autômatos inferidos positivos e negativos. Pesquisar uma forma de caracterizar a construção de exemplos negativos, pois um conjunto deles pode ser formado por qualquer cadeia que não pertença à linguagem. Diante disso, como observado em quase todos os casos, o autômato inferido a partir de exemplos negativos gerava um autômato que reconhecia a linguagem Σ^*, porque o conjunto de exemplos pode se tornar muito abrangente.

Avaliar a possibilidade de uso de autômatos e gramáticas estocásticos, pois o uso de probabilidades estatísticas no processo de inferência pode auxiliar na definição antes de se generalizarem determinados ciclos. Embora o algoritmo de JJ98 sem o uso de tal característica tenha obtido resultados desejados, ao tratar de conjuntos de exemplos muito grandes, a generalização não foi adequada, porque as árvores obtidas eram completas. Com isso, propagavam todas as transições com qualquer símbolo de entrada, que tinham como estado origem e estado final o mesmo estado.

6. REFERÊNCIAS BIBLIOGRÁFICAS

(Abbadingo, 1997)**Competição de Inferência de Gramáticas Regulares**. http://abbadingo.cs.unm.edu/

(Adamowicza, 2004) ADAMOWICZA, Z.; KOLODZIEJCZYK, L. A. **Well-behaved principles alternative to bounded induction.** In: *Theoretical Computer Science,* vol. 322, p. 5-16, 2004.

(Aho, 1972) AHO, A. V.; ULLMAN, J. D. **The Theory of Parsing, Translation and Compiling.**Editora Prentice Hall, 1972.

(Andrei et al, 2004) ANDREI, S.; CHIN W.; CAVADINI, S. V. **Self-embedded context-free grammars with regular counterparts.** In: *Springer-Verlag*, vol.17, p. 349 -365, 2004.

(Angluin, 1992), ANGLUIN, D., **Computational Learning Theory: Survey and Selected Bibliography**. *In: Procedingd of de Twenty-Forth Annual ACM Symposium on Theory of Computing*, 351-369. ACM Press

(Ausiello et al, 1999) AUSIELLO, G.; CRESCENZI, P.; GAMBOSI, G.; KANN, V.; MARCHETTI-SPACCAMELA, A.; PROTASI, M. **Complexity and Approximation - Combinatorial Optimization Problems and Their Approximability Properties**, Berlin: Springer-Verlag, 1999.

(Beltiukov, 2004) BELTIUKOV, A. P. **A strong induction scheme that leads t to polynomially computable realizations.** In: *Theoretical Computer Science,* vol. 322, p. 17 – 39, 2004

.

(Blake, 1998) BLAKE, C.L.; MERZ, C.J., **{UCI} Repository of machine learning databases".** University of California, Irvine, Dept. of Information and Computer Sciences, 1998.

(Carrasco, 1994a) CARRASCO, R.; ONCINA, J. **Learning Stochastic Regular Grammar by Means of State Merging Method**. Lecture Notes Artificial Intelligence. In: *Grammatical Inference and Application,* volume 862, p.139–152, 1994.

(Carrasco, 1994b) CARRASCO, R.; ONCINA, J., editors. **Proceedings of the Second International Colloquium on Grammatical Inference and Applications**, volume 862, 1994.

(Carrasco, 1994c) CARRASCO, R.C.; ONCINA, J., editors. **Grammatical Inference and Applications**, In: *Lecture Notes in Artificial Intelligence*, number 862. Springer-Verlag, 1994.

(Carrasco, 1999) CARRASCO, R.; ONCINA, J. **Learning deterministic regular grammars from stochastic samples in polynomial time**. RAIRO (Theoretical Informatics and Applications), 33(1):1–20, 1999.

(Cavaliere, 2004) CAVALIERE, M.; LEUPOLD, P. **Evolution and observation —a non-standardway to generate formal languages**.In: *Theoretical Computer Science*, vol. 321, p. 233 – 248, 2004.

(Charikar et al, 2002) CHARIKAR, M.; LEHMAN E.; LIU D.; PANIGRAHY R.; PRABHAKARAN M.; RASALA A., SAHAI A.; SHELAT, A. **Approximating The-Smallest Grammar: Kolmogorov Complexity In Natural Models.** STOC/02, May 19-21, 2002.Montreal, Quebec, Canadá. 2002 ACM 1-58113-495-9/02/0005

(Chomsky, 1959). CHOMSKY, N. **On certain formal properties of grammars.** *Information and Control*, 2:137–167, 1959.

(Cicchello, 2003) CICCHELLO, O; KREMER, S.C. **Inducing Grammars from Sparse Data Sets: A Survey of Algorithms and Results.** Journal of Machine Learning Research 4 (2003) 603-632.

(Crescenzi, 2006) **A compendium of NP optimization problems**. Editors: Pierluigi Crescenzi, and Viggo Kann.Site http://www.nada.kth.se/~viggo/wwwcompendium/node1.html com o compêndio de problemas de otimização NPO atualizado, acesso em 01.05.2006.

(Cruz-Alcázar, 1998) CRUZ-ALCÁZAR P.P.; VIDAL-RUIZ, E., **Learning Regular Grammars to Model Style: Comparing Different Coding Schemes**, In: *Grammatical Inference, ICGI'98*, number 1433, in: Lecture Notes in Artificial Intelligence, pages 211-222. Springer Verlag, 1998.

(de la Higuera, 2005) de la HIGUERA, C.; ONCINA, J. **Learning Context-Free Languages,** 2005, Kluwer Academic Publishers. Printed in the Netherlands.

(de la Higuera, 1997) de la HIGUERA, C., **Characteristic sets for polynomial grammatical inference**. Machine Learning, 27:125-138, 1997.

(Dupont et al, 1994) DUPONT, P.; MICLET, L.; VIDAL, E. **What is the search space of the regular inference?** In: *Grammatical Inference and Applications*, ICGI'94, number 862, in: *Lecture Notes in Artificial Intelligence*, pages 25-37. Springer Verlag, 1994.

(Dupont, 1994) DUPONT, P., **Regular grammatical inference from positive and negative samples by genetic search : the GIG method**. In: *Grammatical Inference and Applications*, ICGI'94, number 862. In: *Lecture Notes in Artificial Intelligence*, pages 236-245. Springer Verlag, 1994.

(Dupont, 1996) DUPONT, P., **Incremental regular inference. In Grammatical Inference : Learning Syntax from Sentences**, ICGI'96, number 1147. In: *Lecture Notes in Artificial Intelligence*, pages 222-237. Springer Verlag, 1996.

(Geertzenl, 2004) GEERTZENL, J.; van ZAANEN, M. **Grammatical Inference Using Suffix Trees**, G. Paliouras and Y. Sakakibara (Eds.) In: *Grammatical Inference and Applications*, ICGI 2004, in: *Lecture Notes in Artificial Intelligence*, LNAI 3264, p. 163–174, 2004.

(Gold, 1967) GOLD, E. M., **Language identification in the limit**. *Information and Control*, 10:447–474, 1967.

(Gold, 1978) GOLD, E. M., **Complexity of automaton identification from given data**. *Information and Control*, 37:302–320, 1978.

(Goldberg, 1989) Goldberg, D. E.; **Genetic Algorithms in Search, Optimization and Machine Learning,** Ed. Addison Wesley, Reading, Massachusetts, 1989.

(Gowachin, 2004) **Competição de Inferência de Gramáticas Regulares.** http://www.irisa.fr/Gowachin/

(Graña et al, 2003) GRAÑA, J.; ANDRADE, G.; VILARES, J. **Compilation of Constraint-Based Contextual Rules for Part-of-Speech Tagging into Finite State Transducers,** , J.-M. Champarnaud and D. Maurel (Eds.): CIAA 2002, LNCS 2608, pp. 128–137, 2003.Springer-Verlag Berlin Heidelberg 2003.

(Gregor, 1994) GREGOR, J. **Data-driven inductive inference of finite-state automata.** International Journal of Pattern Recognition and Artificial Intelligence, 8(1):305-322, 1994.

(Hopcroft, 1979) HOPCROFT, J. E.; ULLMAN, J. D. **Introduction to Automata Theory, Languages and Computation**. Addison-Wesley, 1979.

(Iwai, 2000) IWAI M., **Gramáticas Adaptativas**, Tese de Doutorado, USP, São Paulo, 2000.

(Jackson, 2001) JACKSON, Q. T.; LANGAN, C. M. **Adaptive Predicates in Empty-Start Natural Language Parsing Noesis**-E, v. 1, n. 6, November, 2001.

(José Neto, 1987) JOSÉ NETO, J. **Introdução à Compilação.** Rio de Janeiro: Editora LTC, 1987.

(José Neto, 1993) JOSÉ NETO, J. **Contribuição à Metodologia de Construção de Compiladores.** Tese (Livre docência em Engenharia) – Escola Politécnica, Universidade de São Paulo, São Paulo, 1993.

(José Neto, 1998) JOSÉ NETO, J.; IWAI, M. K. ***A*daptative Automata for Syntax Learning.** *Anais Conferência Latino Americana de Informática* – CLEI 1998 MEMORIAS. pp. 135-149, Quito, Equador, 1998.

(José Neto, 2001) JOSÉ NETO, J. ***A*daptive Rule-Driven Devices - General Formulation and Case Study.** *Lecture Notes in Computer Science*.Watson, B.W. and Wood, D. (Eds.): *Implementation and Application of Automata 6th International Conference*, CIAA 2001, Vol.2494, Pretoria, South Africa, July 23-25, pp. 234-250 2001.

(Juillé, 1998) JUILLÉ, H.; POLLACK, J.B. **A stochastic search approach to grammar induction**. In: *Grammatical Inference*, number 1433, in: *Lecture Notes in Artificial Intelligence*, pages 126-137. Springer-Verlag, 1998.

(Kearns, 1994) KEARNS, M. J.; VAZIRANI, U. V. **An introduction to computational lerning theory**. Cambridge, MA: MIT Press.

(Kessey, 2004) KEESEY, J. **CFG Based File Compression**, http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Fall03/Keesey/index.shtml?report.shtml. Acesso em: 08/jan/2004

(Klein, 2002) KLEIN, A.; KUTRIB, M. **Self-Assembling Finite Automata.** In: O. H. Ibarra, L. Zhang (eds.), *Computing and Combinatorics,* (COCOON 2002), LNCS 2387, Springer 2002, 310-3.

(Lang et al,1998) LANG, K. J.; PEARLMUTTER, B. A.; PRICE, R. **Results of the Abbadingo One DFA Learning Competition and a New Evidence Driven State Merging Algorithm**.In: *Grammatical Inference*, number 1433, In: *Lecture Notes in Artificial Intelligence,* pages 1-12. Springer-Verlag, 1998.

(Lang, 1992) LANG, K.J. **Random DFA's can be approximately learned from sparse uniform examples**. In: *Proceedings of the Fifth ACM Workshop on Computational Learning Theory,* pages 45–52, New York, N.Y., 1992.

(Lang, 1999) LANG, K. J. **Faster algorithms for finding minimal consistent dfas**.1999.

(Langley, 2000) LANGLEY, P.; STROMSTEN, S. **"Learning Context-Free Grammars with a Simplicity Bias"**. *Proceedings of the Eleventh European Conference on Machine Learning (ECML 2000), Lecture Notes in Artificial Intelligence* 1810, Springer – Verlag, pp. 220–228, Barcelona, Spain, 2000.

(Levy, 1978) LEVY, L.S.; JOSHI, A. K. **Skeletal Strucutural Descriptions**. In: *Information and Control,*v.39, p.192-211, 1978.

(Lewis, 1998) LEWIS, H. R.; PAPADIMITRIOU, C. H. **Elements of the theory of computation.** New Jersey, Prentice-Hall, Inc, 1998.

(Li, 1994) LI M.;VALIANT, L. G. (Eds.). **Special issue on computational learning theory**. *Machine Learning*, 14 (1).

(Lima, 2002) LIMA, A.M. **Laboratório de geração de classificadores de seqüências.** Dissertação de Mestrado, Universidade de São Paulo, São Paulo, 2002.

(LTA, 2006) Laboratório de Tecnologia Adaptativa - http://www.pcs.usp.br/~lta acessado em 01.05.2006

(Lucas, 2002) LUCAS, S.M.; REYNOLDS, J. T. **Learning DFA: Evolution versus Evidence driven State Merging**.

(Matsuno, 2004) MATSUNO, I. P.; ROCHA, R. L. A**., Aplicação do Problema da Gramática Mínima ao Projeto do Aspecto Sintático de Linguagens de Programação**. In: *Anais do Simpósio Brasileiro de Linguagem de Programação – SBLP 2004*, pp 229-241, Niterói, Rio de Janeiro, 2004.

(Menezes, 2002) MENEZES, P.F.B. **Linguagens Formais e Autômatos**. 4ª ed. Sagra Luzzatto. Porto Alegre, 2002.

(Menezes, 2003) MENEZES, P.F.B.; CAMPANI, C. A. P. **Introdução à Complexidade de Kolmogorov.** Notas de aula. 2003.

(Miclet et al, 2004) . MICLET, L.; ONCINA, J.; CARRASCO, R.; CASACUBERTA, P.; EYRAUD, R.; EZEQUEL, P.; FERNAU, H.; MURGUE, T.; THOLLARD, F.; VIDAL, E. **Applications of Grammatical Inference**. Grammatical Inference 2004.

(Mitchell, 1997) MITCHELL, T., ***Machine Learning*** , Ed. McGraw-Hill, 1997.

(Nakamura, 2000) NAKAMURA, K.; ISHIWATA, Y. **Synthesizing context free grammars from sample strings based on inductive CYK algorithm**. *Fifth Internacional Colloquium on Gramatical Inference (ICGI 2000)*, LNAI 1891 (Springer-Verlag, 2000) 186-195.

(Nakamura, 2002) NAKAMURA, K.; MATSUMOTO. **Incremental Learning of Context Free Grammars**.*6-th Internacional Colloquium on Gramatical Inference (ICGI 2002)*, LNAI 2484 (Springer-Verlag, 2002) 174-184.

(Nakamura, 2004) NAKAMURA, K. **Incremental Learning of Context Free Grammars by Extended Inductive CYK Algorithm**.In: Grammatical Inference and Applications, ICGI 2004. In: *Lecture Notes in Artificial Intelligence*, p X-Y. Springer Verlag, 2004.

(Nevill-Manning, 1996) NEVILL-MANNING, C.G.; WITTEN, I.H. **I*nferring Sequential Structure.*** Thesis Doctor, University of Waikato, 1996.

(Nevill-Manning, 1997) NEVILL-MANNING, C.G.; WITTEN, I.H. **Compression and Explanation Using Hierarchical Grammars.** The Computer Journal, 40(2/3):103-116,1997

.

(Nocka , 2004) NOCKA, R.; NIELSEN, F. **On domain-partitioning induction criteria: worst-case bounds for the worst-case based.** In: *Theoretical Computer Science*, vol. 321 p. 371 – 382, 2004.

(Omphalos, 2004) **Competição de Inferência de Gramáticas Livres de Contexto.** http://www.irisa.fr/symbiose/people/coste/gi_benchs.html

(Oncina, 1992a) ONCINA, J.; GARCIA, P. **Identifying regular languages in polynomial time.** In: H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of Series, in: *Machine Perception and Artificial Intelligence*, pages 99-108. World Scientific, 1992.

(Oncina, 1992b) ONCINA, J.; GARCIA, P. **Inferring regular languages in polynomial update time**. In: N. Pérez de la Blanca, A. Sanfeliu, and E.Vidal, editors, *Pattern Recognition and Image Analysis*, volume 1 of Series, in: *Machine Perception and Artificial Intelligence*, pages 49-61. World Scientific, 1992.

(Parekh et al., 1998) PAREKH, R.; NICHITIU, C.; HONAVAR, V. **A polynomial time incremental algorithm for learning DFA**. In: *Grammatical Inference*, ICGI'98, number 1433, in: *Lecture Notes in Artificial Intelligence*, pages 37-49. Springer Verlag, 1998.

(Parekh, 2000) PAREKH, R.; HONAVAR, V.**Grammar Inference, Automata Induction, and Language Acquisition**.R. Dale, H. Moisl, and H. Somers (eds.), Handbook of Natural Language Processing, chapter 29, pp. 727–764, Marcel Dekker Inc., 2000.

(Pariente, 2004) PARIENTE, C. A. B.. **Gramáticas Livres de Contexto Adaptativas com Verificação de Aparência.** Tese de Doutorado, USP, São Paulo, 2004.

(Pedrazzi et al, 2004) PEDRAZZI, T.; MATSUNO, I. P.; ROCHA, R.L.A. **Uma Proposta de Ensino da Disciplina de Linguagens Formais e Autômatos para um Curso de Engenharia da Computação.** In: *COBENGE 2004, Anais*, UnB:Brasília, 2004.

(Petasis et al, 2004a) PETASIS, G.; PALIOURAS, G.; SPYROPOULOS, C. D.; HALATSIS.C. **eg-GRIDS: Context-Free Grammatical Inference from Positive Examples using Genetic Search**.2004.

(Petasis et al, 2004b) PETASIS, G.; PALIOURAS, G.; KARKALETSIS, V.; HALATSIS, C.; SPYROPOULOS, C. D. **"e-GRIDS: Computationally Efficient Grammatical Inference from Positive Examples"**, *Grammars, Special Issue*, 2004. Available from http://217.125.102.104/special4.asp

(Pistori, 2002), PISTORI, H.; JOSÉ NETO, J., **AdapTree - Proposta de um Algoritmo para Indução de Árvores de Decisão Baseado em Técnicas Adaptativas.** Anais Conferência Latino Americana de Informática - CLEI 2002. Montevideo, Uruguai, Novembro, 2002.

(Pistori, 2003) PISTORI, H. **Tecnologia Adaptativa em Engenharia da Computação: Estado da Arte e Aplicações.** 2003. Tese (Doutorado em Engenharia Elétrica) – Escola Politécnica, Universidade de São Paulo, São Paulo.

(Rocha, 2000a) ROCHA, R.L.A. **Um Método de Escolha Automática de Soluções Usando Tecnologia Adaptativa.**São Paulo, 2000, 211p. Tese(Doutorado) Escola Politécnica, Universidade de São Paulo.

(Rocha, 2000b) ROCHA, R. L. A.; JOSÉ NETO, J. **Autômato Adaptativo, Limites e Complexidade em omparação com Maquina de Turing**. In: *Proceedings of the second Congress of Logic Applied to Technology* – LAPTEC 2000. Proceedings, São Paulo, p. 33-48, 2000.

(Ron, 95) RON, D. **Automata Learning and its Applications**. PhD thesis, Hebrew University, 1995.

(Rubinstein, 1993) RUBINSTEIN, R.; SHUTT, J. N. **Self-Modifying Finite Automata.** Techinical Report WPI-CS-TR-93-11, Worcester Polytechnic Institute, Worcester, Massachusetts, December, 1993.

(Rubinstein, 1995) S. RUBINSTEIN, R.S., SHUTT, J. N. **Self-Modifying Finite Automata - Basic Definitions and Results**. *Computer Science Technical Report Series*. Worcester Polytechnic Institute, WPI-CS-TR-95-2, Agosto, 1995

(Rytter, 2002) RYTTER, W. **Application of Lempel –Ziv Factorization to the Approximation of Grammar-Based Compression.** In: *Combinatorial Pattern Matching, Lecture Notes in Computer Science*,Vol.2373, Springer, Berlin, June 2002, pp.20 –31.

(Sakakibara et al, 1994) SAKAKIBARA, Y.; UNDERWOOD M. BROWN, R. C.; MIAN, I. S.; HAUSSLER, D. **Stochastic ontext-free grammars for modeling RNA**. In: Lawrence Hunter, editor, *Proceedings of the 27th nnual Hawaii International Conference on System Sciences*. Volume 5 : Biotechnology Computing, pages 284–294, Los Alamitos, CA, USA, January 1994. IEEE Computer Society Press.

(Sakakibara, 1992) SAKAKIBARA, Y. **Efficient learning of context-free grammars from positive structural examples**. *Information and Computation*, 97:23-60, 1992.

(Sakakibara, 1995) SAKAKIBARA, Y. **Grammatical Inference: An old and new paradigm**.*Lecture Notes Artificial Intelligence*, 997:1–24, 1995.

(Sakakibara, 1997) Y. SAKAKIBARA. **Recent advances of grammatical inference**. *Theoretical Computer Science*, (185):15-45, 1997.

(Sakakibara, 1999) SAKAKIBARA, Y.; KONDO, M. **GA-based learning of context-free grammars using tabular representations.** *Proceedings 16th International Conference of Machine Learning* (Morgan Kaufmann, 1999) 354-360.

(Sakakibara, 2000) SAKAKIBARA, Y. ; MURAMATSU,H., **Learning of context-free grammars partially structured examples**, Fifth International Colloquium on Grammatical Inference(ICGI 2000), LNAI 1891 (Springer-Verlag, 2000) 229-240.

(Shutt, 1993) SHUTT, J. N. **Recursive Adaptable Grammars**. Masther Thesis, Worcester Polytechnic Institute, Worcester, Massachusetts, 1993.

(Shutt, 1995) SHUTT, J. N. **Self-Modifying Finite Automata - Power and Limitations**.*Computer Science Technical Report Series*. Worcester Polytechnic InstituteWPI-CS-TR-95-4, December 1995

(Shutt, 1999) SHUTT, J. N. **Recursive Adaptable Grammars**. Techincal Report WPI-CS-TR-99-03, Worcester Polytechnic Institute, Worcester, Massachusetts, January, 1999.

(Tomita, 1982) TOMITA, M. **Dynamic construction of finite-automata from examples using hill-climbing**. In: *4th Annual Cognitive Science Conference*, pages 105-108, 1982.

(Trakhtenbrot, 1973) TRAKHTENBROT, B.; BARZDIN, Y. **Finite automata: Behavior and synthesis**. North Holland Publishing Company, Amsterdam, 1973.

(Vervoort, 2002) VERVOORT, M. **Emile 4.4.6 User Guide** (Universiteit van Amsterdam, 2002).

(Vieira, 2004) VIEIRA, D.da C.G. **Geração de Classificadores de Seqüências Genéticas Utilizando Inferência de Linguagens Regulares**. Dissertação apresentada ao Instituto de Matemática e Estatística da Universidade de São Paulo para obtenção do grau de Mestre em Ciência da Computação. São Paulo, SP, 2004.

(Wyard, 1994) WYARD, P. **Representational issues for context free grammars induction using genetic algorithms**. In: *Grammatical Inference and Applications*, ICGI'94, number 862, in: *Lecture Notes in Artificial Intelligence*, pages 222-235. Springer Verlag, 1994.

(Ziv, 1977) ZIV, J.; LEMPEL, A. **A Universal Algorithm for Sequential Data Compression.** *IEEE Transactions on Information Theory* ,IT-23(3):337-343,1977.

(Ziv, 1978) ZIV, J.; LEMPEL, A. **Compression of Individual Sequences via Variable-rate Coding.** *IEEE Transactions on Information Theory* ,IT-24:530-536,1978.