César Alberto Bravo Pariente

Gramáticas Livres de Contexto

Adaptativas
com verificação de aparência

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

São Paulo
2004

César Alberto Bravo Pariente

Gramáticas Livres de Contexto

Adaptativas
com verificação de aparência

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

Área de concentração:
Sistemas Digitais

Orientador:
Prof. Dr. João José Neto

São Paulo
2004

Ficha Catalográfica

Bravo Pariente, César Alberto
Gramáticas Livres de Contexto Adaptativas
com verificação de aparência. São Paulo, 2004. **??** p. 241

Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Gramáticas Adaptativas. 2. Autômatos Adaptativos. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais. II. Título.

Dedico este trabalho a todos aqueles que não param de sonhar; mas, sobretudo, àqueles que têm a sensatez de apreciar os sonhos com moderação.

Data: Mon, 24 Nov 2003 11:04:46 -0800
De: Ray Bradbury < ps.mbl@planetary.org>
Para: My Friends <specialevent@lists.planetary.org>
Responder-para: The Planetary Society <tps.mbl@planetary.org>
Assunto: Thank You for Celebrating a Martian Dream
Dear Friend,

I want to thank you for the birthday greeting you sent me via The Planetary Society. The giant birthday card covered with messages from my friends around the world was presented to me by the Society at a wonderful party during the week of Mars' closest approach to Earth. I appreciate each and every one of you, and all the warm wishes you sent me.

People often ask me how I stay so young, how I've kept such a "youthful"outlook. The answer is simple: Live a life in which you cram yourself with all kinds of metaphors, all kinds of activities, and all kinds of love. On that August night when the Society celebrated my birthday I shared with my friends a dream I have: Some night one hundred years from now, there'll be a boy on Mars reading late at night, with a flashlight under the covers, and he'll look out at the Martian landscape – which will be bleak, rocky and red, and not very romantic. I hope he will be reading my book, The Martian Chronicles.

That's my dream, and that's why I spent that evening celebrating my birthday with my friends at The Planetary Society, who work hard every day to make such a dream possible. There's much work left to be done to achieve my dream and we could use your help. Please join me as a member of The Planetary Society. Working together we will make this grand adventure come to fruition.

Thank you.
Ray Bradbury
Join The Planetary Society at:
http://store.yahoo.com/planetarysociety/about-us.html

"A veces ocurren cosas lindas en la vida, pena que el efecto dura poco"
Marcelo Esteban Coniglio

"Lo malo de la felicidad es que, cuando se vá, nos deja su recuerdo"
Autor desconocido

Agradecimentos

Agradeço a meu orientador, prof. Doutor João José Neto, pela orientação e à CAPES pela bolsa usufruída. Agradeço também ao programa PROAP da Escola Politécnica da Universidade de São Paulo, que me forneceu auxílio financeiro para assistir e participar da Oitava Conferência Internacional sobre Implementação e Aplicação de Autômatos, CIAA 2003, realizada em Santa Barbara, Califórnia, EUA, entre 16 e 18 de julho de 2003. Essa viagem foi possível pelo apoio de vários amigos, em especial, Marcelo Coniglio e Luiz Junqueira.

Agradeço também aos funcionários da biblioteca da Poli-Elétrica que várias vezes conseguiram referências bibliográficas para minha pesquisa pelo serviço **COMUT**. Além deles, foram de grande ajuda os serviços de acesso às bibliotecas digitais da ACM (www.acm.org/dl), ScienceDirect, do Portal CAPES, o serviço http://www.springerlink.com da editora Springer-Verlag, e as bases de dados públicas CiteSeer http://citeseer.nj.nec.com/cs, e DBLP http://www.informatik.uni-trier.de/ ley/db/index.html. De fora do Brasil, algumas outras pessoas me ajudaram também com o material bibliográfico: Da a Índia, a professora Kamala Krithivasam, me enviou por correio postal, seus artigos sobre *autômatos variantes no tempo*. Dos EUA, John Shutt, teve a gentileza de responder, por e-mail, algumas questões sobre os exemplos na sua dissertação de mestrado. Do Canadá, Quinn Tyler Jackson esteve sempre pronto a trocar, por e-mail, idéias, exemplos, material bibliográfico e licenças educacionais de seu produto de software *Meta-S*. A todos eles, meus sinceros agradecimentos.

Meus agradecimentos também para os funcionários do departamento e para o pessoal técnico. As disciplinas e os trabalhos da pós-graduação foram amenizados pela amizade dos colegas da pós-graduação, notadamente: Fabrizio Leonardi, Renata Romariz Recco, e Mario Donato Marino; tive também oportunidade de fazer amizades em outros departamentos da Poli-Elétrica: lembranças especiais para os quase-sempre bem comportados programadores do GAGTD e, ultimamente, para o irreverente time de programação do GEPEA.

Resumo

Este trabalho descreve o formalismo das *gramáticas livres de contexto adaptativas com verificação de aparência*.

Esses dispositivos gramaticais possuem como núcleo uma gramática livre de contexto subjacente e, como mecanismo de auto-modificação, uma ou várias *funções adaptativas* que determinam quais produções são aplicáveis em cada passo de uma derivação. A *verificação de aparência* se refere a uma forma especial de aplicar algumas produções, escolhidas pelo projetista da gramática, sem alterar a forma sentencial nessa aplicação.

É provado que esse formalismo tem poder de máquina de Turing demons-- trando, em forma construtiva, sua equivalência com quatro formalismos gramaticais baseados em gramáticas livres de contexto com mecanismos de controle, que tem esse poder. São desenvolvidos dois analisadorers para linguagens dependentes de contexto a partir de um desses outros quatro formalismos. Um deles, que é baseado em *autômatos-pilha*, opera em forma *ascendente*; o outro, baseado em *autômatos finitos adaptativos*, opera em forma *descendente*.

Abstract

This work introduces and describes the formalism of the *context-free adaptive grammar with appearance checking*.

Such gramatical devices have as its kernel a subjacent context-free grammar and, as mechanism of self-modification, one or several *adaptive functions* which determines the productions able to be applied at each step of a derivation. The *appearance checking* refers to a special way to apply some productions, choosen by the designer of the grammar, without changing the sentential form in this application.

It is proved that this formalism has Turing Machine power, proving, by construction, its equivalence with four grammatical formalisms based on context-free grammars and with control mechanisms, with such power. Two parsers have been developed for context-dependent languages from one of these four formalisms. One of them is based on *stack-automata*, and operates in a *bottom-up* fashion. The other is based on *adaptive automata*, and operates in a *top-down* fashion.

Sumário

Lista de Figuras

Lista de Abreviaturas

símbolo	descrição
TM	Turing Machine
LBA	Linear Bounded Automata
PDA	Pushdown Automata
FA	Finite Automata
$TVPDA$	Time-Varying Pushdown Automata
$PTVPDA$	Periodically Time-Varying Pushdown Automata
$TVFA$	Time-Varying Finite Automata
$PTVFA$	Periodically Time-Varying Finite Automata
AA	Adaptive Automata
$SMFA$	Self-Modifying Finite Automata
RAG	Recursive Adaptable Grammars
$cfmg$	Context-Free Matrix Grammar
$cfpg$	Context-Free Programmed Grammar
$cftvg$	Context-Free Time-Varying Grammar
$cfptvg$	Context-Free Periodically Time-Varying Grammar
$cfrc$	Context-Free Grammar with Regular Control Language
AG	Adaptive Grammar
$cfagwac$	Context-Free Adaptive Grammar with appearance checking
1-SA	Bottom-up Parser based on Stack-Automata
pAA	Top-down Parser based on Adaptive Finite Automata
r-FA	Self-Assembling Deterministic Finite Automata
r-NFA	Self-Assembling Non-deterministic Finite Automata

Lista de Símbolos

Da Teoria dos conjuntos, e das linguagens formais, utilizamos os seguintes símbolos:

símbolo	descrição
$\cup$	união de conjuntos
$\cap$	intersecção de conjuntos
$\setminus$	diferença de conjuntos
$\emptyset$	conjunto vazio
$\|A\|$	cardinalidade do conjunto A
2^A	conjunto potência do conjunto A
$\mathbb{N}$	conjunto dos números naturais
λ	palavra vazia
$\|w\|$	comprimento da palavra w
w^n	concatenação de n cópias da palavra w
$\mathcal{L}_3$	família das linguagens de tipo 3 na hierarquia de Chomsky
$\mathcal{L}_2$	família das linguagens de tipo 2 na hierarquia de Chomsky
$\mathcal{L}_1$	família das linguagens de tipo 1 na hierarquia de Chomsky
$\mathcal{L}_0$	família das linguagens de tipo 0 na hierarquia de Chomsky
$\mathcal{REG}$	família das linguagens de regulares
$\mathcal{CF}$	família das linguagens livres de contexto
$\mathcal{CS}$	família das linguagens dependentes de contexto
$\mathcal{RE}$	família das linguagens recursivamente enumeráveis
G	$= (N, T, P, S)$ gramática na hierarquia de Chomsky, com • N : o conjunto de não-terminais, • T : o conjunto de terminais, • P : o conjunto de produções, • $S \in N$: o símbolo inicial da gramática G.

Notação utilizada para as gramáticas com mecanismos de controle:

símbolo	descrição	página
GM	$= (N, T, M, S)$ gramática matricial, com • N : o conjunto de não-terminais, • T : o conjunto de terminais, • M : o conjunto de matrizes, • $S \in N$: o símbolo inicial da gramática G.	35
(GM, F)	$= (N, T, M, S, F)$ gramática matricial, com • $G = (N, T, M, S)$: uma gramática matricial, • $F \subset P$: o conjunto de produções que se aplicam em modo de verificação de aparência.	40
GT	$= (G, v)$ gramática variante no tempo, com • $G = (N, T, P, S)$: uma gramática na hierarquia de Chomsky, • $v : \mathbb{N} \to 2^P \setminus \{\emptyset\}$: a função de variação.	52
(GT, F)	$= (G, v, F)$ gramática variante no tempo com verificação de aparência para as produções em F, com • $GT = (G, v)$: uma gramática variante no tempo, • $F \subset P$: o conjunto de produções que se aplicam em modo de verificação de aparência.	59
GP	$= (G, s, f)$ gramática programada, com • $G = (N, T, P, S)$: uma gramática na hierarquia de Chomsky, • $s : P \to 2^P$: a função *sucesso*. • $f : P \to 2^P$: a função *falha*.	68

continua...

símbolo	descrição	página
GR	$= (G, C)$ gramática com linguagem de controle C, com • $G = (N, T, P, S)$: uma gramática na hierarquia de Chomsky, • C : uma linguagem sobre o alfabeto P.	89
(GR, F)	$= (G, C, F)$ gramática com linguagem de controle C e com verificação de aparência para produções em F, com • $GR = (G, C)$: uma gramática com linguagem de controle C. • $F \subset P$: o conjunto de produções que se aplicam em modo de verificação de aparência.	89
GA	$= (G, [P_0], [F], \mathcal{A})$ gramática livre de contexto adaptativa, com • $G = (N, T, P, S)$: a gramática livre de contexto subjacente, • $P_0 \subset P$ e $[P_0]$: o conjunto de produções livres de contexto adaptativas iniciais, • $F \subset P$ e $[F]$: o conjunto de produções adaptativas que se aplicam em modo de verificação de aparência, • $\mathcal{A} : P \to 2^P$: a função adaptativa.	121

Notação para as classes das linguagens geradas pelas gramáticas controladas:

símbolo	descrição	página
$\mathcal{M}$	Família das linguagens geradas pelas gramáticas livres de contexto matriciais sem produções-λ	36
$\mathcal{M}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto matriciais	37
$\mathcal{M}_{ac}$	Família das linguagens geradas pelas gramáticas livres de contexto matriciais sem produções-λ e com verificação de aparência	41
$\mathcal{M}_{ac}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto matriciais com verificação de aparência	41
$\mathcal{M}_{esq}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto matriciais sob derivações mais à esquerda	43
$\mathcal{M}_{esq}$	Família das linguagens geradas pelas gramáticas livres de contexto matriciais sem produções-λ sob derivações mais à esquerda	44
$\mathcal{T}_{ac}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto periodicamente variantes no tempo com verificação de aparência	59
$\mathcal{T}$	Família das linguagens geradas pelas gramáticas livres de contexto periodicamente variantes no tempo sem produções-λ	60
$\mathcal{T}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto periodicamente variantes no tempo	60
$\mathcal{T}_{ac}$	Família das linguagens geradas pelas gramáticas livres de contexto periodicamente variantes no tempo sem produções-λ e com verificação de aparência	60
$\mathcal{P}$	Família das linguagens geradas pelas gramáticas livres de contexto programadas sem produções-λ	70
$\mathcal{P}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto programadas	70
$\mathcal{P}_{ac}$	Família das linguagens geradas pelas gramáticas livres de contexto programadas sem produções-λ e com verificação de aparência	70

continua...

símbolo	descrição	página
$\mathcal{P}_{ac}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto programadas com verificação de aparência	70
$\mathcal{R}$	Família das linguagens geradas pelas gramáticas livres de contexto com linguagem de controle regular sem produções-λ	96
$\mathcal{R}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto com linguagem de controle regular	96
$\mathcal{R}_{ac}$	Família das linguagens geradas pelas gramáticas livres de contexto com linguagem de controle regular sem produções-λ e com verificação de aparência	96
$\mathcal{R}_{ac}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto com linguagem de controle regular com verificação de aparência	96
$\mathcal{LA}$	Família das linguagens geradas pelas gramáticas livres de contexto adaptativas sem produções-λ	124
$\mathcal{LA}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto adaptativas	124
$\mathcal{LA}_{ac}$	Família das linguagens geradas pelas gramáticas livres de contexto com linguagem de controle regular sem produções-λ e com verificação de aparência	124
$\mathcal{LA}_{ac}^{\lambda}$	Família das linguagens geradas pelas gramáticas livres de contexto adaptativas com verificação de aparência	124

Outros símbolos utilizados:

- $\mathcal{L}(i,j,k)$ família das linguagens geradas pelas gramáticas (G,C) com:
  - G : uma gramática de tipo i na hierarquia de Chomsky,
  - C : a linguagem de controle gerada por uma gramática de tipo i na hierarquia de Chomsky,
  - $k = 1$ indica que se considera um conjunto não vazio de produções a serem aplicadas em modo de verificação da aparência e $k = 0$ indica que não há produções a serem aplicadas em modo de verificação da aparência.

Capítulo 1

Introdução

Neste capítulo é apresentado o objetivo desta tese, assim como a motivação que levou ao seu desenvolvimento. Também faremos uma resenha sobre dispositivos com estrutura dinâmica[1], baseados em máquinas de estados e baseados em gramáticas, que identificam a classe das linguagens Recursivamente Enumeráveis para situar o presente trabalho no marco geral da teoria da computação. Na última seção é apresentada a estrutura da tese dando uma visão geral sobre os assuntos tratados nos demais capítulos.

1.1 Objetivo

O objetivo desta tese é representar linguagens recursivamente enumeráveis usando dispositivos dinâmicos baseados em gramáticas livres de contexto. Os dispositivos em questão são uma especialização das gramáticas adaptativas desenvolvidas em Iwai[1].

Para alcançar esse objetivo fazemos, no capítulo 2 deste trabalho, um estudo sobre *gramáticas livres de contexto com mecanismos de controle*[2], [3], [4], que são dispositivos gramaticais com poder computacional de máquina de Turing. Como conseqüência desse estudo, atingimos nosso objetivo ao provar, no capítulo 3, a equivalência do formalismo proposto nesta tese com as *gramáticas livres de contexto com mecanismos de controle*.

Outro resultado obtido, durante a elaboração deste trabalho, é a dedução de um analisador não-determinístico para linguagens dependentes de contexto expressas por uma das *gramáticas livres de contexto com mecanismos de controle* estudadas

[1] que mudam, durante sua operação, as regras que inicialmente os definem

no capítulo 2[5].

Além disso, foi obtido também um analisador determinístico, baseado em autômatos finitos adaptativos, para linguagens dependentes de contexto expressas por uma das *gramáticas livres de contexto com mecanismos de controle* cujo conjunto de produções é tal que o único não terminal que produz uma seqüência de não-terminais é o símbolo inicial da gramática.

1.2 Motivação

Em teoria de computação, as linguagens formais têm sido agrupadas em quatro classes, denotadas $\mathcal{L}_i$, $0 \leq i \leq 3$, segundo o tipo da gramática, na hierarquia de Chomsky, que as gera[2]. A essas quatro classes correspondem quatro formalismos basedos em máquinas de estado que servem como reconhecedores para cada uma das classes das linguagens formais: *Máquinas de Turing* (TM pelas suas siglas em inglês *Turing Machine*), *Autômato Linearmente Limitado* (LBA pelas suas siglas em inglês *Linear Bounded Automata*), *Autômato de Pilha* (PDA pelas suas siglas em inglês *Pushdown Automata*), *Autômato Finito* (FA pelas suas siglas em inglês *Finite Automata*). A seguinte tabela mostra a identificação entre as classes das linguagens, os tipos de gramáticas e as diferentes máquinas[2]:

Classes das Linguagens	Exemplo	Gramáticas	Máquinas
$\mathcal{L}_0$ Recursivamente Enumeráveis ($\mathcal{RE}$)	$-$	Tipo 0	TM
$\mathcal{L}_1$ Dependentes de Contexto	$a^n b^n c^n$	Tipo 1	LBA
$\mathcal{L}_2$ Livres de Contexto	$a^n b^n$	Tipo 2	PDA
$\mathcal{L}_3$ Regulares	$a^n b^m$	Tipo 3	FA

Denotamos com $\mathcal{L}(TM)$, $\mathcal{L}(LBA)$, $\mathcal{L}(PDA)$ e $\mathcal{L}(FA)$ as classes das linguagens aceitas, respectivamente, pelas máquinas TM, LBA, PDA, e FA. Essas classes de linguagens estão relacionadas como a seguir: (em particular, as linguagens de programação, denotadas como LP, encontram-se numa posição intermediaria entre a classe $\mathcal{L}_1$ e a classe $\mathcal{L}_2$[2]):

$$\mathcal{L}_3 = \mathcal{L}(FA) \subset \mathcal{L}_2 = \mathcal{L}(PDA) \subseteq LP \subseteq \mathcal{L}_1 = \mathcal{L}(LBA) \subset \mathcal{L}_0 = \mathcal{L}(TM)$$

Tanto as máquinas de estado quanto as gramáticas consideradas nas inclusões acima são dispositivos de *estrutura estática*; isto quer dizer que operam segundo um conjunto fixo de regras, definido antes do inicio de sua operação, e que não muda durante o transcurso da mesma.

Existe, porém, outro tipo de dispositivos, o dos que possuem *estrutura dinâmica*; isto quer dizer, que antes do início de sua operação possui um conjunto de regras

preparadas de tal forma que podem ser modificadas durante sua operação. Isto implica na possibilidade de que esse conjunto de regras cresça indefinidamente, ou seja, sua estrutura pode ser *potencialmente infinita*, mesmo que a cada "passo" da operação a estrutura se mantenha finita.

Nas seções 1.4 e 1.5 vamos fazer uma resenha de seis de tais dispositivos que identificam a classe $\mathcal{RE}$ das linguagens recursivamente enumeráveis.

Por um lado, vamos considerar as seguintes máquinas de estado com estrutura dinâmica: *Autômato de Pilha Variante no Tempo* (*TVPDA* pelas suas siglas em inglês *Time-Varying Pushdown Automata*[6]), *Autômato Finito Auto-Modificável* (*SMFA* pelas suas siglas em inglês *Self-Modifiying Finite Automata*[7]), e *Autômato Adaptativo* (*AA* pelas suas siglas em inglês *Adaptive Automata*[8]).

Por outro lado, vamos considerar os seguintes dispositivos gramaticais com estrutura dinâmica: *Gramáticas-W* (em inglês *W-grammars*[10]; vamos denotá-las W-G nesta secão), *Gramáticas Adaptáveis Recursivas* (*RAG* pelas suas siglas em inglês *Recursive Adaptable Grammars*[11]), e *Gramaticas-§* (em inglês: §-*grammars*; nesta seção usaremos o símbolo § para denotá-las[12]).

A seguinte tabela compara estes dispositivos com estrutura dinâmica do ponto de vista de possível crescimento ilimitado do número de regras que os definem.

	TVPDA	*SMFA*	*AA*	*G-W*	*RAG*	§
Núcleo inicial finito	*Sim*	Sim	Sim	Sim	Sim	Sim
Estrutura potencialmente ilimitada	Não	Sim	Sim	Sim	Sim	Sim

Portanto existe uma máquina de estado, com poder de máquina de Turing, que possui estrutura dinâmica e cujo conjunto de regras não cresce indefinidamente. Pode-se então colocar a seguinte pergunta:

> *Existem dispositivos gramaticais com estrutura dinâmica e poder computacional de máquina de Turing que operem com um conjunto limitado de não-terminais e produções?*

Essa pergunta pode ser respondida da seguinte forma:

1. Sim, existem; no capítulo 2 deste trabalho vamos estudar os projetados, entre 1965 e 1970, por Ábrahám[13], Rozenkrantz[14], Salomaa[15], e Ibarra[16] (outros dispositivos similares podem ser consultados em Salomaa[2], Dassow[3], [4] e no volume 2 de Rozenberg[17]):

2. Todos eles são baseados em gramáticas livres de contexto:

(a) Gramáticas com linguagem de controle regular (GR),

(b) Gramáticas Matriciais (GM),

(c) Gramáticas Programadas (GP),

(d) Gramáticas Periodicamente Variantes no Tempo (GT).

Comparamos a seguir esses dispositivos gramaticais com estrutura dinâmica com as gramáticas adaptativas como definidas em Iwai[1]:

	GR	*GM*	*GP*	*GT*	*GA*
Núcleo inicial finito	*Sim*	Sim	Sim	Sim	Sim
Estrutura potencialmente ilimitada	Não	Não	Não	Não	Sim

Como as gramáticas adaptativas possuem produções dependentes de contexto e podem incrementar, sem limite, a sua quantidade de produções e não-terminais, podemos colocar as seguintes perguntas:

> *Qual o poder computacional das gramáticas adaptativas baseadas em núcleos livres de contexto?*
>
> *Será que as gramáticas adaptativas baseadas em núcleos livres de contexto podem operar com conjuntos de não-terminais e produções que não crescem indefinidamente?*

Vale mencionar que a colocação dessas perguntas foi, pela sua vez, motivada por uma procura por uma simplificação da notação para o formalismo das gramáticas adaptativas, de modo que fosse acessível a um maior número de pessoas, e que não aconteça com elas – por culpa da notação complexa – o que aconteceu, por exemplo, com o poderoso formalismo das gramáticas-W o qual, embora permita especificar completamente linguagens de programação, foi usado poucas vezes para tal fim[10].

Essas perguntas motivaram o trabalho desenvolvido nesta tese e são respondidas afirmativamente pelo resultado central deste trabalho, que enunciamos a seguir:

Teorema 1.2.1 *As gramáticas livres de contexto adaptativas com verificação de aparência*[2] *geram a classe das linguagens recursivamente enumeráveis. Mais ainda, basta usar funções adaptativas posteriores, sem geradores, nem variáveis, nem ações adaptativas iniciais nem ações adaptativas finais. O único parâmetro requerido é uma produção e ele pode ser passado implicitamente usando uma notação que é compatível com a notação dos Autômatos Adaptativos[8] e com a notação das gramáticas adaptativas[1].*

[2] a "verificação de aparência" é definida no capítulo 2

Esse resultado é o resumo de nosso estudo de dispositivos gramaticais adaptativos baseados em gramáticas livres de contexto: no capítulo 3 desta tese provamos que as gramáticas livres de contexto adaptativas com verificação de aparência são equivalentes a todos os dispositivos gramaticais apresentados no capítulo 2, no sentido que podem simulá-los e podem ser simulados por eles.

1.3 Dispositivos que identificam a classe $\mathcal{RE}$

Nas duas seções seguintes vamos descrever vários dispositivos que identificam a classe das linguagens Recursivamente Enumeráveis e que compartilham a característica de possuir uma estrutura dinâmica no sentido que eles alteram as regras de sua definição inicial durante sua execução; em cada caso, damos uma breve descrição do dispositivo e, para ilustrar os conceitos introduzidos, apresentamos um exemplo que os exercita. Iniciamos nossa exposição com dispositivos dinâmicos baseados em máquinas de estados e, a seguir, apresentamos dispositivos dinâmicos baseados em gramáticas.

Na seção 1.4 vamos resenhar as seguintes máquinas de estado com estrutura dinâmica: *Autômato de Pilha Variante no Tempo TVPDA*[6], *Autômato Finito Auto-Modificável SMFA*[11], e *Autômato Adaptativo AA*[8]. Devemos advertir que os *autômatos finitos auto-modificáveis r-DFA*[21], apresentados na subseção 1.4.4, não identificam a classe $\mathcal{RE}$ das linguagens recursivamente enumeráveis mas foram incluídos na seção 1.4 por serem um desenvolvimento recente, de 2002, compatível com os estudados nesta seção.

Na seção 1.5 vamos resenhar os seguintes dispositivos gramaticais com estrutura dinâmica: *Gramáticas-W*[10], *Gramáticas Adaptáveis Recursivas*, *RAG*[11], e *Gramaticas-*§[12].

1.4 Máquinas de estado que identificam a classe $\mathcal{RE}$

Nesta seção apresentamos três dispositivos dinâmicos baseados em máquinas de estados; o primeiro apresenta a interessante característica de alterar seu conjunto de transições segundo uma função definida dos números naturais no conjunto potência do conjunto de suas transições originais alterando assim sua estrutura topológica; o segundo dispositivo é um autômato finito dotado de memória na forma de regis--tradores e que possui ações de modificação que lhe permitem criar ou apagar dinamicamente – e de forma controlada– estados e transições; o terceiro é um autômato de pilha dotado de funções adaptativas que lhe permitem criar ou eliminar dinamicamente estados e transições e ainda procurar por transições que satisfazem um padrão especificado, de modo a usar as informações obtidas como variáveis dentro das funções adaptativas.

1.4.1 Autômatos Variantes no Tempo

Autômatos de Pilha Variantes no Tempo ($TVPDA$ pelas suas siglas em inglês: *Time-Variant Pushdown Automata*[6]) são dispositivos baseados em autômatos de pilha não-determinísticos, dotados de uma função que muda seu conjunto de transições durante o reconhecimento de uma palavra. Quando a função que muda o conjunto de transições é periódica obtém-se a subclasse $PTVPDA$ e quando o autômato de pilha subjacente é determinístico obtém se a subclasse $TVDPDA$. As três classes estão relacionadas pelo seguinte resultado[6]:

$$\mathcal{L}_2 = \mathcal{L}(PTVPDA) \subset \mathcal{L}(TVPDA), \qquad \mathcal{L}(TVDPDA) \subset \mathcal{L}(TVPDA) = \mathcal{L}_0$$

onde $\mathcal{L}_2$ é a classe das linguagens livres de contexto, e onde $\mathcal{L}_0$ é a classe das linguagens recursivamente enumeráveis.

Autômatos Finitos Variantes no Tempo (TVA pelas suas siglas em inglês: *Time-Variant Finite Automata*[18]) são dispositivos baseados em autômatos finitos os quais são dotados de uma função periódica que muda seu conjunto de transições durante o reconhecimento de uma palavra. O poder computacional destes dispositivos inclui todas as linguagens regulares e algumas, mas não todas as linguagens livres de contexto não-regulares. Em Krithivasan[19] é estudada uma versão semelhante deste modelo, chamada Autômato Finito Variante no Tempo Generalizado, na qual são permitidas transições em vazio, o que resulta em uma expansão de seu poder computacional que lhes permite reconhecer linguagens recursivamente enumeráveis.

Exemplo 1.4.1 $TVPDA$ **que aceita a linguagem** $L = \{a^n b^n c^n : n \geq 1\}$**.** *No* $TVPDA$ *apresentado graficamente na Fig. 1.1 a seguir as quádruplas que etiquetam as transições têm o significado seguinte:*

(índice da variação, caractere lido, símbolo a desempilhar, símbolo a empilhar)

em particular, a representação dos símbolos a empilhar ou desempilhar se faz com a convenção de que o topo da pilha fica mais à esquerda.

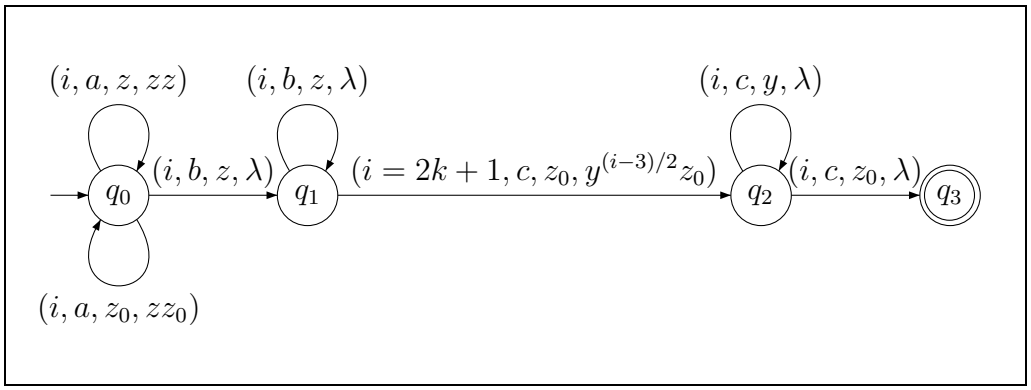


Figura 1.1: $TVPDA$ que reconhece a linguagem $L = \{a^n b^n c^n : n \geq 1\}$.

Vamos discutir como esse $TVPDA$ *reconhece as palavras da linguagem* $L = \{a^n b^n c^n : n \geq 1\}$*. O reconhecimento começa com a pilha contendo o símbolo de fundo de pilha* z_0 *e acaba com a pilha vazia. A estrutura topológica, no que se refere aos estados* q_0 *e* q_1*, está baseada no autômato de pilha clássico, que reconhece a linguagem de duplo balanceamento* $\{a^n b^n : n \geq 1\}$*: para cada símbolo "a" lido é empilhado um símbolo "z", a seguir a quantidade de símbolos "b"que podem ser lidos é controlada usando os símbolos "z" na pilha, impondo a condição de que para cada símbolo "b" lido seja desempilhado um símbolo "z". Já na transição do estado* q_1 *para o estado* q_2*, executada para um valor ímpar do índice, são empilhados tantos símbolos "y" quantos símbolos "c" faltam por ser lidos para completar o triplo balanceamento. A variação desse* $TVPDA$ *é que a transição que empilha os "y" somente existe quando o índice é ímpar. As outras transições existem tanto para valores pares quanto para valores ímpares do índice* i*.*

A seguir apresentamos uma tabela com o reconhecimento da palavra $a^3 b^3 c^3$*.*

i	*Pilha inicial*	*Estado inicial*	*Símbolo lido*	*Estado final*	*Pilha final*
1	z_0	q_0	a	q_0	zz_0
2	zz_0	q_0	a	q_0	zzz_0
3	zzz_0	q_0	a	q_0	$zzzz_0$
4	$zzzz_0$	q_0	b	q_1	zzz_0
5	zzz_0	q_1	b	q_1	zz_0
6	zz_0	q_1	b	q_1	z_0
7	z_0	q_1	c	q_2	yyz_0
8	yyz_0	q_2	c	q_2	yz_0
9	yz_0	q_2	c	q_2	z_0
10	z_0	q_2	λ	q_3	λ

1.4.2 Autômatos Finitos Auto-Modificáveis

Autômatos finitos auto-modificáveis ($SMFA$, pelas suas iniciais em inglês: *Self Modifying Finite Automata*[11]). Projetados por Shutt e Rubinstein em 1993 são autômatos finitos dotados de registradores que representam memória e que podem ter certas restrições impostas para seu uso; $SMFA$ possuem poder computacional de Máquina de Turing[20]. Para explicar os diferentes conceitos referentes ao $SMFA$, vamos detalhar um breve exemplo, apresentado originalmente em Shutt[20].

Exemplo 1.4.2 $SMFA$ **de 2-registros que aceita a linguagem** $L = \{a^n b^n c^n : n \geq 0\}$**.** *No SMFA apresentado graficamente na Fig 1.2 a seguir os 2 registros mencionados são representados pelos meta-nomes* new_b*,* old_b*,* new_c *e* old_c*, com a seguinte semântica: toda vez que uma ação de auto-modificação, que faz referência a* new_b*, é executada deve se criar um novo estado b (nós distinguiremos estes estados criados colocando um subíndice:* b_1*,* b_2*,* b_3*, ...). Vamos mostrar como esse*

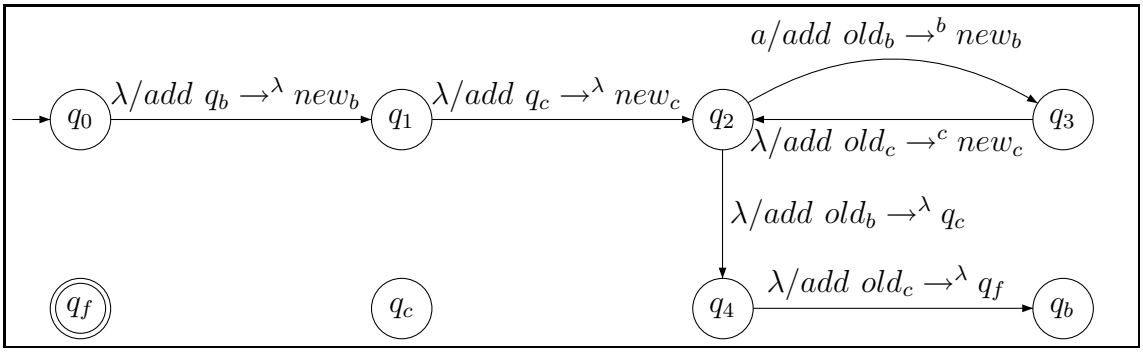


Figura 1.2: O $SMFA$ antes de iniciar o reconhecimento.

$SMFA$ reconhece a cadeia abc. Quando é executada uma ação de auto-modificação, que faz referência a old_b, então essa referência deve ser substituída pelo mais recentemente criado "estado b"; considerações análogas valem para as referências new_c e old_c. Ao percorrer a primeira transição em vazio, a correspondente ação de auto-modificação é executada e é adicionado um novo "estado b" (que ganha o nome b_1) e uma transição correspondente (Fig. 1.3). Agora, ao partir do estado q_1 para o

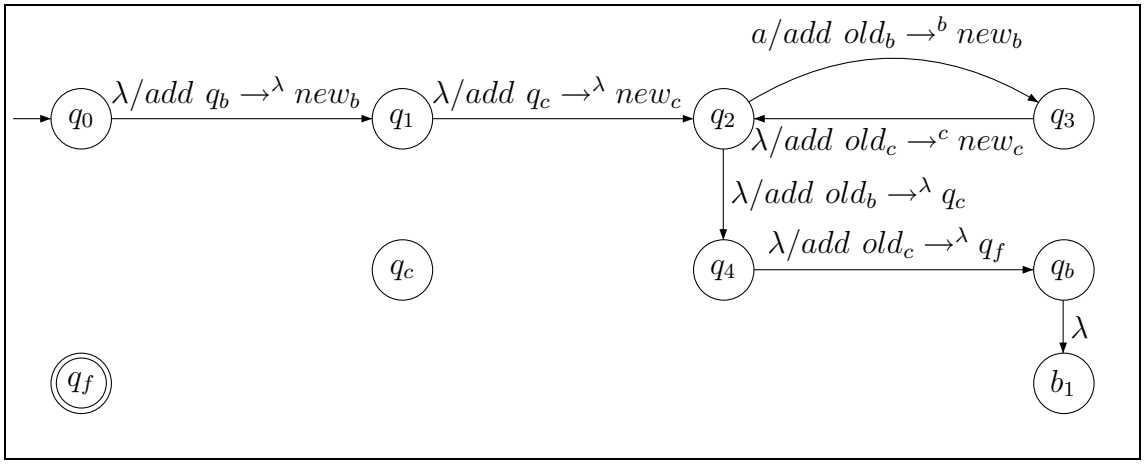


Figura 1.3: $SMFA$ após de transitar do estado q_0 até o estado q_1.

estado q_2, adicionado um novo "estado c" (que ganha o nome c_1) e uma transição correspondente. Agora o controle está no estado q_2, e até agora não introduzimos nenhuma transição que consumisse uma letra. (Fig. 1.4).

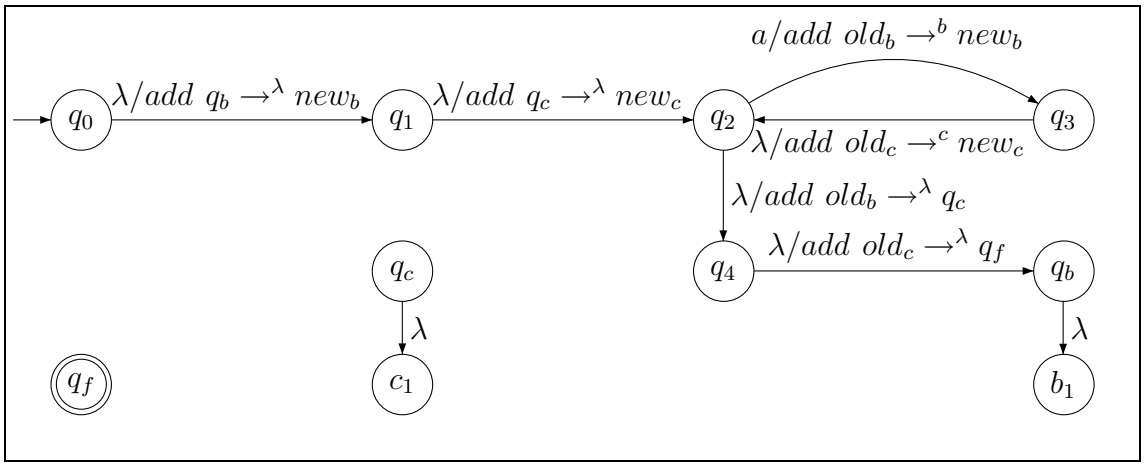


Figura 1.4: $SMFA$ após de transitar do estado q_1 até o estado q_2.

No estado q_2 temos duas transições alternativas; ignoramos a transição em vazio (que não conduz a reconhecimento) e partimos para o estado q_3 criando um novo "estado b" (que ganha nome b_2) e uma transição correspondente (Fig. 1.5). A única

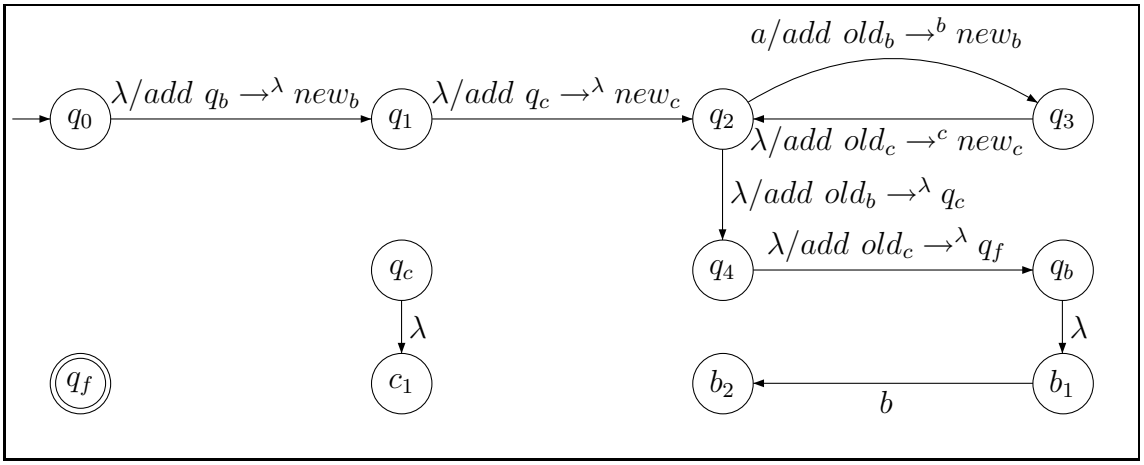


Figura 1.5: $SMFA$ após de transitar do estado q_2 até o estado q_3.

possibilidade agora é voltar ao estado q_2 criando um novo "estado c", chamado c_2, e a transição correspondente. Nestes dois últimos passos temos, agora sim, introduzido transições que consomem símbolos do alfabeto e que, de fato, são as outras duas letras que devem ser consumidas para o reconhecimento da palavra que estiver sendo examinada (Fig. 1.6).

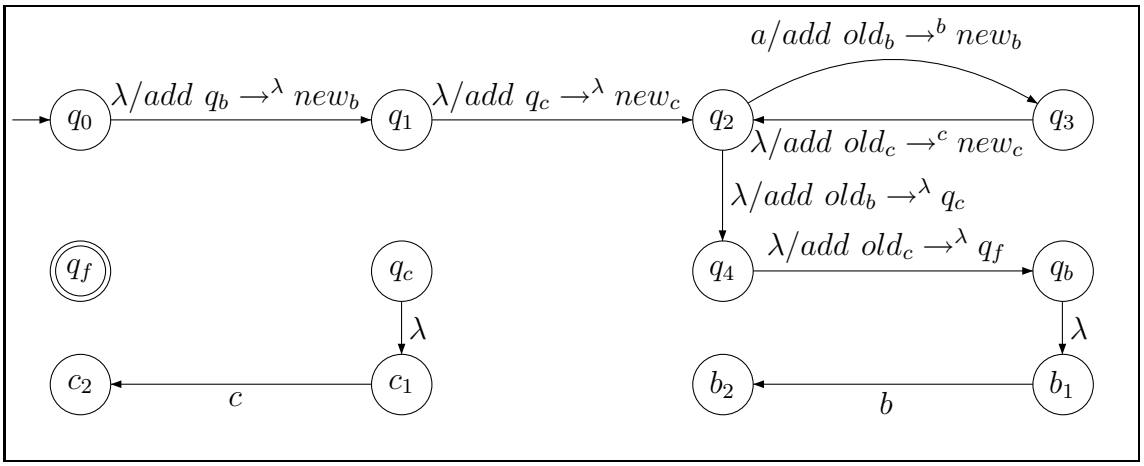


Figura 1.6: $SMFA$ após de transitar do estado q_3 até o estado q_2.

Agora, a transição do estado q_2 ao estado q_4 cria uma nova transição do "estado b" mais recentemente criado - b_2- para o estado q_c: (Fig. 1.7).

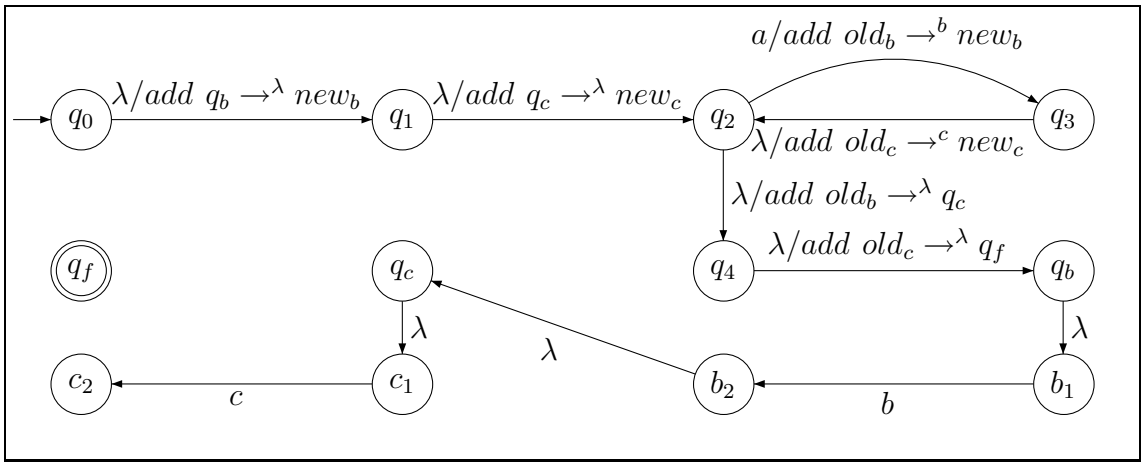


Figura 1.7: $SMFA$ após de transitar do estado q_2 até o estado q_4.

E, agora, a transição do estado q_4 ao estado q_b cria uma nova transição do "estado c" mais recentemente criado - c_2- para o estado final q_f: (Fig. 1.8).

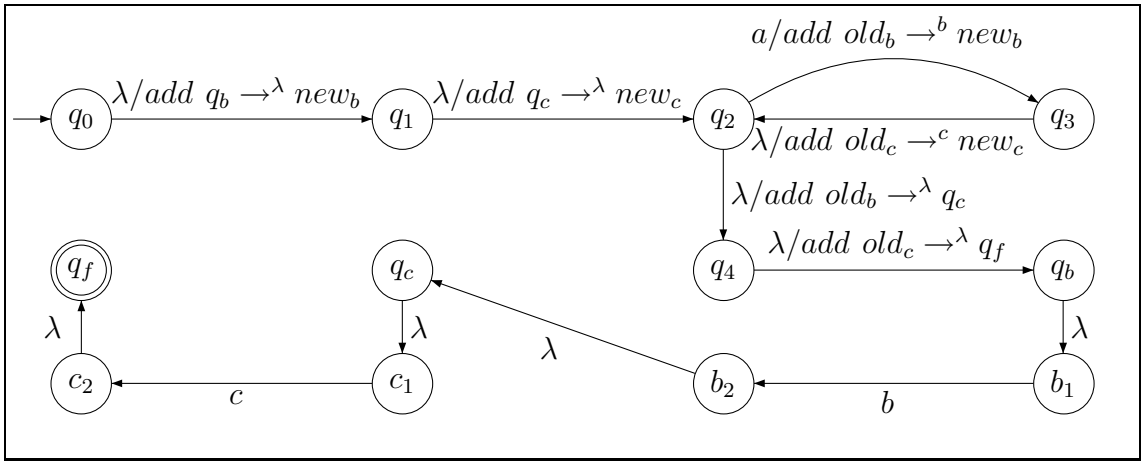


Figura 1.8: $SMFA$ após de reconhecer a cadeia *abc*.

A seqüência de transições ilustrada desde a Fig. 1.3 até a Fig 1.8 conduz ao reconhecimento da cadeia abc.

Finalmente observemos que o $SMFA$ original reconhece a cadeia vazia λ gerando quatro transições vazias, como mostra o gráfico a seguir. (Fig. 1.9).

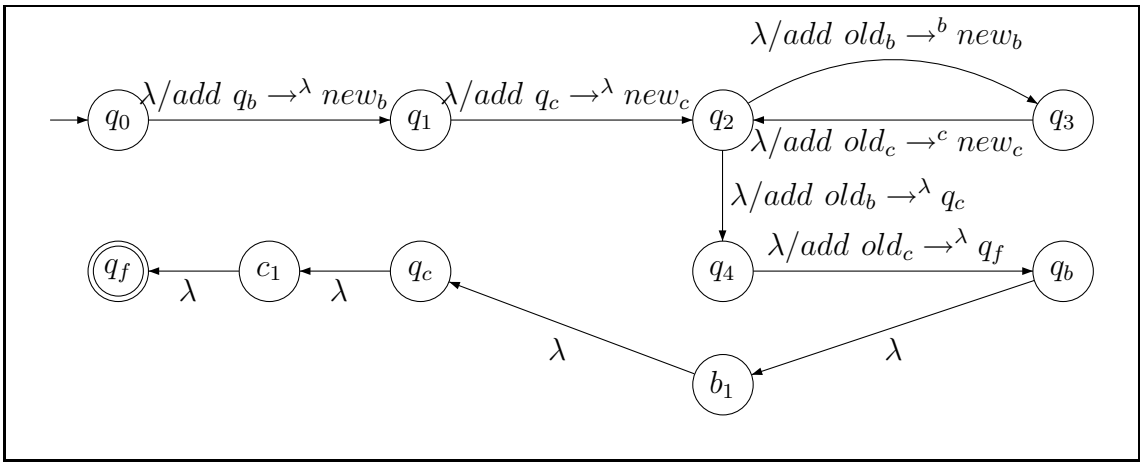


Figura 1.9: $SMFA$ após de reconhecer a cadeia vazia λ.

1.4.3 Autômatos Adaptativos

Autômatos Adaptativos,definidos originalmente em[8], (AA pelas suas inicias em inglês: *Adaptive Automata*[9]) são formados por autômatos de pilha estruturados[8], dotados de um mecanismo de auto-modificação baseado em três primitivas, denominadas busca, inserção e eliminação, que permitem identificar conjuntos de transições que satisfazem um certo padrão, inserir uma nova transição ou eliminar transições que satisfazem um padrão específico. Como no caso dos autômatos auto-modificáveis, vamos nos servir de um exemplo para ilustrar o funcionamento dos Autômatos Adaptativos.

Exemplo 1.4.3 (AA que aceita a linguagem $L = \{a^n b^n c^n : n \geq 1\}$) *No autômato adaptativo S, apresentado na Fig. 1.10 a seguir, a seta dupla etiquetada com o nome do autômato adaptativo, representa uma chamada recursiva a si próprio; o seu funcionamento é governado pelas duas funções adaptativas definidas a seguir:*

$\mathcal{A}_1()\{$
$+[(q_1, b) \rightarrow q_2]$
$-[(q_0, a) \rightarrow, \{\mathcal{A}_1()\}q_1]$
$+[(q_0, a) \rightarrow, \{\mathcal{A}_2()\}q_1]$
$\}$

O efeito desta primeira função adaptativa é o de criar uma transição que consome "b" (para o caso de estarmos reconhecendo a palavra "abc") e trocar ela própria, $\mathcal{A}_1()$, por uma outra função $\mathcal{A}_2()$, a qual, junto com a transição que consome "b", tratarão as outras sentenças da linguagem.

$\mathcal{A}_2()\{$
$q, p*;$
$?\,[(q, b) \rightarrow q_2]$
$-\,[(q, b) \rightarrow q_2]$
$+\,[(q, b) \rightarrow p]$
$+\,[(p, b) \rightarrow q_2]$
$\}$

A função adaptativa $\mathcal{A}_2()$ descobre de qual estado "q" parte uma transição que consome a letra "b" e que chega ao estado q_2; a seguir, é intercalado um estado "p" entre "q" e q_2 e duas transições que consomem a letra "b"; o efeito de tudo isso é que agora temos mais uma letra "b" consumida no caminho que vai desde o estado q_1 até o estado q_2.

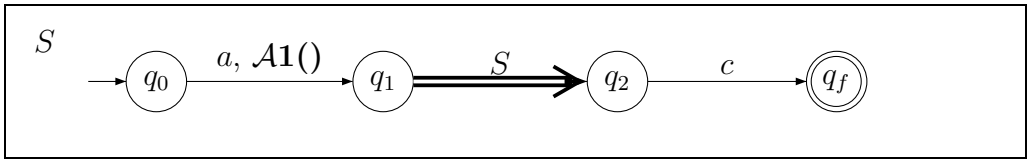


Figura 1.10: O autômato adaptativo S antes do reconhecimento. A pilha está vazia.

Vamos mostrar agora como este autômato adaptativo reconhece a palavra $a^3b^3c^3$. Na Fig. 1.11 vemos como fica o autômato adaptativo depois de ter consumido a primeira letra "a":

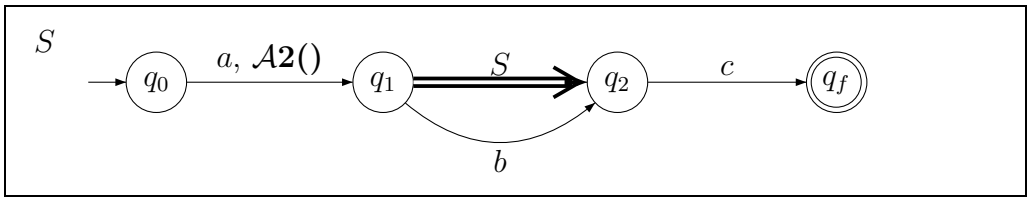


Figura 1.11: O autômato adaptativo S depois do consumo do primeiro símbolo a. Pilha vazia. Controle em q_1.

Nesse ponto o controle se encontra no estado q_1 e o autômato adaptativo faz uma chamada a si mesmo para consumir a segunda letra "a" e o controle volta ao estado q_0, com o estado q_2 empilhado pelo mecanismo de chamada da submáquina S. Na Fig. 1.12, vemos o efeito desta primeira chamada recursiva: o autômato adaptativo foi reconfigurado para aceitar a palavra $a^2b^2c^2$; agora, o autômato adaptativo consome a segunda letra "a" deixando o controle novamente em q_1; mas como ainda existe uma última letra "a", ele deve fazer outra chamada recursiva a si próprio

para consumi-la, voltando o controle a q_0. Novamente, a cada chamada recursiva, empilha-se o estado de retorno q_2.

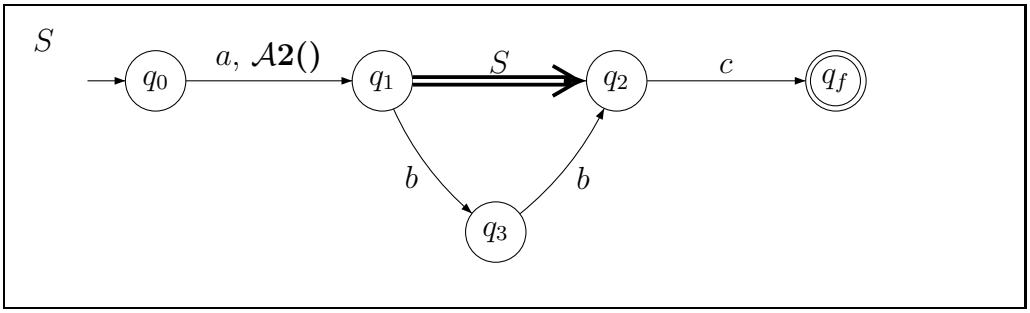


Figura 1.12: O autômato adaptativo S depois do consumo do segundo símbolo a. Pilha: q_2.

O resultado dessa segunda e última chamada recursiva é apresentado na Fig. 1.13 a seguir: o autômato adaptativo conta agora com uma seqüência de transições que lhe permitem consumir as três letras "b", depois de ter consumido as três letras "a"; como é esse exatamente o caso, transita pelos estados q_3, q_4 e q_2 consumindo as três letras "b".

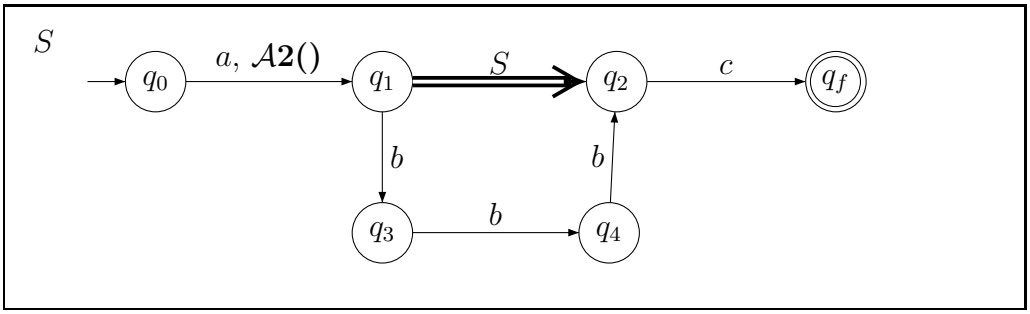


Figura 1.13: O autômato adaptativo S depois do consumo do terceiro símbolo a. Pilha: q_2q_2.

Finalmente o autômato adaptativo consome a primeira letra "c", e atinge o seu estado final q_f, o que conduz a um retorno de chamada de submáquina; esse retorno fará voltar o controle ao estado q_2, onde o autômato adaptativo irá consumir a segunda letra "c", atingindo novamente o estado final e fazendo um outro retorno de chamada de submáquina para compensar a primeira chamada recursiva e, finalmente, consumir a terceira letra "c".

1.4.4 Autômatos Finitos Auto-Montáveis

Autômatos finitos auto-montáveis (denotados $k - FA$, do inglês: *Self-Assembling Finite Automata of degree k*, projetados por Klein e Kutrib em 2002[21] são conjuntos de autômatos finitos $< M_0, \ldots, M_m >$ dotados de "estados de entrada" e "estados de saída" que permitem combiná-los durante o reconhecimento; cada autômato no conjunto é chamado "módulo", os estados de entrada e de saída são chamados "estados de interface" e a quantidade estados de interface é chamada "grau". Um $u : v$ módulo é um módulo que possui grau $u + v$ e tal que os estados $\{r_1, ..., r_u\}$ são de entrada e os estados $\{r_{u+1}, ..., r_{u+v}\}$ são de saída. Um módulo com grau zero é designado como módulo inicial M_0. A montagem dos módulos depende de "transições de montagem" $\delta(q, a)$ que tem como valores ternas da forma:

$$(j, (p_1, ..., p_u), (p_{u+1}, ..., p_{u+v}))$$

e são tais que $\{p_1, ..., p_{u+v}\}$ são diferentes e $q \in \{p_1, ..., p_u\}$; $j \leq m$ indica que a transição deve instanciar e montar o autômato corrente com o $u : v$ módulo M_j. A montagem se realiza da seguinte forma: Uma nova instância do módulo M_j é criada de tal modo que todos os seus estados sejam diferentes dos estados do autômato corrente; os estados $\{p_1, ..., p_u\}$ no autômato corrente são identificados com os estados de entrada da nova instância de M_j e os estados $\{p_{u+1}, ..., p_{u+v}\}$ são identificados com os estados de saída da nova instância de M_j. As classes das linguagens aceitas pelos $k - FA$, não-determinísticos e determinísticos de grau k são denotadas respectivamente, $\mathcal{L}(k - NFA)$, $\mathcal{L}(k - DFA)$. O valor do grau determina o poder computacional do $k - FA$: o aumento do grau determina uma hierarquia infinita de classes de linguagens com inclusão própria, tanto para os $k - FA$ determinísticos quanto para os não-determinísticos e que, além disso, é contida propriamente na classe das linguagens dependentes de contexto, i.e.:

$$\mathcal{L}(k - NFA) \subset \mathcal{L}((k + 1) - NFA) \subset \mathcal{L}_1,$$

$$\mathcal{L}(k - DFA) \subset \mathcal{L}((k + 1) - DFA) \subset \mathcal{L}_1$$

onde $\mathcal{L}_1$ é a classe das linguagens dependentes de contexto. Mas ainda, quando o $k - FA$ não possui laços (denotada com subindice lf pelo termo em inglês "loop-free"), é possível a subclassificação:

$$\mathcal{L}_{lf}(k - NFA) \subset \mathcal{L}_{lf}(k - NFA),$$

$$\mathcal{L}_{lf}(k - DFA) \subset \mathcal{L}_{lf}(k - DFA)$$

onde $\mathcal{L}_{lf}(k - NFA)$, $\mathcal{L}_{lf}(k - DFA)$ denotam as classes das linguagens aceitas pelos $k - FA$, não-determinísticos e determinísticos respectivamente, de grau k que não têm laços. Em particular, a família das linguagens regulares coincide com as famílias $\mathcal{L}(1 - NFA)$, $\mathcal{L}_{lf}(1 - NFA)$, $\mathcal{L}(1 - DFA)$, $\mathcal{L}_{lf}(1 - DFA)$ e, a classe das linguagens livres de contexto é idêntica a $\mathcal{L}_{lf}(2 - NFA)$.

Exemplo 1.4.4 $3 - DFA$ **que aceita a linguagem** $L = \{a^n b^n c^n : n \geq 1\}$**.** *O módulo* M_0 *possui, no estado* q_2*, duas transições de montagem; a transição* $\delta(q_2, a) = (1, (q_2, q_6), (q_4))$ *faz a montagem com o módulo* M_1 *e a transição* $\delta(q_2, b) = (2, (q_2, q_6), (q_4))$ *faz a montagem com o módulo* M_2*.* M_1 *e* M_2 *possuem estados de entrada* $\{r_1, r_2\}$ *e estado de saída* r_3*. Na primeira montagem do módulo* M_1 *seus*

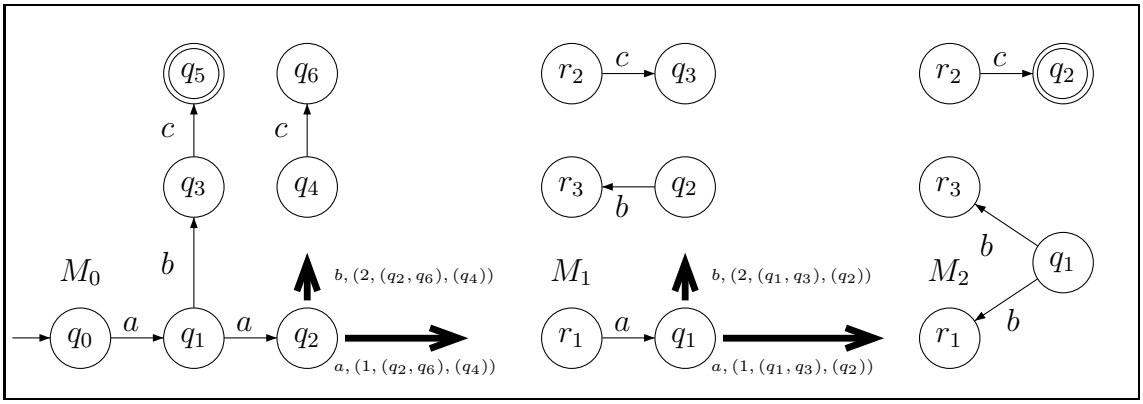


Figura 1.14: Os módulos do 3-DFA que aceita $L = \{a^n b^n c^n : n \geq 1\}$.

estados r_1, r_2 *e* r_3 *são identificados com os estados* q_2*,* q_6 *e* q_4 *respectivamente do módulo* M_0 *e seus outros estados são renomeados; Na segunda montagem do módulo* M_1 *seus estados* r_1, r_2 *e* r_3 *são identificados com os estados* $q_1^{'}$*,* $q_3^{'}$ *e* $q_2^{'}$ *respectivamente do módulo* M_0 *e seus outros estados são novamente renomeados; finalmente na primeira montagem do módulo* M_1 *seus estados* r_1, r_2 *e* r_3 *são identificados com os estados* $q_1^{''}$*,* $q_3^{''}$ *e* $q_2^{''}$ *respectivamente do módulo* M_0 *e seus estados* q_1 *e* q_2 *renomeados para* $q_1^{'''}$ *e* $q_2^{'''}$*.*

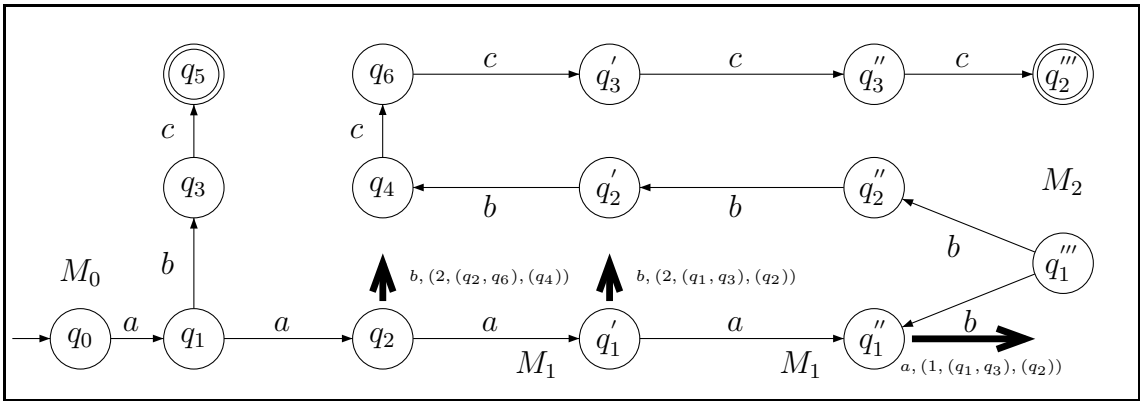


Figura 1.15: O 3-DFA após de aceitar $a^4 b^4 c^4$.

Nesta seção estudamos quatro modelos de máquinas de estados com *estrutura dinâmica*, isto é, que se modifica durante a operação da máquina. Três desses modelos, *autômatos de pilha variantes no tempo TVPDA*, *autômatos auto-modificáveis SMFA*, e *autômatos adaptativos AA*, possuem poder computacional de máquina de Turing; os *autômatos finitos auto-montáveis r-DFA*, identificam uma hierarquia infinita de linguagens estritamente contida na classe das linguagens dependentes de contexto, mas foram incluídos na presente seção por serem um desenvolvimento recente compatível com os outros três.

Desses quatro modelos somente os *autômatos de pilha variantes no tempo TVPDA*, operam sempre com uma quantidade finita de estados, alterando durante sua operação as transições entres esses estados. Os outros três modelos podem criar, sem limite, novos estados durante sua operação mas, na prática, o reconhecimento tem sucesso sempre com uma quantidade finita de estados, embora essa quantidade não seja limitada a priori.

Estamos interessados nessa caraterística: identificar modelos com poder computacional de máquina de Turing que operem com uma quantidade limitada de recursos, fixada a priori. Na seção seguinte vamos estudar três modelos gramaticais, que possuem poder de máquina de Turing mas que podem incrementar, sem limite, a quantidade de produções que os definem. No capítulo 2 estudaremos quatro modelos gramáticais que possuem a caraterística de interese, isto é, que possuem poder de máquina de Turing e que operam com uma quantidade limitada de não-terminais e produções e, no capítulo 3, propomos um modelo gramatical adaptativo, que possue também a caraterística de interese.

1.5 Gramáticas que identificam a classe $\mathcal{RE}$

1.5.1 Gramáticas-W

Gramáticas-W[10], conhecidas também como gramáticas de dois níveis, são um formalismo que permite gerar linguagens recursivamente enumeráveis; assim sendo, elas possuem o poder computacional da Máquina de Turing. Em nossa discussão, nós seguiremos Shutt[11] e Marcotty[22].

Uma gramática-W consiste de dois conjuntos finitos de regras: as metaproduções e as hiper-regras. As metaproduções são produções livres de contexto, com o sinal "::" separando seu lado esquerdo do direito, ";" separarando alternativas no lado direito, e "." indicando o final do lado direito; os não-terminais das metaproduções são denominadas meta-noções, e são escritos em letras maiúsculas; os terminais das metaproduções são chamados proto-noções, e são escritos em letras minúsculas, com brancos permitidos por exemplo:

[MP1] ALPHA :: a; b; c; d; e; f; g; h; i; j; k; l; m; n; o; p; q; r; s; t; u; v; w; x; y; z .
[MP2] NOTION :: ALPHA; NOTION ALPHA .

As hiper-regras são protótipos para produções livres de contexto nas quais encontramos meta-noções definidas em alguma metaprodução; para as hiper-regras o sepa-rador entre os seus lados esquerdo e direito é ":", o separador de alternativas no lado direito é ";", o símbolo "," é usado para separar diferentes proto-noções dentro da mesma alternativa e "." tem a mesma função que para as metaproduções; por exemplo:

[HR1] NOTION sequence : NOTION ; NOTION sequence, NOTION.

essas meta-noções podem serem substituídas por alguma palavra que elas definem, segundo alguma metaprodução, de acordo com a *regra de substituição uniforme*[3]: *Todas as ocorrências da mesma meta-noção na hiper-regra devem ser substituídas obrigatoriamente pela mesma proto-noção*; deste modo pode-se obter um novo conjunto de produções "em tempo de derivação". Assim, por exemplo, usando as metaproduções [**MP1**] e [**MP2**] para derivar as palavras "value" e "identifier", podemos obter, da hiper-regra [**HR1**], as seguintes duas produções livres de contexto:

value sequence : value ; value sequence, value .

[3] às vezes chamam isto de "substituição consistente"

identifier sequence : identifier ; identifier sequence, identifier .

Para concluir nossos comentários sobre gramáticas-W incluímos como exemplo uma gramática-W que gera uma linguagem dependente de contexto.

Exemplo 1.5.1 Gramática-W que gera a linguagem $L = \{a^n b^n c^n : n \geq 1\}$**.** *Neste exemplo apresentamos uma gramática-W com duas metaproduções e três hiperregras que gera a linguagem de triplo-balanceamento.*

[MP1] N :: n; Nn .

[MP2] L :: a; b; c .

[HR1] S : Na, Nb, Nc .

[HR2] nL : L .

[HR3] nNL : L, NL .

Vamos ver como derivar a palavra $a^3b^3c^3$*, segundo esta Gramática-W. Com efeito, substituindo na hiper-regra* [**HR2**]*, a meta-noção* **L** *pelas três palavras que ela gera, segundo a meta-produção* [**MP2**] *obtemos as três produções seguintes:*

1. *na : a .*
2. *nb : b .*
3. *nc : c .*

Agora substituímos na hiper-regra [**HR1**] *a meta-noção* **N** *pelas palavras que ela gera segundo a meta-produção* [**MP1**]*, obtemos o seguinte conjunto infinito de produções:*

4. *S : na, nb, nc .*
5. *S : nna, nnb, nnc .*
6. *S : nnna, nnnb, nnnc .*

...

Agora substituímos na hiper-regra [**HR3**] *as meta-noções* **N** *e* **L***, definidas nas meta-produções* [**MP1**] *e* [**MP2**]*, e obtemos novamente um conjunto infinito de produções:*

7. *nna : a, na .*
8. *nnna : a, nna .*
9. *...*
10. *nnb : b, nb .*
11. *nnnb : b, nnb .*
12. *...*
13. *nnc : c, nc .*
14. *nnnc : c, nnc .*
15. *...*

E, finalmente, derivamos a palavra segundo as regras 6, 8, 11, 14, 7, 10, 13, 1, 2 e 3 da seguinte forma:

$$S \Rightarrow_6 nnnannnbnnnc \Rightarrow_8 annannnbnnnc \Rightarrow_{11} annabnnbnnnc \Rightarrow_{14} annabnnbcnnc$$

$$\Rightarrow_7 aanabnnbcnnc \Rightarrow_{10} aanabbnbcnnc \Rightarrow_{13} aanabbnbccnc$$

$$\Rightarrow_1 aaabbnbccnc \Rightarrow_2 aaabbbccnc \Rightarrow_3 aaabbbccc.$$

1.5.2 Gramáticas Adaptáveis Recursivas de Shutt

Este formalismo gramatical não distingue terminais, não terminais, meta-sintaxe, nem atributos; ao invés disso, cada gramática adaptável recursiva (**RAG** pelas suas iniciais em inglês: *Recursive Adaptable Grammar*), possui um único domínio chamado Álgebra de Respostas, cujos valores (as respostas) podem atuar como sintaxe, meta-sintaxe ou semântica, dependendo de como são usados. Neste resumo seguimos informalmente Shutt[11].

Definição 1.5.1 (Regras livres) *Uma regra livre r sobre V é uma construção da forma:*

$$< v_0, e_0 > \rightarrow t_0 < e_1, v_1 > t_1 ... < e_{n-1}, v_{n-1} > t_{n-1} < en, v_n > t_n$$

onde $n \geq 0$, os v_k são variáveis distintas, os t_k são respostas (usualmente terminais, mas nem sempre) e os e_k são polinômios sobre as variáveis $v_0, ..., v_n$. O domínio de todas as regras livres sobre V é denotado R_V.

Definição 1.5.2 (Pares livres) *Os pares livres são os pares ordenados de polinômios em uma regra livre,* $< v_0, e_0 >$ *e todos os* $< e_k, v_k >$*. O primeiro elemento de um par livre tem papel meta-sintático, enquanto o segundo tem papel semântico. Os* t_k *no lado direito (as respostas fora de pares livres) de uma regra atuam como sintaxe, no sentido que se convertem em elementos da palavra (usualmente formada por terminais) que é gerada pela gramática.*

Definição 1.5.3 (Interpretação semântica) *Numa derivação* $< a, c > \rightarrow b$*, com respostas em* $a, b, c \in A_G$*, diz-se que o valor meta sintático a sintetiza (ou computa) o valor semântico c quando gera o valor sintático b ou, mais simplesmente, que associa o valor semântico c ao valor sintático b.*

Definição 1.5.4 (Regras versus Respostas) *O domínio RG das regras livres está relacionado ao domínio das respostas* A_G *através de uma função regra:*

$$\rho_G : A_G \rightarrow 2^{R_G}$$

Ao ligar as variáveis livres de uma regra livre como r anteriormente definida, a variável mais à esquerda v_0 *deve ser ligada a alguma resposta* $a \in A_G$ *tal que* $r \in \rho_G(a)$*.*

Definição 1.5.5 (Regras ligadas) *Seja x uma resposta da álgebra de respostas da RAG G:* $x \in A_G$*. O conjunto de regras ligadas, denotado* $\beta_G(x)$*, é construído fazendo todas as possíveis ligações de respostas a variáveis nas regras livres de* $\rho_G(x)$ *com a única restrição que* v_0*, em cada regra livre, deve ser ligado a x.*

Pela quantidade de detalhes na definição deste formalismo vamos usar o recurso de examinar seus conceitos à luz do seguinte exemplo.

Exemplo 1.5.2 *Gramática Adaptável Recursiva para a linguagem* $L = \{www : w \in \Sigma^*\}$*, onde* Σ *é um alfabeto finito.*

$$\begin{array}{lcl} S & : & < v_0, v_3 > \rightarrow < W, v_1 >< v_1, v_2 >< v_1, v_3 > \\ W & : & < v_0, \lambda > \rightarrow \lambda \\ & & < v_0, v_1 v_2 > \rightarrow < v_0, v_1 >< C, v_2 > \\ C & : & \forall z \in \Sigma, < v_0, z > \rightarrow z \end{array}$$

Os terminais são λ *(a palavra vazia) e os elementos de* Σ*; os não terminais são S (o símbolo inicial), W e C. O não-terminal C gera todas e cada letra z de* Σ*, e*

associa a ela o valor semântico z; para ver isto, consideremos o alfabeto $\Sigma = \{a, b\}$. *O conjunto de regras livres de C é:*

$$\rho_G(C) = \{< v_0, a > \rightarrow a, < v_0, b > \rightarrow b\}$$

e como é necessário fazer a ligação $v_0 = C$, *para as regras ligadas, então concluímos que* $\beta_G(C)$ *consiste das seguintes regras de produção:*

$$\beta_G(C) = \{< C, a > \rightarrow a, < C, b > \rightarrow b\}$$

que permitem derivar as letras do alfabeto Σ. *Por sua vez, o não-terminal W gera qualquer palavra* $w \in \Sigma^*$, *baseando-se no efeito de C, e associa a ela o valor semântico w; com efeito, continuando com nossa ilustração podemos ver que o conjunto de regras livres:*

$$\rho_G(W) = \{< v_0, \lambda > \rightarrow \lambda, < v_0, v_1v_2 > \rightarrow < v_0, v_1 >< C, v_2 >\}$$

com a ligação da variável $v_0 = W$, *nos fornece uma regra ligada:*

$$< W, \lambda > \rightarrow \lambda$$

e uma regra livre:

$$< W, v_1v_2 > \rightarrow < W, v_1 >< C, v_2 >$$

Agora é necessário determinar quais as ligações adequadas para v_1 *e* v_2. *Essa última regra somente pode ser reduzida a uma resposta se* $< C, v_2 >$ *aparece no lado esquerdo de uma regra em* $\beta_G(C)$. *O únicos lados esquerdos de regras em* $\beta_G(C)$ *são* $< C, a >$ *e* $< C, b >$; *deste modo concluímos que* v_2 *tem que ser ligado a a ou então a b. Assim, a regra anterior produz o seguinte par de regras parcialmente ligadas:*

$$< W, v_1a > \rightarrow < W, v_1 >< C, a >,$$
$$< W, v_1b > \rightarrow < W, v_1 >< C, b > .$$

Finalmente devemos encontrar o conjunto útil de ligações para v_1. *A única restrição é que as ligações devem permitir reduzir a regra a uma resposta. Mas isso acontece justamente quando* $v_1 = wa$ *ou* $v_1 = wb$, *para algum w que permite reduzir a regra a uma resposta. Deste modo, podemos construir o conjunto de todos os valores úteis de* v_1 *por indução: o caso-base é que pode ser ligado a* λ *(já que* λ *é redutível a* λ*), e para cada valor útil w, temos que wa e wb são também úteis.*

Temos assim que o conjunto de regras livres $\rho_G(W)$ *consiste de um conjunto infinito de produções, fazendo a ligação das variáveis* v_1, v_2, *como ilustra a tabela a seguir:*

v_1	v_2	*Produção obtida*	*interpretação semântica*
λ	a	$< W, a > \rightarrow < W, \lambda >< C, a >= \lambda a = a$	a
λ	b	$< W, a > \rightarrow < W, \lambda >< C, b >= \lambda b = b$	b
a	a	$< W, aa > \rightarrow < W, a >< C, a >= aa$	aa
a	b	$< W, ab > \rightarrow < W, a >< C, b >= ab$	ab
b	a	$< W, ba > \rightarrow < W, b >< C, a >= ba$	ba
b	b	$< W, bb > \rightarrow < W, b >< C, b >= bb$	bb
e assim por diante...			

A última linha da tabela faz referência ao fato de que, para continuar ilustrando derivações, são necessários resultados da Álgebra das respostas que escapam ao escopo desta discussão. Finalmente o símbolo inicial S usa W para gerar uma palavra arbitrária $w \in \Sigma^*$, que fica ligada a v_1; por sua vez, as variáveis v_2 e v_3 ficam ligadas a λ, e os outros dois pares restantes da produção de S também geram w pois $r(t) = \{< v_0, \lambda > \rightarrow t\}$ (ou seja, terminais usados como meta-sintaxe geram linguagens unitárias formadas por eles próprios, e sintetizam o valor semântico λ.); assim sendo, a regra livre: $S :< v_0, v_3 > \rightarrow < W, v_1 >< v_1, v_2 >< v_1, v_3 >$, nos fornece a regra livre (referente às variáveis v_1, v_2 e v_3) $< S, v_3 > \rightarrow < W, v_1 >< v_1, v_2 >< v_1, v_3 >$, e, ligando as variáveis v_2 e v_3 a λ, obtemos a regra livre (referente à variável v_1) $< S, \lambda > \rightarrow < W, v_1 >< v_1, \lambda >< v_1, \lambda >$, que gera um conjunto de produções como o listado na tabela a seguir:

v_1	*Produção obtida*	*interpretação semântica*
λ	$< S, \lambda > \rightarrow < W, l >< \lambda, \lambda >< v_1, \lambda >= \lambda$	λ
a	$< S, \lambda > \rightarrow < W, a >< a, \lambda >< a, \lambda >= aaa$	aaa
b	$< S, \lambda > \rightarrow < W, b >< b, \lambda >< b, \lambda >= bbb$	bbb
aa	$< S, \lambda > \rightarrow < W, aa >< aa, \lambda >< aa, \lambda >= aaaaaa$	$aaaaaa$
ab	$< S, \lambda > \rightarrow < W, ab >< ab, \lambda >< ab, \lambda >= ababab$	$ababab$
ba	$< S, \lambda > \rightarrow < W, ba >< ba, \lambda >< ba, \lambda >= bababa$	$bababa$
bb	$< S, \lambda > \rightarrow < W, bb >< bb, \lambda >< bb, \lambda >= bbbbbb$	$bbbbbb$
e	*assim*	*por diante...*
e assim por diante...		

1.5.3 O Meta-cálculo e as Meta-gramáticas de Jackson

Desenvolvido por Quinn Tyler Jackson[12], o cálculo-§[4] é outra infraestrutura para um formalismo gramatical, com poder de máquina de Turing[40]. O cálculo-§ é o cálculo das transformações de uma gramática durante um ou vários reconhecimentos. Uma gramática-§ é um conjunto de funções, produções e predicados que agem em alto nível manipulando estruturas sintáticas em lugar de manipular símbolos de baixo nível. As gramáticas-§ agem como reconhecedores, consumindo texto de entrada e efetuando reduções até chegar ao símbolo inicial (raiz da gramática).

Enquadrando seu trabalho no contexto do discurso de Shutt[11], Jackson define que as gramáticas que podem alterar suas produções de acordo com os dados de entrada são adaptativas ("adaptive" no original). Essa característica das gramáticas-§ pode ser explorada, por exemplo, na remoção da ambigüidade (operação comumente denominada, em inglês, "disambiguation"), que age como uma modificação local nas gramáticas-§, efetuada durante o reconhecimento de uma sentença. As gramáticas-§ podem também realizar mudanças globais e, deste modo, esse formalismo representa uma unificação da taxonomia proposta por Shutt, que dividia as gramáticas "adaptáveis" ("adaptable" no original) em "declarativas", que podiam fazer mudanças locais, e "imperativas", que podiam operar mudanças globais.

O modelo das gramáticas-§ tem evoluído no decorrer do tempo: em Jackson[12], era usado um autômato auto-modificável[7], para responsabilizar-se pelas mudanças locais; em Jackson[23], é apresentado um autômato de pilha "estendido", que possui uma árvore de busca de tipo *trie*[5], para armazenar os terminais coletados; nesse trie a gramática-§ pode efetuar mudanças locais ou globais, que mudam a interpretação semântica de um dado terminal.

É conveniente ressaltar que o trabalho de Jackson sobre gramáticas-§ e o trabalho de Iwai[1] foram desenvolvidos como pesquisas independentes, sem comunicação entre os autores, mas possuem alguns paralelos de nomenclatura e notação; por exemplo, ambos autores denominaram seus modelos de *gramáticas adaptativas* e usaram o símbolo ←, embora com significados diferentes, sendo que Jackson retirou posteriormente esse símbolo da descrição de seu modelo. Por outro lado as gramáticas adaptativas de Iwai são dispositivos geradores enquanto as gramáticas-§ operam como reconhecedores.

Todos esses desenvovimentos teóricos têm dado origem a várias implementações, a última das quais é o *Analisador Adaptativo* Meta-S (em inglês: Adaptive Parser),

[4] O símbolo § se lê "Meta-S"; traduzimos §-*calculus* e §-*grammar* por "Meta-cálculo" e "Meta-gramáticas" e os denotamos cálculo-§ e gramática-§.

[5] Uma árvore para armazenar palavras em que há um nó para cada prefixo comum. As palavras são armazenadas nas folhas do trie.

um ambiente de desenvolvimento para gramáticas-§; o Meta-S foi desenvolvido em C-Builder da Borland, e tem a capacidade de operar como um ambiente de execução para as gramáticas-§ nele desenvolvidas, podendo ainda gerar (no estilo yacc, porém com produto muito mais complexo e completo) o analisador correspondente a uma gramática-§ como um conjunto de classes C++ que podem ser incorporadas em qualquer projeto, em qualquer plataforma que possua um compilador C++[6].

A seguir definimos os conceitos básicos e introduzimos alguns exemplos, extraídos de Jackson[23] e [24], para esclarecer a notação.

Definição 1.5.6 *Uma gramática-§ é uma séptupla* $G = (N, T, C, S, Q, P, \Phi)$ *onde:*

- N *é o conjunto finito de não-terminais,*

- T *é o conjunto finito de terminais,*

- C *é um conjunto de conjuntos (chamados "classes"), potencialmente infinito,*

- $S \in N$ *é o símbolo inicial da gramática-§*

- $Q \subset (N \cup C)$ *é um conjunto distinguido, potencialmente infinito, de predicados (denominados também "qualificadores de terminais")*

- P *é um conjunto de produções da forma* $X \rightarrow Y$ *tal que:*

$$X \in (N \cup C) \text{ e } Y \in (N \cup T \cup C \cup Q \cup \Phi)^*$$

  e Y *pode, através de expressões* ϕ *da forma* $< \alpha >_\beta$ *com* $\alpha \in (N \cup T \cup C \cup \Phi)^*$*, adicionar ou retirar elementos no conjunto* C*; essas operações podem ser globais ou locais.*

- Φ *é um conjunto finito de expressões* ϕ*.*

As expressões ϕ adicionam ou eliminam elementos do conjunto C, e são responsáveis pelas mudanças na gramática.

Definição 1.5.7 (Expressões ϕ) *Uma expressão* ϕ *é da forma* $< \alpha >_{-b!}$ *ou* $\phi(\Delta)_{-b!}$ *onde*

- α *é um padrão de busca, dado por uma expressão regular, por exemplo* $[a-z]^+$

[6]Uma versão de avaliação do Meta-S pode ser obtida para propósitos de pesquisa no endereço eletrônico: qjackson@shaw.ca.

- $\Delta \in N \cup T \cup \{\beta'\}$, *tal como "(" β' ")" onde* β' *é o elemento mais recentemente adicionado ao trie* β

- β *é um "named trie", tal que* $\beta \in C$, *no qual é depositado o resultado da pesquisa de* α.

- *Os símbolos* $-$ *e* ! *são opcionais e significam "retirar" e "local" respectivamente. O comportamento-padrão é adicionar* β *globalmente.*

Definição 1.5.8 (Expressões Ψ) *As expressões* Ψ *comportam-se como predicados e permitem interseção de subgramáticas durante o reconhecimento. Uma expressão* Ψ *é da forma* $\Psi(\alpha)^{\theta}$ *onde*

- α *é um ou vários elementos de* C, *ou então um ou mais terminais*

- $\theta \in Q$

o reconhecimento continua se e somente se θ *é satisfeito, aceitando* α.

Exemplo 1.5.3 *Consideremos a gramática-§ G:*

$S \rightarrow < A >_{N!} \Psi(\text{“(”}\ N\ \text{“)”})^{W}$;

$A \rightarrow w$, *onde* $w \in T^{*}$;

$W \rightarrow$ *“(” '*$[a-z]+$*' “)”;*

A segunda produção aceita uma cadeia arbitrária de carateres; essa cadeia então é testada com o predicado W *que somente aceita cadeias de letras minúsculas entre parêntesis. A idéia é que o terminal aceito por* A *seja posteriormente qualificado por* W; *é por isso que os predicados são chamados também de qualificadores de terminais.*

Exemplo 1.5.4 (Mudanças globais e locais) *Consideremos as produções*

$$A1 \rightarrow < expr >_{x!} \text{ /* mudança local */}$$

$$A2 \rightarrow < expr >_{x} \text{ /* mudança global*/}$$

No caso de $A1$*, qualquer coisa que seja coletada em expr é vinculada "localmente"(no espaço da entrada) ao trie* x *e esse vínculo dura enquanto a produção* $A1$ *está ativa; no caso de* $A2$ *qualquer coisa coletada em expr é vinculada "globalmente"(em tempo de parsing) ao trie* x *e pode ser acessado fora de* $A2$*. Como exemplo de mudança local, consideremos*

$$B \rightarrow A1 \text{ “ : ” } A1$$

$$A1 \rightarrow <' [a - Z]+' >_{x!} \Psi(x)^C \text{ “,” } A1.x$$

$$C \rightarrow' [a - z]+'$$

com texto-fonte "a,a:b,b". A primeira vez que a produção $A1$ *é utilizada, a expressão* $A1.x$ *é vinculada à "a"; na segunda vez que* $A1$ *é utilizada, após a leitura do símbolo ":",* $A1.x$ *é vinculada à "b", sendo que o vínculo anterior é esquecido. Em outras palavras* $A1.x$ *é "local" à* $A1$*, na nomenclatura das gramáticas-*§*, diz-se que* $A1.x$ *"varia no espaço da entrada", pois as mudanças acontecem à medida que o texto-fonte é examinado.*

Como exemplo de mudança global, consideremos

$$B \rightarrow A1 \text{ “ : ” } A2$$

$$A1 \rightarrow <' [a - Z]+' >_{x} \Psi(x)^C \text{ “,” } A1.x$$

$$A2 \rightarrow A1.x$$

$$C \rightarrow' [a - z]+'$$

neste caso, o texto-fonte "a,a:b,b"é rejeitado, pois na primeia vez que a produção $A1$ *é utilizada, a expressão* $A1.x$ *é vinculada globalmente a "a", isto quer dizer que a terceira produção, em qualquer momento posterior do parsing, é interpretada como* $A2 \rightarrow a$*.*

Exemplo 1.5.5 Gramática adaptativa do cálculo-§ que gera a linguagem $L = \{a^n b^n c^n : n \geq 0\}$**.** *Consideremos a gramática-§* $G = (N, T, C, S, Q, P, \Phi)$*:*

- $N = \{S, A, B\}$,
- $T = \{$"a", "b", "c"$\}$,
- $C = \{S.W!, S.X!, S.Y!\}$,
- S, *o símbolo inicial da gramática*
- $Q = \{S.\Psi(WX)^A, S.\Psi(XY)^B\}$,
- $P = \{S \rightarrow < a^+ >_{W!} < b^+ >_{X!} < c^+ >_{Y!} \Psi(WY)^A \Psi(XY)^B, A \rightarrow a[A]b, B \rightarrow b[B]c\}$,
- $\Phi = \{< a^+ >< S.W! >, < b^+ >_{S.X!}, < c^+ >_{S.Y!}\}$.

onde a produção

$$S \rightarrow < a^+ >_{W!} < b^+ >_{X!} < c^+ >_{Y!} \Psi(WX)^A \Psi(XY)^B,$$

significa

1. *Coletar todas as letras "a" e colocar o resultado em* $S.W!$,
2. *Coletar todas as letras "b" e colocar o resultado em* $S.X!$,
3. *Coletar todas as letras "c" e colocar o resultado em* $S.Y!$,
4. *Concatenar* $S.W!$ *e* $S.X!$ *e comparar o resultado com* A,
5. *Se o passo anterior deu igualdade, concatenar* $S.X!$ *e* $S.Y!$ *e comparar o resultado com* B,
6. *Se o passo anterior deu igualdade então foi reconhecido o símbolo inicial* S.

Analisemos a cadeia aaabbbccc usando a gramática-§ G*:*

1. $S.W! = aaa$
2. $S.X! = bbb$
3. $S.Y! = ccc$

4. *Concatenando $S.W!$ com $S.X!$ obtemos $S.W!S.X! = aaabbb$ que satisfaz o predicado A.*

5. *Concatenando $S.X!$ com $S.Y!$ obtemos $S.X!S.Y! = bbbccc$ que satisfaz o predicado B.*

e portanto a gramática-§ G aceita $aaabbbccc$. O número de passos conceituais permanece constante em 6*, enquanto com as gramáticas dependentes de contexto tradicionais, à medida que n cresce também cresce o número de passos nas derivações.*

Nesta seção estudamos três modelos gramaticais com poder de máquina de Turing: *gramáticas-W*, *gramáticas adaptáveis recursivas*, e *gramática-§*; os três modelos podem incrementar, sem limite, durante sua operação, a quantidade de produções e símbolos não-terminais que os definem. Os dois primeiros modelos são dispositivos geradores enquanto o terceiro, opera como dispositivo reconhecedor. Na seguinte seção comparamos esses modelos gramaticais com os modelos de máquinas de estado, estudados na seção anterior, para situar o modelo proposto nesta tese face às caracteristicas mínimas que objetivamos para obter poder de máquina de Turing.

1.6 Poder computacional vs núcleo mínimo

Depois de ter discutido acerca de alguns dispositivos com estrutura dinâmica, baseados em máquinas de estado e em gramáticas, que identificam a classe das linguagens recursivamente enumeráveis, vamos agora salientar os requisitos mínimos, em cada caso, para obter poder computacional máximo.

1.6.1 Em autômatos: núcleo para linguagens de tipo 3

No caso das máquinas de estados finitos, os autômatos finitos auto-modificáveis[7] possuem como núcleo um autômato finito e têm poder de máquina de Turing (Teorema 5.2, pág. 10[25]). O mesmo podemos dizer dos autômatos adaptativos: em Neto[26] foi demostrado que é possível simular um autômato de pilha estruturado usando um autômato finito dotado de funções adaptativas que fazem as construções topológicas necessárias para simular as possíveis chamadas de submáquina do autômato de pilha estruturado, obtendo assim um autômato finito adaptativo equivalente. Como o autômato adaptativo tem como substrato um autômato de pilha estruturado, basta substituir este autômato de pilha estruturado pelo seu autômato finito adaptativo equivalente para ver que, também no caso dos autômatos

adaptativos, basta ter um mecanismo estático para linguagens de tipo 3, para obtermos poder computacional máximo. Esses dois autômatos podem incrementar, sem limite, a sua quantidade de estados e transições.

Em contraposição aos anteriores, os autômatos de pilha variantes no tempo, e os autômatos finitos variantes no tempo geralizados, possuem, núcleo *estático*; isto quer dizer que não são gerados nem estados nem transições durante sua operação; o que acontece é que, sobre um conjunto fixo de estados, as transições variam durante a operação dos dispositivos.

Em ambos os casos, quando a quantidade de estados e transições pode crescer sem limite ou permanece constante, o núcleo mínimo necessário para garantir poder de máquina de Turing é um autômato finito.

Mas não basta adicionar algum tipo de mecanismo de modificação das transições, durante a operação de um autômato finito, para obter um formalismo com poder de máquina de Turing. Como contra-exemplos temos:

1. O *autômato finito variante no tempo* definido em Salomaa[18], que identifica todas as linguagens regulares e algumas (mas não todas) as linguagens livres de contexto.

2. O *autômato finito auto-montável de grau k* definido em Klein[21], que identifica, dependendo do valor de k, uma hierarquia infinita de linguagens, todas elas contidas propriamente na classe das linguagens dependentes de contexto.

1.6.2 Em gramáticas: núcleo para linguagens de tipo 2

Tanto as gramáticas-W[10], quanto as gramáticas adaptáveis recursivas[11] e as gramáticas-§ possuem, como substrato estático, dispositivos gramaticais a gerar linguagens de tipo 2; mas, como no caso dos autômatos auto-modificáveis e dos autômatos adaptativos, que podem incrementar, sem limite, a sua quantidade de estados e transições, essas gramáticas podem também ampliar indefinidamente o número de suas produções (não-terminais e regras).

No capítulo 2 deste trabalho apresentamos quatro formalismos gramaticais dinâmicos baseados em gramáticas livres de contexto que têm o poder computacional da máquina de Turing[15] e que não apresentam a desvantagem de gerar um número ilimitado de produções quando usados para gerar as sentenças de uma linguagem formal; adicionalmente, pode-se provar[2], que usar gramáticas livres de contexto é o requisito mínimo para atingir poder computacional máximo, pois a classe de dispositivos obtidos substituindo-se as gramáticas livres de contexto por gramáticas regulares tem o mesmo poder computacional do que as gramáticas regulares. Os

quatro mecanismos com os quais as gramáticas livres de contexto são controladas são diferentes entre si, mas todos eles são equivalentes, no sentido de que o dispositivo dinâmico obtido identifica a classe das linguagens recursivamente enumeráveis. Assim, nos casos estudados, o núcleo mínimo para obter poder computacional de máquina de Turing é uma gramática de tipo 2.

1.6.3 Nossa opção: Gramáticas Livres de Contexto Adaptativas

No espírito da discussão anterior, nós pesquisamos os requisitos mínimos para que uma gramática adaptativa ainda possua poder computacional máximo; nesta pesquisa, conseguimos provar que, com um projeto cuidadoso das funções adaptativas, basta requerer apenas um conjunto de produções livres de contexto para a gramática adaptativa; mais ainda, as funções adaptativas não requerem variáveis, nem geradores, e também não é requerido gerar novas produções. O único parâmetro requerido é uma produção, mas esse parâmetro pode ser tornado implícito, sobrecarregando a notação das produções numa forma que é compatível com a notação usada em Neto[5] e Iwai[1].

Todas estas caraterísticas representam uma grande simplificação do modelo original introduzido em Iwai[1]; exploramos essa simplificação em uma ferramenta que serve como plataforma para especificação completa de linguagens usando os dispositivos gramaticais estudados nesta tese; essa ferramenta é apresentado no capítulo 4 deste trabalho.

1.7 Estrutura desta tese

A estrutura desta tese é apresentada a seguir.

Capítulo 2 - Gramáticas com mecanismos de controle

Apresenta quatro dispositivos recursivamente enumeráveis baseados em gramáticas livres de contexto: gramáticas livres de contexto matriciais; gramáticas livres de contexto periodicamente variantes no tempo; gramáticas livres de contexto programadas e gramáticas livres de contexto com conjunto de controle regular.

O funcionamento desses dispositivos é ilustrado com diversos exemplos e seu poder computacional é caraterizado através de teoremas que os mostram equivalentes a gramáticas de tipo zero, na hierarquia de Chomsky. O objetivo deste

capítulo é proporcionar um substrato teórico para a proposta de um dispositivo gramátical adaptativo, baseado em gramáticas livres de contexto, que identifica a classe das linguagens recursivamente enumeráveis, apresentado no capítulo 3.

Capítulo 3 - Gramáticas livres de contexto adaptativas

Neste capítulo são apresentadas as gramáticas livres de contexto adaptativas como nossa proposta de dispositivos gramáticais adaptativos, baseados em gramáticas livres de contexto que identificam a classe das linguagens recursivamente enumeráveis. Para fins de comparação, este capítulo começa apresentando, em uma notação mais econômica do que a original, o conceito da gramática adaptativa conforme originalmente desenvolvido, e, a seguir, prova-se que o formalismo proposto nesta tese identifica a classe das linguagens recursivamente enumeráveis demonstrando, em forma construtiva, a equivalência desse formalismo com os quatro formalismos estudados no capítulo 2.

O capítulo termina ilustrando o método das provas mencionadas no paragrafo anterior apresentando, as correspondentes gramáticas livres de contexto adaptativas que simulam as gramáticas controladas dos exemplos do capítulo 2.

Capítulo 4 - Analisadores para gramáticas controladas

Neste capítulo é estudado o problema de analisar linguanges dependentes de contexto geradas pelos dispositivos gramáticais estudados no capítulos 2 e pelas gramáticas livres de contexto adaptativas, estudadas no capítulo 3. Fazendo-se uso da equivalência entres esses cinco dispositvos, provada no capítulo 3, o problema é reducido ao caso de obter analisadores para linguagens geradas por gramáticas livres de contexto com linguagem de controle regular. São apresentados dois analisadores para essa linguagens.

O primeiro é um analisador ascendente, que opera em forma não determinística; esse analisador usa como infraestrutura um autômato pilha[30]; que é obtido por uma especialização do algoritmo *wirth2ape*, que transforma expressões de Wirth em autômatos de pilha estruturados[29]; especificamente a construção aproveita o fato ue o algoritmo *wirth2ape*, quando alimentado com expressões regulares produz como saída um autômato finito. Neste caso a expressão regular é a que controla as derivações das palavras da linguagem gerada pela gramática livre de contexto com linguagem de controle regular. Foram também desenvolvidos dois algoritmos para eliminar transições-λ do autômato finito.

O segundo é uma analisador descendente, determinístico, baseado em um autômato adaptativo mas sua construção está limitada ao caso de gramáticas livres

de contexto com linguagem de controle regular, nas quais as produções são tais que o unico não-terminal que produz uma seqüência de não-terminais é o símbolo inicial da gramática.

Capítulo 5 - Contribuições e temas para futuras pesquisas

Enumera os resultados obtidos com o desenvolvimento desta tese. Este capítulo também identifica trabalhos futuros que podem ser desenvolvidos pelo aprofundamento das pesquisas correspondentes a alguns dos assuntos desenvolvidos nesta tese.

Apêndice A

Apresenta alguns resultados gerais de linguagens formais, que são requesitos para a compreensão do capítulo 2 desta tese.

Referências bibliográficas

Enumera as referências bibliográficas citadas ao longo desta tese.

Capítulo 2

Gramáticas com mecanismos de controle

Neste capítulo estudamos quatro dispositivos gramaticais, com o mesmo poder computacional de máquina de Turing, baseados em gramáticas livres de contexto. Esses dispositivos são denominados: *gramátcas livres de contexto matriciais*[13], *gramátcas livres de contexto periodicamente variantes no tempo*[15], *gramátcas livres de contexto programadas*[14] e *gramátcas livres de contexto com linguagem de controle regular*[50], [51], [52], [49].Em cada caso, o poder computacional das gramáticas livres de contexto é expandido impondo restrições sobre a seqüência de produções que podem ser aplicadas numa derivação e introduzindo uma forma nova de aplicar uma produção. Essas restrições são denominadas, em geral, *mecanismo de controle*, e a nova noção de aplicação é chamada *modo de verificação de aparência*[49]. O modo de verificação de aparência permite fazer "mais um passo" numa derivação sem alterar a forma sentencial sob consideração. Se o mecanismo de controle é um conjunto de matrizes, então as produções numa mesma matriz são aplicadas "ao mesmo tempo". Quando o mecanismo de controle é uma função periódica variante no tempo, então, a cada passo de uma derivação, essa função periódica especifica quais produções são aplicáveis. Se o mecanismo de controle é uma linguagem regular, então a concatenação das produções aplicadas numa derivação deve corresponder a uma palavra dessa linguagem regular. Esses primeiros tipos de gramáticas com mecanismo de controle separam o conjunto P das produções em dois subconjuntos: as que podem ser aplicadas em modo de verificação de aparência, agrupadas no conjunto $F \subset P$, e as que são aplicadas usando "o modo" usual de aplicação de produção nas gramáticas da hierarquia de Chomsky, agrupadas na diferença dos conjuntos P e F, denotada $P \setminus F$. Já no caso das gramáticas programadas, uma mesma produção pode ser aplicada "nos dois modos", e, dependendo de qual modo foi usado, o conjunto de produções disponíveis para o próximo passo fica determinado por uma de duas

funções: *a função sucesso* para o modo das gramáticas na hierarquia de Chomsky e *a função falha* para o modo de verificação de aparência. A maioria das demonstrações dos resultados deste capítulo usa um método construtivo para simular um dispositivo gramatical com outro; incluímos essas demonstrações pois elas serviram como base para as demonstrações apresentadas no próximo capítulo, no qual provamos a equivalência de todos os dispositivos apresentados neste capítulo com as gramáticas livres de contexto adaptativas com verificação de aparência.

Este capítulo segue de perto as seções 3, 4, 5 e 6 do capítulo V, da parte II, do livro *Formal Languages*, escrito por Salomaa[2] e representa uma simplificação sobre o original; Essa simplificação foi obtida assumindo, como hipótese, que podemos *diferençar todas as produções das gramáticas consideradas nos mecanismos de controle*. Isto quer dizer que, mesmo que uma produção apareça duas vezes (em duas matrizes, por exemplo), consideramos cada ocorrência da produção como uma produção diferente; o texto original não faz essa simplificação e assim, em cada demonstração, são incluídas construções adicionais para levar em conta as várias ocorrências de uma mesma produção; essas construções adicionais obscurecem o método central das demonstrações. Acreditamos que nossa simplifcação preserva a essência desse método, mas o leitor interessado em conhecer os resultados em sua forma mais geral deve recorrer à referência citada. Como temos atualizado a notação original, nossa versão dos resultados pode servir como um roteiro para esse fim.

Foram acrescentados exemplos para ilustrar as construções utilizadas para demonstrar os principais resultados apresentados no capítulo. Em particular, desenvolvemos uma notação gráfica para gramáticas com conjunto de controle regular, que foi a base do trabalho que apresentamos na conferência CIAA2003[5], e que utilizamos no capítulo 4 para obter analisadores dependentes de contexto. Algumas demonstrações foram complementadas e outras foram totalmente reescritas. No resumo do capítulo, ressaltamos esses acréscimos e modificações.

2.1 Gramáticas Matriciais

Gramáticas matriciais[13], são baseadas em gramáticas de tipo i, $0 \leq i \leq 3$, segundo a hieraquia de Chomsky, e suas produções são aplicadas em *grupos*; esses grupos são denominados de "matrizes"e as produções numa matriz são aplicadas numa forma sentencial, de esquerda à direita, segundo a ordem em que aparecem na matriz, considerada como "matrizes-linha". Nesta seção definem-se os conceitos básicos das gramáticas matriciais, são apresentados alguns exemplos, define-se uma notação para o caso específico das gramáticas matriciais baseadas em gramáticas livres de contexto, é definido o conceito de *aplicação de produção em modo de verificação de aparência*, que permite aplicar mais um passo numa derivação *quando nenhuma produção é aplicável no sentindo da hierarquia de Chomsky*, e estuda-se uma noção de *derivação mais à esquerda*, que permite caracterizar as linguagens dependentes de contexto que não contém a palavra vazia.

2.1.1 Conceitos básicos

Definição 2.1.1 (Gramática Matricial) *Uma gramática matricial é uma quádrupla* $GM = (N, T, M, S)$ *onde*

1. N *é o alfabeto de símbolos não-terminais,*
2. T *é o alfabeto de símbolos terminais tal que* $N \cap T = \emptyset$*,*
3. $M = \{m_1, m_2, ..., m_n\}$ *é um conjunto finito de matrizes, que são seqüências não vazias* $m_i = [p_{i_1}, ..., p_{i_{k(i)}}]$ *com* $k(i) \geq 1$ *e* $1 \leq i \leq n$*, onde cada* p_{i_j} *para* $1 \leq j \leq k(i)$*, é um par ordenado:*

   $$p_{i_1} = (E, D) \text{ onde } E \in (N \cup T)^* N (N \cup T)^*, D \in (N \cup T)^*.$$

   Esses pares são chamados "produções", e denotados $E \to D$*. Nessas condições, as matrizes podem ser escritas como:*

   $$m_i = [p_{i_1} = E_{i_1} \to D_{i_1}, ..., p_{i_{k(i)}} = E_{i_{k(i)}} \to D_{i_{k(i)}}]$$

4. $S \in N$ *é o símbolo inicial .*

Definição 2.1.2 *Seja* $GM = (N, T, M, S)$ *uma gramática matricial e seja* P *a coleção de todas as produções que figuram nas matrizes de* GM*. Dizemos que* GM *é de tipo* i *na hierarquia de Chomsky com* $i = 0, 1, 2, 3$ *ou "de comprimento crescente", ou "linear", ou "sem produções-*λ*" se e somente se a gramática* $G = (N, T, P, S)$ *goza da propriedade correspondente.*

Definição 2.1.3 (Passo de derivação) *Seja $G = (N, T, M, S)$ uma gramática matricial e $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$; dizemos que X deriva diretamente Y e escrevemos*

$$X \Rightarrow Y$$

se existe um inteiro $k \geq 1$ e cadeias $w_{k+1}, w_i, r_i, s_i \in (N \cup T)^*$, $1 \leq i \leq k$, tais que:

(i) $w_1 = X$ *e* $w_{k+1} = Y$,

(ii) $m = [p_1 = E_1 \rightarrow D_1, ..., p_k = E_k \rightarrow D_k]$ *é uma das matrizes de G, e*

(iii) Para cada $i = 1, \ldots, k$ *tem-se* $w_i = r_i E_i s_i$ *e* $w_{i+1} = r_i D_i s_i$

Nestas condições também dizemos que $X \Rightarrow Y$ é válida com especificação (m, r_1). Diremos também que a matriz m foi aplicada na forma sentencial X e que a notação $X \Rightarrow Y$ simboliza "um passo de derivação".

Notação 2.1.1 *O fechamento reflexivo-transitivo da relação binária $\Rightarrow$ é denotado como $\Rightarrow^*$.*

Definição 2.1.4 *Seja $G = (N, T, M, S)$ uma gramática matricial e $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$. Uma derivação de n passos, desde X até Y, é uma seqüência finita de formas sentenciais $w_i \in (N \cup T)^*$, $0 \leq i \leq n$, tais que*

(i) $w_0 = X$,

(ii) $w_{i-1} \Rightarrow w_i$, *para* $0 \leq i \leq n$, *isto é,* w_{i-1} *deriva diretamente* w_i,

(iii) $w_n = Y$.

Nestas condições, denotamos $X \Rightarrow^n Y$. Assim, a aplicação de uma matriz m sobre a forma sentencial X é uma derivação de um passo.

Definição 2.1.5 *A linguagem gerada pela gramática matricial $G = (N, T, M, S)$, é*

$$L(G) = \{w \in T^* : S \Rightarrow^* w\}.$$

Notação 2.1.2 *Adotamos, para a família das linguagens geradas pelas gramáticas matriciais, a seguinte notação:*

1. *A família das linguagens geradas por gramáticas livres de contexto matriciais sem produções-λ é denotada como $\mathcal{M}$.*

2. *A família das linguagens geradas por gramáticas livres de contexto matriciais é denotada* $\mathcal{M}^{\lambda}$.

Exemplo 2.1.1 *Gramática matricial* G_1 *para a linguagem* $L_1 = \{a^n b^n c^n : n \geq 1\}$. *Consideremos a gramática matricial:*

$$G_1 = (\{S, C\}, \{a, b, c\}, M, S)$$

com

$$M = \{m_1, m_2, m_3\},$$

onde

$$m_1 = [S \to SC], \quad m_2 = [S \to aSb, C \to cC], \quad m_3 = [S \to ab, C \to c].$$

Derivação de aaabbbccc:

$$S \Rightarrow_{m_1} SC \Rightarrow_{m_2} aSbcC \Rightarrow_{m_2} aaSbbccC \Rightarrow_{m_3} aaabbbccc.$$

Vamos provar, por indução matemática, que $L(G) = \{a^n b^n c^n : n \geq 1\}$.

Caso base. n = 1.
Com efeito:

$$S \Rightarrow_{m_1} SC \Rightarrow_{m_3} abc.$$

Hipótese indutiva. *Seja* $n > 1$ *e suponhamos que para todo* k, *com* $1 \leq k < n$, *temos* $a^k b^k c^k \in L(G_1)$.

Com efeito, consideremos $a^n b^n c^n = a^{n-1} a b b^{n-1} c^{n-1} c$. *Pela hipótese indutiva existe uma derivação de* $a^{n-1} b^{n-1} c^{n-1}$, *digamos* $S \Rightarrow^* a^{n-1} b^{n-1} c^{n-1}$; *usamos essa derivação de* $a^{n-1} b^{n-1} c^{n-1}$ *para obtermos uma derivação de* $a^n b^n c^n$; *para tanto observamos que uma derivação segundo a gramática matricial* G_1 *é sempre formada por uma aplicação da matriz* m_1, *seguida de várias aplicações da matriz* m_2 *e, finalmente, de uma aplicação da matriz* m_3; *assim a derivação de* $a^{n-1} b^{n-1} c^{n-1}$ *fornecida pela hipótese indutiva pode ser decomposta da seguinta forma:*

$$S \Rightarrow_{m_1} SC \Rightarrow^*_{m_2} a^{n-2} S b^{n-2} b c^{n-2} C \Rightarrow_{m_3} a^{n-1} b^{n-1} c^{n-1}$$

consideramos apenas a aplicação da matriz m_1 *e as aplicações da matriz* m_2, *incluimos mais uma aplicação da matriz* m_2 *e, finalmente, uma aplicação da matriz* m_1 *para obtermos uma derivação de* $a^n b^n c^n$:

$$S \Rightarrow_{m_1} SC \Rightarrow^*_{m_2} a^{n-2} S b^{n-2} b c^{n-2} C \Rightarrow_{m_2} a^{n-1} S b^{n-1} b c^{n-1} C \Rightarrow_{m_3} a^n b^n c^n$$

Isto completa a indução e termina a prova.

Observação 2.1.1 *Uma pequena modificação do exemplo anterior permite incluir a palavra vazia na linguagem reconhecida pela gramática matricial* G_1*:*

$$G_1' = (\{S, A, C\}, \{a, b, c\}, M, S, \emptyset)$$

com

$$M = \{m_1, m_2, m_3\},$$

onde

$$m_1 = [S \to AC]\,, \quad m_2 = [A \to aA, C \to bCc]\,, \quad m_3 = [A \to \lambda, C \to \lambda]\,.$$

Exemplo 2.1.2 *Gramática matricial* G_2 *para a linguagem* $L_2 = \{ww : w \in \{a, b\}^+\}$. *Consideremos a gramática matricial:*

$$G_2 = (\{S, A, B\}, \{a, b\}, M, S)$$

com

$$M = \{m_1, m_2, m_3, m_4, m_5\},$$

onde

$m_1 = [S \to AB]$, $m_2 = [A \to aA, B \to aB]$, $m_3 = [A \to bA, B \to bB]$,
$m_4 = [A \to a, B \to a]$, $m_5 = [A \to b, B \to b]$.

Derivação de ababbbababbb*:*

$$S \Rightarrow_{m_1} AB \Rightarrow_{m_2} aAaB \Rightarrow_{m_3} abAabB \Rightarrow_{m_2} abaAabaB$$
$$\Rightarrow_{m_3} ababAababB \Rightarrow_{m_3} ababbAababbB \Rightarrow_{m_5} ababbbababbb.$$

Vamos provar, por indução matemática no comprimento $|w|$ *da palavra* w*, que* $L(G_2) = \{ww : w \in \{a, b\}^+\}$.

Caso Base: $|w| = 1$
Neste caso $w = a$ *ou* $w = b$.

$$S \Rightarrow_{m_1} AB \Rightarrow_{m_4} aa$$
$$S \Rightarrow_{m_1} AB \Rightarrow_{m_5} bb$$

Hipótese indutiva*: Seja* $w \in \{a, b\}^+$ *tal que* $|w| = n > 1$ *e suponhamos que para todas as palavras* $u \in \{a, b\}^+$*, com* $|u| < |w|$ *temos que* $u \in L(G)$.

Sem perda de generalidade, podemos supor que $w = ua$ *pois o caso* $w = ub$ *é análogo. Pela sua vez* $u = va$ *ou* $u = vb$. *Em os ambos casos, pela hipótese indutiva, temos:*

(i) $S \Rightarrow_{m_1} AB \Rightarrow^* mAmB \Rightarrow_{m_4} vava = uu,$

(ii) $S \Rightarrow_{m_1} AB \Rightarrow^* mAmB \Rightarrow_{m_5} vbvb = uu.$

De (i) *obtemos*

$$S \Rightarrow_{m_1} AB \Rightarrow^* mAmB \Rightarrow_{m_2} vaAvaB = uAuB \Rightarrow_{m_4} uaua = ww.$$

De (ii) *obtemos*

$$S \Rightarrow_{m_1} AB \Rightarrow^* mAmB \Rightarrow_{m_3} vbAvbB = uAuB \Rightarrow_{m_4} uaua = ww.$$

Isto completa a indução e termina a prova.

2.1.2 Gramáticas Matriciais com verificação de aparência

Definição 2.1.6 (Gramática Matricial com verificação de aparência) *Seja $GM = (N, T, M, S)$ uma gramática matricial. Seja P o conjunto de todas as produções que aparecem nas matrizes de M e seja F um subconjunto de P. Se $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$, dizemos que X deriva diretamente Y em modo de verificação de aparência, e escrevemos*

$$X \Rightarrow^{ac} Y$$

se existir um inteiro $k \geq 1$ e cadeias $w_{k+1}, w_i, r_i, s_i \in (N \cup T)^*$, $1 \leq i \leq k$, tais que:

(i) $w_1 = X$ *e* $w_{k+1} = Y$,

(ii) $m = [p_1 = E_1 \rightarrow D_1, ..., p_k = E_k \rightarrow D_k]$ *é uma das matrizes de* M, *e*

(iii) Para cada $i = 1, \ldots, k$ *tem-se* $w_i = r_i E_i s_i$ *e, neste caso,* $w_{i+1} = r_i D_i s_i$ *ou então o não terminal* E_i *não aparece em* w_i *e a ocorrência da produção* $p_i = E_i \rightarrow D_i$ *pertence a* F, *e, neste caso,* $w_{i+1} = w_i$.

Nestas condições dizemos que $GM = (N, T, M, S, F)$ ou, mais brevemente, $G = (GM, F)$ é uma gramática matricial com verificação de aparência.

Notação 2.1.3 *O fechamento reflexivo-transitivo da relação binária $\Rightarrow^{ac}$ é denotado por $\Rightarrow^{ac^*}$.*

Definição 2.1.7 (Derivação) *Seja $GM = (N, T, M, S)$ uma gramática matricial e $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$; uma derivação de n passos, desde X até Y, é uma seqüência finita de formas sentenciais $w_i \in (N \cup T)^*$, $0 \leq i \leq n$, tais que*

(i) $w_0 = X$,

(ii) $w_{i-1} \Rightarrow^{ac} w_i$, *para* $0 \leq i \leq n$, *isto é,* w_{i-1} *deriva diretamente* w_i *em modo de verificação de aparência,*

(iii) $w_n = Y$.

Nestas condições denotamos $X \Rightarrow^{ac^n} Y$. Assim, a aplicação de uma matriz m sobre a forma sentencial X corresponde a uma derivação de um único passo.

Definição 2.1.8 *A linguagem gerada pela gramática matricial com verificação de aparência $GM = (N, T, M, S, F)$ é*

$$L_{ac}(GM, F) = \{w \in T^* : S \Rightarrow^{ac^*} w\}.$$

Observação 2.1.2 *Para qualquer gramática matricial GM temos:*

$$L(GM) = L_{ac}(GM, \emptyset).$$

Notação 2.1.4 *Introduzimos uma notação para a família das linguagens geradas pelas gramáticas matriciais com verificação de aparência, da seguinte forma:*

1. *A família das linguagens da forma $L_{ac}(GM, F)$ onde GM é uma gramática livre de contexto é denotada $\mathcal{M}_{ac}^{\lambda}$.*

2. *A família das linguagens da forma $L_{ac}(GM, F)$ onde GM é uma gramática livre de contexto sem produções-λ é denotada $\mathcal{M}_{ac}$.*

Proposição 2.1.1

$$\mathcal{M} \subseteq \mathcal{M}^{\lambda} \subseteq \mathcal{M}_{ac}^{\lambda}, \qquad \mathcal{M} \subseteq \mathcal{M}_{ac} \subseteq \mathcal{M}_{ac}^{\lambda}.$$

Prova A primeira inclusão é garantida pela notação 2.1.2; a segunda e a terceira são conseqüência da observação 2.1.2; a quarta inclusão é garantida pela notação 2.1.4.
□

Exemplo 2.1.3 *Gramática matricial com verificação de aparência $G_3 = (GM, F)$ para a linguagem $L = \{a^{2^n} : n \geq 0\}$. Consideremos a gramática matricial com verificação de aparência:*

$$G_3 = (\{S, A, B, C, U\}, \{a\}, M, S, F)$$

com

$$M = \{m_1, m_2, m_3, m_4, m_5\},$$

onde

$$m_1 = [B \to U, A \to U, S \to CC], \quad m_2 = [S \to U, C \to B], \quad m_3 = [C \to U, B \to S],$$
$$m_4 = [B \to U, C \to U, S \to A], \quad m_5 = [S \to U, A \to a].$$

$$F = \{S \to U, A \to U, B \to U, C \to U\}.$$

Derivação de a^{2^3}:
Com efeito:

$$
\begin{array}{lllll}
S \Rightarrow_{m_1} CC & \Rightarrow_{m_2} BC & \Rightarrow_{m_2} BB \\
\Rightarrow_{m_3} BS & \Rightarrow_{m_3} SS & \Rightarrow_{m_1} CCS \\
\Rightarrow_{m_1} CCCC & \Rightarrow_{m_2} CCBC & \Rightarrow_{m_2} BCBC \\
\Rightarrow_{m_2} BCBB & \Rightarrow_{m_2} BBBB & \Rightarrow_{m_3} BBBS \\
\Rightarrow_{m_3} BBSS & \Rightarrow_{m_3} BSSS & \Rightarrow_{m_3} SSSS \\
\Rightarrow_{m_1} CCSSS & \Rightarrow_{m_1} CCCCSS & \Rightarrow_{m_1} CCCCCCS \\
\Rightarrow_{m_1} CCCCCCCC & \Rightarrow_{m_2} CBCCCCCC & \Rightarrow_{m_2} CBCBCCCC \\
\Rightarrow_{m_2} CBCBCBCC & \Rightarrow_{m_2} CBCBCBCB & \Rightarrow_{m_2} CBCBCBBB \\
\Rightarrow_{m_2} CBCBBBBB & \Rightarrow_{m_2} CBBBBBBB & \Rightarrow_{m_2} BBBBBBBB \\
\Rightarrow_{m_3} SBBBBBBB & \Rightarrow_{m_3} SBSBBBBB & \Rightarrow_{m_3} SBSBSBBB \\
\Rightarrow_{m_3} SBSBSBSB & \Rightarrow_{m_3} SSSBSBSB & \Rightarrow_{m_3} SSSSSBSB \\
\Rightarrow_{m_3} SSSSSSSB & \Rightarrow_{m_3} SSSSSSSS & \Rightarrow_{m_4} SSSSSSSA \\
\Rightarrow_{m_4} ASSSSSSA & \Rightarrow_{m_4} AASSSSSA & \Rightarrow_{m_4} AAASSSSA \\
\Rightarrow_{m_4} AAASSSAA & \Rightarrow_{m_4} AAAASSAA & \Rightarrow_{m_4} AAAAASAA \\
\Rightarrow_{m_4} AAAAAAAA & \Rightarrow_{m_5} AAAAAAAa & \Rightarrow_{m_5} AAAAAAaa \\
\Rightarrow_{m_5} AAAAAaaa & \Rightarrow_{m_5} AAAAaaaa & \Rightarrow_{m_5} AAAaaaaa \\
\Rightarrow_{m_5} AAaaaaaa & \Rightarrow_{m_5} Aaaaaaaa & \Rightarrow_{m_5} aaaaaaaa
\end{array}
$$

Vamos provar, por indução matemática, que $L(G_3) = \{a^{2^n} : n \geq 0\}$.

Caso base. $n = 0$.
Com efeito:

$$S \Rightarrow_{m_4} A \Rightarrow_{m_5} a = a^1 = a^{2^0}.$$

Hipótese indutiva. *Seja $n > 0$ e suponhamos que para todo k, com $0 \leq k < n$, temos $a^{2^k} \in L(G_3)$.*

Com efeito, consideremos $S^{2^n} = S^{2^{n-1}+2^{n-1}} = S^{2^{n-1}}.S^{2^{n-1}}$. Pela hipótese indutiva existe uma derivação de $S^{2^{n-1}}$, digamos $S \Rightarrow^* S^{2^{n-1}}$; usamos essa derivação de $S^{2^{n-1}}$ para obtermos uma derivação de $S^{2^{n-1}}$:

$$S \Rightarrow_{m_1} CC \Rightarrow_{m_2} BC \Rightarrow_{m_2} BB \Rightarrow_{m_3} BS \Rightarrow_{m_3} SS \Rightarrow^* AS^{2^{n-1}} \Rightarrow^* S^{2^{n-1}}S^{2^{n-1}} = S^{2^n}.$$

Isto completa a indução e termina a prova.

2.1.3 Gramática Matricial sob derivações mais à esquerda

Definição 2.1.9 (Derivações mais à esquerda, em um passo) *Consideremos uma gramática matricial $G = (N, T, M, S)$. Definimos a relação binária $\Rightarrow_{esq}$ sobre $(N \cup T)^*$ como segue: Se $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$, dizemos que X deriva diretamente Y e escrevemos*

$$X \Rightarrow_{esq} Y$$

se para alguma matriz m e alguma cadeia r_1 sobre $(N \cup T)^*$ tem-se que

(i) $X \Rightarrow_{esq} Y$ *é válida com especificação* (m, r_1),

(ii) Para nenhuma matriz m', *nenhum* $Z \in (N \cup T)^*$ *e nenhum* $r_1' \in (N \cup T)^*$ *tal que* $|r_1'| < |r_1|$, *tem-se que* $X \Rightarrow_{esq} Z$ *é válida com especificação* (m', r_1').

Notação 2.1.5 *O fechamento reflexivo-transitivo da relação binária $\Rightarrow_{esq}$ é denotado como $\Rightarrow^*_{esq}$.*

Definição 2.1.10 (Derivação mais à esquerda) *Seja $G = (N, T, M, S)$ uma gramática matricial sob derivações mais à esquerda e $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$. Uma derivação mais à esquerda de n passos desde X até Y é uma seqüência finita de formas sentenciais $w_i \in (N \cup T)^*$, $0 \leq i \leq n$, tais que:*

(i) $w_0 = X$,

(ii) $w_{i-1} \Rightarrow_{esq} w_i$, *para* $0 \leq i \leq n$, *isto é,* w_{i-1} *deriva diretamente* w_i,

(iii) $w_n = Y$.

Nestas condições, denotamos $X \Rightarrow^n_{esq} Y$. Assim, a aplicação de uma matriz m sobre a forma sentencial X é uma derivação de um passo.

Definição 2.1.11 *A linguagem gerada pela gramática matricial sob derivações mais à esquerda $G = (N, T, M, S)$ é*

$$L_{esq}(G) = \{w \in T^* : S \Rightarrow^*_{esq} w\}.$$

Notação 2.1.6 *Introduzimos uma notação para a família das linguagens geradas pelas gramáticas matriciais sob derivações mais à esquerda, da seguinte forma:*

1. *A família das linguagens da forma $L_{esq}(G)$ onde G é uma gramática livre de contexto é denotada $\mathcal{M}^{\lambda}_{esq}$.*

2. *A família das linguagens da forma $L_{esq}(G)$ onde G é uma gramática livre de contexto sem produções-λ é denotada $\mathcal{M}_{esq}$.*

Exemplo 2.1.4 *Gramática matricial G_3' sob derivações mais à esquerda para a linguagem $L = \{a^{2^n} : n \geq 0\}$. Consideremos a gramática matricial sob derivações mais à esquerda*

$$G_3' = (\{S, X, Y, Z, W, A, B, C\}, \{a\}, M, S)$$

com

$$M = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9\},$$

onde

$$\begin{array}{lll} m_1 = [S \to a], & m_2 = [S \to YA], & m_3 = [X \to ZZ, A \to A], \\ m_4 = [Y \to ZW, A \to B], & m_5 = [Z \to X, B \to B], & m_6 = [W \to Y, B \to A], \\ m_7 = [W \to Y, B \to C], & m_8 = [X \to a, C \to C], & m_9 = [Y \to a, C \to \lambda]. \end{array}$$

Cada derivação consiste de três laços:

(i) Trocar os X, Y por Z e duplicar seu número,

(ii) Trocar os Z por X, Y,

(iii) Trocar os X, Y por a.

Os não-terminais A, B e C indicam qual laço está ativo. Pela definição da relação $\Rightarrow_{esq}$ a matriz m_4 não pode ser aplicada quando a matriz m_3 ainda é aplicável; do mesmo modo, as matrizes m_6 e m_7 não podem ser aplicadas enquanto a matriz m_5 ainda for aplicável. Assim, a linguagem $L = \{a^{2^n} : n \geq 0\}$ pertence à classe $\mathcal{M}_{esq}^{\lambda}$. Pode-se ainda provar que $L \in \mathcal{M}_{esq}^{\lambda}$ trocando os não-terminais A, B e C por subíndices, nos outros não-terminais. Essa última afirmativa também é um caso particular de teorema 2.1.2*, que será estabelecido mais adiante: "toda linguagem dependente de contexto que não contém a palavra vazia pode ser pode ser gerada por uma gramática livre de contexto matricial, sem produções λ, sob derivações mais à esquerda".*

Planejar a demonstração por indução neste caso não é tão intuitivo como no caso que as derivações não são mais à esquerda; é melhor, neste caso, começar desenhando as árvores de derivação de algumas palavras; nos dois diagramas a seguir foram desenhadas árvores de derivação para as palavras a, a^{2^1}, a^{2^2} e a^{2^3}. Para poupar espaço unimos todas as árvores em uma só; nessa única árvore marcou-se com a palavra "proibido" os passos de derivação que não são mais à esquerda; as derivações mais à esquerda foram marcadas com "+ esq.". Usamos a notação m_j^i para indicar que a matriz m_j foi aplicada i vezes.

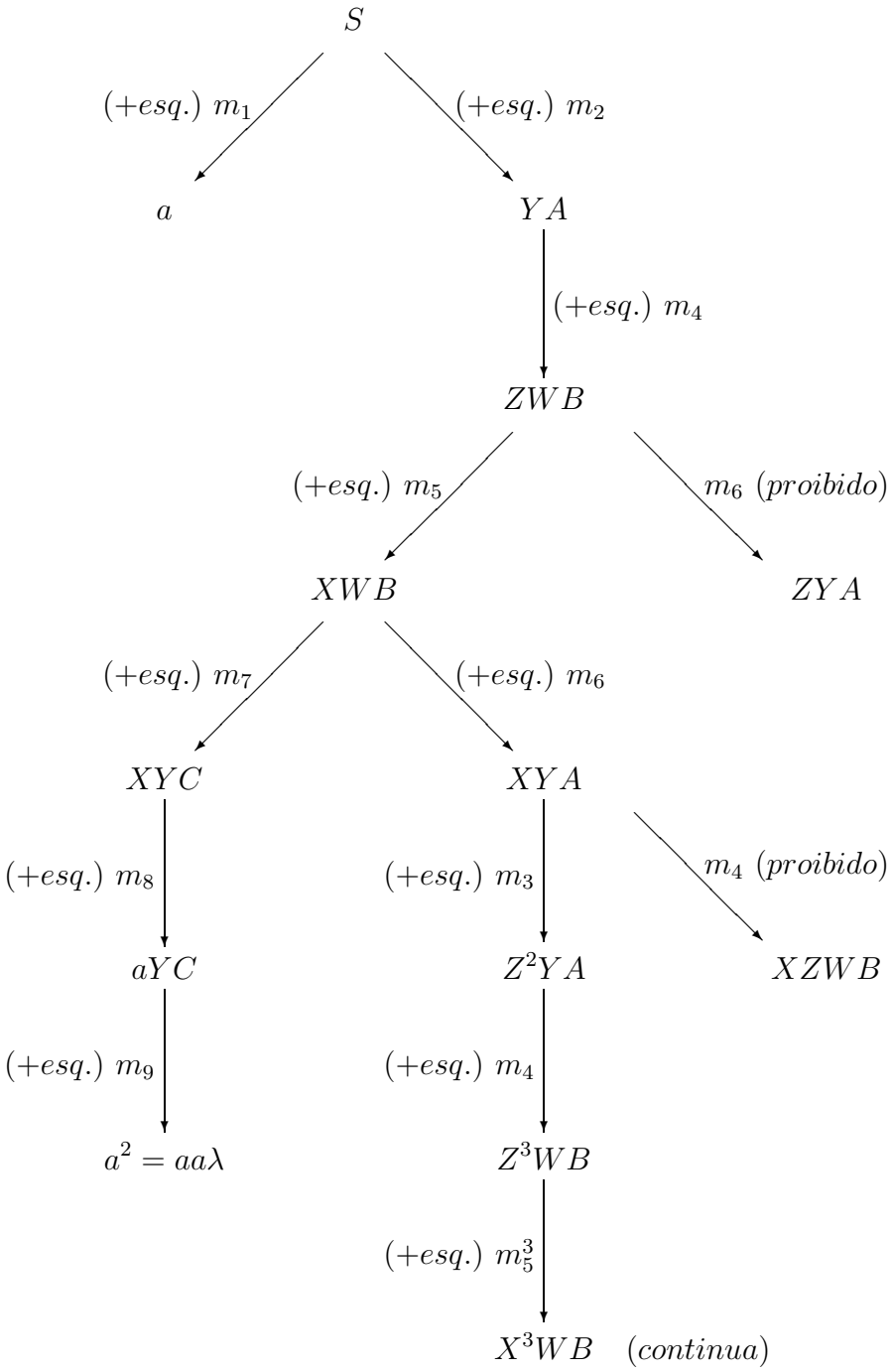


Diagrama 1. Derivações de a e a^2.

(continuação)

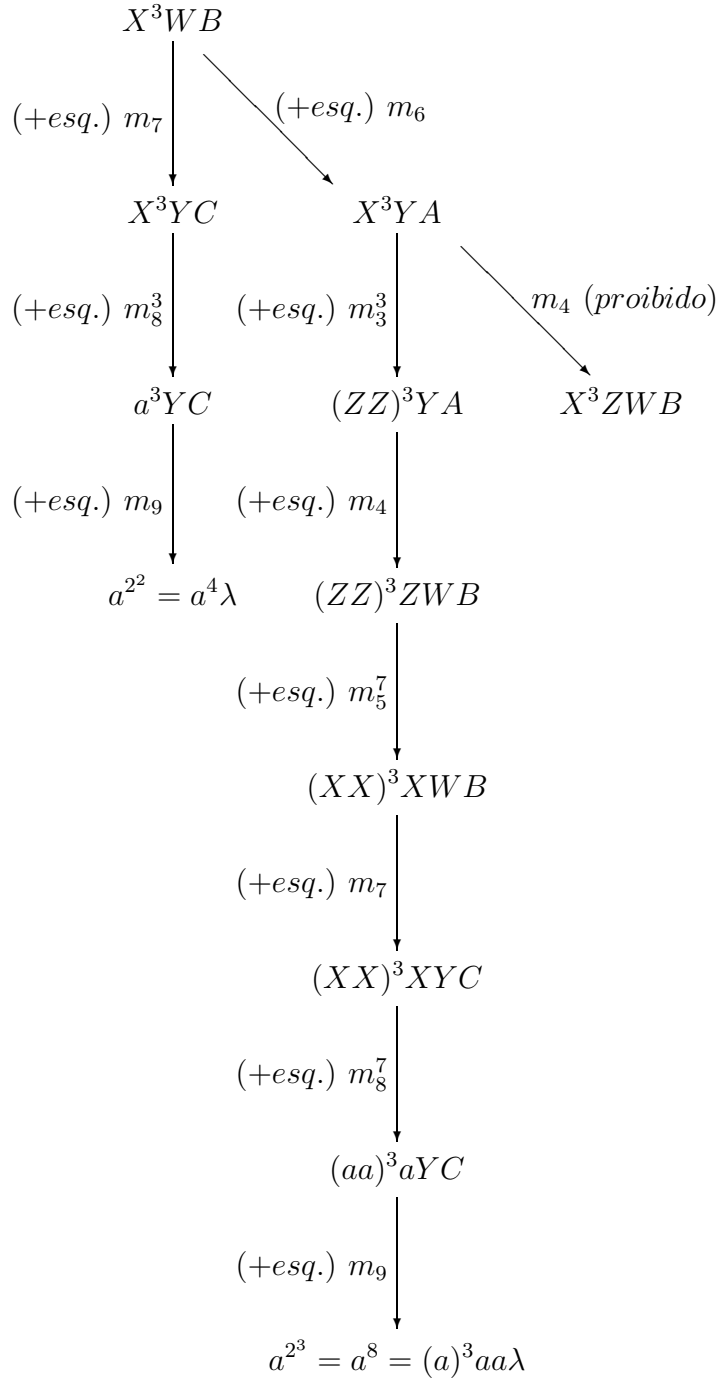


Diagrama 2. Derivações de a^{2^2} e a^{2^3}.

A árvore acima é continuação da árvore da página anterior. Analisando as

árvores de derivação das palavras a^{2^1}, a^{2^2} e a^{2^3} podemos chegar às seguintes conclussões:

1. *todas compartilham os primeiros três passos de derivação quando são aplicadas as matrizes m_2, m_4 e m_5,*
2. *depois de aplicar $2^i - 1$ vezes a matriz m_5 (denotado na árvore como $m_5^{2^i-1}$), pode ser aplicada a matriz m_7 ou a matriz m_6; a primeira leva a uma derivação de a^{2^i}, a segunda a uma derivação de $a^{2^{i+1}}$,*
3. *a parte final da derivação de a^{2^i} pode ser dividida em cinco partes da seguinte forma:*
   (a) *uma aplicação da matriz m_4, que leva à forma sentencial $(ZZ)^{2^{i-1}-1}YA$,*
   (b) *2^i-1 aplicações da matriz m_5, que leva à forma sentencial $(ZZ)^{2^{i-1}-1}ZWB$,*
   (c) *uma aplicação da matriz m_7, que leva à forma sentencial $(XX)^{2^{i-1}-1}XWB$,*
   (d) *2^i-1 aplicações da matriz m_8, que leva à forma sentencial $(aa)^{2^{i-1}-1}aYC$,*
   (e) *uma aplicação da matriz m_9, que leva à palavra $a^{2^n} = (a)^{2^n-1}aa\lambda$.*

Com estas observações a demonstração por indução matemática pode ser esquematizada como segue:

Para demonstrar: A gramática matricial sob derivações mais à esquerda G_3' gera a linguagem $L = \{a^{2^n} : n \geq 0\}$

Caso Base *O caso $n = 0$ e imediato pela aplicação da matriz m_1 e não faz parte do esquema da indução; portanto procedemos com $n = 1$:*

$$S \Rightarrow_{esq\ m_2} YA \Rightarrow_{esq\ m_4} ZWB \Rightarrow_{esq\ m_5} XWB$$

$$\Rightarrow_{esq\ m_7} XYC \Rightarrow_{esq\ m_8} AYC \Rightarrow_{esq\ m_9} a^2 = aa\lambda.$$

Hipótese indutiva *Seja $n > 1$ e suponhamos que $a^{2^n} \in L_{esq}(G_3')$. Vamos provar que $a^{2^{n+1}} \in L_{esq}(G_3')$.*

Com efeito, já que $a^{2^n} \in L_{esq}(G_3')$ então existe uma derivação de a^{2^n} segundo a gramática G_3', digamos $S \Rightarrow_{esq}^* a^{2^n}$. Segundo nossas observações acima, essa derivação pode ser decomposta da seguinte forma

$$S \Rightarrow_{esq}^* (ZZ)^{2^{n-1}-1}YA \Rightarrow_{esq\ m_4} (ZZ)^{2^{n-1}-1}ZWB \Rightarrow_{esq\ m_5^{2^n-1}} (XX)^{2^{n-1}-1}XWB$$

$$\Rightarrow_{esq\ m_7} (XX)^{2^{n-1}-1}YC \Rightarrow_{esq\ m_8^{2^n-1}} (aa)^{2^{n-1}-1}aYC \Rightarrow_{esq\ m_9} a^{2^n} = (a)^{2^n-1}aa\lambda$$

Vamos obter uma derivação de $a^{2^{n+1}}$ segundo G_3' desconsiderando os últimos a^{2^n+2} passos, isto é, desconsiderando as últimas aplicações de m_7, m_8 e m_9. Com efeito:

$$S \Rightarrow_{esq}^{*} (ZZ)^{2^{n-1}-1}YA \Rightarrow_{esq\ m_4} (ZZ)^{2^{n-1}-1}ZWB \Rightarrow_{esq\ m_5^{2^n-1}} (XX)^{2^{n-1}-1}XWB$$

$$(X)^{2^n-1}WB \Rightarrow_{esq\ m_6} (X)^{2^n-1}YA \Rightarrow_{esq\ m_3^{2^n-1}} (ZZ)^{2^n-1}YA \Rightarrow_{esq\ m_4} (ZZ)^{2^n-1}ZWB$$

$$\Rightarrow_{esq\ m_5^{2^n-1}} (XX)^{2^n-1}XWB \Rightarrow_{esq\ m_7} (XX)^{2^n-1}XYC \Rightarrow_{esq\ m_8^{2^n-1}} (aa)^{2^n-1}aYC$$

$$\Rightarrow_{esq\ m_9} a^{2^{n+1}} = (aa)^{2^n-1}aa\lambda.$$

Isto completa a indução e termina a prova.

2.1.4 Caracterização de linguagens através de gramáticas matriciais

Teorema 2.1.1 *Toda linguagem dependente de contexto L que não contém a palavra vazia λ é gerada por uma gramática cujas produções são todas de uma das formas seguintes:*

$$A \to BC, AD \to BC, A \to a$$

onde A, B, C, D são não-terminais e a é um terminal.

Prova. Seja $n \geq 1$ e $G = (N, T, P, S)$ uma gramática; dizemos que G é de grau n se, e somente se, somente ocorrem terminais em produções da forma $A \to a$, onde $A \in N$, $a \in T$ e $\forall p = E \to D \in P$ tem-se $|E| \leq |D| \leq n$. Pelo teorema A.1.2 do apêndice A, podemos assumir que existe um n tal que L é gerado por uma gramática de grau n.

Afirmativa: L é gerada por uma gramática de grau 2.
É suficiente demonstrar que, para toda gramática G de grau $n \geq 3$ existe uma gramática G' de grau $n-1$ equivalente a G: $L(G) = L(G')$.
Seja $G = (N, T, P, S)$ uma gramática de grau $n \geq 3$. Seja $p = E \to D \in P$. Se $|D| \leq 2$, então colocamos $p \in P(G')$. Caso contrário ($|D| > 2$), existem X_1, X_2, X_3, X_4 tais que:

$$E = X_1E', D = X_2, X_3, X_4D'$$

para algumas palavras E', D' (que podem ser iguais à palavra vazia λ). Se $E' = \lambda$ então introduzimos um novo não-terminal Y_1 e incluímos

$$X_1 = X_2Y_1, Y_1 \to X_3X_4D'$$

como produções de $G^{'}$. Se $E = X_1Y_5E^{''}$ para algum não-terminal X_5 e uma palavra $E^{''}$, então introduzimos um novo não terminal Y_2 e incluímos

$$X_5E^{''} \rightarrow Y_2Y_4D^{'}, X_1Y_2 \rightarrow X_2X_3$$

como produções de $G^{'}$.
Esse processo é repetido para cada produção de G. Claramente, a gramática resultante $G^{'}$ é de grau $n-1$, e equivalente a G. Portanto, L é gerado por uma gramática $G = (N, T, P, S)$ de grau 2.
Vamos construir uma gramática $G^{'}$ satisfazendo as condições do teorema 2.1.1. Isto significa que as produções da forma $E \rightarrow D$, onde E, D são não-terminais são eliminadas de G.
Definimos

$$\mathcal{K} = \{w : S \Rightarrow^* w \text{ ou então } w \in T \text{ ou então } w \in N^* \text{ e } |w| \leq 2\}$$

Esse conjunto finito pode ser determinado usando o método do teorema A.1.3 do apêndice A. O conjunto $P^{'}$ consiste de:

(i) Todas as produções $S^{'} \rightarrow w$ tais que $w \in \mathcal{K}$

(ii) Todas as produções $CA \rightarrow CB$, $AC \rightarrow BC$, tais que $C \in N$ e $A \rightarrow B \in P$, onde $A, B \in N$

(iii) Todas as produções $A \rightarrow BC$, $AD \rightarrow BC$, $A \rightarrow a$ em P onde $A, B, C, D \in N$ e $a \in T$.

Portanto $G^{'}$ satisfaz as condições exigidas. Por construção, $G^{'}$ é portanto equivalente a G.
□

Teorema 2.1.2 *Uma linguagem que não contém a palavra vazia λ é dependente de contexto se e somente se ela é gerada por uma gramática livre de contexto matricial sem produções λ, sob derivações mais à esquerda.*

Prova. Denotamos com $\mathcal{L}_1^{'}$ a família das linguagens dependentes de contexto que não contém a palavra vazia λ. Vamos provar que $\mathcal{L}_1^{'} \subset \mathcal{M}_{esq}$. Para a demonstração de $\mathcal{M}_{esq} \subset \mathcal{L}_1^{'}$ remetemos o leitor ao livro de Salomaa[2]. Seja $b \in T$ uma letra, fixa, arbitrária y seja $L \in \mathcal{L}_1^{'}$. Nestas condições, pelo teorema 2.1.1, existe uma gramática dependente de contexto $G = (N, T, M, S)$ tal que $L_{esq}(G) = Lb$ e cujas produções são de uma das três formas seguintes:

$$\begin{array}{ll} A \rightarrow BC, & A, B, C \in N \\ AD \rightarrow BC, & A, B, C, D \in N \\ A \rightarrow a & A \in N, a \in T \end{array}$$

Definimos novos não-terminais

$$\begin{aligned} N_1 &= \{A_1 : A \in N\} \\ N_2 &= \{A_2 : A \in N\} \\ N_0 &= \{S_0, X_1, X_2, X_3, X_4\} \cup \{X_p : p = AD \to BC, \text{ onde } A, B, C, D \in N\} \end{aligned}$$

e, com eles, a gramática matricial:

$$G_1 = (N \cup N_0 \cup N_1 \cup N_2, T, M, S)$$

onde as matrizes são dadas por:

(i) $[S_0 \to SX_1]$

(ii) $[A \to A_1, X_1 \to X_1]$

(iii) $[A \to B_1C_2, X_1 \to X_2]$, $\forall\ p = A \to BC,\ A, B, C \in N$

(iv) $\left.\begin{array}{l}[A \to B_1, X_1 \to X_p] \\ [D \to C_2, X_p \to X_2] \\ [E \to X_4, X_p \to X_4]\end{array}\right\} E \in N, E \neq D, p = AD \to BC, A, B, C, D \in N$

(v) $\left.\begin{array}{l}[B_1 \to B, X_2 \to X_2] \\ [B_2 \to B, X_2 \to X_1]\end{array}\right\} \forall B \in N$

(vi) $\left.\begin{array}{l}[A \to a, X_1 \to X_3] \\ [A \to a, X_3 \to X_3] \\ [A \to a, X_3 \to b] \\ [A \to a, X_1 \to b]\end{array}\right\} \forall p = A \to a \in P, A \in N, a \in T$

Como cada palavra de $L(G)$ possui uma derivação segundo G tal que as produções $A \to a$ são aplicadas por último (isto é produção de outro tipo é aplicada depois da aplicação de uma das produções $A \to a$) e G_1 simula este tipo de derivações obtemos:

$$L_{esq}(G_1) = L(G)b$$

Por outro lado, se L_1 é uma linguagem sobre T que não contém a palavra vazia λ,

$$L_1 = \bigcup_{b \in T} (\partial_b^r L_1)b$$

onde $\partial_b^r L_1 = \{z : bz \in L_1\}$.

Afirmativa:
Seja L_1 é uma linguagem dependente de contexto sobre T que não contém a palavra

vazia λ Se $b \in T$ então $\partial_b^r L_1$ também é dependente de contexto.
Com efeito: Suponha que L_1 é gerada pela gramática dependente de contexto $G' = (N', T', P', S')$. Definimos a nova gramática $G'' = (N' \cup \{S_1', \#\}, T', P' \cup \{S_1' \rightarrow S'\#, \# \rightarrow \lambda\}, S_1')$. Nestas condições $L(G'') = \partial_b^r L_1$ e, pelo teorema A.1.4 do apêndice, a linguagem $\partial_b^r L_1$ é dependente de contexto.

Finalmente, temos dois casos a considerar:

1. $\lambda \notin \partial_b^r L_1$: Neste caso, $b \notin L_1$ e usamos o resultado acima sobre a linguagem L para concluírmos que:
$$(\partial_b^r L_1)b \in \mathcal{M}_{esq}.$$

2. $\lambda \in \partial_b^r L_1$:, denotamos $L_2 = \partial_b^r L_1 \setminus \{\lambda\}$ e temos que $L_2 b \in \mathcal{M}_{esq}$; como M_{esq} é fechada sob união e $(\partial_b^r L_1) = L_2 b \cup \{b\}$ obtemos
$$(\partial_b^r L_1)b \in \mathcal{M}_{esq}.$$

e destes dois casos e do teorema A.1.4 do apêndice, obtemos que $L_1 \in \mathcal{M}_{esq}$. □

Como nenhuma das matrizes da prova do teorema 2.1.2 contém mais de duas produções, obtém-se o seguinte corolário:

Teorema 2.1.3 *Para qualquer linguagem L em $\mathcal{M}_{esq}$, existe uma gramática livre de contexto matricial G que não contém a palavra vazia λ tal que $L = L(G)$ e nenhuma das matrizes de G contém mais de duas produções.*

Nesta seção foram introduzidas as gramáticas matriciais, baseadas em gramáticas de tipo i, $0 \leq i \leq 3$, segundo a hierarquia de Chomsky, e tais que suas produções são aplicadas em "grupos", denominados *matrizes*. Essas matrizes constituem o primeiro *mecanismo de controle* que estudamos neste capítulo. Foi introduzida também a noção de *aplicação de produção em modo de verificação de aparência*, que será redefinida para cada um dos outros mecanismos de controle estudados nas seguintes três seções.

Também foi provado que as gramáticas matriciais sob derivações mais à esquerda caracterizam as linguagens dependentes de contexto que não contém a palavra vazia. Todos esses conceitos foram ilustrados com diversos exemplos.

Embora as gramáticas matriciais possuem, em geral, como núcleo, uma gramática de qualquer tipo segundo a hierarquia de Chomsky, são de interesse especial aquelas baseadas em gramáticas livres de contexto (de tipo 2 segundo a hierarquia de Chomsky) pois, como será provado na seção 2.4, essas gramáticas possuem poder computacional de máquina de Turing.

2.2 Gramáticas Variantes no Tempo

Gramáticas variantes no tempo[15], são gramáticas de tipo i, $0 \leq i \leq 3$, segundo a hierarquia de Chomsky, nas quais, a cada passo de uma derivação, as produções que podem ser aplicadas são determinadas por uma função, que denominamos *de variação*, que tem como domínio o conjunto dos inteiros positivos e que toma como valores subconjuntos não-vazios do conjunto das produções da gramática. Nesta seção definem-se os conceitos básicos das gramáticas variantes no tempo, estuda-se o caso especial das gramáticas variantes no tempo no qual a função de variação é periódica, redefine-se o conceito, introduzido na seção anterior para gramáticas matriciais, de *aplicação de produção em modo de verificação de aparência*, para o caso das gramáticas variantes no tempo, define-se uma notação para o caso específico das gramáticas variantes no tempo baseadas em gramáticas livres de contexto, e obtém-se uma caracterização das gramáticas matriciais com verificação de aparência, que possuem duas produções em cada matriz, em termos das gramáticas variantes no tempo com função de variação periódica, baseadas em gramáticas livres de contexto.

2.2.1 Conceitos básicos

Definição 2.2.1 (Gramática variante no tempo) *Uma gramática variante no tempo GT de tipo i, com $0 \leq i \leq 3$, é um par ordenado (G, v) onde $G = (N, T, P, S)$ é uma gramática de tipo i e $v : \mathbb{N} \to 2^P \setminus \{\emptyset\}$[1] é a função de variação (uma função dos números naturais no conjunto de subconjuntos não-vazios de P).*

Definição 2.2.2 (Passo de derivação) *Seja $GT = (G, v)$ uma gramática variante no tempo de tipo i, com $0 \leq i \leq 3$, e $X, Y \in (N \cup T)^*$. Dizemos que X deriva diretamente Y e escrevemos*

$$X \Rightarrow Y$$

se existe um inteiro $j \geq 1$ e cadeias r, s sobre $(N \cup T)^*$ tais que:

(i) $X = rEs$ *e* $Y = rDs$,

(ii) $p = E \to D$ *pertence a* $v(j)$,

(iii) a produção a ser aplicada no passo de derivação seguinte deve pertencer a $v(j+1)$.

Nestas condições também dizemos que a produção p foi aplicada sobre a forma sentencial X e que $X \Rightarrow Y$ é um passo de derivação.

[1] a diferença dos conjuntos 2^P e $\{\emptyset\}$.

Notação 2.2.1 *O fechamento reflexivo-transitivo da relação binária $\Rightarrow$ é denotado como $\Rightarrow^*$.*

Definição 2.2.3 *Seja $GT = (G, v)$ uma gramática variante no tempo e $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$. Uma derivação de n passos desde X até Y é uma seqüência finita de formas sentenciais $w_i \in (N \cup T)^*$, $0 \leq i \leq n$, tais que*

(i) $w_0 = X$,

(ii) $w_{i-1} \Rightarrow w_i$, *para* $0 \leq i \leq n$, *isto é,* w_{i-1} *deriva diretamente* w_i,

(iii) $w_n = Y$.

Nestas condições denotamos $X \Rightarrow^n Y$. Assim, a aplicação da produção p sobre a forma sentencial X é uma derivação de um só passo.

Definição 2.2.4 *A linguagem gerada pela gramática variante no tempo (G, v), onde $G = (N, T, P, S)$, é*

$$L(G, v) = \{w \in T^* : S \Rightarrow^* w\}.$$

Definição 2.2.5 *Uma linguagem L é "variante no tempo de tipo i" se e somente se*

$$L = L(G, v)$$

para alguma gramática (G, v) variante no tempo de tipo i.

Teorema 2.2.1 *Toda linguagem L é variante no tempo de tipo* 3.

Prova. Seja L uma linguagem sobre o alfabeto $\Sigma = \{a_1, \ldots, a_n\}$. Se L é vazia então L é variante no tempo de tipo 3. Suponhamos que $L \neq \emptyset$, digamos $L = \{w_i : i \geq 1\}$, onde $w_i = a_{i_1}, \ldots, a_{i_{|w_i|}}$, se $|w_i| > 0$ e, se $|w_i| = 0$, então $w_i = \lambda$.
Consideremos a gramática regular $G = (\{S, X\}, \Sigma, P, S)$, onde

$$P = \{S \to S, S \to X, X \to \lambda, \} \cup \{X \to a_i X : 1 \leq i \leq n\}.$$

Definimos agora $v : \mathbb{N} \to 2^P$

$$v(i) = \{S \to S, p\}.$$

onde a produção p está definida indutivamente a seguir:

Caso base $i = 1$:

Suponha que $w_1 = c_1, \ldots, c_{|w_1|}$, com $|w_1| \geq 0$ e $c_i \in \Sigma$ então $S \to X \in v(1)$ e, para $1 \leq j \leq |w_1|$

$$\begin{array}{rcl} X \to c_j X & \in & v(1+j), \\ S \to \lambda & \in & v(2+|w_1|). \end{array}$$

Hipótese indutiva: Suponha que para algum $i \geq 1$ a segunda produção em $v(j)$ tenha sido definida para todos os valores j tais que

$$j \leq |w_i| + 2i = j_0.$$

Definição indutiva Consideremos a palavra $w_{i+1} = b_1 \ldots b_{|w_{i+1}|}$, onde $b_i \in \Sigma$. Então:

$$\begin{array}{rcl} S \to X & \in & v(j_0+1) \\ X \to b_j X & \in & v(j_0+1+j), 1 \leq j \leq |w_{i+1}| \\ X \to \lambda & \in & v(j_0+2+|w_{i+1}|). \end{array}$$

Isto termina a definição de v.
Agora mostramos que $L = L(G, v)$. Com efeito, pela definição de v, as derivações segundo (G, v) começam aplicando j_0 vezes a produção $S \to S$ (aqui $j_0 = 0$ ou então $j_0 = |w_i| + 2i$); depois disso, aplica-se uma vez a produção $S \to X$ do conjunto $v(j_0 + 1)$; depois disso, a cada passo da derivação, é aplicada a produção $X \to b_j X$ do conjunto $v(j_0 + 1 + j)$ para cada $1 \leq j \leq |w_{i+1}|$; depois disso, para terminar a derivação da palavra w_i é aplicada a produção $X \to \lambda$ do conjunto $v(j_0+2+|w_{i+1}|)$. □

Observação 2.2.1 *A idéia central, na prova do teorema anterior, é definir uma seqüência de derivações (letra-a-letra e mais à esquerda) das palavras de L. A função pode ser não computável, pois depende de uma enumeração das palavras da linguagem.*

Exemplo 2.2.1 Gramática variante no tempo de tipo 3 para a linguagem $L = \{a^n b^n : n \geq 1\}$**.** *Neste exemplo mostramos como é construida a função de variação v do teorema* 2.2.1*. Consideremos*

$$L = \{a^n b^n : n \geq 1\} = \{ab = w_1, a^2b^2 = w_2, a^3b^3 = w_3, a^4b^4 = w_4, \ldots\}$$

Derivação de *ab:*

$$\begin{array}{lllll} w_1 = a_1 a_2 = ab & S \to A & \in v(1) & = v(1) \\ & A \to aA & \in v(1+1) & = v(2) \\ & A \to bA & \in v(1+2) & = v(3) \\ & A \to \lambda & \in v(2+2) & = v(4) \end{array}$$

Derivação de a^2b^2:

$$\begin{array}{lllll} w_{i+1} = w_2 = a_1a_2a_3a_4 = aabb & S \to A & \in v(4+1) & = v(5) \\ & A \to aA & \in v(4+1+1) & = v(6) \\ & A \to aA & \in v(4+1+2) & = v(7) \\ & A \to bA & \in v(4+1+3) & = v(8) \\ & A \to bA & \in v(4+1+4) & = v(9) \\ & A \to \lambda & \in v(4+2+4) & = v(10) \end{array}$$

2.2.2 Gramáticas Periodicamente Variantes no Tempo

Definição 2.2.6 *Uma gramática variante no tempo (G, v) é dita "periodicamente variante no tempo" se e somente se a função v for periódica, isto é, se existe um inteiro $k \geq 1$ tal que*

$$v(j+k) = v(j), \text{ para todo } j.$$

Exemplo 2.2.2 Gramática periodicamente variante no tempo $G_4 = (G, v)$ para a linguagem $L = \{a^nb^nc^n : n \geq 1\}$. *Consideremos a seguinte gramática livre de contexto sem produções-λ periodicamente variante no tempo, com período igual a 3, $G_4 = (\{S, A, B, C, D\}, \{a, b, c\}, P, S)$, onde o conjunto de produções é definido como segue:*

$$\begin{array}{l} P = \{ \\ p_0 = S \to ABC, \quad p_1 = A \to aA, \quad p_2 = B \to bB, \quad p_3 = C \to cC, \\ p_4 = A \to a, \qquad p_5 = B \to D, \quad p_6 = C \to c \qquad p_7 = D \to b, \\ \} \end{array}$$

e a função v é definida da seguinte forma:

$$v(i) = \begin{cases} \{p_0, p_3, p_6\}, & i = 1, \\ \{p_1, p_4, p_7\}, & i = 2, \\ \{p_2, p_5\}, & i = 3, \\ v(i-3), & i \geq 3. \end{cases}$$

Derivação de $a^3b^3c^3$. *Com efeito:*

$$S \Rightarrow^{ac}_{p_1} ABC \Rightarrow^{ac}_{p_2} aABC \Rightarrow^{ac}_{p_3} aAbBC \Rightarrow^{ac}_{p_4} aAbBcC \Rightarrow^{ac}_{p_2} aaAbBcC \Rightarrow^{ac}_{p_3} aaAbbBcC$$

$$\Rightarrow^{ac}_{p_4} aaAbbBccC \Rightarrow^{ac}_{p_6} a^3b^2Bc^2C \Rightarrow^{ac}_{p_5} a^3b^2Dc^2C \Rightarrow^{ac}_{p_8} a^3b^2Dc^3 \Rightarrow^{ac}_{p_7} a^3b^3c^3.$$

Vamos demonstrar, por indução matemática em n, que $a^n b^n c^n \in L(G)$.

Caso base: $n = 1$. *Como antes,*

$$S \Rightarrow^{ac}_{p_1} ABC \Rightarrow^{ac}_{p_6} aBC \Rightarrow^{ac}_{p_5} aDC \Rightarrow^{ac}_{p_8} aDc \Rightarrow^{ac}_{p_7} abc$$

Hipótese indutiva: *Seja $n > 1$ e suponhamos que $(\forall k)(1 \leq k < n), a^k b^k c^k \in L(G)$.*

Consideremos $a^n b^n c^n$. Pela hipótese indutiva existe uma derivação para $a^{n-1} b^{n-1} c^{n-1}$, digamos $S \Rightarrow^* a^{n-1} b^{n-1} c^{n-1}$. As derivações segundo G consistem de uma aplicação de p_1 seguida de repetições de aplicações das produções p_2, p_3 e p_4, nessa ordem e, por último, uma aplicação de cada uma das produções p_6, p_5, p_8, p_7 nessa ordem; portanto podemos decompor a derivação de $a^{n-1} b^{n-1} c^{n-1}$ do seguinte modo:

$$S \Rightarrow^* a^{n-2} A b^{n-2} B c^{n-2} C \Rightarrow^{ac}_{p_6} a^{n-1} b^{n-2} B c^{n-2} C \Rightarrow^{ac}_{p_5} a^{n-1} b^{n-2} D c^{n-2} C$$

$$\Rightarrow^{ac}_{p_8} a^{n-1} b^{n-2} D c^{n-1} \Rightarrow^{ac}_{p_7} a^{n-1} b^{n-1} c^{n-1}.$$

Ficando apenas com a primeira parte desta derivação (que inclui as repetições de p_2, p_3, p_4) podemos obter uma derivação para $a^n b^n c^n$, do seguinte modo:

$$S \Rightarrow^* a^{n-2} A b^{n-2} B c^{n-2} C \Rightarrow^{ac}_{p_2} a^{n-1} A b^{n-2} B c^{n-2} C \Rightarrow^{ac}_{p_3} a^{n-1} A b^{n-1} B c^{n-2} C$$

$$\Rightarrow^{ac}_{p_4} a^{n-1} A b^{n-1} B c^{n-1} C \Rightarrow^{ac}_{p_6} a^n b^{n-1} B c^{n-1} C \Rightarrow^{ac}_{p_5} a^n b^{n-1} D c^{n-1} C$$

$$\Rightarrow^{ac}_{p_8} a^n b^n c^{n-1} C \Rightarrow^{ac}_{p_7} a^n b^n c^n.$$

A idéia é, simplesmente, fazer mais uma iteração nas repetições de p_2, p_3, p_4. Isto completa a indução e termina a prova.

Observação 2.2.2 *É possível encontrar gramáticas periodicamente variantes no tempo, com diferentes períodos, que geram a mesma linguagem. Por exemplo, a gramática periodicamente variante no tempo $G_4^{'} = (G, v)$ também gera a linguagem $L = \{a^n b^n c^n : n \geq 1\}$. Consideremos a seguinte gramática livre de contexto sem produções-λ periodicamente variante no tempo, com período igual a 2, $G_4^{'} = (\{S, A, C, D\}, \{a, b, c\}, P, S)$, onde o conjunto de produções é definido como segue:*

$$\begin{array}{l} P = \{ \\ p_1 = S \to AC, \quad p_2 = A \to aAb, \quad p_3 = C \to cC, \\ p_4 = A \to ab, \quad\ p_5 = C \to D, \quad\ p_6 = D \to c. \\ \} \end{array}$$

e a função v é definida da seguinte forma:

$$v(i) = \begin{cases} \{p_1, p_3, p_5\}, & i = 1, \\ \{p_2, p_4, p_6\}, & i = 2, \\ v(i-2), & i \geq 3. \end{cases}$$

Note-se também que, as gramáticas G_4 do exemplo 2.2.2 e a gramática G_4' são gramáticas periodicamente variante no tempo livres de contexto.

Exemplo 2.2.3 Gramática periodicamente variante no tempo $G_5 = (G, v)$ para a linguagem $L = \{ww : w \in \{a, b\}^+\}$. *Consideremos a seguinte gramática livre de contexto $G_5 = (\{S, A, B, C, D\}, \{a, b, c\}, P, S)$, onde o conjunto de produções é definido como segue:*

$$\begin{array}{l} P = \{ \\ p_1 = S \to AB, \quad p_2 = A \to aA, \quad p_3 = B \to aB, \quad p_4 = C \to C, \quad p_5 = A \to A, \\ p_6 = A \to bC, \quad p_7 = B \to bD, \quad p_8 = B \to B, \quad p_9 = D \to D, \quad p_{10} = A \to \lambda, \\ p_{11} = B \to \lambda, \quad p_{12} = C \to aA, \quad p_{13} = D \to aB, \quad p_{14} = C \to bC, \\ p_{15} = D \to bD, \quad p_{16} = C \to \lambda, \quad p_{17} = D \to \lambda \\ \} \end{array}$$

com a função v definida da seguinte forma:

$$v(i) = \begin{cases} \{p_1, p_5, p_9\}, & i = 1, \\ \{p_2, p_6, p_{10}, p_{12}, p_{14}, p_{16}\}, & i = 2, \\ \{p_3, p_7, p_{11}, p_{13}, p_{15}, p_{17}\}, & i = 3, \\ \{p_4, p_8\} & i = 4, \\ v(i-4), & i \geq 5. \end{cases}$$

Derivação de bb*:*

$$S \Rightarrow^{ac}_{p_1} AB \Rightarrow^{ac}_{p_6} bCB \Rightarrow^{ac}_{p_7} bCbD \Rightarrow^{ac}_{p_4} bCbD \Rightarrow^{ac}_{p_9} bCbD \Rightarrow^{ac}_{p_{16}} b\lambda bD = bbD \Rightarrow^{ac}_{p_{17}} bb\lambda = bb.$$

Vamos provar, por indução matemática no comprimento $|w|$ da palavra w, que $L(G) = \{ww : w \in \{a, b\}^+\}$.

Caso Base: $|w| = 1$
Neste caso $w = a$ ou $w = b$, resta apenas ver o primeiro caso. Com efeito:

$$S \Rightarrow^{ac}_{p_1} AB \Rightarrow^{ac}_{p_2} aAB \Rightarrow^{ac}_{p_3} aAaB \Rightarrow^{ac}_{p_8} aAaB$$

$$\Rightarrow^{ac}_{p_5} aAaB \Rightarrow^{ac}_{p_{10}} a\lambda aB = aaB \Rightarrow^{ac}_{p_{11}} aa\lambda = aa.$$

Hipótese indutiva: *Seja* $w \in \{a,b\}^+$ *tal que* $|w| = n > 1$ *e suponhamos que para todas as palavras* $u \in \{a,b\}^+$, *com* $|u| < |w|$ *temos que* $uu \in L(G)$.

Nestas condições, $w = ua$ *ou* $w = ub$, *para alguma palavra* $u \in \{a,b\}^+$. *Por sua vez,* $u = xa$ *ou* $u = xb$, *para alguma palavra* $u \in \{a,b\}^+$. *Consideramos apenas o caso* $u = xa$ *e* $w = ua$, *pois os outros casos são análogos.*

Pela hipótese indutiva, existe uma derivação de uu, *digamos:*

$$S \Rightarrow^* xaAxaB \Rightarrow^{ac}_{p_{10}} xa\lambda xaB = xaxaB \Rightarrow^{ac}_{p_{11}} xaxa\lambda = xaxa = uu.$$

Desconsiderando as duas últimas produções aplicadas, obtemos uma derivação para ww *do seguinte modo:*

$$S \Rightarrow^* xaAxaB = uAuB \Rightarrow^{ac}_{p_2} uaAuB \Rightarrow^{ac}_{p_3} uaAuaB =$$

$$wAwB \Rightarrow^{ac}_{p_8} wAwB \Rightarrow^{ac}_{p_5} wAwB \Rightarrow^{ac}_{p_{10}} w\lambda wB = wwB \Rightarrow^{ac}_{p_{11}} ww\lambda = ww.$$

Isto completa a indução e termina a prova.

2.2.3 Gramáticas variantes no tempo com verificação de aparência

Definição 2.2.7 (Modo de verificação de aparência) *Seja* $GT = (G, v)$ *uma gramáti-ca variante no tempo, com* $G = (N, T, P, S)$, $F \subset P$ *e* $X, Y \in (N \cup T)^*$. *Dizemos que* X *deriva diretamente* Y *e escrevemos*

$$X \Rightarrow^{ac} Y$$

se existe um inteiro $j \geq 1$ *e cadeias* r, s *sobre* $(N \cup T)$ *tais que:*

(i) $X = rEs$ *e* $Y = rDs$,

(ii) $p = E \rightarrow D$ *pertence a* $v(j)$,

(iii) a produção a ser aplicada no passo de derivação seguinte deve pertencer a $v(j+1)$,

ou então

(i) E *não aparece em* X *e, neste caso,* $Y = X$

(ii) $p = E \to D$ *pertence a* $v(j) \cap F$,

(iii) *a produção a ser aplicada no passo de derivação seguinte deve pertencer a* $v(j+1)$.

Nestas condições, também dizemos que a produção p foi aplicada na forma sentencial X *em modo de verificação de aparência e que* $X \Rightarrow^{ac} Y$ *é um passo de derivação em modo de verificação de aparência, e dizemos que* $(GT, F) = (G, v, F)$ *é uma gramática variante no tempo com verificação de aparência para as produções em* F.

Notação 2.2.2 *O fechamento reflexivo-transitivo da relação binária* $\Rightarrow^{ac}$ *é denotado como* $\Rightarrow^{ac^*}$.

Definição 2.2.8 *Seja* (G, v, F) *uma gramática variante no tempo com verificação de aparência para as produções em* F, *onde* $G = (N, T, P, S)$ *é uma gramática, e* $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$; *uma derivação de n passos desde* X *até* Y *é uma seqüência finita de formas sentenciais* $w_i \in (N \cup T)^*$, $0 \leq i \leq n$, *tais que*

(i) $w_0 = X$,

(ii) $w_{i-1} \Rightarrow w_i$, *para* $0 \leq i \leq n$, *isto é,* w_{i-1} *deriva diretamente* w_i,

(iii) $w_n = Y$.

Nestas condições denotamos $X \Rightarrow^{ac^n} Y$. *Assim, a aplicação da produção p sobre a forma sentencial* X *é uma derivação de um passo.*

Definição 2.2.9 *Seja* $G = (N, T, P, S)$ *uma gramática e* $F \subset P$ *um subconjunto das produções de* G. *A linguagem gerada pela gramática variante no tempo* (G, v, F) *com verificação de aparência para as produções em* F *é*

$$L_{ac}(G, v, F) = \{w \in T^* : S \Rightarrow^{ac^*} w\}.$$

Notação 2.2.3 *Introduzimos uma notação para a família das linguages geradas pelas gramáticas variantes no tempo com verificação de aparência, da seguinte forma:*

1. *A família das linguagens da forma* $L_{ac}(G, v, F)$ *onde* G *é uma gramática livre de contexto e* v *é uma função periódica, é denominada a "família das linguages livres de contexto periodicamente variantes no tempo com verificação de aparência" e é denotada* $\mathcal{T}_{ac}^{\lambda}$. *Algumas vezes, na literatura especializada, as gramáticas que geram estas linguagens são denotadas ptvcfac, pela sua denominação em inglês: periodically time-varying context-free grammars with appearance checking.*

2. *A família das linguagens da forma $L_{ac}(G, v, F)$ onde G é uma gramática livre de contexto sem produções-λ e v é uma função periódica, é denotada $\mathcal{T}_{ac}$.*

3. *A família das linguagens da forma $L_{ac}(G, v, F)$ onde G é uma gramática livre de contexto, v é uma função periódica e $F = \emptyset$, é denotada $\mathcal{T}^{\lambda}$.*

4. *A família das linguagens da forma $L_{ac}(G, v, F)$ onde G é uma gramática livre de contexto sem produções-λ, v é uma função periódica é denominada e $F = \emptyset$, é denotada $\mathcal{T}$.*

Proposição 2.2.1

$$\mathcal{T} \subseteq \mathcal{T}^{\lambda} \subseteq \mathcal{T}_{ac}^{\lambda}, \qquad \mathcal{T} \subseteq \mathcal{T}_{ac} \subseteq \mathcal{T}_{ac}^{\lambda}.$$

Teorema 2.2.2 *Toda linguagem L gerada por uma gramática matricial tal que todas as suas matrizes consistem de duas produções, pertence a $\mathcal{T}^{\lambda}$.*

Prova. Seja $G = (N, T, M, S)$ uma gramática matricial tal que suas matrizes são da forma:

$$\left[E_1^i \to D_1^i, E_2^i \to D_2^i\right], 1 \leq i \leq u$$

onde $E_1^i, E_2^i \in N$. Vamos definir uma gramática periodicamente variante no tempo (G_1, v) tal que $L(G) = L(G_1, v)$. Com efeito, consideremos

$$G_1 = (N \cup \{Y_j^i : 1 \leq i \leq u, 1 \leq j \leq 2\}, T, P, S)$$

Os novos não-terminais Y_j^i são introduzidos para evitar derivações que usem produções (primeira e segunda) de matrizes diferentes. O conjunto de produções é:

$$P = \{E_j^i \to D_j^i Y_j^i, Y_j^i \to Y_j^i, Y_j^i \to \lambda : 1 \leq i \leq u, 1 \leq j \leq 2\}$$

e a função periódica é:

$$v(k) = \begin{cases} \{E_1^i \to D_1^i Y_1^i : 1 \leq i \leq u\}, & k = 1, \\ \{E_2^i \to D_2^i Y_2^i : 1 \leq i \leq u\}, & k = 2, \\ \{Y_2^h \to Y_2^h\} \cup \{Y_1^i \to Y_1^i : 1 \leq i \leq u\}, & k = 2h+1, 1 \leq h \leq u-1, i \neq h \\ \{Y_1^h \to Y_1^h\} \cup \{Y_2^i \to Y_2^i : 1 \leq i \leq u\}, & k = 2h+2, 1 \leq h \leq u-1, i \neq h \\ \{Y_1^i \to \lambda : 1 \leq i \leq u\}, & k = 2u+1, \\ \{Y_2^i \to \lambda : 1 \leq i \leq u\}, & k = 2u+2, \end{cases}$$

Assim definida, a função v é periódica com período $2u+2$. Vamos provar agora que a linguagem gerada pela gramática (G_1, v) coincide com a linguagem gerada pela gramática G.

Com efeito, consideremos a aplicação de uma matriz do tipo $[E_1^i \to D_1^i, E_2^i \to D_2^i]$ em uma derivação segundo G. É possível obter o mesmo efeito em uma derivação segundo (G_1, v) em $2u+2$ passos. A única diferença é que, durante os dois primeiros passos, os não-terminais Y_1^i, Y_2^i são introduzidos, e durante os dois últimos passos, esses não-terminais são apagados. Assim, qualquer derivação segundo G pode ser simulada por uma derivação segundo (G_1, v); portanto, $L(G) \subseteq L(G_1, v)$.

Observamos agora que uma derivação segundo (G_1, v) pode ser dividida em subderivações, cada uma das quais consiste de $2u + 2$ passos. Consideremos uma subderivação tal que se, para algum i, os não-terminais Y_1^i, Y_2^i, foram introduzidos nos dois primeiros passos, então pode-se obter o mesmo efeito, em uma derivação segundo G, aplicando-se a matriz $[E_1^i \to D_1^i, E_2^i \to D_2^i]$. Por outro lado se os não-terminais Y_1^p, Y_2^q são introduzidos durante os primeiros dois passos, então pelo fato de as produções $Y_{3-j}^h \to Y_{3-j}^h, j = 1, 2$ pertencerem a $v(k)$, ambas as produções $Y_1^p \to Y_1^p$ e $Y_1^q \to Y_2^q$ poderarão ser aplicadas ao mesmo tempo. Portanto, para algum k, não existe produção aplicável no k-ésimo passo. Assim a derivação termina sem produzir palavra alguma. Portanto $L(G_1, v) \subseteq L(G)$. Como também $L(G) \subseteq L(G_1, v)$ então $L(G) = L(G_1, v)$.
□

Teorema 2.2.3 *Seja $G = (N, T, M, S)$ uma gramática matricial com matrizes*

$$\left[E_1^i \to D_1^i, E_2^i \to D_2^i\right]$$

onde $i = 1 \ldots u$, e são tais que os conjuntos de símbolos que aparecem na primeira e na segunda produção são disjuntos (isto é, nenhum símbolo aparece simultaneamente em $E_1^i D_1^i$ e $E_2^j D_2^j$, onde $1 \leq i, j \leq u$). Seja F o conjunto das ocorrências das primeiras produções das matrizes de G, e seja $\Rightarrow^{ac}$ definido com respeito a esse F. Então

$$\{w \in T^* : P_0 S_0 \Rightarrow^{ac^*} w\} \in \mathcal{T}_{ac}^{\lambda} \tag{2.1}$$

onde P_0 é um dos não-terminais que aparecem nas primeiras produções das matrizes e S_0 é um dos não-terminais que aparecem nas segundas produções das matrizes.

Prova. Suponhamos primeiro que F é vazio. Então a linguagem $\{w \in T^* : P_0 S_0 \Rightarrow^{ac^*} w\}$ é gerada pela gramática matricial com matrizes $[E_1^i \to D_1^i, E_2^i \to D_2^i]$ e pela matriz $\left[X_0' \to X_0', X_0' \to P_0 S_0\right]$ onde X_0' é o novo símbolo não-terminal inicial. Pelo teorema 2.2.2 a linguagem em questão pertence a $\mathcal{T}^{\lambda}$ e, portanto, a $\mathcal{T}_{ac}^{\lambda}$.

Consideremos agora o caso de F não ser vazio, e suponhamos que ele contém k

produções, com $1 \leq k \leq u$. Sem perda de generalidade (renumerando adequadamente as produções se for necessário), podemos supor que as produções em F sejam:

$$E_1^i \to D_1^i, \qquad 1 \leq i \leq k.$$

Vamos definir agora uma gramática periodicamente variante no tempo (G_1, v_1) e um subconjunto F_1 do conjunto de produções de G_1 tal que

$$\{w \in T^* : P_0 S_0 \Rightarrow^{ac^*} w\} = L_{ac}(G_1, v_1, F_1)$$

Com efeito, consideremos uma gramática periodicamente variante no tempo com verificação de aparência (G_1, v_1) onde $G_1 = (N_1, T, P_1, X_0')$ sendo o conjunto das produções P_1 formado pelas seguintes produções:

$$E_j^i \to D_j^i Y_j^i, \quad 1 \leq i \leq u, 1 \leq j \leq 2, \tag{2.2}$$

$$Y_j^i \to Y_j^i, \quad 1 \leq i \leq u, 1 \leq j \leq 2, \tag{2.3}$$

$$Y_j^i \to \lambda, \quad 1 \leq i \leq u, 1 \leq j \leq 2, \tag{2.4}$$

$$X_0' \to P_0 S_0 U \tag{2.5}$$

$$U \to \lambda \tag{2.6}$$

$$U \to U^i U Y_1^i \qquad 1 \leq i \leq k \tag{2.7}$$

$$E_1^i \to Z^i E_1^i \qquad 1 \leq i \leq k \tag{2.8}$$

$$U^i \to Z^i \qquad 1 \leq i \leq k \tag{2.9}$$

$$Z^i \to \lambda \qquad 1 \leq i \leq k \tag{2.10}$$

$$Z^i \to Z \qquad 1 \leq i \leq k \tag{2.11}$$

e o conjunto de não-terminais N_1 formado pela união do conjunto de não-terminais N de G com os novos não-terminais X_0',Y_j^i, U, U^i, Z, Z^i onde os índices assumem valores como definido em $(2.2) - (2.11)$. O conjunto F_1 é formado pelas produções (2.5) e $(2.8) - (2.11)$.

Vamos agora definir a função de variação v_1, com período $(2u + 3) + 4k$ em ter-

mos da função de variação v usada na demonstração do teorema 2.2.2:

$$v_1(j) = \begin{cases} X_0' \to P_0 S_0 U, & j = 1, \\ v(1) \cup \{U \to \lambda\} \cup \{U \to U^i U Y_1^i : 1 \leq i \leq k\}, & j = 2, \\ v(j-1), & 3 \leq j \leq 2u+3, \\ \hline E_1^1 \to Z^1 E_1^1 & j = (2u+3)+1 \\ U^1 \to Z^1 & j = (2u+3)+2 \\ Z^1 \to \lambda & j = (2u+3)+3 \\ \ldots & \ldots \\ E_1^k \to Z^k E_1^k & j = (2u+3)+3k-2 \\ U^k \to Z^k & j = (2u+3)+3k-1 \\ Z^k \to \lambda & j = (2u+3)+3k \\ \hline Z^1 \to Z & j = (2u+3)+3k+1 \\ \ldots & \\ Z^k \to Z & j = (2u+3)+4k \end{cases}$$

A seguir mostraremos que as linguagens coincidem. Com efeito, a produção (2.5) é aplicada no inicio de uma derivação segundo (G_1, v_1, F_1) e, depois disso, é aplicada apenas em modo de verificação de aparência pois X_0' não aparece en nenhum outro ponto.

A cada aplicação da matriz $[E_1^i \to D_1^i, E_2^i \to D_2^i]$ correspondem $(2u+2)+4k$ passos em uma derivação segundo (G_1, v_1, F_1). Apenas os primeiros dois destes passos são essenciais: os outros servem para verificar que os dois primeiros passos foram executados corretamente. Os não-terminais Y_j^i permitem verificar que as produções aplicadas nos dois primeiros passos vêm da mesma matriz.

Suponha que se deseja aplicar a produção

$$E_1^i \to D_1^i, 1 \leq i \leq k$$

no primeiro passo de uma derivação na qual E_1^i não aparece na forma sentencial que está sendo examinada. Depois disso, aplicamos a produção (2.7). Essa aplicação estaria incorrecta se, e somente se, para algum i, tivéssemos ambos os não-terminais E_1^i e U^i na forma sentencial examinada depois do passo inicial. Pela hipótese sobre as matrizes, isso acontece no caso quando E_1^i e U^i aparecem na forma sentencial examinada após o passo $(2u+2)$. Por outro lado, isso significa que o não-terminal Z (que não pode ser eliminado) irá ser introduzido depois do passo $(2u+2)$. Pela escolha de F_1, esse é o único caso no qual Z é introduzido. Portanto, somente aplicações corretas de (2.7) levam a palavras formadas apenas por terminais.

Uma derivação segundo (G_1, v_1, F_1) é terminada pela aplicação da produção (2.6).

Se a produção (2.6) é usada prematuramente, então no passo seguinte será introduzido o não terminal Y_2^i, o qual não poderá ser eliminado, e, portanto, não será obtida uma sentença desta derivação.
□

Salomaa[2], observa que as construções das duas provas precendentes podem ser refeitas para o caso de gramáticas matriciais sem produções-λ, obtendo assim, como corolario a proposição seguinte:

Proposição 2.2.2

$$\mathcal{M} \subseteq \mathcal{T}, \qquad \mathcal{M}_{ac} \subseteq \mathcal{T}_{ac}.$$

Queremos construir uma gramática livre de contexto periodicamente variante no tempo com verificação de aparência para a linguagem $L = \{a^{2^n} : n \geq 1\}$. Não existe um tal exemplo na literatura consultada: Salomaa[2], Rozenberg[17], Dassow[4]. Uma idéia é aplicar o teorema 2.4.3 a uma gramática livre de contexto matricial com verificação de aparência para a linguagem L; mas essa gramática matricial teria que ter duas produções em cada matriz. A única gramática livre de contexto matricial com verificação de aparência para a linguagem L, é a gramática G_3 do exemplo 2.1.3, que não satisfaz a hipótese de ter duas produções em cada matriz.

Outra idéia seria usar como base a demonstração do teorema 2.1.2 para tentar transformar essa gramática G_3 em uma gramática gramática livre de contexto matricial com verificação de aparência com duas produções por matriz. Mas essa demonstração se refere a gramáticas matriciais com derivações mais à esquerda e, além disso, não considera produções aplicadas em modo de verificação de aparência.

Vamos seguir outro caminho. Usaremos um exemplo da seção 4, no qual o teorema 2.4.6 é usado para transformar uma gramática livre de contexto com linguagem de controle regular com verificação de aparência em uma gramática livre de contexto matricial com verificação de aparência com duas produções em cada matriz; desta forma, poderemos aplicar o teorema 2.2.3 e obter uma gramática livre de contexto periodicamente variante no tempo com verificação de aparência para a linguagem em questão.

Exemplo 2.2.4 *A partir da gramática livre de contexto matricial com verificação de aparência GM do exemplo 2.4.4, na qual cada matriz possui apenas duas produções, vamos obter uma gramática livre de contexto periodicamente variante no tempo com verificação de aparência para a linguagem $L = \{a^{2^n} : n \geq 1\}$. Usamos o método do teorema 2.2.3*

Neste caso temos que a quantidade de produções que podem ser aplicadas em modo de verificação de aparência é $k = 2$ (observe-se que este valor coincide, por acaso,

com a quantidade de produções nas matrizes) e a quantidade de matrizes é $u = 7$.

Primeiro renumeramos as matrizes, de modo que as produções a serem aplicadas em modo de verificação de aparência sejam as primeiras produções (aquelas mais a esquerda) das duas primeiras matrizes.

De (2.2) *do teorema 2.2.3 obtemos as quatorze produções seguintes:*

i		
1	$S \to XY_1^1,$	$q_0 \to q_2Y_2^1$
2	$A \to XY_1^2,$	$q_2 \to q_0Y_2^2$
3	$S \to AAY_1^3,$	$q_0 \to q_0Y_2^3$
4	$S \to aY_1^4,$	$q_0 \to q_1Y_2^4$
5	$A \to SY_1^5,$	$q_2 \to q_2Y_2^5$
6	$S \to aY_1^6,$	$q_1 \to q_1Y_2^6$
7	$S \to aY_1^7,$	$q_1 \to Y_2^7$

De (2.3) *e* (2.4) *do teorema 2.2.3 temos outras vinte e oito produções:*

$$Y_1^i \to Y_1^i, Y_2^i \to Y_2^i, Y_1^i \to \lambda, Y_2^i \to \lambda. \quad 1 \leq i \leq 7,$$

De (2.5) *do teorema 2.2.3 obtemos a produção:*

$$X_0^{'} \to Sq_0U$$

De (2.6) *do teorema 2.2.3 obtemos a produção:*

$$U \to \lambda$$

A seguinte tabela reune as últimas dez produções obtidas a partir de (2.7), (2.8), (2.9) e (2.10):

(2.i)		
(2.7)	$U \to U^1UY_1^1,$	$U \to U^2UY_2^1$
(2.8)	$S \to Z^1S$	$A \to Z^2A$
(2.9)	$U^1 \to Z^1$	$U^2 \to Z^2$
(2.10)	$Z^1 \to \lambda$	$Z^2 \to \lambda$
(2.11)	$Z^1 \to Z$	$Z^2 \to Z$

Definimos agora a função periódica v *com período* $(2u+3)+4k = (2\times7+3)+4\times2 =$

25*:*

$v(1) = \{X_0' \to Sq_0U\}$

$v(2) =$
$\{S \to XY_1^1, A \to XY_1^2, S \to AAY_1^3, S \to aY_1^4, A \to SY_1^5, S \to aY_1^6, S \to aY_1^7\}$
$\cup\{U \to \lambda\} \cup \{U \to U^1UY_1^1, U \to U^2UY_2^1\}$

$v(3) =$
$\{q_0 \to q_2Y_2^1, q_2 \to q_0Y_2^2, q_0 \to q_0Y_2^3, q_0 \to q_1Y_2^4, q_2 \to q_2Y_2^5, q_1 \to q_1Y_2^6, q_1 \to Y_2^7,$

$$v(j) = \begin{cases} \{Y_2^h \to Y_2^h\} \cup \{Y_1^i \to Y_1^i : 1 \leq i \leq 6, i \neq h\} & j = 2h+2, 1 \leq h \leq 6 \\ \{Y_1^h \to Y_1^h\} \cup \{Y_2^i \to Y_2^i : 1 \leq i \leq 6, i \neq h\}, & j = 2h+3, 1 \leq h \leq 6 \\ \{Y_1^i \to \lambda : 1 \leq i \leq 7\}, & j = 16 \\ \{Y_2^i \to \lambda : 1 \leq i \leq 7\}, & j = 17 \\ S \to Z^1S, & j = 18 \\ U^1 \to Z^1, & j = 19 \\ Z^1 \to \lambda, & j = 20 \\ A \to Z^2A, & j = 21 \\ U^2 \to Z^2, & j = 22 \\ Z^2 \to \lambda, & j = 23 \\ Z^1 \to Z, & j = 24 \\ Z^2 \to Z, & j = 25 \\ v(j-25). & \forall j \geq 26 \end{cases}$$

O conjunto das produções que podem ser aplicadas em modo de verificação de aparência é definido como:

$$F_1 = \{ \quad X_0' \to Sq_0U, \quad S \to Z^1S, \quad A \to Z^2A, \quad U^1 \to Z^1, \quad U^2 \to Z^2, \quad Z^1 \to \lambda, \quad Z^2 \to \lambda, \quad Z^1 \to Z, \quad Z^2 \to Z \quad \}$$

Definimos a gramática livre de contexto $G = (N_1, T, P_1, X_0')$, onde o conjunto de não-terminais é definido como:

$$N_1 = N \cup Q \cup \{X_0', U, Z\} \cup \{U^i, Z^i, Y_1^i, Y_2^i : 1 \leq i \leq 7\}$$

e o conjunto de produções P_1 é formado pelas cinqüenta e quatro produções definidas acima. E assim a gramática $GT = (G, v, F_1)$ é uma gramática livre de contexto periodicamente variante no tempo que reconhece a linguagem $\{a^{2^n} : n \geq 0\}$.

$$L(G, v, F_1) = \{a^{2^n} : n \geq 0\}.$$

Nesta seção foram introduzidas as gramáticas variantes no tempo, baseadas em gramáticas de tipo i, $0 \leq i \leq 3$, segundo a hierarquia de Chomsky, e tais que as produções que podem ser aplicadas, em cada passo de uma derivação, são determinadas por uma *função de variação* que tem como domínio o conjunto dos inteiros positivos e que toma como valores os subconjuntos do conjunto das produções da gramática. Essa função de variação é o segundo *mecanismo de controle* estudado neste capítulo.

A noção de *aplicação de produção em modo de verificação de aparência*, introduzida na seção anterior para as gramáticas matriciais, foi redefinida para o caso das gramáticas variantes no tempo e as gramáticas matriciais com verificação de aparência, com duas produções em cada matriz foram caracterizadas em termos de das gramáticas variantes no tempo com verificação de aparência.

Como no caso das gramáticas matriciais, foi definida uma notação especial para as gramáticas variantes no tempo baseadas em gramáticas livres de contexto, pois, como será provado na seção 2.4, essas gramáticas possuem poder de máquina de Turing.

2.3 Gramáticas Programadas

Gramáticas programadas[14], são baseadas em gramáticas de tipo i, $0 \leq i \leq 3$, segundo a hierarquia de Chomsky, e possuem, como mecanismo de controle, duas funções, denominadas respectivamente, *função de sucesso* e *função de falha*, que determinam, a cada passo de uma derivação, quais produções podem ser aplicadas após ter sido aplicada uma produção. Essas funções tem como domínio o conjunto das produções da gramática e tomam como valores subconjuntos do conjunto das produções da gramática. Mais especificamente, a função sucesso determina quais produções podem ser aplicadas depois da aplicação de uma produção *no sentido das gramáticas na hierarquia de Chomsky*, enquanto a função de falha determina as produções que podem ser aplicadas depois da aplicação de uma produção *no sentido verificação de aparência*. Desta forma, gramáticas programadas que po-- ssuem função falha com valor constante igual ao conjunto vazio $\emptyset$, são denominadas *gramáticas programadas sem verificação de aparência*. Nesta seção são definidos os conceitos básicos das gramáticas programadas, define-se uma notação para o caso específico das gramáticas programadas baseadas em gramáticas livres de contexto, são apresentados alguns exemplos, é provado que as gramáticas livres de contexto programadas com verificação de aparência identificam a classe das linguagens recursivamente enumeráveis, e prova-se, em forma construtiva, que as classes das linguagens geradas por gramáticas livres de contexto matriciais, com ou sem verificação de aparência, estão contidas nas respectivas classes das gramáticas livres de contexto programadas com ou sem verificação de aparência.

Definição 2.3.1 (Gramática Programada) *Uma gramática programada de tipo i, com $0 \leq i \leq 3$, é uma tripla $GP = (G, s, f)$ onde $G = (N, T, P, S)$ é uma gramática de tipo i, e "s","f" são funções do conjunto de produções P sobre conjunto dos subconjuntos de P, isto é:*

$$s : P \to 2^P$$

e

$$f : P \to 2^P.$$

Definição 2.3.2 (Passo de derivação) *Seja (G, s, f) uma gramática programada com $G = (N, T, P, S)$. Se $X, Y \in (N \cup T)^*$, dizemos que X deriva diretamente Y, e escrevemos:*

$$X \Rightarrow Y$$

se existirem cadeias r, s, sobre $(N \cup T)^*$ tais que:

(i) $X = rEs$ *e* $Y = rDs$,

(ii) $p = E \to D$ *é uma produção de* G*, e*

(iii) A próxima produção a ser aplicada deve pertenecer a $s(p)$*.*

Nestas condições, também dizemos que $X \Rightarrow Y$ *é válida.*

Definição 2.3.3 (Modo de verificação de aparência) *Seja* (G, s, f) *uma gramática programada com* $G = (N, T, P, S)$*. Se* $X, Y \in (N \cup T)^*$*, dizemos que* X *deriva diretamente* Y *em modo de verificação de aparência, e escrevemos*

$$X \Rightarrow^{ac} Y$$

se $X \Rightarrow Y$ *for válida, ou então cada uma das seguintes condições deve ser satisfeita:*

(i) $Y = X$*,*

(ii) $p = E \to D$ *é uma produção de* P*, sendo que* E *não aparece em* X*, e*

(iii) A próxima produção a ser aplicada deve pertenecer a $f(p)$*.*

Nestas condições, também dizemos que $X \Rightarrow^{ac} Y$ *é válida.*

Notação 2.3.1 *O fechamento reflexivo-transitivo da relação binária* $\Rightarrow$ *é denotado como* $\Rightarrow^*$*, e o fechamento reflexivo-transitivo de* $\Rightarrow^{ac}$ *é denotado como* $\Rightarrow^{ac^*}$*.*

Definição 2.3.4 (Derivação) *Seja* (G, s, f) *uma gramática programada com* $G = (N, T, P, S)$ *e* $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$*; uma derivação de* n *passos desde* X *até* Y*, em modo de verificação de aparência é uma seqüência finita de formas sentenciais* $w_i \in (N \cup T)^*$*,* $0 \leq i \leq n$*, tais que*

(i) $w_0 = X$*,*

(ii) $w_{i-1} \Rightarrow^{ac} w_i$*, para* $0 \leq i \leq n$*, isto é,* w_{i-1} *deriva diretamente* w_i *em modo de verificação de aparência,*

(iii) $w_n = Y$*.*

Nestas condições, denotamos $X \Rightarrow^{ac^n} Y$*. Assim, a aplicação da produção* p *na forma sentencial* X *é uma derivação de um passo em modo de verificação de aparência.*

Definição 2.3.5 *A linguagem gerada pela gramática programada* (G, s, f)*, é*

$$L(G, s, f) = \{w \in T^* : S \Rightarrow^* w\}.$$

Definição 2.3.6 *A linguagem gerada pela gramática programada (G, s, f), com verificação de aparência é*

$$L_{ac}(G, s, f) = \{w \in T^* : S \Rightarrow^{ac^*} w\}.$$

Notação 2.3.2 *Introduzimos uma notação para a família das linguagens geradas pelas gramáticas programadas com verificação de aparência, da seguinte forma:*

1. *A família das linguagens da forma $L_{ac}(G, s, f)$, onde G é uma gramática livre de contexto, é denotada $\mathcal{P}_{ac}^{\lambda}$.*

2. *A família das linguagens da forma $L_{ac}(G, s, f)$, onde G é uma gramática livre de contexto, sem produções-λ, é denotada $\mathcal{P}_{ac}$.*

3. *A família das linguagens da forma $L_{ac}(G, s, f)$, onde G é uma gramática livre de contexto, e $f(p) = \emptyset, \forall p \in P$, é denotada $\mathcal{P}^{\lambda}$.*

4. *A família das linguagens da forma $L_{ac}(G, s, f)$, onde G é uma gramática livre de contexto, sem produções-λ e $f(p) = \emptyset, \forall p \in P$, é denotada $\mathcal{P}$.*

Proposição 2.3.1

$$\mathcal{P} \subseteq \mathcal{P}^{\lambda} \subseteq \mathcal{P}_{ac}^{\lambda}, \qquad \mathcal{P} \subseteq \mathcal{P}_{ac} \subseteq \mathcal{P}_{ac}^{\lambda}.$$

Exemplo 2.3.1 Gramática programada $G_7 = (G, s, f)$ para a linguagem $L = \{a^n b^n c^n : n \geq 1\}$. *Consideremos a seguinte gramática livre de contexto $G_7 = (\{S, A, B, C\}, \{a, b, c\}, P, S)$, onde o conjunto de produções e o conjunto de etiquetas estão definidos e são associados como segue:*

p	$s(p)$	$f(p)$
$p_1 = S \to ABC$	$\{p_2, p_5\}$	$\emptyset$
$p_2 = A \to aA$	$\{p_3\}$	$\emptyset$
$p_3 = B \to bB$	$\{p_4\}$	$\emptyset$
$p_4 = C \to cC$	$\{p_2, p_5\}$	$\emptyset$
$p_5 = A \to a$	$\{p_6\}$	$\emptyset$
$p_6 = B \to b$	$\{p_7\}$	$\emptyset$
$p_7 = C \to c$	$\emptyset$	$\emptyset$

Nestas condições a gramática livre de contexto programada $G_7 = (G, s, f)$ gera a linguagem dependente de contexto: $L = \{a^n b^n c^n : n \geq 1\}$.

Derivação de $a^3b^3c^3$*:*
Com efeito:

$$S \Rightarrow^{ac}_{p_1} ABC \Rightarrow^{ac}_{p_2} aABC \Rightarrow^{ac}_{p_3} aAbBC \Rightarrow^{ac}_{p_4} aAbBcC \Rightarrow^{ac}_{p_2} aaAbBcC \Rightarrow^{ac}_{p_3} aaAbbBcC$$
$$\Rightarrow^{ac}_{p_4} aaAbbBccC \Rightarrow^{ac}_{p_5} a^3bbBccC \Rightarrow^{ac}_{p_6} a^3b^3ccC \Rightarrow^{ac}_{p_7} a^3b^3c^3.$$

Vamos demonstrar, por indução matemática, que $a^nb^nc^n \in L(G_7)$.

Caso base. $\mathbf{n = 1}$ *Com efeito:*

$$S \Rightarrow^{ac}_{p_1} ABC \Rightarrow^{ac}_{p_5} aBC \Rightarrow^{ac}_{p_6} abC \Rightarrow^{ac}_{p_7} abc.$$

Hipótese indutiva *Seja* $n > 1$ *e suponhamos que* $a^kb^kc^k \in L(G_7)$, $(\forall k)(1 \leq k < n)$.

Consideremos $a^nb^nc^n$. *Pela hipótese indutiva, existe uma derivação para* $a^{n-1}b^{n-1}c^{n-1}$, *digamos,* $S \Rightarrow^* a^{n-1}b^{n-1}c^{n-1}$; *toda derivação segundo* G_7 *é formada por uma aplicação da produção* p_1, *seguida de múltiplas aplicações das produções* p_2, p_3 *e* p_4, *nessa ordem e, finalmente, uma aplicação de cada uma das produções* p_5, p_6 *e* p_7, *nessa ordem; assim, podemos decompor a derivação de* $a^{n-1}b^{n-1}c^{n-1}$, *fornecida pela hipótese indutiva, da seguinta forma:*

$$S \Rightarrow^* (\text{repetições de } p_2 - p_3 - p_4)\ a^{n-2}Ab^{n-2}Bc^{n-2}C$$
$$\Rightarrow^{ac}_{p_5} a^{n-1}b^{n-2}Bc^{n-2}C \Rightarrow^{ac}_{p_6} a^{n-1}b^{n-1}c^{n-2}C \Rightarrow^{ac}_{p_7} a^{n-1}b^{n-1}c^{n-1}.$$

Vamos usar esta derivação de $a^{n-1}b^{n-1}c^{n-1}$ *para construir uma derivação de* $a^nb^nc^n$. *Com efeito, conservando a parte que contém as repetições da aplicação das produções* p_2, p_3 *e* p_4, *podemos fazer mais uma iteração e obter:*

$$S \Rightarrow^* (\text{repetições de } p_2 - p_3 - p_4)a^{n-2}Ab^{n-2}Bc^{n-2}C$$
$$\Rightarrow^{ac}_{p_2} a^{n-1}Ab^{n-2}Bc^{n-2}C \Rightarrow^{ac}_{p_3} a^{n-1}Ab^{n-1}Bc^{n-2}C \Rightarrow^{ac}_{p_4} a^{n-1}Ab^{n-1}Bc^{n-1}C$$
$$\Rightarrow^{ac}_{p_5} anb^{n-1}Bc^{n-1}C \Rightarrow^{ac}_{p_6} a^nb^nc^n - 1CC \Rightarrow^{ac}_{p_7} a^nb^nc^n.$$

Isto completa a indução e termina a prova.

Observação 2.3.1 *Existe uma representação das gramáticas programadas com diagramas de fluxo que resulta útil para planejar as demonstrações por indução; a Fig. 2.1 ilustra o diagrama de fluxo da gramática do exemplo anterior; essa representação está baseada na seqüencialidade da aplicação das produções, imposta pelas funções*

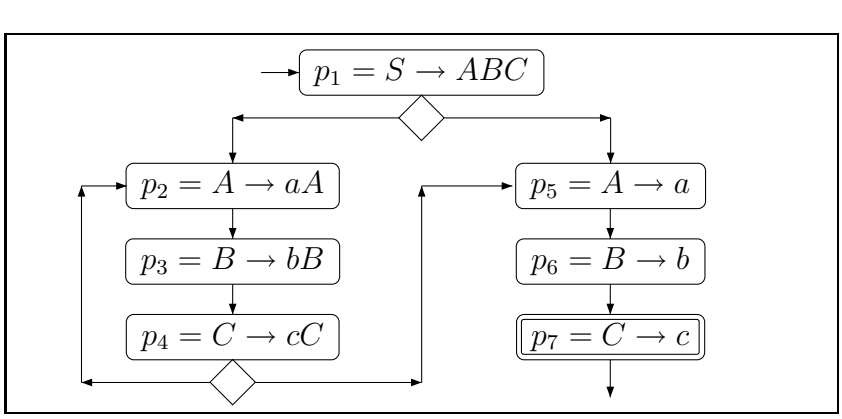


Figura 2.1: O diagrama de fluxo da gramática G_7

de sucesso e falha das gramáticas programadas. Representamos cada produção por uma caixa retangular e traçamos uma seta de uma caixa a outra se a segunda caixa pertencem ao conjunto sucesso ou falha da primeira; utilizamos uma seta especial, que não provém de caixa alguma, para indicar o papel especial do símbolo inicial da gramática. As setas podem ser interpretadas como uma relação "sucessor" e, se alguma caixa é sucessor de algum de seus sucessores, fazemos a seta voltar até onde ela apareceu pela primeira vez. Desse modo, conseguimos evitar diagramas potencialmente infinitos e capturamos a notação usual para processos iterativos dos diagramas de fluxo. Usamos uma caixa dupla para indicar que a execução da gramática programada termina nessa caixa. No caso em que a gramática possua produções aplicáveis em modo de verificação de aparência, colocam-se etiquetas "s" e "f" para indicar se as setas correspondem a valores das funções sucesso ou falha, respectivamente; omitimos as etiquetas se $f(p) = \emptyset$, $\forall p \in P$.

Observação 2.3.2 *Uma pequena modificação no exemplo anterior permite obter uma gramática livre de contexto programada* $G_7' = (G, s, f)$ *para a linguagem* $L = \{a^n b^n c^n : n \geq 0\}$. *Consideremos a seguinte gramática livre de contexto* $G_7' = (\{S, A, C\}, \{a, b, c\}, P, S)$, *onde o conjunto de produções e o conjunto de etiquetas são definidos e associados como segue:*

p	$s(p)$	$f(p)$
$p_1 = S \to AC$	$\{p_2, p_4\}$	$\emptyset$
$p_2 = A \to aAb$	$\{p_3\}$	$\emptyset$
$p_3 = C \to cC$	$\{p_2, p_4\}$	$\emptyset$
$p_4 = A \to ab$	$\{p_5\}$	$\emptyset$
$p_5 = C \to c$	$\emptyset$	$\emptyset$
$p_6 = S \to \lambda$	$\emptyset$	$\emptyset$

O diagrama de fluxo da gramática G_7' é apresentado na Fig. 2.2. Como se pode observar, trata-se de uma variação do diagrama de fluxo da gramática G_7. Uma derivação segundo a gramática G_7' pode ser formada por uma única aplicação da produção p_6, ou então, por um laço no qual são aplicadas repetidamente as produções p_2 e p_3, sendo que, a cada aplicação da produção p_2, é possível optar por aplicar a produção p_4 e, neste caso, a derivação termina com a aplicação do p_5.

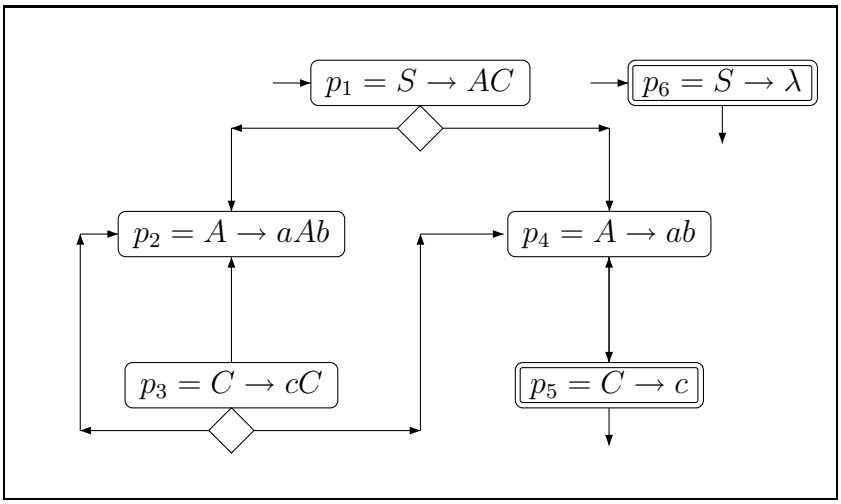


Figura 2.2: O diagrama de fluxo da gramática G_7'

Esta observação nos mostra que o digrama de fluxo de uma gramática programada pode não ser *conexo*. Essa característica pode ser evitada, no caso da gramática programada G_7', eliminando-se a produção p_6 e definindo-se as produções p_4 e p_5 como $p_4 = A \to \lambda$ e $p_5 = C \to \lambda$.

Exemplo 2.3.2 Gramática programada $G_8 = (G, s, f)$ para a linguagem $L = \{ww : w \in \{a, b\}^+\}$. *Consideremos a seguinte gramática livre de contexto programada $G_8 = (\{S, A, B\}, \{a, b\}, P, S)$, onde o conjunto de produções e as funções de sucesso e falha são definidos como segue:*

p	$s(p)$	$f(p)$
$p_1 = S \to AB$	$\{p_2, p_3, p_6, p_7\}$	$\emptyset$
$p_2 = A \to aA$	$\{p_4\}$	$\emptyset$
$p_3 = A \to bA$	$\{p_5\}$	$\emptyset$
$p_4 = B \to aB$	$\{p_2, p_3, p_6, p_7\}$	$\emptyset$
$p_5 = B \to bB$	$\{p_2, p_3, p_6, p_7\}$	$\emptyset$
$p_6 = A \to a$	$\{p_8\}$	$\emptyset$
$p_7 = A \to b$	$\{p_9\}$	$\emptyset$
$p_8 = B \to a$	$\{p_1\}$	$\emptyset$
$p_9 = B \to b$	$\{p_1\}$	$\emptyset$

Derivação de bbaabbbaab:
Com efeito:

$$S \Rightarrow^{ac}_{p_1} AB \Rightarrow^{ac}_{p_3} bAB \Rightarrow^{ac}_{p_5} bAbB \Rightarrow^{ac}_{p_3} bbAbB \Rightarrow^{ac}_{p_5} bbAbbB \Rightarrow^{ac}_{p_2} bbaAbbB$$

$$\Rightarrow^{ac}_{p_4} bbaAbbaB \Rightarrow^{ac}_{p_2} bbaaAbbaB \Rightarrow^{ac}_{p_4} bbaaAbbaaB \Rightarrow^{ac}_{p_7} bbaabbbaaB \Rightarrow^{ac}_{p_9} bbaabbbaab.$$

Vamos analisar o diagrama de fluxo da gramática G_8:

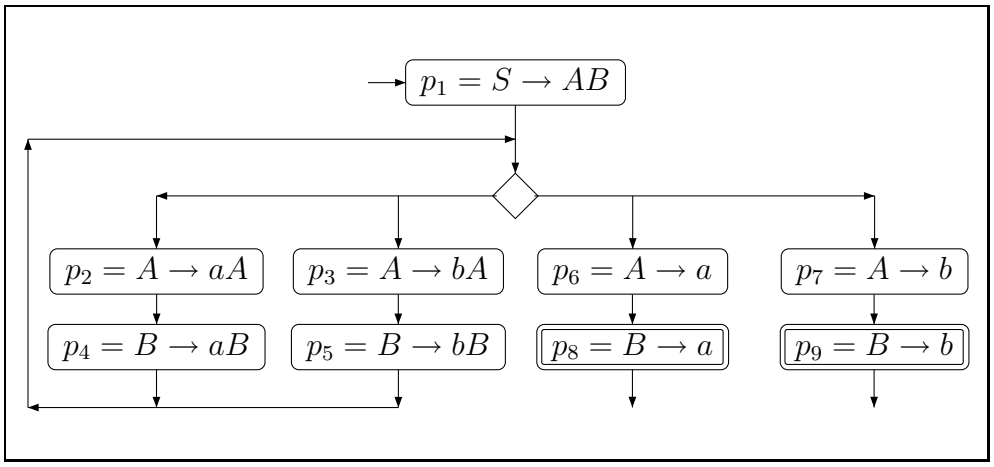


Figura 2.3: O diagrama de fluxo da gramática G_8.

Como fica evidente no diagrama, a "programação" desta gramática faz que duas cópias de uma mesma palavra sejam geradas lado a lado, letra a letra e de esquerda a direita; Finalmente, quando falta apenas uma letra para terminar a geração,

é escolhido um entre dois pares de produções para gerar a -mesma- última letra nas duas cópias da palavra que estiver sendo gerada.

Segundo a análise efetuada podemos agora planejar nossa prova por indução usando os circuitos do diagrama de fluxo acima para re-utilizar a derivação da hipótese indutiva no passo indutivo.

Vamos provar, por indução matemática, que $L(G_8) = \{ww : w \in \{a,b\}^+\}$.

Caso base. $|w| = 1$.
Com efeito, sem perda de generalidade podemos supor que $w = a$*:*

$$S \Rightarrow^{ac}_{p_1} AB \Rightarrow^{ac}_{p_6} aB \Rightarrow^{ac}_{p_8} aa.$$

Hipótese indutiva.
Seja $|w| > 1$ *e suponhamos que para toda palavra* $v \in \{a,b\}^+$*, com* $|v| < |w|$*, existe uma derivação segundo a gramática* G_8 *para* vv*, digamos:* $S \Rightarrow^* vv$.

Consideremos agora $w = ua$*. Sem perda de generalidade, podemos assumir que* $u = xa$*, com* $x \in \{a,b\}^*$*, pois o caso* $u = xb$ *é análogo. Pela hipótese indutiva, existe uma derivação de* uu *segundo a gramática:* $S \Rightarrow^* uu$*; do grafo acima sabemos que esta derivação deve ter o formato:*

$$S \Rightarrow^* xAxB \Rightarrow^{ac}_{p_6} xaxB \Rightarrow^{ac}_{p_8} xaxa = uu$$

Construímos agora uma derivação para ww*, baseada na derivação para* uu*:*

$$S \Rightarrow^* xAxB \Rightarrow^{ac}_{p_2} xaAxB \Rightarrow^{ac}_{p_4} xaAxaB = uAuB \Rightarrow^{ac}_{p_4} uauB \Rightarrow^{ac}_{p_8} uaua = ww.$$

Isto completa a indução e termina a prova.

Exemplo 2.3.3 Gramática programada $G_9 = (G, s, f)$ **para a linguagem** $L = \{a^{2^n} : n \geq 0\}$**.** *Consideremos a seguinte gramática livre de contexto programada* $G_9 = (\{S, A\}, \{a\}, P, S)$*, na qual o conjunto de produções e as funções de sucesso e falha são definidos como segue:*

p	$s(p)$	$f(p)$
$p_1 \quad = \quad S \to AA$	$\{p_1\}$	$\{p_2\}$
$p_2 \quad = \quad A \to S$	$\{p_2\}$	$\{p_1, p_3\}$
$p_3 \quad = \quad S \to a$	$\{p_3\}$	$\emptyset$

Derivação de a^4**:**

$$S \Rightarrow^{ac}_{p_1} AA \Rightarrow^{ac}_{p_1} AA \Rightarrow^{ac}_{p_2} SA \Rightarrow^{ac}_{p_2} SS \Rightarrow^{ac}_{p_2} SS \Rightarrow^{ac}_{p_1} AAS \Rightarrow^{ac}_{p_1} AAAA \Rightarrow^{ac}_{p_1} AAAA$$

$$\Rightarrow^{ac}_{p_2} ASAA \Rightarrow^{ac}_{p_2} ASAS \Rightarrow^{ac}_{p_2} ASSS \Rightarrow^{ac}_{p_2} SSSS \Rightarrow^{ac}_{p_2} SSSS \Rightarrow^{ac}_{p_3} aSSS \Rightarrow^{ac}_{p_3} aSSa$$
$$\Rightarrow^{ac}_{p_3} aSaa \Rightarrow^{ac}_{p_3} aaaa = a^4.$$

Vejamos o diagrama de fluxo desta gramática:

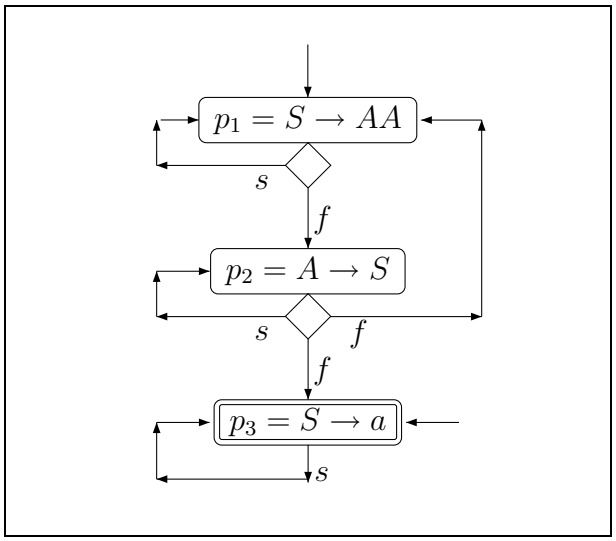


Figura 2.4: O diagrama de fluxo da gramática G_9.

Segundo o diagrama de fluxo acima, as derivações segundo a gramática G_9 são de dois tipos: o primeiro é simplesmente uma derivação formada por uma aplicação da produção p_3; no segundo tipo, cada derivação é formada por três laços; no primeiro laço é aplicada, uma ou mais vezes, a produção p_1, até que a aplicação da produção p_1 falhe, isto é, que a produção p_1 seja aplicada em modo de verificação de aparência; nesse caso começa o segundo laço, no qual é aplicada, uma ou mais vezes, a produção p_2, até que a aplicação da produção p_2 falhe, isto é, que a produção p_2 seja aplicada em modo de verificação de aparência. Nesse caso, começa o terceiro laço, no qual é aplicada uma ou mais vezes, a produção p_3.

Vamos provar, por indução matemática em n, que $L(G_9) = \{a^{2^n} : n \geq 0\}$.

Caso Base $m = 0$.
Com efeito, a produção p_3 em modo de verificação de aparência, permite derivar diretamente a:

$$S \Rightarrow^{ac}_{p_3} a = a^1 = a^{2^0}.$$

Hipótese indutiva: *Seja $n > 0$ e suponhamos que $a^{2^k} \in L(G_9)$, $(\forall k)(0 \leq k < n)$.*

Por hipótese indutiva, existe um derivação de $S^{2^{n-1}}$ *segundo* G_9*, digamos,* $S \Rightarrow^* S^{2^{n-1}}$*. Além disso, podemos obter uma derivação de "*SS*" da seguinte forma:*

$$S \Rightarrow^{ac}_{p_1} AA \Rightarrow^{ac}_{p_1} AA \Rightarrow^{ac}_{p_2} SA \Rightarrow^{ac}_{p_2} SS$$

Aplicando a hipótese indutiva a cada uma das duas ocorrências de S *na forma sentencial à direita:*

$$SS \Rightarrow^* a^{2^{n-1}} a^{2^{n-1}} = a^{2^{n-1}+2^{n-1}} = a^{2x2^{n-1}} = a^{2^n}$$

Isto completa a indução e termina a prova.

Observação 2.3.3 *Uma pequena modificação do exemplo anterior permite obter uma gramática programada* $G_9^{'}$ *tal que* $L(G_9^{'}) = \{a^{2^n} : n \geq 1\}$*. Com efeito, consideremos a seguinte gramática livre de contexto programada* $G = (\{S, A\}, \{a\}, P, S)$*, onde o conjunto de produções e as funções de sucesso e falha são definidos como segue:*

p	$s(p)$	$f(p)$
$p_1 \quad = \quad S \rightarrow AA$	$\{p_1\}$	$\{p_2, p_3\}$
$p_2 \quad = \quad A \rightarrow S$	$\{p_2\}$	$\{p_1\}$
$p_3 \quad = \quad A \rightarrow a$	$\{p_3\}$	$\emptyset$

Teorema 2.3.1 *Uma linguagem L é de tipo 0 se e somente se ela for gerada por uma gramática livre de contexto programada com verificação de aparência, i.e.*

$$\mathcal{RE} = \mathcal{P}_{ac}^{\lambda}.$$

Prova. A parte "se" é conseqüência da tese de Church.

Para provar a parte "somente se" consideremos uma linguagem L arbitrária ge-- rada pela gramática $G = (N, T, P, S)$. Provaremos que L é gerada, com verificação de aparência, por uma gramática livre de contexto programada $G_p = (G_1, s, f)$.

Podemos assumir, sem perda de generalidade, que $N \cup T$ possui $m - 1$ símbolos, com $m \geq 3$. Vamos codificar as palavras sobre $N \cup T$ como números, usando uma notação similar à notação m-ádica descrita na **nota 9.1** do livro de Salomaa[2]. Em outras palavras, estamos considerando uma função injetora g de $N \cup T$ no conjunto $\{1, \ldots, m-1\}$. Estendemos essa função definindo

$$g(x) = \begin{cases} 0 & \text{se } x = \lambda, \\ \Sigma_{i=1}^{n} g(a_i) m^{n-i} & \text{se } x = a_1 \ldots a_n. \end{cases}$$

assim a função g assim estendida é uma bijeção entre Σ^* e o conjunto dos inteiros não-negativos.

Ao simular G, a nossa gramática programada G_p vai trabalhar com os códigos das palavras ao invés de com as próprias palavras. Grosso modo isso é feito como segue. No inicio, G_p recebe a palavra $A^{g(S)}$ onde A é um não-terminal. Consideremos um passo

$$\alpha E \beta \Rightarrow \alpha D \beta, \qquad E \to D$$

em uma derivação segundo G. A entrada de G_p são $g(\alpha E \beta)$ cópias do não-terminal especial A; a gramática G_p pode, não-deterministicamente, separar uma palavra, e, a seguir, pode examinar uma palavra com $g(\alpha)$ ocorrências de A, $g(E)$ ocorrên-cias de B, e $g(\beta)$ ocorrências de C. Essa é uma decomposição possível; B e C são não-terminais. Agora G_p pode substituir as $g(E)$ ocorrências de B por $g(D)$ ocorrências de B. Então as $g(\alpha)$ ocorrências de A, $g(E)$ ocorrências de B, e $g(\beta)$ ocorrências de C são combinadas para criar $g(\alpha D \beta)$. Depois disso G_p pode separar novamente a palavra $\alpha D \beta$, ou então decodificar as $g(\alpha D \beta)$ ocorrências de A como a palavra $\alpha D \beta$.

O conjunto de produções de G_p está dividido em várias partes, denominadas *subrotinas*. Usamos diagramas de fluxo para descrever as interconexões entre as várias subrotinas. A expressão "saída" num campo "go-to" de uma subrotina indica que a subrotina seguinte é chamada. Uma seta apontando para uma produção indica que se ingressa na subrotina por essa produção.

Em nossas considerações, somente importa o número de não-terminais tais como A, B, C, mas não sua ordem. Para evitar repetir isso durante a prova, introduzimos a notação

$$E \Rightarrow^*: D,$$

o que significa que a subrotina em questão gera, a partir de qualquer palavra, onde o número de ocorrências de cada símbolo (que também ocorre em E) coincide com aqueles de E, uma palavra em que os números de ocorrências de cada letra coincide com aqueles em D. Por exemplo $A^2B^3 \Rightarrow^*: ABC$ significa que qualquer palavra com dois As e três Bs gera uma palavra com um A, um B e um C. Ocorrências de letras diferentes de A, B ou C permanecem inalteradas.

Para separar e recombinar palavras aplica-se um pouco de aritmética. Primeiro definimos duas subrotinas aritméticas. Para cada inteiro não negativo i, denotamos com $[i/m]$ a parte inteira na divisão de i por m; o resto dessa divisão é denotado com $res(i/m)$. Portanto

$$i/m = [i/m] + res(i/m)/m.$$

Definimos agora a subrotina $DIV(A, B)$:

			$s(p)$	$f(p)$	
$\rightarrow$	$d_1 =$	$A \rightarrow A'$	$\{d_2\}$	$\{d_{2m}\}$	
	$d_i =$	$A \rightarrow \lambda$	$\{d_{i+1}\}$	$\{d_{m+i-1}\}$	$i = 2, \ldots, m-1$
	$d_m =$	$A \rightarrow \lambda$	$\{d_1\}$	$\{d_{2m-1}\}$	
	$d_{m+i} =$	$A \rightarrow B^i$	$\{d_{2m}\}$	$\emptyset$	$i = 1, \ldots, m-1$
	$d_{2m} =$	$A' \rightarrow A$	$\{d_{2m}\}$	saída	

Pode-se verificar que $DIV(A, B)$ satisfaz, para todo inteiro não-negativo i,

$$A^i \Rightarrow^*: A^{[i/m]}B^{res(i/m)}$$

Note-se que m é o número fixado pela cardinalidade do alfabeto $N \cup T$, e que A, B são parâmetros, ou seja, poderíamos considerar $DIV(C, D)$.

Agora definimos uma outra subrotina aritmética $MULT(A, B, C)$?

			$s(p)$	$f(p)$
$\rightarrow$	$e_1 =$	$A \rightarrow A'A''$	$\{e_1\}$	$\{e_2\}$
	$e_2 =$	$A'' \rightarrow \lambda$	$\{e_3\}$	$\{e_5\}$
	$e_3 =$	$B \rightarrow B'C$	$\{e_3\}$	$\{e_4\}$
	$e_4 =$	$B' \rightarrow B$	$\{e_4\}$	$\{e_2\}$
	$e_5 =$	$A' \rightarrow A$	$\{e_5\}$	saída

Essa subrotina multiplica quaisquer números inteiros não negativos i e j fazendo:

$$A^iB^j \Rightarrow^*: A^iB^jC^{ij}.$$

A seguir listamos algumas subrotinas úteis. Em cada caso, damos o nome, seguido pela semântica da subrotina e finalmente a descrevemos como uma gramática programada.

$ER(A)$ apaga todas as ocorrências da letra A. $CH(A)$ verifica a ocorrência ou não da letra A, sem mudar nada. A subrotina seguinte depende do resultado dessa verificação. $G_1(A,B)$ satisfaz, para todo $i \geq 1$, $A^i \Rightarrow^* A^iB$. $G_2(A,B,C)$ satisfaz, para todo $i \geq 1$, $A^i \Rightarrow^* A^iB^iC$. $G_3(A)$ satisfaz, para todo $i \geq 1$, $A^i \Rightarrow^* A^{mi}$. $G_4(A,B)$ troca todas as letras A por B. G_5 satisfaz $S \rightarrow \#A^{g(S)}$.

$Nome$	p	$s(p)$	$f(p)$
$ER(A)$	$p_1 = A \rightarrow \lambda$	$\{p_1\}$	saída
$CH(A)$	$p_2 = A \rightarrow A$	$\{p_1\}$	saída
$G_1(A,B)$	$p_3 = A \rightarrow AB$	$\{p_1\}$	saída
$G_2(A,B,C)$	$\rightarrow \quad p_4 = \quad A \rightarrow AC$	$\{p_5\}$	saída
	$\quad\quad p_5 = \quad A \rightarrow A'$	$\{p_5\}$	$\{p_6\}$
	$\quad\quad p_6 = \quad A \rightarrow AB$	$\{p_6\}$	saída
$G_3(A)$	$\rightarrow \quad p_7 = \quad A \rightarrow A'$	$\{p_5\}$	$\{p_8\}$
	$\quad\quad p_8 = \quad A \rightarrow A^m$	$\{p_6\}$	saída
$G_4(A,B)$	$p_9 = A \rightarrow B$	$\{p_9\}$	saída
G_5	$p_{10} = S \rightarrow \#A^{g(S)}$	saída	$\emptyset$

$G_6(A,B)$ satisfaz, para todas as palavras w, z sobre $(N \cup T)$, com $wz \neq \lambda$,

$$A^{g(wz)} \Rightarrow^* A^{g(w)}B^{g(z)}$$

ela está baseada na equação

$$g(wz) = g(w)m^{|z|} + g(z)$$

que é válida para quaisquer palavras w e z sobre $(N \cup T)$; em particular, para uma letra $a \in (N \cup T)$: $g(za) = g(z)m + g(a)$ isto é, $g(w)$ e $g(a)$ são, respectivamente, a parte inteira e o resíduo, na divisão de $g(wa)$ por m. Pela esclolha de m e da definição de g, $1 \leq g(a) < m$. Se, ao invés de fazermos isso, tivéssemos escolhido usar notação m-ádica, existiria a possibilidade de $g(a) = m$ e deveriam que ser feitas mudanças na equação da divisão e na subrotina $DIV(A,B)$

A subrotina $G_6(A,B)$ é definida pela Fig. 2.5. Dada $A^{g(u)}$, a subrotina separa u em duas palavras $u = wz$. Inicialmente $z = \lambda$. O laço no diagrama de fluxo toma, sucessivamente, um símbolo da extremidade direita de w e o anexa a z. O bloco $DIV(A,D)$ recebe uma palavra da forma

$$A^{g(wa)}B^{g(z)}E^{m^{|z|}}$$

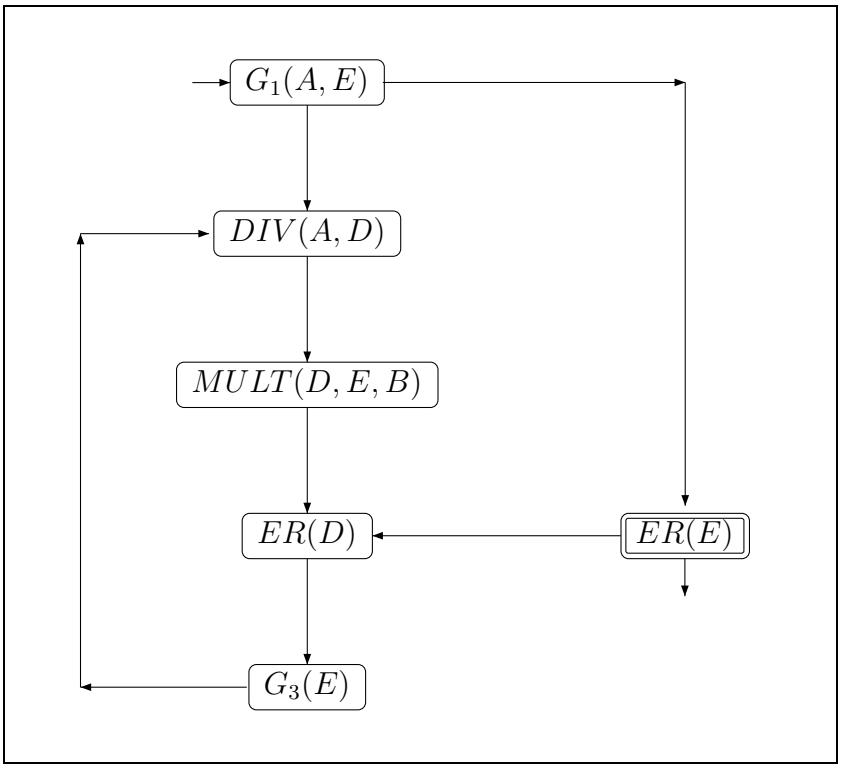


Figura 2.5: A subrotina $G_6(A,B)$.

sendo, inicialmente, $z = \lambda$.

A subrotina $DIV(A,D)$ transforma essa palavra, no sentido da relação $\Rightarrow^*$, em: $A^{g(w)}D^{g(a)}B^{g(z)}E^{m^{|z|}}$. O bloco seguinte produz a seguinte igualdade no sentido da relação $\Rightarrow^*$:

$$A^{g(w)}D^{g(a)}B^{g(z)}E^{m|z|}B^{g(a)m^{|z|}} = A^{g(w)}D^{g(a)}E^{m^{|z|}}B^{g(z)}$$

depois disso, os Ds são apagados. A seguir, os Es são apagados e com isso a subrotina termina ou então volta-se ao bloco $DIV(A,D)$ com a palavra

$$A^{g(w)}B^{g(az)}E^{m^{a|z|}},$$

reiniciando o laço. Note-se que o caminho dirigido de $G_1(A,E)$ a $ER(E)$ torna possível que Q seja vazia.

A subrotina $G_7(A,B)$, definida na Fig. 2.6, produz o efeito inverso ao de G_6: Para duas palavras w, z sobre $N \cup T$, a subrotina $G_7(A,B)$ satisfaz

$$A^{g(w)}B^{g(az)} \Rightarrow^*: A^{g(wz)}.$$

Se não houver Bs, então $z = \lambda$ e saímos sem mudar o expoente de A. Caso ccontrário, inicia-se o laço que produz $E^{m^{|z|}}$, isto é, entra-se no bloco $MULT(A, E, C)$, com a palavra $A^{g(w)}B^{g(z)}E^{m^{|z|}}$.

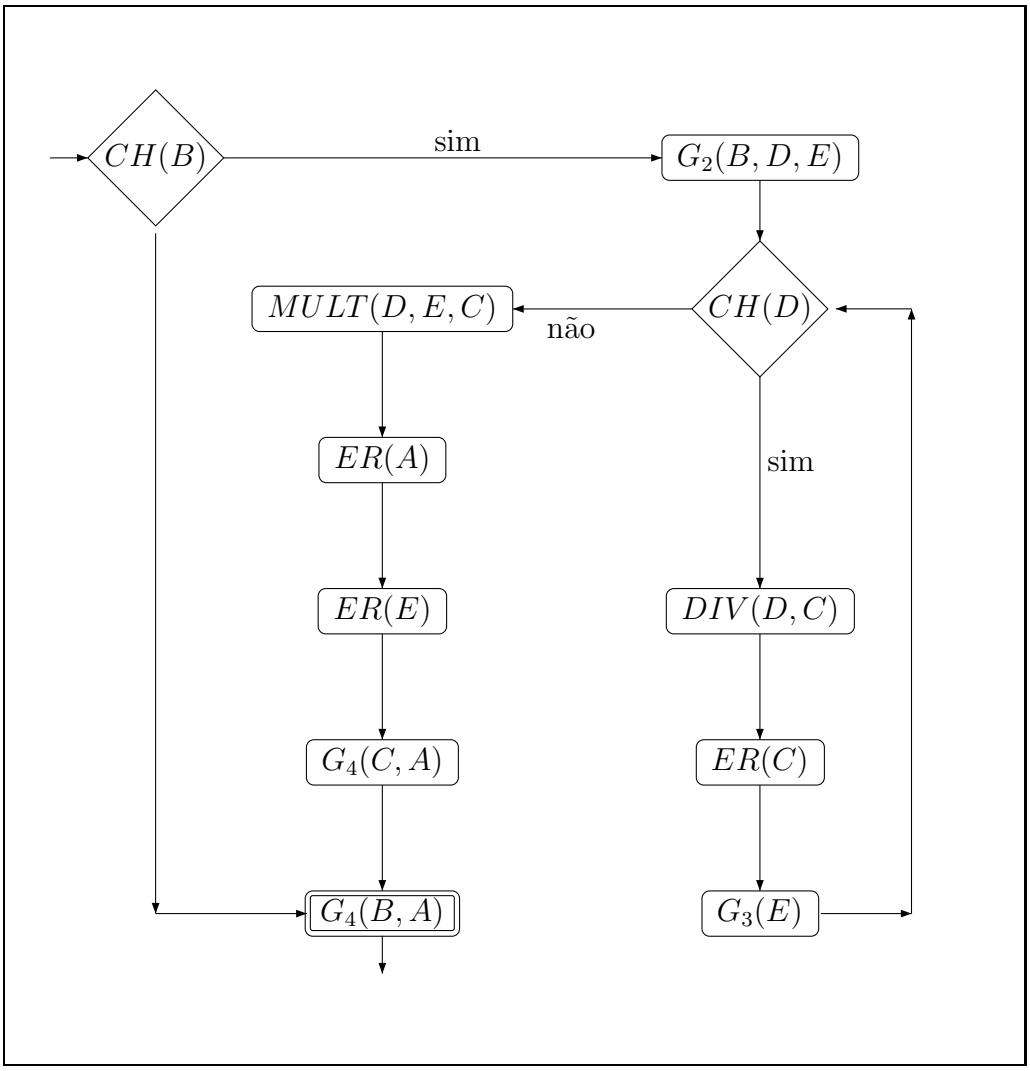


Figura 2.6: A subrotina $G_7(A, B)$.

Depois desse bloco, o número de Cs coincide com $g(w)m^{|w|}$ e o número de Bs coincide com $g(z)$. Considerando que os Bs e os Cs são convertidos em As, obtemos, no total, $g(wz)$ ocorrências de A.

A subrotina G_8 produz o símbolo w a partir da palavra $\#A^{g(w)}$, para todo $w \in (N \cup T)^*$. Nesse caso, estamos considerando a relação usual $\Rightarrow$ ao invés de $\Rightarrow^*$:. Assim, G_8 decodifica palavras. O conjunto das produções de G_8 contém as

produções de $DIV(A,B)$, em que cada h_1^1 é o campo falha de d_{2m}, e das seguintes produções:

			sucesso	*falha*	
$\rightarrow$	$h_1^1 =$	$B \rightarrow \lambda$	$\{h_2^1\}$	$\{h_1^5\}$	
	$h_i^1 =$	$B \rightarrow \lambda$	$\{h_{i+1}^1\}$	$\{h_{i-1}^2\}$	$i = 2, \ldots, m-1$
	$h_m^1 =$	$B \rightarrow \lambda$	$\emptyset$	$\{h_{m-1}^2\}$	
	$h_i^2 =$	$A \rightarrow A$	$\{h_1^3\}$	$\{h_i^4\}$	$i = 1, \ldots, m-1$
	$h_i^3 =$	$\# \rightarrow \#g^{-1}(i)$	$\{d_1\}$	$\emptyset$	$i = 1, \ldots, m-1$
	$h_i^4 =$	$\# \rightarrow g^{-1}(i)$	$\emptyset$	$\emptyset$	$i = 1, \ldots, m-1$
	$h_i^5 =$	$\# \rightarrow \lambda$	$\emptyset$	$\emptyset$	

A parte da subrotina G_8 formada por $DIV(A,B)$ recebe uma palavra da forma

$$\#QA^{g(xa)}, \qquad a \in (N \cup T), w = xay.$$

e a converte na palavra

$$\#QA^{g(x)}B^{g(a)}.$$

A produção $\{h_j^1\}$ apaga os Bs, mas o seu número i é retido pelos diferentes campos falha. A produção $\{h_j^2\}$ verifica se existem ou não As, conservando ainda o número de Bs. Então $a = g^{-1}(i)$ é escrito à direita de #. Se não houver mais As então a produção $\{h_j^4\}$ apaga também #. Caso contrario aplica-se a produção $\{h_j^3\}$. Depois se volta a entrar no laço, com a palavra

$$\#aQA^{g(x)},$$

em que $x = x_1 b$ para algum $b \in (N \cup T)$. Se, no inicio, $x = \lambda$, e, conseqüentemente G_8 recebe a palavra # como dado, então a seqüência de produções $d_1, d_{2m}, h_1^1, h_1^5$ produz λ. Esse é o único caso em que a aplicação de h_1^1 não tem sucesso.

A última subrotina G_9 satisfaz, para qualquer produção $M \rightarrow R \in P$, a relação

$$B^{g(M)} \Rightarrow^* B^{g(R)}.$$

Definimos

$$u = max\{g(M) : M \rightarrow R \in P\}$$

e introduzimos etiquetas $t_1^1, t_2^1, \ldots, t_v^1$ para cada uma das produções de P. Para cada $i = 1, \ldots, u$; seja $T(i)$ o conjunto (possivelmente vazio) das etiquetas daquelas produções em P com lado esquerdo igual a $g^{-1}(i)$. Seja R_i, $i = 1, \ldots, v$ o lado direito da produção com etiqueta t_i^1. A subrotina G_9 é formada pelas seguintes produções:

			sucesso	*falha*	
	$t_i^1 =$	$D \rightarrow B^{g(R_i)}$	saída	$\emptyset$	$i = 1, \ldots, v$
$\rightarrow$	$t_1^2 =$	$B \rightarrow D$	$\{t_2^2\}$	$\emptyset$	
	$t_i^2 =$	$B \rightarrow \lambda$	$\{t_{i+1}^2\}$	$T(i-1)$	$i = 2, \ldots, u$
	$t_{u+1}^2 =$	$B \rightarrow \lambda$	$\emptyset$	$T(u)$	

Como o lado esquerdo das produções em P é diferente de λ, a aplicação de t_1^2 sem-

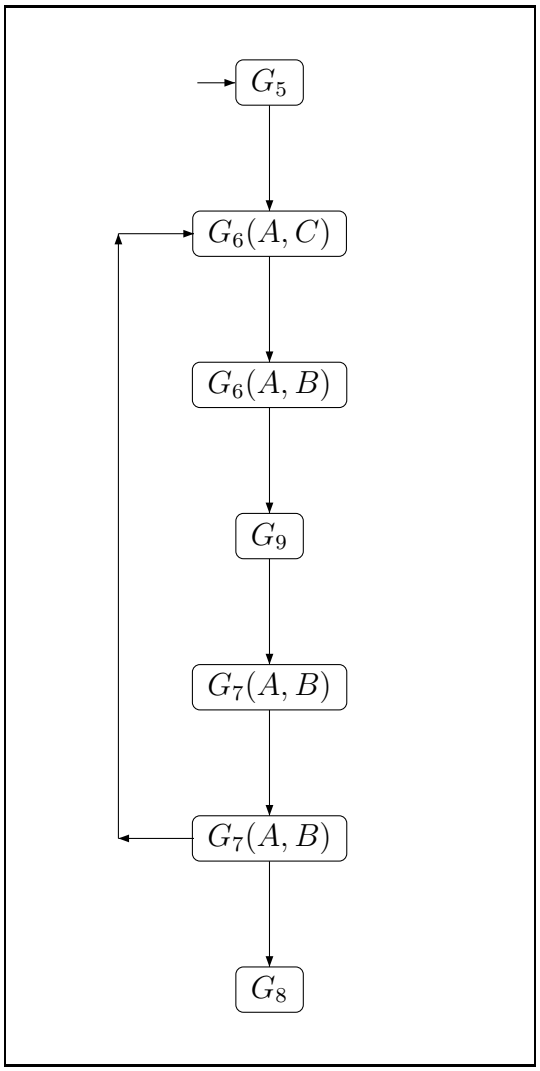


Figura 2.7: A gramática programada $G_p = (N, T, P_p, S_p)$.

pre tem sucesso; as produções t_i^2 apagam os Bs, conservam seu número, e depois re-escrevem o D como uma potência de B.

A gramática programada $G_p = (N, T, P_p, S_p)$ está definida, em termos das subrotinas acima, na Fig. 2.7. O alfabeto de não-terminais é $N = \{A, B, C\}$.

A gramática G_p inicia sua operação invocando a subrotina G_5. Inicialmente, faz

$S = z$; a seguir a subrotina $G_6(A, C)$ recebe como dado uma palavra da forma

$$\#A^{g(z)}, \qquad z \neq \lambda$$

onde $S \Rightarrow^* z$ segundo a gramática G. As duas chamadas à subrotina G_6 produzem, não-deteterministicamente, no sentido de da relação $\Rightarrow^*$:, a palavra

$$\#A^{g(x)}B^{g(w)}C^{g(y)}, \qquad z = xwy.$$

Se w está no lado direito de alguma produção de G, digamos $w \to r \in P$, então a subrotina G_9 converte a palavra acima na palavra

$$\#A^{g(x)}B^{g(r)}C^{g(y)}.$$

As duas chamadas à subrotina G_7 produzem a palavra

$$\#A^{g(xry)} = \#A^{g(z_1)}$$

onde $z \to z_1$ segundo G e conseqüêntemente $S \Rightarrow_G^* z_1$. Nesse ponto, se z_1 contém pelo menos um não-terminal, o laço deve ser reiniciado. No caso de z_1 conter apenas terminais, então z_1 é decodificada pela chamada da subrotina G_8.
□

Teorema 2.3.2 *Toda linguagem L de tipo 0 é imagem homomórfica de alguma linguagem em $\mathcal{P}_{ac}$.*

Prova. Seja L uma linguagem de tipo 0 sobre T. Pelo teorema 2.3.1 existe uma gramática livre de contexto programada (G, s, f) tal que

$$L = L_{ac}(G, s, f)$$

Seja (G_1, s_1, f_1) uma gramática livre de contexto programada obtida de (G, s, f) adicionando o novo terminal c e substituindo toda produção $A \to \lambda$ por $A \to c$.

Seja h o homomorfismo de $T \cup \{c\}$ em T^* definido por

$$h(a) = \begin{cases} \lambda, \text{ se } a = c \\ a, \text{ se } a \in T \end{cases}$$

então

$$L = h(L_{ac}(G, s, f)).$$

□

Teorema 2.3.3 *Cada uma das famílias das gramáticas matriciais está incluida na família correspondente das gramáticas programadas, isto é,*

$$\mathcal{M} \subseteq \mathcal{P}, \qquad \mathcal{M}_{ac} \subseteq \mathcal{P}_{ac}, \qquad \mathcal{M}^{\lambda} \subseteq \mathcal{P}^{\lambda}, \qquad \mathcal{M}^{\lambda}_{ac} \subseteq \mathcal{P}^{\lambda}_{ac}.$$

Prova. Consideramos primeiro as duas primeiras inclusões. Seja $G = (N, T, M, S)$ uma gramática matricial sem produções-λ e F um subconjunto do conjunto P de todas as produções nas matrizes de G. Vamos definir uma gramática livre de contexto programada sem produções-λ (G_1, s, f) tal que

$$L_{ac}(G, F) = L_{ac}(G_1, s, f).$$

Mais ainda, se $F = \emptyset$, então também a imagem de P por f é $\emptyset$.

Introduzimos $N_1 = \{X_a : a \in T\}$ e consideramos os conjuntos P_1 igual ao conjunto de todas as produções obtidas de P, substituindo por $a \in T$ por X_a e

$$P_2 = \{X_a \to a : a \in T\}$$

Definimos agora a gramática livre de contexto e sem produções-λ: $G_1 = (N \cup N_1, T, P_1 \cup P_2, S)$.

Para definir as funções s, f consideremos a matriz arbitrária

$$[p_1 = E_1 \to D_1, \ldots p_n = E_n \to D_n,], n \geq 1$$

no conjunto M; seja H o subconjunto de $P_1 \cup P_2$ formado pelas primeiras produções nas matrizes de M. Definimos as funções sucesso e falha:

$$s(p) = \begin{cases} \{p_{j+1}\}, & \text{se } p = p_j, 1 \leq j < n, \\ H \cup P_2, & \text{se } p = p_n, \\ P_2, & \text{se } p \in P_2. \end{cases}$$

$$f(p) = \begin{cases} \{p_{j+1}\}, & \text{se } p = p_j \in F, 1 \leq j < n, \\ \emptyset, & \text{se } p = p_j \notin F, 1 \leq j < n, \\ s(p_n), & \text{se } p = p_n \in F, \\ \emptyset, & \text{se } p = p_n \notin F. \\ \emptyset, & \text{se } p \in P_2. \end{cases}$$

Por construção $L_{ac}(G, F) = L_{ac}(G_1, s, f)$ e, se $F = \emptyset$, então a imagem da função falha é $\emptyset$. Isso prova as duas primeiras inclusões.

Consideramos agora as duas últimas inclusões. Nestes casos, a gramática $G =$

(N, T, M, S) é uma gramática matricial que pode possuir produções-λ e F um subconjunto do conjunto P de todas as produções produções nas matrizes de G. Introduzimos dois novos não-terminais $N_1 = \{S', Y\}$ e um par de novas produções:

$$P_1 = \{S' \to SY, Y \to \lambda\}$$

Vamos definir agora $G_1 = (N \cup N_1, T, P \cup P_1, S')$: Para definir as funções s, f consideremos a matriz arbitrária

$$[p_1 = E_1 \to D_1, \ldots p_n = E_n \to D_n,], n \geq 1$$

no conjunto M; seja H o subconjunto de $P \cup P_1$ formado pelas primeiras produções nas matrizes de M. Definimos as funções sucesso e falha:

$$s(p) = \begin{cases} \{p_{j+1}\}, & \text{se } p = p_j, 1 \leq j < n, \\ H \cup P_1, & \text{se } p = p_n, \\ \{p\}, & \text{se } p = S' \to SY, \\ H, & \text{se } p = Y \to \lambda. \end{cases}$$

$$f(p) = \begin{cases} \{p_{j+1}\}, & \text{se } p = p_j \in F, 1 \leq j < n, \\ \emptyset, & \text{se } p = p_j \notin F, 1 \leq j < n, \\ s(p_n), & \text{se } p = p_n \in F, \\ \emptyset, & \text{se } p = p_n \notin F. \\ \emptyset, & \text{se } p \in P_1. \end{cases}$$

Por construção $L_{ac}(G, F) = L_{ac}(G_1, s, f)$ e, se $F = \emptyset$, então a imagem da função falha é $\emptyset$. Isso prova as duas últimas inclusões.
□

Nesta seção, foram introduzidas as gramáticas programadas, baseadas em gramáticas tipo i, $0 \leq i \leq 3$, segundo a hierarquia de Chomsky, e tais que as produções que podem ser aplicadas, em cada passo de uma derivação, dependem da produção, aplicada no passo anterior, ter sido aplicada no sentido das gramáticas na hierarquia de Chomsky ou de ter sido aplicada em modo de verificação de aparência. Essas alternativas são expressadas definindo, para cada produção, as funções de *sucesso* e *falha* que tomam como valores subconjuntos do conjunto das produções da gramática; em particular, gramáticas programadas com função falha constante, e igual ao conjunto vazio, são denominadas *gramáticas programadas sem verificação de aparência.* Essas funções de sucesso e falha constituem o terceiro mecanismo de controle estudado neste capítulo. Todos esses conceitos foram ilustrados com diversos exemplos. Foi definida uma notação para o caso da gramáticas programadas baseadas em gramáticas livres de contexto, e foi provado que as gramáticas livres de contexto programadas identificam a classe das linguagens recursivamente enumeráveis; também foi provado que as classes das linguagens geradas pelas gramáticas livres de contexto matriciais, estão contidas nas classes das linguagens geradas pelas gramáticas livres de contexto programadas.

2.4 Gramáticas com Linguagem de Controle

Gramáticas com linguagem de controle[51] são baseadas em gramáticas de tipo i, $0 \leq i \leq 3$, segundo a hierarquia de Chomsky e possuem, como mecanismo de controle, uma outra linguagem sobre o alfabeto formado pelo conjunto das produções da gramática, à qual deve pertencer, como palavra, a concatenação das produções aplicadas durante uma derivação. Essa segunda linguagem (gerada por uma outra gramática de tipo j, $0 \leq j \leq 3$, segundo a hierarquia de Chomsky) é denominada *linguagem de controle* e é o quarto – e último – mecanismo de controle que estudamos neste capítulo. Outros mecanismos de controle podem ser consultados nas obras de Salomaa[2], e Dassow e Paun[3], [4]. Nesta seção são definidos os conceitos básicos das gramáticas com linguagem de controle, a noção de *aplicação de produção em modo de verificação de aparência*, estudada nas três seções anteriores, é redefinida para o caso das gramáticas com linguagem de controle, é definida uma notação para o caso das gramáticas com linguagem de controle regular baseadas em gramáticas livres de contexto, a noção de gramática de tipo i é estendida para tipo (i, j, k) onde i é o tipo da gramática *controlada*, j é o tipo da *linguagem de controle*, $k = 1$ indica que existem produções a serem aplicadas em modo de verificação de aparência, e $k = 0$ indica que nenhuma produção é aplicada em modo de verificação de aparência. Prova-se, em forma construtiva, que toda linguagem gerada por uma gramática matricial, periódica variante no tempo ou programada com o sem verificação de aparência pode ser gerada por uma gramática com linguagem de controle regular com o sem verificação de aparência. Também é provado, em forma construtiva, que as gramáticas livres de contexto com conjunto de controle regular e com o sem verificação de aparência podem ser geradas por gramáticas livres de contexto matriciais, que possuem duas produções em cada matriz. É provado também que as gramáticas livres de contexto, matriciais, periódicas variantes no tempo, programadas e com conjunto de controle regular, com verificação de aparência identificam a classe das linguagens recursivamente enumeráveis.

Definição 2.4.1 *Seja $G = (N, T, P, S)$ uma gramática e $F \subseteq P$ um subconjunto das produções de G. Seja $\mathcal{D}$ uma derivação segundo G e u uma palavra sobre P. Então diz-se que u é uma palavra de controle para $\mathcal{D}$ se e somente se uma única das seguintes condições for satisfeita:*

(i) para algumas formas sentenciais $r, s \in (N \cup T)^*$, $\mathcal{D}$ é a derivação $X \Rightarrow^* Y$, onde $X = rEs$ e $Y = rDs$ e $u = E \to D \in P$,

(ii) para algumas formas sentenciais $r, s \in (N \cup T)^*$, $\mathcal{D}$ é a derivação que consiste apenas da palavra w e $u = \lambda$ ou então $u = E \to D \in F$ e E não aparece em w.

(iii) para algumas formas sentenciais X, Y, Z, temos que $\mathcal{D}$ é a derivação

$$X \Rightarrow^* Y \Rightarrow^* Z,$$

$u = u_1u_2$, e u_1 é uma palavra de controle da derivação $X \Rightarrow^* Y$ e u_2 é uma palavra de controle para a derivação $Y \Rightarrow^* Z$.

Definição 2.4.2 *Seja C uma linguagem sobre o alfabeto P. A linguagem gerada por G, com linguagem de controle C e com verificação de aparência para produções em F, é definida como:*

$$L_{ac}(G, C, F) = \{w \in T^* : \text{ existe uma palavra de controle } u \in C, \forall S \Rightarrow^* w\}.$$

Se $F = \emptyset$ então também dizemos que a linguagem é gerada com linguagem de controle C, e a denotamos $L(G, C)$.

A seguintes duas definições estendem o conceito de "linguagem de tipo i na hierarquia de Chomsky" para considerar o caso de linguagens de controle e verificação de aparência. O "tipo estendido" será uma terna (i, j, k) onde o primeiro componente indica o tipo, na hierarquia de Chomsky, da linguagem que "está sendo controlada", o segundo componente indica o tipo, na hierarquia de Chomsky, da linguagem de controle e terceiro componente indica se existem produções a serem aplicadas em modo de verificação de aparência.

Definição 2.4.3 *Uma linguagem é de tipo $(i, j, 1)$ se e somente se é da forma $L_{ac}(G, C, F)$ onde G é uma gramática de tipo i e C é uma linguagem de tipo j.*

Definição 2.4.4 *Uma linguagem é de tipo $(i, j, 0)$ se e somente se é da forma $L(G, C)$ onde G é uma gramática de tipo i e C é uma linguagem de tipo j. No caso de $i = 3$, a gramática G pode ser linear à esquerda ou à direita.*

Notação 2.4.1 *A família de todas as linguagens de tipo (i, j, k) é denotada como:*

$$\mathcal{L}(i, j, k), \qquad i, j \in \{0, 1, 2, 3\} \text{ e } k \in \{0, 1\}$$

Definição 2.4.5 *Linguagens e gramáticas livres de contexto sem produções-λ são denominadas, abreviadamente, como linguagens e gramáticas de tipo $2 - \lambda$.*

1. *Uma linguagem é de tipo $\mathcal{L}(2-\lambda, j, 1)$ se e somente se for da forma $L_{ac}(G, C, F)$ onde G é uma gramática de tipo $2 - \lambda$ sendo C uma linguagem de tipo j.*

2. *Uma linguagem é de tipo* $\mathcal{L}(2-\lambda, j, 0)$ *se e somente se for da forma* $L(G, C)$ *onde* G *é uma gramática de tipo* $2-\lambda$ *sendo* C *uma linguagem de tipo* j.

Exemplo 2.4.1 Gramática com linguagem de controle regular $G_{10} = (G, C)$ **para a linguagem** $L = \{a^n b^n c^n : n \geq 1\}$**.** *Consideremos a seguinte gramática com linguagem de controle regular e sem verificação de aparência:*

$$G_{10} = (\{S, A, C\}, \{a, b, c\}, \{p_1, p_2, p_3, p_4, p_5\}, S, \{p_1(p_2p_3)^n p_4 p_5 : n \geq 0\}, \emptyset)$$

com

$$p_1 = S \to AC, p_2 = A \to aAb, p_3 = C \to cC, p_4 = A \to ab, p_5 = C \to c.$$

Derivação de $a^3b^3c^3$*:*
Com efeito:

$$S \Rightarrow^{ac}_{p_1} AC \Rightarrow^{ac}_{p_2} aAbC \Rightarrow^{ac}_{p_3} aAbcC \Rightarrow^{ac}_{p_2} aaAbbcC \Rightarrow^{ac}_{p_3} aaAbbccC$$

$$\Rightarrow^{ac}_{p_4} aaabbbccC \Rightarrow^{ac}_{p_5} aaabbbccc.$$

Vamos mostrar, por indução matemática, que $L(G_{10}) = \{a^n b^n c^n : n \geq 1\}$.

Caso base: $n = 1$
Como antes

$$S \Rightarrow^{ac}_{p_1} AC \Rightarrow^{ac}_{p_4} abC \Rightarrow^{ac}_{p_5} abc.$$

Hipótese indutiva: *Seja* $n > 1$ *e suponhamos que* $(\forall k)(1 \leq k < n), a^k b^k c^k \in L(G_{10})$.

Consideremos $a^n b^n c^n$*. Pela hipótese indutiva existe uma derivação para* $a^{n-1}b^{n-1}c^{n-1}$*, digamos* $S \Rightarrow^* a^{n-1}b^{n-1}c^{n-1}$*. Como* G *deriva segundo palavras regulares sobre o alfabeto de suas produções, então:*

$$S \Rightarrow^{ac}_{p_1} AC \Rightarrow^{ac}_{p_2} aAbC \Rightarrow^{ac}_{p_3} aAbcC \cdots \Rightarrow^{ac}_{p_2} a^{n-2}Ab^{n-2}c^{n-3}C \Rightarrow^{ac}_{p_3} a^{n-2}Ab^{n-2}c^{n-2}C$$
$$\Rightarrow^{ac}_{p_4} a^{n-1}b^{n-1}c^{n-2}C \Rightarrow^{ac}_{p_5} a^{n-1}b^{n-1}c^{n-1}$$

Ficando apenas com a primeira parte desta derivação (que inclui as repetições de p_2 *e* p_3*) podemos obter uma derivação para* $a^n b^n c^n$*, do modo seguinte:*

$$S \Rightarrow^{ac}_{p_1} AC \Rightarrow^{ac}_{p_2} aAbC \Rightarrow^{ac}_{p_3} aAbcC \cdots \Rightarrow^{ac}_{p_2} a^{n-2}Ab^{n-2}c^{n-3}C \Rightarrow^{ac}_{p_3} a^{n-2}Ab^{n-2}c^{n-2}C$$
$$\Rightarrow^{ac}_{p_2} a^{n-1}Ab^{n-1}c^{n-2}C \Rightarrow^{ac}_{p_3} a^{n-1}Ab^{n-1}c^{n-1}C \Rightarrow^{ac}_{p_4} a^n b^n c^{n-1}C \Rightarrow^{ac}_{p_5} a^n b^n c^n.$$

A idéia é, simplesmente, fazer mais uma iteração das repetições de p_2p_3.

Observação 2.4.1 *Já que as produções aplicadas em uma derivação seguem uma expressão regular, é conveniente, para fins de visualização do processo de indução matemática, considerar um grafo orientado, em que cada aresta é etiquetada pela transição correspondente na expressão regular; considerando que se trata de substituir o lado esquerdo da produção pelo seu lado direito, optou-se por representar a produção $p = A \to B$, na forma A/B. A idéia é olhar para esse grafo como se fosse o autômato finito que representa a expressão regular; reforçamos essa idéia utilizando a notação usual de autômatos finitos para denotar os estados inicial e final, querendo significar onde deve começar e terminar uma derivação. Veja a Fig. 2.8.*

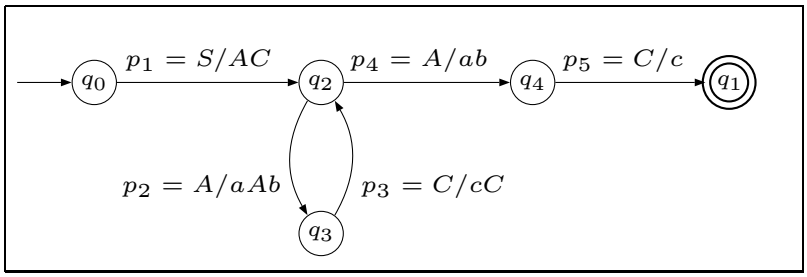


Figura 2.8: O autômato finito que aceita $p_1(p_2p_3)^*p_4p_5$ associado com G_{10}.

Observação 2.4.2 *Uma pequena modificação do exemplo anterior permite obter uma outra gramática com conjunto de controle regular $G_{10}^{'}$ tal que $L(G_{10}^{'}) = \{a^nb^nc^n : n \geq 1\}$ Com efeito, consideremos a seguinte gramática com linguagem de controle regular:*

$$G_{10}^{'} = (\{S, A, B, C\}, \{a, b, c\}, \{p_1, p_2, p_3, p_4, p_5\}, S, \{p_1(p_2p_3p_4)^np_5p_6p_7 : n \geq 1\}, \emptyset)$$

com

$$p_1 = S \to ABC, p_2 = A \to aA, p_3 = B \to bB, p_4 = C \to cC,$$
$$p_5 = A \to a, p_6 = B \to b, p_7 = C \to c.$$

Neste caso, as derivações segundo a gramática $G_{10}^{'}$, são formadas pela aplicação da produção p_1, seguida por um laço, onde são aplicadas, as produções p_2, p_3 e p_4 nessa ordem e no qual, cada vez que a produção p_2 pode ser aplicada, é possível optar por aplicar a produção p_4 e, neste caso, a derivação termina com a aplicação das produções p_5, p_6 e p_7, nessa ordem.

Exemplo 2.4.2 Gramática com linguagem de controle regular $G_{11} = (G, C)$ para a linguagem $L = \{ww : w \in \{a, b\}^+\}$. *Consideremos a gramática com linguagem de controle regular sem verificação de aparência:*

$$G_{11} = (\{S, A, B\}, \{a, b, c\}, P, S, \{p_1(p_2p_6|p_3p_7)^n(p_5p_9|p_4p_8) : n \geq 0\}, \emptyset)$$

com

$$P = \{p_1 = S \to AB, p_2 = A \to aA, p_3 = A \to bA, p_4 = A \to a, p_5 = A \to b,$$
$$p_6 = B \to aB, p_7 = B \to bB, p_8 = B \to a, p_9 = B \to b.\}$$

Derivação de *bbaabbbaab: Com efeito:*

$$S \Rightarrow^{ac}_{p_1} AB \Rightarrow^{ac}_{p_3} bAB \Rightarrow^{ac}_{p_7} bAbB \Rightarrow^{ac}_{p_3} bbAbB \Rightarrow^{ac}_{p_7} bbAbbB \Rightarrow^{ac}_{p_2} bbaAbbB$$
$$\Rightarrow^{ac}_{p_6} bbaAbbaB \Rightarrow^{ac}_{p_2} bbaaAbbaB \Rightarrow^{ac}_{p_6} bbaaAbbaaB$$
$$\Rightarrow^{ac}_{p_5} bbaabbbaaB \Rightarrow^{ac}_{p_8} bbaabbbaab.$$

Para provar que a gramática gera exatamente a linguagem dos palíndromos pares vamos usar nossa representação gráfica. Neste caso o autômato finito é mostrado na Fig. 2.9:

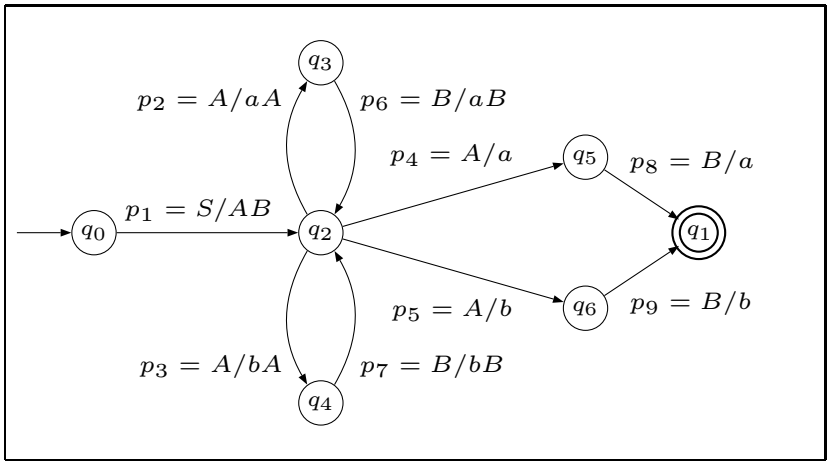


Figura 2.9: Autômata finito que aceita $p_1(p_2p_6|p_3p_7)^*(p_5p_9|p_4p_8)$ associado com G_{11}.

Do autômato da Fig 2.9 segue que, antes de aplicar qualquer um dos dois pares de produções finais, a gramática gera formas sentenciais do tipo: $wAwB$, onde w é uma palavra de $\{a, b\}^+$. Daqui a indução no comprimento de w pode ser planejada considerando dois casos:

1. *$w = ua$, ou*
2. *$w = ub$.*

Sem perda de generalidade, podemos considerar apenas o caso (1), pois o caso (2) é similar.

Caso base. $|w| = 1$.

Com efeito:

$$S \Rightarrow^{ac}_{p_1} AB \Rightarrow^{ac}_{p_4} aB \Rightarrow^{ac}_{p_8} aa.$$

Hipótese indutiva. *Seja $|w| > 1$ e suponhamos que para toda palavra $v \in \{a,b\}^+$, com $|v| < |w|$, existe uma derivação segundo a gramática para $vv : S \Rightarrow^* vv$.*
Consideremos agora $w = ua$. Sem perda de generalidade, podemos assumir que $u = xa$, com $x \in \{a,b\}^*$, pois o caso $u = xb$ é análogo. Pela hipótese indutiva, existe uma derivação de uu segundo a gramática: $S \Rightarrow^* uu$; do autômato acima sabemos que esta derivação deve ter o formato:

$$S \Rightarrow^* xAxA \Rightarrow^{ac}_{p_4} xaxB \Rightarrow^{ac}_{p_8} xaxa = uu$$

Construímos agora uma derivação para ww, baseada na derivação para uu. Com efeito:

$$S \Rightarrow^* xAxA \Rightarrow^{ac}_{p_2} xaAxA \Rightarrow^{ac}_{p_6} xaAxaA = uAuA \Rightarrow^{ac}_{p_4} uauB \Rightarrow^{ac}_{p_8} uaua = ww.$$

Isto completa a indução e termina a prova.

Exemplo 2.4.3 Gramática com linguagem de controle regular $G_{12} = (G, C)$ para a linguagem $L = \{a^{2^n} : n \geq 0\}$. *Consideremos a gramática com linguagem de controle regular e com verificação de aparência:*

$$G_{12} = (\{S, A, X\}, \{a\}, \{p_1, p_2, p_3, p_4, p_5\}, S, (p_1^* p_2 p_3^* p_4)^* p_5 p_5^*, \{p_2, p_4\})$$

com

$$p_1 = S \to AA, p_2 = S \to X, p_3 = A \to S, p_4 = A \to X, p_5 = S \to a.$$

Derivação de a^{2^3} :
Com efeito:

$$\begin{array}{llll}
S & \Rightarrow^{ac}_{p_1} AA & \Rightarrow^{ac}_{p_2} AA & \Rightarrow^{ac}_{p_3} SA \quad \Rightarrow^{ac}_{p_3} SS \\
& \Rightarrow^{ac}_{p_4} SS & \Rightarrow^{ac}_{p_1} AAS & \Rightarrow^{ac}_{p_1} AAAA \quad \Rightarrow^{ac}_{p_2} AAAA \\
& \Rightarrow^{ac}_{p_3} SAAA & \Rightarrow^{ac}_{p_3} SSAA & \Rightarrow^{ac}_{p_3} SSSA \quad \Rightarrow^{ac}_{p_3} SSSS \\
& \Rightarrow^{ac}_{p_4} SSSS & \Rightarrow^{ac}_{p_1} AASSS & \Rightarrow^{ac}_{p_1} AAAASS \quad \Rightarrow^{ac}_{p_1} AAAAAAS \\
& \Rightarrow^{ac}_{p_1} AAAAAAAA & \Rightarrow^{ac}_{p_2} AAAAAAAA & \Rightarrow^{ac}_{p_3} SAAAAAAA \quad \Rightarrow^{ac}_{p_3} SSAAAAAA \\
& \Rightarrow^{ac}_{p_3} SSSAAAAA & \Rightarrow^{ac}_{p_3} SSSSAAAA & \Rightarrow^{ac}_{p_3} SSSSSAAA \quad \Rightarrow^{ac}_{p_3} SSSSSSAA \\
& \Rightarrow^{ac}_{p_3} SSSSSSSA & \Rightarrow^{ac}_{p_3} SSSSSSSS & \Rightarrow^{ac}_{p_4} SSSSSSSS \quad \Rightarrow^{ac}_{p_5} aSSSSSSS \\
& \Rightarrow^{ac}_{p_5} aaSSSSSS & \Rightarrow^{ac}_{p_5} aaaSSSSS & \Rightarrow^{ac}_{p_5} aaaaSSSS \quad \Rightarrow^{ac}_{p_5} aaaaaSSS \\
& \Rightarrow^{ac}_{p_5} aaaaaaSS & \Rightarrow^{ac}_{p_5} aaaaaaaS & \Rightarrow^{ac}_{p_5} a^{2^3}.
\end{array}$$

O autômato associado com a gramática G_{12} é apresentado na Fig. 2.10. O que nos diz esse autômato sobre as derivações da gramática? Em primeiro lugar, que a derivação de $a = a^{2^0}$ é conseguida aplicando, uma única vez, a produção p_5. Por outro lado, pode-se notar que, a cada aplicação da produção p_1 no estado q_0,

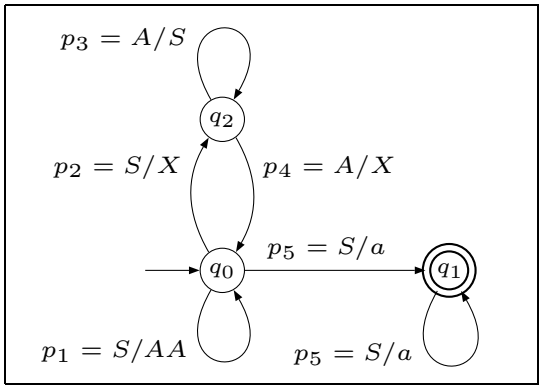


Figura 2.10: Autômato finito que aceita $(p_1^* p_2 p_3^* p_4)^* p_5 p_5^*$.

corres-podem duas aplicações da produção p_3 no estado q_2. Essa observação fornece a chave para a demonstração por indução: A derivação da palavra a^{2^n} pode ser baseada (decomposta) em duas derivações da "metade" dessa palavra, ou seja, em duas derivações da palavra $a^{2^{n-1}}$, uma vez que $a^{2^n} = a^{2^{n-1}}.a^{2^{n-1}}$.

Vamos provar, por indução matemática, que a aplicação das produções da gramática acima geram formas sentenciais do tipo S^{2^n}, com $n \geq 0$, do que se conclui que $L(G_{12}) = \{a^{2^n} : n \geq 0\}$.

Caso base. $n = 1$. *(O caso $n = 0$ é trivial e não influi no processo indutivo)*
Com efeito:

$$S \Rightarrow_{p_1}^{ac} AA \Rightarrow_{p_2}^{ac} AA \Rightarrow_{p_3}^{ac} AS \Rightarrow_{p_3}^{ac} SS = S^{2^1}.$$

Hipótese indutiva. *Seja $n > 1$ e suponhamos que para todo k, com $0 \leq k < n$, temos $S^{2^k} \in L(G_{12})$.*
Com efeito, consideremos $S^{2^n} = S^{2^{n-1}+2^{n-1}} = S^{2^{n-1}}.S^{2^{n-1}}$. Pela hipótese indutiva existe uma derivação de $S^{2^{n-1}}$, digamos $S \Rightarrow^* S^{2^{n-1}}$; usamos essa derivação de $S^{2^{n-1}}$ para obtermos uma derivação de S^{2^n}:

$$S \Rightarrow_{p_1}^{ac} AAS \Rightarrow^* AS^{2^{n-1}} \Rightarrow^* S^{2^{n-1}} S^{2^{n-1}} = S^{2^n}.$$

Isto completa a indução e termina a prova.

Proposição 2.4.1 *As seguintes inclusões valem para todo* i, j, k, r, s

$$\mathcal{L}_i \subseteq \mathcal{L}(i,j,k). \tag{2.12}$$

$$\mathcal{L}_j \subseteq \mathcal{L}(i,j,k). \tag{2.13}$$

$$\mathcal{L}(i,j,0) \subseteq \mathcal{L}(i,j,1). \tag{2.14}$$

$$\mathcal{L}(i,j,k) \subseteq \mathcal{L}(i,s,k), \text{ se } j \geq s. \tag{2.15}$$

$$\mathcal{L}(i,j,k) \subseteq \mathcal{L}(r,j,k), \text{ se } i \geq r, \text{ e não acontece } i = 2 \text{ e } r = 1. \tag{2.16}$$

Na última inclusão, considera-se que $3 \geq 2 - \lambda \geq 2 \geq 1 \geq 0$. *Mais ainda: para o segundo e para o último caso são válidas as seguintes inclussões estritas:* $\mathcal{L}_j \subset \mathcal{L}(2-\lambda, j, k)$, $\mathcal{L}(i,j,k) \subset \mathcal{L}(2-\lambda, j, k)$.

Prova.

Prova de (2.16)**:** Segundo a hierarquia de Chomsky, toda linguagem de tipo i, é uma linguagem de tipo r, para $i \geq r$.

Prova de (2.15)**:** Segundo a hierarquia de Chomsky, toda linguagem de controle de tipo i, é uma linguagem de tipo s, para $i \geq s$.

Prova de (2.14)**:** Provamos primeiro que $\mathcal{L}(i,3,0) \subseteq \mathcal{L}(i,3,1)$. Com efeito, seja $L \in \mathcal{L}(i,3,0)$. Por definição, existe uma gramática $G = (N, T, P, S)$, de tipo i, e uma expressão regular definindo o conjunto C, tal que $L = L(G, C)$. Introduzimos um não-terminal novo W ($W \notin N$) e definimos o conjunto $F = \{p_\lambda = W \to \lambda\}$ e temos que $L = L(G', C, F)$, onde $G' = (N', T, P', S)$, $N' = N \cup \{W\}$, e $P' = P \cup F$: a produção p_λ se aplica por último, em modo de verificação de aparência.

Provamos agora $\mathcal{L}(i,j,0) \subseteq \mathcal{L}(i,j,1)$ com $j = 0, 1, 2$. Com efeito, seja $L \in \mathcal{L}(i,j,0)$. Por definição, existe uma gramática $G = (N, T, P, S)$, de tipo i, e uma linguagem C sobre o alfabeto P, tal que $L = L(G, C)$. Neste caso, o novo conjunto de não-terminais é definindo como $N' = P \cup \{S_0, W\}$, o conjunto de produções a serem aplicadas em modo de verificação de aparência é definido como $F = \{p_W = W \to X\}$, e o novo conjunto de produções é definido como $P' = P \cup F \cup \{p_0 = S_0 \to SW, p_\lambda = W \to \lambda\}$, e obtemos $L = L(G', C, F)$, onde $G' = (N', T, P', S_0)$: primeiro aplica-se a produção p_0, introduzindo o não-terminal W na forma sentencial e ficando, portanto, proibido aplicar, em modo de verificação de aparência, a produção p_W até depois de ter aplicado a produção p_λ. A aplicação da produção p_λ (no sentido da hierarquia de Chomsky) não altera a derivação da palavra segundo G, pois $W \notin N$.

Prova de (2.12)**:** Basta provar $\mathcal{L}_i \subseteq \mathcal{L}(i,3,0)$, pois com isso, a afirmativa é conseqüência de (2.14) e (2.15). Com efeito, Se seja L um linguagem de

tipo i; portanto existe uma gramática $G = (N, T, P, S)$, de tipo i segundo a hierarquia de Chomsky, tal que $L = L(G)$. Se $P = \{p_1, \ldots, p_n\}$, então definindo C como o conjunto expressado pela expressão regular $(p_1| \ldots |p_n)^*$ temos que $L = L(G, C)$.

Prova de (2.13)**:** Consideremos primeiro o caso de uma linguagem L de tipo j, sobre o alfabeto $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ que contém a palavra vazia λ. Por definição, existe uma gramática $G = (N, \Sigma, P, S)$, tipo j, tal que $L = L(G)$. Definimos a gramática regular $G_1 = (\{S_0\}, \Sigma, P_1, S_0)$, onde $S_0 \notin N$ é um novo não-terminal e o conjunto de produções P_1 é definido como:

$$\begin{aligned} p_i &= S_0 \to \sigma_i S_0, \quad 1 \leq i \leq n \\ p_{n+1} &= S_0 \to \lambda. \end{aligned}$$

e obtemos que $L = L(G_1, Cp_{n+1})$, onde C é a linguagem definida pelo isomorfismo $\phi(\sigma_i) = p_i$, $1 \leq i \leq n$; isto mostra que $L \in \mathcal{L}(3, j, 0)$.

Se a linguagem L não contém a palavra vazia λ, definimos o conjunto de produções P_2:

$$\begin{aligned} p_{n+1} &= S_0 \to S_1. \\ p_i &= S_1 \to \sigma_i S_1, \quad 1 \leq i \leq n \\ p_{n+1} &= S_1 \to \lambda. \end{aligned}$$

onde $S_0, S_1 \notin N$ são novos não-terminais, e temos que a gramática $G_2 = (\{S_0, S_1\}, \Sigma, P_2, S_0)$, mostra que $L \in \mathcal{L}(2 - \lambda, j, 0)$.

□

Notação 2.4.2 *A seguinte notação se aplica à família das linguagens geradas por gramáticas livres de contexto com linguagens de controle regulares:*

$$\mathcal{L}(2 - \lambda, 3, 0) = \mathcal{R}, \qquad \mathcal{L}(2, 3, 0) = \mathcal{R}^{\lambda}$$

$$\mathcal{L}(2 - \lambda, 3, 1) = \mathcal{R}_{ac}, \qquad \mathcal{L}(2, 3, 1) = \mathcal{R}_{ac}^{\lambda}$$

Teorema 2.4.1 *Toda linguagem gerada por uma gramática matricial, periódica variante no tempo ou programada de tipo i, onde $i \in \{3, 2 - \lambda, 2, 1, 0\}$, pertence à família $\mathcal{L}(i, 3, 0)$. Toda linguagem gerada por uma gramática matricial, periódica variante no tempo ou programada de tipo i, onde $i \in \{3, 2 - \lambda, 2, 1, 0\}$, com verificação de aparência, pertence à família $\mathcal{L}(i, 3, 1)$.*

Prova. Consideramos primeiro o caso das gramáticas matriciais.

Seja $G = (N, T, M, S)$ uma gramática matricial, P o conjunto de todas as produções que aparecem nas matrizes de M e $F \subset P$ um subconjunto das produções.

Suponhamos que existam k matrizes em M. Para cada matriz

$$m_i = [p_1 = E_1 \to D_1, \ldots p_n = E_n \to D_n,], n \geq 1$$

denotamos

$$w_i = p_1 \ldots p_n \in P^*$$

e definimos

$$C = (w_i \cup \ldots \cup w_k)^*$$

e a gramática $G_1 = (N, T, P, S)$. Então

$$L_{ac}(G, F) = L_{ac}(G_1, C, F).$$

Consideramos a seguir o caso das gramáticas periódicas variantes no tempo.

Seja uma gramática periódica variante no tempo (G, v) de período k, onde $G = (N, T, P, S)$, e seja $F \subset P$ um subconjunto das produções de G. Denotamos

$$P_j = v(j) \cap P, \qquad 1 \leq j \leq k$$

e definimos

$$C = (P_1 \ldots P_k)^*(\{\lambda\} \cup P_1 \cup P_1P_2 \cup \ldots \cup P_1P_2 \ldots P_{k-1}).$$

Então

$$L_{ac}(G, v, F) = L_{ac}(G, C, F).$$

Consideramos finalmente o caso das gramáticas programadas.

Seja uma gramática programada (G, s, f), em que $G = (N, T, P, S)$, "s" e "f" são funções de P em 2^P, e seja $F \subset P$ um subconjunto das produções de G. Assumindo que existam n produções p_j, $1 \leq j \leq n$, em P, i.e. $|P| = n$, e definimos

$$G_1 = (N \cup \{Y\}, T, P_1, S)$$

onde

$$P_1 = \{p_j = E_j \to D_j : 1 \leq j \leq n\} \cup \{q_j = E_j \to \alpha(Y) : 1 \leq j \leq n\}$$

e o lado direito das produções no último conjunto é definido como:

$$\alpha(Y) = \begin{cases} Y & \text{se } G \text{ é de tipo 3, 2 ou 0,} \\ \beta\delta\gamma & \text{se } G \text{ é de tipo 1 e } E_j = \beta X\gamma, D_j = \beta\delta\gamma. \end{cases}$$

Considerando o conjunto P_1 das produções de G_1, definimos:

$$s_1 : P_1 \to 2^{P_1} \text{ e } f_1 : P_1 \to 2^{P_1}.$$

$$s_1(p) = \begin{cases} \{p_h : p_h \in s(p_j)\} \cup \{q_h : p_h \in s(p_j)\}, & \text{se } p = p_j, 1 \leq j \leq n \\ \emptyset, & \text{se } p = q_j, 1 \leq j \leq n. \end{cases}$$

$$f_1(p) = \begin{cases} \emptyset, & \text{se } p = p_j, 1 \leq j \leq n, \\ \{p_h : p_h \in f(p_j)\} \cup \{q_h : p_h \in f(p_j)\}, & \text{se } p = q_j, 1 \leq j \leq n. \end{cases}$$

Assim definiu-se uma gramática programada (G_1, s_1, f_1) para a qual

$$L_{ac}(G, s, f) = L_{ac}(G_1, s_1, f_1).$$

Definimos agora uma relação binária $\mathcal{R}$ no conjunto P_1 das produções de G_1: Sejam $p, q \in \mathcal{R}$,

$$p\mathcal{R}q \Leftrightarrow q \in s_1(p) \cup f_1(p).$$

Seja agora C a linguagem, sobre o alfabeto P_1, que comprende todas as palavras

$$w = \sigma_1 \ldots \sigma_m, \text{ com } m \geq 1 \text{ e } \sigma_i \mathcal{R} \sigma_{j+1}, 1 \leq j \leq m-1.$$

Pelo teorema A.1.1 do apêndice A, o conjunto C é regular, e temos:

$$L_{ac}(G_1, s_1, f_1) = L_{ac}(G_1, C, \{q_1, \ldots, q_n\}).$$

Já que o não-terminal Y não pode ser eliminado, as produções q_j somente podem ser aplicadas em modo de verificação de aparência. Além disso, na última igualdade, o conjunto $\{q_1, \ldots, q_n\}$ pode ser substituído pelo conjunto vazio se a imagem de f (e portanto a imagem de f_1) for o conjunto vazio. Assim, as últimas duas igualdades completam a demonstração.
□

Teorema 2.4.2 *A família $\mathcal{L}(0, 3, 1)$ é igual à família $\mathcal{L}_0$ das linguagens de tipo 0.*

Prova. É conseqüência da inclusão (2.12) da proposição 2.4.1 e da tese de Church.
□

Teorema 2.4.3 *A família $\mathcal{L}(1, 3, 1)$ é igual à família $\mathcal{L}_1$ das linguagens dependentes de contexto.*

Prova. Se L é uma linguagem dependente de contexto, então existe uma gramática dependente de contexto $G = (N, T, P, S)$ tal que $L(G) = L$. Neste caso, satisfaz-se $L(G) = L(G, P^*, \emptyset)$ e, pelo conteúdo (2.13) da proposição 2.4.1, temos que $\mathcal{L}_1 \subset$

$\mathcal{L}(1,3,1)$. Falta provar que, se $G = (N, T, P, S)$ é uma gramática de tipo 1, C é uma linguagem regular sobre o alfabeto P e $F \subset P$, então a linguagem arbitrária:

$$L = L_{ac}(G, C, F)$$

é dependente de contexto. A prova é feita em duas etapas: na primeira é eliminada a verificação de aparência e, na segunda, é eliminada a linguagem de controle.

Sem perda de generalidade, podemos assumir que $S \to \lambda$ não pertence a F. Com efeito, examinando C, podemos determinar se a palavra vazia λ pertence a L ou não e, em caso negativo, sabemos que, se $L \setminus \{\lambda\}$ é dependente de contexto, então L também é dependente de contexto. Por outro lado, se a produção $S \to \lambda$ é aplicada, em modo de verificação de aparência, na derivação de palavras não vazias, então podemos substituí-la por uma outra produção $S \to Y$, em que Y é um novo não-terminal.

Eliminação da verificação de aparência: Introduzimos uma gramática de comprimento crescente G_1, uma linguagem de controle C_1 e um novo símbolo terminal b tais que:

$$L(G_1, C_1) = \{bwb : w \in L_{ac}(G, C, F)\} \tag{2.17}$$

O conjunto de não-terminais de G_1 é

$$N_1 = N \cup \{S_1, B\} \cup \{[\alpha, p] : \alpha \in (N \cup T), p \in F\}.$$

O conjunto de terminais de G_1 é definido como:

$$T_1 = T \cup \{b\}.$$

e o símbolo inicial é S_1. O conjunto de produções de G_1 é definido como:

$$P_1 = P \cup \{p_1 = S_1 \to BSB, p_2 = B \to b\} \cup (\cup_p (A_p \cup E_p \cup B_p)).$$

em que A_p, E_p e B_p são definidos a seguir: Consideremos um $p = E \to D \in F$ arbirtrário e suponhamos que $E = x_1 \dots x_r$, $r \geq 1$, onde cada $x_i \in (N \cup T)$. O conjunto A_p é formado pelas produções

$$\begin{array}{cc}
[\alpha, p]\,\beta \to \alpha\,[\beta, p]\,, & \alpha, \beta \in (N \cup T), \alpha \neq x_1, \\
[x_1, p]\,\beta \to x_1\,[\beta, p]\,, & \beta \in (N \cup T), \backslash\{x_2\}, \\
[x_1, p]\,x_2\beta \to x_1\,[x_2, p]\,\beta & \beta \in (N \cup T), \backslash\{x_3\}, \\
\dots & \\
[x_1, p]\,x_2 \dots x_{r-1}\beta \to x_1\,[x_2, p] \dots x_{r-1}\beta, & \beta \in (N \cup T), \backslash\{x_r\}.
\end{array}$$

O conjunto E_p é formado pelas produções

$$\begin{array}{rl} [\alpha, p]\, B \to \alpha B, & \alpha, \beta \in (N \cup T), \\ & \text{Se } r \neq 1 \text{ ou } \alpha \neq x_1, \\ [\alpha_1, p]\, \alpha_2 B \to \alpha_1 \alpha_2 B, & \alpha_i \in (N \cup T), r > 2, \\ \vdots & \vdots \\ [\alpha_1, p]\, \alpha_2 \ldots \alpha_{r-1} B \to \alpha_1 \alpha_2 \ldots \alpha_{r-1} B, & \alpha_i \in (N \cup T), r > 2. \end{array}$$

O conjunto B_p é formado pelas produções

$$B\alpha \to B\,[\alpha, p]\,, \alpha \in (N \cup T).$$

Agora definimos uma substituição regular sobre os elementos de P:

$$\sigma(p) = \begin{cases} p & \text{se } p \in P \setminus F, \\ p \cup B_p (A_p)^* E_p & \text{se } p \in F. \end{cases}$$

Agora definimos a linguagem regular de controle:

$$C_1 = p_1 \sigma(C) p_2^2.$$

Vamos agora verificar que a equação 2.17 é válida. Já que, em toda derivação de palavras de C_1, a primeira produção aplicada é p_1, o que se faz em primeiro lugar é introduzir os símbolos B que marcam os extremos da palavra; além disso, a definição de σ torna possível simular a aplicação de produções $p \in P$, em modo de reescrita ou em modo de verificação de aparência. Especificamente, para simular a aplicação de uma produção em modo de verificação de aparência, começa-se com uma produção de B_p, introduzindo o não-terminal $[\alpha, p]$; depois se aplicam produções em A_p para mover o não-terminal entre colchetes para o extremo direito, de onde pode ser eliminado pela aplicação de uma produção em E_p. Se o lado esquerdo da produção p aparece como uma palavra da forma sentencial corrente, então nenhuma produção em E_p pode ser aplicada e, portanto, a derivação termina sem produzir qualquer palavra. Finalmente, aplica-se a produção p_2 para trocar os marcadores B pelos terminais b.

Eliminação da linguagem de controle: A linguagem regular C_1 é aceita por algum autômato finito determinístico, digamos $(Q, P_1, q_0, Q_F, \delta)$, onde P_1 é conjunto de produções de G_1. Definimos agora a gramática dependente de contexto $G_2 = (N_2, T_2, P_2, S_2)$ com conjunto de não-terminais $N_2 = N_1 \cup Q \cup \{S_2\}$, conjunto de terminais $T_2 = T_1$, símbolo inicial S_2 e conjunto de produções:

$$\begin{array}{rcl} P_2 & = & \{qE \to \delta(q,p)D : q \in Q \text{ e } p = E \to D \in P_1\} \\ & \cup & \{q\alpha \to \alpha q, \alpha q \to q\alpha : q \in Q, \alpha \in (N_1 \cup T)\} \\ & \cup & \{qb \to bb : q \in Q_F\} \\ & \cup & \{S_2 \to q_0 S_1\} \end{array}$$

afirmamos agora que

$$L(G_2) = \{bbwb : w \in L_{ac}(G, C, F)\} = \{bu : u \in L(G_1, C_1)\}.$$

Com efeito, as produções no primeiro conjunto da união que define P_2 mantém o registro da palavra de controle; as produções no segundo conjunto procuram uma posição, na forma sentencial, para continuar a derivação; as produções no terceiro conjunto eliminam o estado final do autômato, no qual a derivação de uma palavra w de $L_{ac}(G, C, F)$ possa ter terminado segundo alguma palavra de controle de C_1. □

Teorema 2.4.4 *A família $\mathcal{L}(3, 3, 1)$ é igual à família $\mathcal{L}_3$ das linguagens regulares.*

Prova. Vamos provar que uma linguagem arbitrária da família $\mathcal{L}(3, 3, 1)$ é de tipo 3. Para isso, consideremos uma linguagem

$$L = L_{ac}(G, C, F_1)$$

em que $G = (N, T, P, S)$ é linear à direita, C é uma linguagem regular e $F \subset P$. O caso para gramáticas lineares à esquerda é similar. Vamos construir uma máquina seqüência generalizada GSM tal que

$$L = GSM(C).$$

Assim, pelo teorema A.1.6 do apêndice A, temos que L é de tipo 3. O alfabeto de estados de GSM é $N \cup \{U\}$, o alfabeto de entrada de GSM é P e o alfabeto de saída de GSM é T.

Para cada produção

$$p = A \to wB, \qquad A, B \in N, \qquad w \in T^*, \qquad p \in P.$$

de G, existe uma produção em GSM

$$Ap \to wB \tag{2.18}$$

e, no caso que $p \in F_1$, GSM também contém as produções

$$Dp \to D, \qquad D \in N \cup \{U\}, \qquad D \neq A. \tag{2.19}$$

Para cada produção

$$p = A \to w, \qquad A \in N, \qquad w \in T^*, \qquad p \in P.$$

de G, existe uma produção em GSM

$$Ap \to wU \tag{2.20}$$

e, no caso que $p \in F_1$, GSM também contém as produções

$$Dp \to D, \qquad D \in N \cup \{U\}, \qquad D \neq A. \tag{2.21}$$

Nessas condições, ja que o número de não-terminais nas derivações segundo G nunca supera um, então as palavras de controle tornam desnecessários os não-terminais. A reescrita é simulada pelas produções (2.18) e (2.20), e o modo de verificação de aparência, pelas produções (2.19) e (2.21).
□

Teorema 2.4.5 *Para $i = 0, 1, 3$, cada uma das seguintes famílias de linguagens é igual à família $\mathcal{L}_i$ de linguagens de tipo i:*

1. *A família das linguagens geradas por gramáticas matriciais de tipo i.*
2. *A família das linguagens geradas por gramáticas matriciais de tipo i com verificação de aparência.*
3. *A família das linguagens geradas por gramáticas periódicas variantes no tempo de tipo i.*
4. *A família das linguagens geradas por gramáticas periódicas variantes no tempo de tipo i com verificação de aparência.*
5. *A família das linguagens geradas por gramáticas programadas de tipo i.*
6. *A família das linguagens geradas por gramáticas programadas de tipo i com verificação de aparência.*

Prova. É conseqüência dos teoremas 2.4.1-2.4.4.
□

Teorema 2.4.6 *Toda linguagem na família $\mathcal{L}(2, 3, 0) = \mathcal{R}^{\lambda}$ é gerada por uma gramática livre de contexto matricial, cujas matrizes consistem, todas, de apenas duas produções. Toda linguagem na família $\mathcal{L}(2, 3, 1) = \mathcal{R}^{\lambda}_{ac}$ é da forma (2.1) descrita no teorema 2.2.3 para alguma gramática livre de contexto matricial, que satisfaz as hipóteses do teorema 2.2.3.*

Prova. Consideremos a primeira afirmativa.

Seja L uma linguagem da família $\mathcal{R}^{\lambda}$, isto é,

$$L = L_{ac}(G_1, C)$$

onde $G_1 = (N, T, P, S)$ é uma gramática livre de contexto, C é uma linguagem regular sobre P. Como C é uma linguagem regular, ela é aceita por um autômato finito determinístico $A = (Q, P, q_0, Q_F, \delta)$. Consideremos agora a gramática livre de contexto matricial

$$G = (N \cup Q \cup \{S_0\}, T, M, S_0)$$

com M definida como segue: Assumindo que $P = \{p_j = E_j \to D_j : 1 \leq j \leq k\}$ então M possui as matrizes:

$[S_0 \to S_0, S_0 \to Sq_0]$
$[p_j = E_j \to D_j, q \to \delta(q, p_j)]$ para $q \in Q, 1 \leq j \leq k$,
$[p_j = E_j \to D_j, q \to \lambda]$ para $q \in Q, 1 \leq j \leq k$, tal que $\delta(q, p_j) \in Q_F$

Pode-se verificar que essa gramática matricial gera a linguagem L.
Consideremos agora a segunda afirmativa.

Seja L uma linguagem da família $\mathcal{R}^{\lambda}_{ac}$, isto é,

$$L = L_{ac}(G_1, C, F_1)$$

onde $G_1 = (N, T, P, S)$ é uma gramática livre de contexto, C é uma linguagem regular sobre P, e F_1 é um subconjunto do conjunto P das produções de G_1. Portanto, C é aceita por um autômato finito determinístico $A = (Q, P, q_0, Q_F, \delta)$. Consideremos agora a gramática livre de contexto matricial

$$G = (N \cup Q, T, M, S)$$

em que M é definida como segue: Assumindo que $P = \{p = E_j \to D_j : 1 \leq j \leq k\}$ então M possui as matrizes:

$[p_j = E_j \to D_j, q \to \delta(q, p_j)]$ para $q \in Q, 1 \leq j \leq k$,
$[p_j = E_j \to D_j, q \to \lambda]$ para $q \in Q, 1 \leq j \leq k$, tal que $\delta(q, p_j) \in Q_F$

Pondo $F = F_1$ temos que a gramática matricial G satisfaz as hipóteses do teorema 2.2.3. Mais ainda,

$$L = \{w \in T^* : P_0 S_0 \Rightarrow^{ac^*} w\}$$

pois as segundas produções nas matrizes servem para determinar se a palavra de controle conduz o autômato A do estado inicial q_0 até algum estado em Q_F.
□

Exemplo 2.4.4 (Aplicação do teorema 2.4.6.) *A partir da gramática livre de contexto com linguagem de controle regular com verificação de aparência G_{12} do exemplo 2.4.3, vamos obter uma gramática livre de contexto matricial com verificação de aparência, na qual cada matriz possui apenas duas produções. Usamos o método do teorema 2.4.6.*

Com efeito consideremos a gramática livre de contexto com linguagem de controle regular com verificação de aparência G_{12} do exemplo 2.4.3:

$$G_{12} = (\{S, A, X\}, \{a\}, \{p_1, p_2, p_3, p_4, p_5\}, S, (p_1^* p_2 p_3^* p_4)^* p_5 p_5^*, \{p_2, p_4\})$$

cujas produções são definidas como:

$$p_1 = S \to AA, p_2 = S \to X, p_3 = A \to S, p_4 = A \to X, p_5 = S \to a.$$

a linguagem de controle é definida como:

$$C = (p_1^* p_2 p_3^* p_4)^* p_5 p_5^*$$

e as produções que podem ser aplicadas em modo de verificação de aparência são:

$$F = \{p_2 = S \to X p_4 = A \to X\}$$

Do exemplo 2.4.3 sabemos que C é aceita pelo autômato finito determinístico $A = (\{q_0, q_1, q_2\}, P, q_0, \{q_1\}, \delta)$.

	p_1	p_2	p_3	p_4	p_5
q_0	q_0	q_2			q_1
q_2			q_2	q_0	
q_1					q_1

Aplicamos o método do teorema 2.4.6 obtemos $GM = (N \cup Q, T, M, S_0)$

i	m_i	*pois*
0	$[p_0 = S_0 \to S_0, \quad S_0 \to Sq_0]$	*é matriz inicial*
1	$[p_1 = S \to AA, \quad q_0 \to q_0]$	$\delta(q_0, p_1) = p_0$
2	$[p_2 = S \to X, \quad q_0 \to q_2]$	$\delta(q_0, p_2) = p_2$
3	$[p_5 = S \to a, \quad q_0 \to q_1]$	$\delta(q_0, p_5) = p_1$
4	$[p_3 = A \to S, \quad q_2 \to q_2]$	$\delta(q_2, p_3) = p_2$
5	$[p_4 = A \to X, \quad q_2 \to q_0]$	$\delta(q_2, p_4) = p_0$
6	$[p_5 = S \to a, \quad q_1 \to q_1]$	$\delta(q_1, p_5) = p_1$
7	$[p_5 = S \to a, \quad q_1 \to \lambda]$	$\delta(q, p) \in Q_F$

Teorema 2.4.7 *Cada uma das famílias $\mathcal{M}^{\lambda}_{ac}$, $\mathcal{T}^{\lambda}_{ac}$, $\mathcal{P}^{\lambda}_{ac}$, $\mathcal{R}^{\lambda}_{ac}$ é igual à família $\mathcal{L}_0$ das linguagens de tipo 0. Conseqüentemente, toda linguagem de tipo 0 é gerada por uma gramática livre de contexto periodicamente variante no tempo com verificação de aparência. Mais ainda, todas as famílias $\mathcal{M}^{\lambda}$, $\mathcal{T}^{\lambda}$, $\mathcal{P}^{\lambda}$, $\mathcal{R}^{\lambda}$. são iguais.*

Prova. As inclusões

$$\mathcal{M}^{\lambda}, \mathcal{T}^{\lambda}, \mathcal{P}^{\lambda}, \subseteq \mathcal{R}^{\lambda} \qquad \text{e} \qquad \mathcal{M}^{\lambda}_{ac}, \mathcal{T}^{\lambda}_{ac}, \mathcal{P}^{\lambda}_{ac}, \subseteq \mathcal{R}^{\lambda}_{ac}$$

são conseqüência do teorema 2.4.1.

As inclusões

$$\mathcal{R}^{\lambda} \subseteq \mathcal{M}^{\lambda}, \subseteq \mathcal{T}^{\lambda}, \qquad \text{e} \qquad \mathcal{R}^{\lambda}_{ac} \subseteq \mathcal{M}^{\lambda}_{ac}, \subseteq \mathcal{T}^{\lambda}_{ac},$$

são conseqüência dos teoremas 2.4.6, 2.4.1, 2.2.2 e 2.2.3.

Já que o teorema 2.3.3 garante que $\mathcal{M}^{\lambda} \subset \mathcal{P}^{\lambda}$ e $\mathcal{M}^{\lambda}_{ac} \subset \mathcal{P}^{\lambda}_{ac}$, temos, pelas inclusões anteriores que

$$\mathcal{T}^{\lambda} = \mathcal{M}^{\lambda} = \mathcal{P}^{\lambda} = \mathcal{R}^{\lambda} \qquad \text{e} \qquad \mathcal{T}^{\lambda}_{ac} = \mathcal{M}^{\lambda}_{ac} = \mathcal{P}^{\lambda}_{ac} = \mathcal{R}^{\lambda}_{ac}$$

e finalmente a afirmativa do teorema decorre do teorema 2.3.1.
□

Teorema 2.4.8 *Qualquer linguagem gerada por uma gramática matricial é gerada por uma gramática livre de contexto matricial, cujas matrizes possuem apenas duas produções. Qualquer linguagem gerada por uma gramática matricial com verificação de aparência é gerada por uma gramática livre de contexto matricial com verificação de aparência, cujas matrizes possuem apenas duas produções.*

Prova. É conseqüência dos teoremas 2.4.1 e 2.4.6
□

Teorema 2.4.9 *Todas as linguagens nas famílias $\mathcal{T}_{ac}$, $\mathcal{M}_{ac}$, $\mathcal{P}_{ac}$, $\mathcal{R}_{ac}$ são dependentes de contexto. Portanto, todas as linguagens das famílias $\mathcal{T}$, $\mathcal{M}$, $\mathcal{P}$, $\mathcal{R}$ são dependentes de contexto.*

Prova. É conseqüência dos teoremas 2.4.1 e 2.4.3
□

Teorema 2.4.10

$$\mathcal{M}^{\lambda}_{esq} = \mathcal{L}_0$$

Prova. É conseqüência do teorema 2.1.2 e do teorema A.1.4 do apêndice A,
□

Proposição 2.4.2

$$\mathcal{T} = \mathcal{M} = \mathcal{P} = \mathcal{R} \qquad e \qquad \mathcal{T}_{ac} = \mathcal{M}_{ac} = \mathcal{P}_{ac} = \mathcal{R}_{ac}$$

Todas essas famílias estão contidas propriamente na família das linguagens dependentes de contexto sem produções-λ. A família $\mathcal{R}_{ac}$ é uma "família abstrata de linguagens" (AFL, pelo seu nome em inglês: abstract family of languages) que é fechada em relação às operações de substituição e homomorfismo restrito.

Prova. Eliminando as transições-λ na prova do teorema 2.4.6 obtemos

$$\mathcal{R} \subset \mathcal{M} \qquad \text{e} \qquad \mathcal{R}_{ac} \subset \mathcal{M}_{ac}$$

Da proposição 2.2.2 e do teorema 2.4.1 e de $\mathcal{R} \subset \mathcal{M}$, obtemos:

$$\mathcal{M} \subseteq \mathcal{T} \subseteq \mathcal{R} \subseteq \mathcal{M}$$

Da proposição 2.2.2 e do teorema 2.3.3 e de $\mathcal{R}_{ac} \subset \mathcal{M}_{ac}$, obtemos:

$$\mathcal{M}_{ac} \subseteq \mathcal{P}_{ac} \subseteq \mathcal{R}_{ac} \subseteq \mathcal{M}_{ac}$$

obtendo assim a primeira afirmativa. A segunda afirmativa é provada de forma similar.
□

Nesta seção foram introduzidas as gramáticas com linguagem de controle, baseadas em gramáticas de tipo i, segundo a hierarquia de Chomsky, e que possuem como mecanismo de controle, uma outra linguagem de tipo j, segundo a hierarquia de Chomsky, à qual deve pertencer, como uma palavra, a concatenação das produções aplicadas numa derivação. O conceito de *aplicação de produção em modo de verificação de aparência* foi redefinido para o caso das gramáticas com linguagem de controle e foi definida uma notação para o caso especifico das gramáticas com linguagem de controle regular baseadas em gramáticas livres de contexto. Foi provado que essa classe e as correspondentes classes das gramáticas matriciais, periódicas variantes no tempo e programadas identificam a classe das linguagens recursivamente enumeráveis. A maioria das provas é *construtiva* isto quer dizer que é exibido um processo para a construção das gramáticas que provam os resultados do capítulo. Essas construções têm inspirado as provas da seção 3.2 do seguinte capítulo, no qual provamos a equivalência do formalismo proposto nesta tese com os quatro formalismos gramaticais estudados neste capítulo.

2.5 Verificação de aparência e decidibilidade

Vamos formular alguns problemas de decisão que mostram a importância do conceito de verificação de aparência na teoria das linguagens formais. A terminologia e os resultados desta seção, enunciados sem demosntração, foram extraídos da obra de Dassow[3].

Definição 2.5.1 (Problemas de decisão) *Seja $\mathcal{G}$ uma família de gramáticas.*

1. **Problema de pertinência:** *Dados $G = (N, T, P, S) \in \mathcal{G}$, e $w \in T^*$, determinar se $w \in L(G)$.*
2. **Problema de equivalência:** *Dadas $G_1, G_2 \in \mathcal{G}$, determinar se $L(G_1) = L(G_2)$.*
3. **Problema de inclusão:** *Dadas $G_1, G_2 \in \mathcal{G}$, determinar se $L(G_1) \subset L(G_2)$.*
4. **Problema de vacuidade:** *Dada $G \in \mathcal{G}$, determinar se $L(G) = \emptyset$.*
5. **Problema de finitude:** *Dada $G \in \mathcal{G}$, determinar se $L(G)$ é finita.*

Teorema 2.5.1 *O problema de vacuidade e o problema de finitude não são decidíveis para gramáticas livres de contexto matriciais com verificação de aparência.*

Teorema 2.5.2 *O problema de vacuidade é decidível para gramáticas livres de contexto matriciais arbitrárias sem verificação de aparência.*

Teorema 2.5.3 *O problema de pertinência é NP-completo para gramáticas livres de contexto matriciais sem produções-λ e com verificação de aparência.*

Teorema 2.5.4 *Não é decidível o problema de determinar se uma gramática matricial, sem produ-ções-λ e com verificação de aparência, gera uma linguagem livre de contexto.*

2.6 Resumo

Neste resumo vamos indicar as modificações e acréscimos que fizemos ao texto original de Salomaa[2]. O final da prova do teorema 2.2.1 foi re-escrito para esclarecer como se fazem as derivações; as provas dos teorema 2.2.3 e 2.4.3, foram reorganizadas no sentido de colocar os argumentos numa ordem seqüêncial; em particular na prova do teorema 2.2.3 foram repetidas algumas construções (feitas no original apenas para dois dos quatro casos a serem considerados), seguindo indicações no texto original da prova. Além disso, incluímos alguns exemplos e demonstrações de nossa autoria; a seguir apresentamos uma lista e uma breve descrição dos mesmos:

1. No exemplo 2.1.4 sobre uma gramática matricial com derivações mais à esquerda que gera a linguagem $L = \{a^{2^n} : n \geq 0\}$, incluímos a árvore de derivação para várias palavras mostrando as derivações mais à esquerda nessas árvores; analisando essas árvores de derivação, planejamos uma demonstração por indução matemática para provar que as derivações mais à esquerda geram as palavras da linguagem e incluímos essa prova no final do exemplo.

2. O exemplo 2.2.1 foi desenvolvido para ilustrar o método usado na prova do teorema 2.2.1, no qual se prova que toda linguagem é variante no tempo de tipo 3, mas a função de variação pode não ser computável.

3. O exemplo 2.2.4 foi desenvolvido para ilustrar o método usado na prova do teorema 2.2.3, no qual uma gramática matricial, com duas produções em cada matriz, é transformada numa gramática periodicamente variante no tempo.

4. Na observação 2.3.2 fazemos uma pequena modificação do exemplo 2.3.1 para obter uma gramática programada que gera a linguagem gerada pela gramática do exemplo 2.3.1 e também a palavra vazia.

5. Na observação 2.4.1 introduzimos o autômato finito com alfabeto de entrada igual ao conjunto de produções da gramática com linguagem de controle regular; esse autômato finito é usado para planejar provas por indução nesse exemplo e nos seguintes.

6. No exemplo 2.4.4 usamos o método da prova do teorema 2.4.6 para obter uma gramática matricial, com duas produções em cada matriz, a partir de uma gramática com linguagem de controle regular; a gramática matricial assim obtida é a mesma que foi usada no exemplo 2.2.4.

As técnicas de demonstração introduzidas neste capítulo serão usadas no próximo capítulo para demonstrar que o formalismo proposto das *gramáticas livres de contexto adaptativas* é equivalente aos quatro formalismos gramáticas estudados neste capítulo e, portanto, possui poder de máquina de Turing.

Capítulo 3

Gramáticas livres de contexto adaptativas

Neste capítulo, apresentamos uma proposta de dispositivo gramatical que identifica a classe das linguagens recursivamente enumeráveis. Para tanto recordamos primeiramente o conceito de gramática adaptativa[1] e, a seguir, introduzimos uma especialização, baseada apenas no formalismo das *produções livres de contexto adaptativas com verificação de aparência*, que possui poder computacional de máquina de Turing. A prova desta afirmativa se apóia na equivalência entre o formalismo proposto e o das gramáticas livres de contexto com mecanismos de controle e verificação de aparência, apresentadas no capítulo 2. As provas são construtivas, e podem ser usadas para traduzir exemplos de gramáticas livres de contexto com mecanismos de controle e verificação de aparência para gramáticas livres de contexto adaptativas com verificação de aparência. Fazemos tais traduções na parte final da seção 3.2.

Esta apresentação das gramáticas adaptativas incorpora uma extensa simplificação da notação original. Para apresentar uma idéia das simplificações realizadas, mostramos na primeira seção deste capítulo uma resenha das gramáticas adaptativas como definidas originalmente e, a seguir, definimos o novo formalismo. Para uma descrição detalhada das gramáticas adaptativas e suas propriedades remetemos o leitor ao trabalho original de Iwai[1].

Os resultados provados na seção 3.2 podem ser resumidos no seguinte enunciado: *as gramáticas livres de contexto adaptativas com verificação de aparência geram a classe das linguagens recursivamente enumeráveis. Mais ainda, basta usar funções adaptativas posteriores*[1] *sem geradores nem variáveis, nem ações elementares de consulta, nem ações adaptativas inicial nem final; adicionalmente, o único parâmetro requerido é uma produção, sendo que esse parâmetro pode ser eliminando sobrecarre-*

[1] Iwai[1] provou que gramáticas adaptativas podem usar apenas funções adaptativas posteriores

gando a notação das produções de uma forma compatível com a notação utilizada em Neto[8] e Iwai[1].

3.1 Gramáticas Adaptativas

Gramáticas adaptativas são dispositivos gramáticais dinâmicos[27], isto é, que podem mudar sua configuração em tempo de utilização significando que a derivação de uma palavra gerada pela gramática adaptativa pode ser interpretada como uma trajetória que percorre diferentes gramáticas.

Na sua especificação original[1], as gramáticas adaptativas podiam, ao realizar uma derivação, incorporar novos símbolos não-terminais, assim como criar ou apagar produções; mais ainda, podiam incluir produções dependentes de contexto na sua definição. Era permitido definir variáveis sem tipo para armazenar nomes de não-terminais, terminais e geradores (apontadores a variáveis que podiam ser usados para "gerar dinâmicamente" novos valores de não-terminais ou produções). Mais ainda, o conjunto de produções podia ser pesquisado especificando um padrão de busca, e o resultado da pesquisa era um conjunto de produções que satisfaziam tal padrão de busca.

A seguir vamos apresentar um resumo destas e outras carateristicas; terminaremos esta seção apresentando um exemplo, extraído de Iwai[1].

Seja $G = (N, T, P, S)$ uma gramática livre de contexto. Consideremos adicionalmente: um alfabeto de símbolos de contexto C, um conjunto de produções D aplicáveis a formas sentenciais com dependências de contexto e um conjunto $\mathbb{A}$ de "funções adaptativas"; estas permitem incluir ou eliminar produções nos conjuntos P e D. Associamos funções adaptativas às produções de modo que seja possível incluir ou eliminar produções durante a derivação de uma palavra, definindo uma relação binária R sobre o produto cartesiano $(P \cup D) \times \mathbb{A}$. Define-se a tripla $(G, \mathbb{A}, \mathcal{R})$ como sendo uma gramática adaptativa, cujo conjunto inicial de produções é $P \cup D$. As funções adaptativas associadas às produções especificam quais produções devem ser retiradas da gramática ou adicionadas a ela em cada passo da derivação de palavras de T^* pela gramática adaptativa $(G, \mathbb{A}, \mathcal{R})$. Vamos formalizar esses conceitos:

Definição 3.1.1 Gramática Adaptativa. *Uma gramática adaptativa é uma tripla* $G = (G^0, \mathbb{A}, R^0)$ *onde*

1. $G^0 = (N^0, T, P^0, C^0, D^0, S)$ *é uma "gramática inicial" tal que*

   1.1 N^0 *é um conjunto finito de símbolos não-terminais,*

   1.2 T *é um conjunto finito de símbolos terminais, tal que* $N^0 \cap T = \emptyset$,

1.3 P^0 *é um conjunto finito de produções livres de contexto, relativas aos não-terminais do conjunto* N^0 *e aos terminais do conjunto* T*,*

1.4 C^0 *é um conjunto finito de símbolos de contexto,*

1.5 N^0*,* T *e* C^0 *são disjuntos dois a dois,*

1.6 D^0 *é um conjunto finito de produções dependentes de contexto relativas aos não-terminais do conjunto* $(N^0 \cup C^0)$ *e aos terminais do conjunto* T*,*

1.7 $S \in N$ *é o símbolo inicial.*

2. $\mathbb{A}$ *é um conjunto finito de funções adaptativas (veja definição 3.1.8)*

3. R^0 *é uma relação binária sobre o produto cartesiano* $(P^0 \cup D^0) \times \mathbb{A}$*.*

Definição 3.1.2 Produções. *Na definição dos conjuntos de produções* P *e* D *utilizaremos um índice superior* i *que indica o número de alterações já realizadas pela gramática adaptativa desde o início do processamento corrente.*

No conjunto das produções das gramáticas adaptativas há dois tipos de produções livres de contexto, pertencendo ao conjunto P^i*, com* $i \in \mathbb{N}$

Tipo 1. *Produções da forma:*

$$A \rightarrow \{\mathcal{A}\}\alpha$$

onde $\alpha \in (N \cup T)^*, A \in N^i$*, e* $\mathcal{A}$ *é uma ação adaptativa opcional.*

Tipo 2. *Produções da forma:*

$$A \rightarrow \phi$$

onde ϕ *é um meta-símbolo que indica o conjunto vazio; regras desse tipo são usadas para o caso de existirem regras que referenciem não-terminais que deverão ser dinâmicamente definidos, pela aplicação de ações adaptativas.*

Adicionalmente, há as produções dependentes de contexto do conjunto D^i*, com* $i \in \mathbb{N}$

Tipo 3. *Possuem um dos formatos seguintes: Sendo* $A, B \in N^i$ *e* $\mathcal{A}$ *uma ação adaptativa opcional,*

$$\alpha A \leftarrow \{\mathcal{A}\}\beta B, \text{ com } \alpha \in C \cup \{\lambda\}, \beta \in C$$

Neste caso, a seta para esquerda indica que o β *está sendo injetado na cadeia de entrada.*

$$\alpha A \rightarrow \{\mathcal{A}\}\beta B, \text{ com } \alpha \in C, \beta \in (T \cup \{\lambda\})^*$$

Neste caso, αA *esta sendo substituído por* βB *enquanto a cadeia* β *é gerada como saída.*

Definição 3.1.3 Geradores. *São "apontadores a variáveis" automaticamente preenchidos ao início da execução da função adaptativa, com valores que não se repetem, preservando este valor apenas até o final da execução da função.*

Nas duas definições a seguir vamos considerar que p é um padrão de busca que corresponde a um dos três tipos de produções descritos na definição 3.1.2, e pode referenciar variáveis ainda não preenchidas.

Definição 3.1.4 Ação elementar de consulta.

$$?\,[p]$$

Consulta a existência de regras que correspondem ao padrão p. Se existirem as regras correspondentes, então preenche toda variável que apareça no padrão p com os valores correspondentes nas regras encontradas.

Definição 3.1.5 Ação elementar de remoção.

$$-\,[p]$$

Consulta a existência de regras que correspondem ao formato indicado para exclusão e, se existirem as regras correspondentes, então as remove; caso contrário, nada ocorre.

Definição 3.1.6 Ação elementar de inserção.

$$+\,[p]$$

onde p é uma produção de um dos três tipos de produções descritos na definição 3.1.2, que pode referenciar variáveis preenchidas e geradores; essa ação inclui no conjunto de regras da gramática adaptativa a regra p. A inserção de uma regra já existente é inócua. A inserção de regras que referenciam variáveis indefinidas é inócua.

Definição 3.1.7 Variável para funções adaptativas. *Variáveis são símbolos que dão nome a elementos cujos valores são desconhecidos no momento da chamada da função adaptativa, mas que devem ser preenchidos, como efeito da execução de ações adaptativas elementares durante sua execução.*

Definição 3.1.8 *O formato geral de uma função adaptativa é o seguinte:*

Nome(lista de parâmetros formais)
{lista de variáveis, lista de geradores:
ação adaptativa opcional inicial
ação elementar 1
ação elementar 2
...
ação elementar n
ação adaptativa opcional final
}

Faremos a definição da aplicação de uma produção considerando, primeiro, as produções de tipo 1, 2 ou 3 sem levar em conta a função adaptativa opcional e, depois, explicaremos a aplicação de produções que incluem funções adaptativas.

Definição 3.1.9 Derivação de um passo. *A definição da relação $\Rightarrow_{G^i}$ envolve duas cadeias de símbolos. A segunda é gerada a partir da primeira pela aplicação de uma única regra de produção pertencente à gramática G. Temos quatro casos a considerar:*

1. *Se $p = A \rightarrow \beta \in P^i$ e $\alpha \in T^*$ e $\gamma \in (N \cup T)^*$ então dizemos que $\alpha A \gamma$ deriva diretamente $\alpha\beta\gamma$ na gramática G^i e denotamos*

$$\alpha A \gamma \Rightarrow_{G^i} \alpha\beta\gamma.$$

   Este tipo de derivação é chamada de "derivação interna".

2. *Se $p = A \rightarrow \phi \in P^i$ e $\alpha \in T^*$ e $\gamma \in (N \cup T)^*$ e se não houver outra produção aplicável então dizemos que $\alpha A \gamma$ não deriva nenhuma sentença (nem sequer a cadeia vazia) na gramática G^i e denotamos*

$$\alpha A \gamma \Rightarrow_{G^i} \phi.$$

3. *Se $p = \alpha A \leftarrow \beta B \in D^i$ onde $\alpha \in C^*$, $\beta \in C$ e $A, B \in N^i$ e, além disso, $\gamma \in T^*$ e $\delta \in (N \cup T)^*$ então dizemos que $\gamma\alpha A\delta$ deriva diretamente $\gamma\beta B\delta$ na gramática G^i e denotamos*

$$\gamma\alpha A\delta \Rightarrow_{G^i} \gamma\beta B\delta.$$

4. *Se $p = \alpha A \rightarrow \beta B \in D^i$ onde $\alpha \in C$, $\beta \in T^*$ e $A, B \in N^i$ e, além disso, $\gamma \in T^*$ e $\delta \in (N \cup T)^*$ então dizemos que $\gamma\alpha A\delta$ deriva diretamente $\gamma\beta B\delta$ na gramática G^i e denotamos*

$$\gamma\alpha A\delta \Rightarrow_{G^i} \gamma\beta B\delta.$$

Qualquer um destes quatro quatro casos é denominado uma "aplicação da produção p no ambiente da gramática" G_i.

Definição 3.1.10 Derivação Adaptativa de um passo. *A aplicação de uma produção adaptativa (produção de tipo 3) da forma $p = \alpha A \leftarrow \{\mathcal{A}\}\beta B \in D^i$ ou $p = \alpha A \rightarrow \{\mathcal{A}\}\beta B \in D^i$ é chamada "derivação adaptativa de um passo". Uma derivação adaptativa de um passo é representada na forma*

$$\omega_i \Rightarrow_{G^i} \{\mathcal{A}\}\omega_{i+1}$$

com $\omega_i = \gamma\alpha A\delta$, $\omega_{i+1} = \gamma\beta B\delta$ e tem o seguinte significado: primeiramente, executa-se a ação adaptativa $\mathcal{A}$, que gera a gramática G^{i+1}; a seguir aplica-se a regra de produção $p' = \alpha A \leftarrow \beta B \in D^i$ ou $p' = \alpha A \rightarrow \beta B \in D^i$ que atua no ambiente da gramática G^i, e, finalmente é feita a migração para a gramática G^{i+1}, descartando-se a gramática G^i.

Definição 3.1.11 Derivação. *Seja $G = (G^0, \mathbb{A}, R^0)$ uma gramática adaptativa. Uma derivação de uma palavra $w \in T^*$ segundo G é uma seqüência finita de derivações de um passo de G tal que*

i. O lado esquerdo da primeira derivação de um passo é o símbolo inicial de G,

ii. O lado direito da última derivação de um passo é w,

iii. O lado direito de cada uma das outras derivações de um passo é igual ao lado esquerdo da seguinte.

Esquematicamente:

$$S \Rightarrow_{G^0} w_1 \Rightarrow_{G^1} w_2 \Rightarrow_{G^2} w_3 ... w_{n-1} \Rightarrow_{G^n} w_n = w.$$

Definição 3.1.12 *A relação $\Rightarrow^*_G$ é o fecho reflexivo transitivo da relação $\Rightarrow_{G^i}$.*

Definição 3.1.13 Linguagem. *Seja $G = (G^0, \mathbb{A}, R^0)$ uma gramática adaptativa. A linguagem $L(G)$ aceita por G é:*

$$L(G) = \{w \in T^* : S \Rightarrow^*_G w\}$$

Exemplo 3.1.1 Gramática adaptativa $G = (G^0, \mathbb{A}, R^0)$ para a linguagem $L = \{a^n b^n c^n : n \geq 0\}$. *Consideremos a gramática adaptativa $G = (G^0, \mathbb{A}, R^0)$, com:*
$G^0 = (N, T, P, C, D, S)$
$N^0 = \{S, E, A, A_1, B, B_1, C, C_1\}$

$T = \{a, b, c\}$
$C^0 = \emptyset$
$D^0 = \emptyset$
$\mathbb{A} = \{\mathcal{A}, \mathcal{B}\}$
$R_0 = \{(A \to \{\mathcal{A}(A, A_1, B_1, C_1)\}aA_1, \mathcal{A}(A, A_1, B_1, C_1)), (A \to \{\mathcal{B}(B_1, C_1)\}\lambda, \mathcal{B}(B_1, C_1))\}$

$$\begin{array}{lll} P^0 = \{ \ p_1^0 = S \to EC, & p_2^0 = E \to AB, & p_3^0 = A \to \{\mathcal{A}(A, A_1, B_1, C_1)\}aA_1, \\ \quad p_4^0 = A \to \{\mathcal{B}(B_1, C_1)\}\lambda, & p_5^0 = B \to B_1, & p_6^0 = C \to C_1, \\ \quad p_7^0 = A_1 \to \phi, & p_8^0 = B_1 \to \phi, & p_9^0 = C_1 \to \phi\}, \end{array}$$

com $\mathcal{A}$ definido como especificado a seguir:

$\mathcal{A}(t, x, y, z) = \{u^*, v^*, w^* :$			$\mathcal{B}(x, y) = \{$
$-[t \to \{\mathcal{A}(t, x, y, z)\}ax]$,	$-[t \to \{\mathcal{B}(y, z)\}\lambda]$,	$+[t \to ax]$,	$+[x \to \lambda]$
$+[x \to \{\mathcal{A}(x, u, v, w)\}au]$,	$+[x \to \{\mathcal{B}(v, w)\}\lambda]$,	$+[y \to bv]$,	$+[y \to \lambda]$
$+[z \to cw]$,	$+[u \to \phi]$,	$+[v \to \phi]$,	$\}$
$+[w \to \phi]$			
$\}$			

Derivação de *aaabbbccc***:**

$$S \Rightarrow_{G^0} EC \Rightarrow_{G^0} ABC \Rightarrow_{G^0} \{\mathcal{A}(A, A_1, B_1, C_1)\}aA_1BC$$
$$\Rightarrow_{G^1} \{\mathcal{A}(A_1, A_2, B_2, C_2)\}aaA_2BC \Rightarrow_{G^2} \{\mathcal{A}(A_2, A_3, B_3, C_3)\}aaaA_3BC$$
$$\Rightarrow_{G^3} \{\mathcal{B}(B_4, C_4)\}aaa\lambda BC \Rightarrow_{G^4} aaaB_1C \Rightarrow_{G^4} aaabB_2C \Rightarrow_{G^4} aaabbB_3C$$
$$\Rightarrow_{G^4} aaabbbB_4C \Rightarrow_{G^4} aaabbb\lambda C \Rightarrow_{G^4} aaabbbC_1 \Rightarrow_{G^4} aaabbbcC_2$$
$$\Rightarrow_{G^4} aaabbbccC_3 \Rightarrow_{G^4} aaabbbcccC_4 \Rightarrow_{G^4} aaabbbccc\lambda.$$

Observação 3.1.1 *A gramática G^4 possui 32 produções e 16 não-terminais. Pode-se observar que a notação $\Rightarrow_{G^i} \{\mathcal{A}()\}$ indica a função adaptativa mas não a produção aplicada; na seção seguinte modificaremos esta notação para indicar a produção e a função adaptativa aplicadas. Anotamos a seguir os conjuntos das produções das gramáticas intermediárias para podermos indicar quais produções foram aplicadas em cada passo da derivação anterior.*

$$\begin{array}{lll} P^1 = (P^0 \setminus \{p_3^0, p_4^0\}) \cup \{p_3^1 = A \to aA_1\} \ \cup & & \\ \{ \ p_{10}^1 = A_1 \to \mathcal{A}(A_1, A_2, B_2, C_2)aA_2, & p_{11}^1 = A_1 \to \mathcal{B}(B_2, C_2)\lambda, & p_{12}^1 = B_1 \to bB_2, \\ \quad p_{13}^1 = C_1 \to cC_2, & p_{14}^1 = A_2 \to \phi, & p_{15}^1 = B_2 \to \phi, \\ \quad p_{16}^1 = C_2 \to \phi \ \} & & \end{array}$$

$$\begin{array}{lll} P^2 = (\ P^1 \setminus \{\ p_{10}^1,\ p_{11}^1\ \}\ ) \cup & \{p_{10}^2 = A_1 \to aA_2\} \cup & \\ \{ \ p_{17}^2 = A_2 \to \mathcal{A}(A_2, A_3, B_3, C_3)aA_3, & p_{18}^2 = A_2 \to \mathcal{B}(B_3, C_3)\lambda, & p_{19}^2 = B_2 \to bB_3, \\ \quad p_{20}^2 = C_2 \to cC_3, & p_{21}^2 = A_3 \to \phi, & p_{22}^2 = B_3 \to \phi, \\ \quad p_{23}^2 = C_3 \to \phi \ \} & & \end{array}$$

$$P^3 = (\ P^2 \setminus \{\ p^2_{17},\ p^2_{18}\ \}\ ) \cup \{\ p^3_{17} = A_2 \to aA_3\ \} \cup$$
$$\{\ p^2_{24} = A_3 \to \mathcal{A}(A_3, A_4, B_4, C_4)aA_4,\quad p^2_{25} = A_3 \to \mathcal{B}(B_4, C_4)\lambda,\quad p^2_{26} = B_3 \to bB_4,$$
$$p^2_{27} = C_3 \to cC_4,\quad p^2_{28} = A_4 \to \phi,\quad p^2_{29} = B_4 \to \phi,$$
$$p^2_{30} = C_4 \to \phi\ \}$$

$$P^4 = P^3 \cup \{\ p^4_{31} = B_4 \to \lambda,\quad p^4_{32} = C_4 \to \lambda\ \}$$

As produções aplicadas foram:

$$p^0_1, p^0_2, p^0_3, p^1_{10}, p^2_{17}, p^3_{25}, p^0_5, p^1_{12}, p^2_{19}, p^3_{26}, p^4_{31}, p^0_6, p^1_{13}, p^2_{20}, p^3_{27}, p^4_{32}.$$

3.2 Gramáticas Livres de Contexto Adaptativas

Nesta seção vamos apresentar as gramáticas livres de contexto adaptativas como dispositivos que identificam a classe das linguagens recursivamente enumeráveis. Esse formalismo é baseado em gramáticas livres de contexto, e este dispositivo especializa as gramáticas adaptativas estudadas na seção anterior.

Depois de definir a notação necessária, enunciamos e provamos a equivalência das gramáticas livres de contexto adaptativas com as gramáticas periodicamente variantes no tempo com verificação de aparência (vide seção 2.4); a prova é feita em duas etapas: na primeira etapa mostramos como construir uma função adaptativa que fornece controle sobre gramáticas livres de contexto, e que permite simular o funcionamento das gramáticas periodicamente variantes no tempo sem verificação de aparência. Na segunda etapa fazemos as consideraçõs necessárias para levar em conta o caso geral, que inclui a verificação de aparência.

Finalmente, o método empregado na prova desta equivalência é utilizado como mecanismo de tradução dos exemplos das gramáticas periodicamente variantes no tempo com verificação de aparência para gramáticas livres de contexto adaptativas com verificação de aparência, representando em nosso formalismo as versões correspondentes aos exemplos apresentados na seção 2.4.

Definição 3.2.1 Ações Adaptativas elementares de produções livres de contexto. *Seja $G = (N, T, P, S)$ uma gramática livre de contexto. Definimos duas funções, que denominaremos como as "ações adaptativas elementares" de "inserção de produções livres de contexto"*

$$+ : P \times 2^P \to 2^P,$$

e de "remoção de produções livres de contexto"

$$- : P \times 2^P \to 2^P,$$

pelas seguintes regras:

$$+(p, Q) = \begin{cases} Q & \text{se } p \in Q, \\ Q \cup \{p\}, & \text{se } p \notin Q, \end{cases} \qquad (\forall Q)(Q \in 2^P).$$

e

$$-(p, Q) = \begin{cases} Q \setminus \{p\}, & \text{se } p \in Q, \\ Q & \text{se } p \notin Q, \end{cases} \qquad (\forall Q)(Q \in 2^P).$$

A idéia é que o resultado da aplicação de (o valor de) uma ação elementar de inserção ou remoção em uma produção da gramática venha a ser o novo conjunto

de produções da gramática livre de contexto adaptativa; isto é, depois de aplicar a ação adaptativa elementar de inserção à produção livre de contexto p no conjunto de produções P da gramática $G = (N, T, P, S)$ (ao qual p pode não pertencer) obtemos a nova gramática $G = (N, T, +(p, P), S)$ e, depois de aplicar a ação adaptativa elementar de remoção à produção livre de contexto p no conjunto de produções P da gramática $G = (N, T, P, S)$ (ao qual p pode pertencer) obtemos a nova gramática $G = (N, T, -(p, P), S)$. Assim, as ações adaptativas elementares de produções livres de contexto, agem sempre sobre o "conjunto de produções corrente" da gramática.

Notação 3.2.1 Ações Adaptativas elementares de produções livres de contexto. *Para manter compatibilidade com a notação dos autômatos adaptativos[9], usaremos colchetes em lugar de parênteses na notação das ações adaptativas elementares. Além disso, para salientar o fato de que as ações elementares tratam indistintamente qualquer valor de seu segundo parâmetro em sua definição, simplesmente omitiremos esse segundo parâmetro (o conjunto de produções corrente, que fica subentendido). Assim poderemos escrever $+[A \to aA]$, $-[S \to ABC]$, etc.*

A definição das funções adaptativas que acompanham uma gramática livre de contexto adaptativa pode ser feita de duas formas:

1. Uma única função adaptativa, que é uma função parcial do conjunto de partes das produções da gramática livre de contexto subjacente nele próprio,

2. Uma família de funções adaptativas, cada uma das quais é uma função total em algum subconjunto de produções.

Observação 3.2.1 *Pode-se passar da primeira à segunda forma ao agrupar as produções que são mapeadas para os mesmos valores segundo esta função parcial.*

Vamos introduzir as funções adaptativas na primeira forma, restando a segunda forma para ser ilustrada através de exemplos, mais adiante.

Definição 3.2.2 Função Adaptativa de produções livres de contexto. *Seja $G = (N, T, P, S)$ uma gramática livre de contexto. Uma função adaptativa de produções livres de contexto de G*

$$\mathcal{A} : P \times 2^P \to 2^P$$

é uma regra de transformação do conjunto de produções da gramática livre de contexto G. A transformação é especificada usando-se uma quantidade finita de ações

adaptativas elementares aae_i de produções livres de contexto, associadas a produções p_i específicas,

$$\mathcal{A}(p, Q) = \{aae_i[p_i] : 1 \leq i \leq n\}, \qquad (p, Q) \in P \times 2^P,$$

tal que qualquer aplicação de ação elementar de remoção precede a todas as aplicações de ações elementares de inserção; o valor da função adaptativa é calculado segundo as seguintes regras:

$$\begin{array}{ll} Q_0 & = Q \\ Q_i & = aae_i(p_i, Q_{i-1}), (\forall i)(1 \leq i \leq n) \\ \mathcal{A}(p, Q) & = Q_n, \qquad (\forall Q)(Q \in 2^P) \end{array}$$

Notação 3.2.2 *A Função Adaptativa de produções livres de contexto opera sobre o "conjunto de produções corrente" da gramática; dessa forma, o segundo parâmetro é omitido; enfatizaremos isto omitindo o segundo parâmetro da notação e escreveremos, simplesmente, $\mathcal{A}(p)$.*

Definição 3.2.3 Produção livre de contexto adaptativa. *Consideremos uma gramática livre de contexto $G = (N, T, P, S)$, uma função adaptativa $\mathcal{A}$ das produções livres de contexto de G e uma produção $p = L \rightarrow R \in P$, $L \in N$, $R \in (N \cup T)^*$. A produção livre de contexto adaptativa associada a p é:*

$$[p] = L \rightarrow \{\mathcal{A}()\}R,$$

cuja semântica é:

1. *Aplica-se a produção livre de contexto p,*
2. *Aplica-se a função adaptativa $\mathcal{A}$ na produção p: $\mathcal{A}(p)$*

O conjunto de todas as produções livres de contexto adaptativas associadas à gramática G é $[P]$.

Estendemos agora a definição 3.2.1 para incluir as produções adaptativas livres de contexto.

Definição 3.2.4 Ações Adaptativas elementares de produções livres de contexto adaptativas. *Seja $G = (N, T, P, S)$ uma gramática livre de contexto e $\mathcal{A}$ uma função adaptativa das produções livres de contexto de G. Definimos duas funções que denominaremos "ações adaptativas elementares" de "inserção"*

$$+ : [P] \times 2^{[P]} \rightarrow 2^{[P]},$$

e de "remoção"

$$- : [P] \times 2^{[P]} \to 2^{[P]},$$

pelas seguintes regras:

$$+([p],[Q]) = \begin{cases} [Q] & \text{se } [p] \in [Q], \\ [Q] \cup \{[p]\} & \text{se } [p] \notin [Q], \end{cases} \qquad (\forall [Q])([Q] \in 2^{[P]}).$$

e

$$-([p],[Q]) = \begin{cases} [Q] \setminus \{[p]\}, & \text{se } [p] \in [Q], \\ [Q] & \text{se } [p] \notin [Q], \end{cases} \qquad (\forall [Q])([Q] \in 2^{[P]}).$$

Depois de aplicar a ação adaptativa elementar de inserção à produção adaptativa livre de contexto $[p]$ do conjunto de produções adaptativas $[P]$ associadas à gramática $G = (N, T, P, S)$ (ao qual $[p]$ pode não pertencer) obtemos a nova gramática $G = (N, T, +(p, P), S)$ com conjunto de produções livres de contexto adaptativas $[+(p, P)]$ e, depois de aplicar a ação adaptativa elementar de remoção à produção adaptativa livre de contexto $[p]$ no conjunto de produções adaptativas $[P]$ da gramática $G = (N, T, P, S)$ (ao qual $[p]$ pode pertencer) obtemos a nova gramática $G = (N, T, -(p, P), S)$ com conjunto de produções livres de contexto adaptativas $[-(p, P)]$. Assim, as ações adaptativas elementares de produções adaptativas livres de contexto, agem sempre sobre o "conjunto corrente" de produções adaptativas associadas à gramática. G.

Notação 3.2.3 Ações Adaptativas elementares de produções livres de contexto adaptativas. *Novamente, para manter compatibilidade com Neto[9], usaremos colchetes em lugar de parênteses na notação das ações adaptativas elementares. Além disso, para salientar o fato de que as ações elementares tratam indistintamente qualquer valor de seu segundo parâmetro em sua definição, simplesmente omitiremos esse segundo parâmetro (que fica subentendido). Assim se $p_1 = A \to aA$, $p_2 = S \to ABC$ então escrevereremos $+[[p_1]] = +[A \to \{\mathcal{A}\}aA]$, $-[[p_2]] = -[S \to \{\mathcal{A}\}ABC]$, etc. Daqui em diante nos referiremos às ações adaptativas elementares de produções livres de contexto adaptativas simplesmente como "ações adaptativas elementares".*

Definição 3.2.5 Função Adaptativa. *Seja $G = (N, T, P, S)$ uma gramática livre de contexto. Uma função adaptativa de produções livres de contexto adaptativas de G*

$$\mathcal{A} : [P] \times 2^{[P]} \to 2^{[P]},$$

é uma regra de transformação do conjunto de produções livres de contexto adaptativas associadas à gramática G. A transformação é especificada usando-se uma

quantidade finita de ações adaptativas elementares aae_i, aplicadas em produções livres de contexto adaptativas $[p_i]$ específicas,

$$\mathcal{A}([p],[Q]) = \{aae_i[[p_i]] : 1 \leq i \leq n\},$$

tal que qualquer aplicação da ação elementar de remoção precede a todas as aplicações da ação elementar de inserção; o valor da função adaptativa é calculado segundo as regras:

$$\begin{array}{ll} [Q_0] & = [Q] \\ [Q_i] & = aae_i([p_i],[Q_{i-1}]), (\forall i)(1 \leq i \leq n) \\ \mathcal{A}([p],[Q]) & = [Q_n], \qquad (\forall [Q])([Q] \in 2^{[P]}) \end{array}$$

Notação 3.2.4 *A Função Adaptativa de produções livres de contexto adaptativas opera sobre o "conjunto de produções corrente" da gramática. Dessa forma, o segundo parâmetro é omitido; enfatizaremos isto omitindo o segundo parâmetro da notação e escreveremos, simplesmente, $\mathcal{A}([p])$.*

Observação 3.2.2 *Pela definição acima, uma função adaptativa pode conter, na definição das ações adaptativas elementares que a definem, chamadas de funções adaptativas. Em particular, ela pode conter chamadas a si própria; é por isso que podemos omitir os parâmetros da definição das funções adaptativas: cada ação elementar adaptativa tem como parâmetro uma produção livre de contexto adaptativa $[p]$ que indica qual a função adaptativa que se aplica depois de aplicar a produção p.*

Notação 3.2.5 *O formato geral de uma função adaptativa é o seguinte*

$\mathcal{A}()$
{
ação adaptativa elementar 1
ação adaptativa elementar 2
...
ação adaptativa elementar n
}

a semântica de uma função adaptativa requer que sejam aplicadas em primeiro lugar, simultaneamente, todas as ações adaptativas elementares de remoção que a definem e, depois, sejam aplicadas, simultaneamente, todas as ações adaptativas elementares de inserção que a definem.

Definição 3.2.6 Gramática livre de contexto adaptativa. *Consideremos uma gramática livre de contexto $G = (N, T, P, S)$, uma função adaptativa $\mathcal{A}$ das produções livres de contexto de G e P_0, F dois subconjuntos de P. Nestas condições, a quádrupla $GA = (G, [P_0], [F], \mathcal{A})$ é dita uma gramática livre de contexto adaptativa, sendo:*

1. *G é a gramática livre de contexto subjacente,*
2. *$[P_0]$ é o conjunto das produções livres de contexto adaptativas iniciais de GA,*
3. *$[F]$ é o conjunto das produções adaptativas que se aplicam em modo de verificação de aparência,*
4. *$\mathcal{A}$ é a função adaptativa de GA.*

Definição 3.2.7 Aplicação de produção livre de contexto adaptativa. *Sejam $GA = (G, [P_0], [F], \mathcal{A})$ uma gramática livre de contexto adaptativa, onde $G = (N, T, P, S)$, e $[p] = E \to \{\mathcal{A}()\}D \in [P]$, $E \in N$, $D \in (N \cup T)^*$, uma produção livre de contexto adaptativa de GA. Se $X, Y \in (N \cup T)^*$, dizemos que X deriva diretamente Y e escrevemos*

$$X \Rightarrow Y$$

se existem $r, s \in (N \cup T)^*$ tais que $X = rEs$ e $Y = rDs$. Assim sendo, se o conjunto de produções livres de contexto adaptativas de GA é P_{i-1}, $i \geq 1$, então o "conjunto de produções livres de contexto adaptativas aplicáveis depois da aplicação de $[p]$" é definido como $P_i = \mathcal{A}(p)$. Nestas condições também dizemos que $X \Rightarrow Y$ é válida. Costuma-se dizer que a produção $[p]$ foi aplicada à forma sentencial X.

Estendemos agora o conceito de aplicação de produção adaptativa para cobrir o caso da verificação de aparência, parafraseando a definição correspondente para o caso das gramáticas livres de contexto programadas (capítulo 2, seção 3):

Definição 3.2.8 Aplicação de produção livre de contexto adaptativa em modo de verificação de aparência. *Sejam $GA = (G, [P_0], [F], \mathcal{A})$ uma gramática livre de contexto adaptativa, onde $G = (N, T, P, S)$, e, $[p] = E \to \{\mathcal{A}()\}D \in [P]$ uma produção livre de contexto adaptativa de GA. Se $X, Y \in (N \cup T)^*$, dizemos que X deriva diretamente Y, em modo de verificação de aparência, e escrevemos:*

$$X \Rightarrow^{ac} Y$$

se $X \Rightarrow Y$ for válida, ou então, se cada uma das seguintes condições for satisfeita:

(i) $Y = X$,

(ii) $[p] = E \to \{\mathcal{A}()\}D$ *é uma produção de* $[F]$ *tal que* E *não aparece em* X.

Nestas condições, também dizemos que $X \Rightarrow^{ac} Y$ é válida.

Definição 3.2.9 Derivação de k passos. *Sejam $X, Y \in (N \cup T)^*$. Uma derivação de k passos desde X até Y é uma seqüência finita de formas sentenciais $w_i \in (N \cup T)^*, 0 \leq i \leq k$ tais que*

1. $X = w_0$,
2. $w_{i-1} \Rightarrow^{ac} w_i$ *é válida, para* $1 \leq i \leq k$,
3. $Y = w_k$

Nestas condições denotamos $X \Rightarrow^{ac^k} Y$. Assim, a aplicação de uma produção é uma derivação de um passo $X \Rightarrow^{ac^1} Y$. Consideramos $[P_0]$ como o "conjunto de produções aplicáveis imediatamente após o passo zero" e consideramos $[P_i]$, $1 \leq i \leq k$, como o "conjunto de produções aplicáveis imediatamente após o passo i".

Definição 3.2.10 *O fechamento reflexivo-transitivo da relação binária $\Rightarrow$ é denotado como $\Rightarrow^*$ e o fechamento reflexivo-transitivo da relação binária $\Rightarrow^{ac}$ é denotado como $\Rightarrow^{ac^*}$.*

Definição 3.2.11 *Seja $G = (N, T, P, S)$ uma gramática livre de contexto. A linguagem gerada por uma gramática livre de contexto adaptativa $GA = (G, [P_0], [F], \mathcal{A})$ é $L(GA) = \{w \in T^* : S \Rightarrow^{ac^*} w\}$*

Observação 3.2.3 *Em geral, $P_0 \subseteq P$ e, embora o "conjunto de produções aplicáveis" $[P_i]$ possa mudar durante o curso de uma derivação, ele sempre corresponde a um subconjunto P_i do conjunto de produções da gramática livre de contexto subjacente $G = (N, T, P, S)$. Portanto, no caso das gramáticas livres de contexto adaptativas, uma derivação não é uma trajetória entre um conjunto potencialmente infinito de gramáticas criadas a partir de uma gramática inicial G^0, mas uma trajetória entre os elementos de um conjunto finito de gramáticas, todas as quais possuem conjuntos de produções que correspondem a um subconjunto do conjunto de produções P da gramática livre de contexto subjacente G. É por isso que, em lugar de usar G^i como subíndice nas derivações $\Rightarrow_{G^i}$, indicaremos explicitamente qual produção livre de contexto adaptativa $[p]$ foi aplicada escrevendo $\Rightarrow_{[p]}$. Reservamos ainda $\Rightarrow^{ac}_{[p]}$ exclusivamente para indicar que a produção se aplica em modo de verificação de aparência, isto é $[p] \in [F]$, e usamos $\Rightarrow_{[p]}$ para toda produção $[p]$ tal que $[p] \in [P \setminus F]$. Assim mesmo, se nenhuma das produções, aplicadas numa derivação de k passos, $X \Rightarrow^{ac^k} Y$, pertence ao conjunto $[F]$, então escreveremos, simplesmente, $X \Rightarrow^k Y$.*

Notação 3.2.6 *Quando for necessário distinguir uma particular gramática adaptativa GA segundo a qual foi feita uma derivação de k passos, com ou sem verificação*

de aparência, e para os respectivos fechamentos reflexivo-transitivos, escreveremos $X \Rightarrow_{GA}^{ac^k} Y$, $X \Rightarrow_{GA}^{k} Y$ $X \Rightarrow_{GA}^{ac^*} Y$, *e* $X \Rightarrow_{GA}^{*} Y$, *respectivamente. Utilizaremos esta conveção notacional também para gramaticas matriciais* GM, *programadas* GP, *variantes no tempo* GV *e com linguagem de controle regular* GR.

Notação 3.2.7 *Definimos agora algumas classes de linguagens associadas às gramáticas adaptativas livres de contexto:*

1. $\mathcal{LA}_{ac}^{\lambda}$ *Classe das linguagens geradas pelas gramáticas livres de contexto adaptativas com verificação de aparência.*

2. $\mathcal{LA}_{ac}$ *Subconjunto de* $\mathcal{LA}_{ac}^{\lambda}$ *obtido considerando apenas as gramáticas livres de contexto que não geram* λ.

3. $\mathcal{LA}^{\lambda}$ *Subconjunto de* $\mathcal{LA}_{ac}^{\lambda}$ *obtido considerando apenas as gramáticas livres de contexto adaptativas com* $F = \emptyset$.

4. $\mathcal{LA}$ *Subconjunto de* $\mathcal{LA}^{\lambda}$ *obtido considerando apenas as gramáticas livres de contexto que não geram* λ.

Proposição 3.2.1

$$\mathcal{LA} \subseteq \mathcal{LA}^{\lambda} \subseteq \mathcal{LA}_{ac}^{\lambda} \qquad \mathcal{LA} \subseteq \mathcal{LA}_{ac} \subseteq \mathcal{LA}_{ac}^{\lambda}$$

Enunciamos a seguir o resultado central desta seção.

Teorema 3.2.1 *A classe* $\mathcal{LA}_{ac}^{\lambda}$ *das linguagens geradas pela classe das gramáticas livres de contexto adaptativas coincide com a classe* $\mathcal{RE}$ *das linguagens recursivamente enumeráveis.*

Para provar o teorema demostraremos inicialmente o lema seguinte:

Lema 3.2.2 *A classe* $\mathcal{LA}^{\lambda}$ *das linguagens geradas pela classe das gramáticas livres de contexto adaptativas sem verificação de aparência contém a classe* $\mathcal{T}^{\lambda}$ *das linguagens geradas pela classe das gramáticas periodicamente variantes no tempo sem verificação de aparência.*

Prova Seja $GT = (G, v, \emptyset) = (G, v)$ uma gramática periodicamente variante no tempo sem verificação de aparência, com $G = (N, T, P, S)$, e seja $L = L(GT)$ a linguagem que ela gera. Vamos construir uma gramática adaptativa GA tal que $L(GA) = L(GT)$.

Com efeito, seja p o período da função de variação $v : N \to 2^P$:

$$v(i) = \begin{cases} P_1, & i = 1 \\ P_2, & i = 2 \\ ..., & ... \\ P_p, & i = p \\ v(i-p), & i \geq p. \end{cases}$$

em que os $P_i \subseteq P$, i.e., são subconjuntos do conjunto das produções P. Definimos agora uma gramática livre de contexto adaptativa sem verificação de aparência $GA = (G, [P_1], \emptyset, \mathcal{A})$ que coincide com a anterior no conjunto de não-terminais e no conjunto de terminais, possui o mesmo símbolo inicial e seu conjunto de produções é definido como:

$$[P_1] = \{E \to \{\mathcal{A}_1()\}D : E \to D \in P_1\}.$$

Além disso, definimos as suas funções adaptativas da seguinte forma:

$$\mathcal{A}_1() = \begin{cases} -[P_1] \\ +[P_2] \end{cases}$$

$$\mathcal{A}_2() = \begin{cases} -[P_2] \\ +[P_3] \end{cases}$$

$$...$$

$$\mathcal{A}_p() = \begin{cases} -[P_p] \\ +[P_1] \end{cases}$$

com

$$-[P_i] = \{-[E \to \{\mathcal{A}_i()\}D] : E \to D \in P_i\}$$
$$+[P_i] = \{+[E \to \{\mathcal{A}_{i+1}()\}D] : E \to D \in P_{i+1}\}$$

para $1 \leq i \leq p-1$, e

$$-[P_p] = \{-[E \to \{\mathcal{A}_p()\}D] : E \to D \in P_p\}$$
$$+[P_p] = \{+[E \to \{\mathcal{A}_1()\}D] : E \to D \in P_1\}$$

Vamos mostrar agora que $L(GA) = L(GT)$.

$L(GT) \subset L(GA)$
Com efeito seja $w \in L(GT)$, i.e., existe uma derivação de w segundo GT, digamos:

$$S \Rightarrow^*_{GT} w$$

Como G deriva palavras segundo sua função periódica de variação v, podemos concluir que o comprimento da derivação acima, denotado $|S \Rightarrow^*_{GT} w|$, satisfaz à seguinte equação:

$$|S \Rightarrow^*_{GT} w| = kp + r$$

sendo k um número inteiro positivo e $0 \leq r \leq p-1$. Esquematicamente, podemos escrever a derivação acima como:

$$S \Rightarrow^{kp}_{GT} w_{kp} \Rightarrow^{kp+1}_{GT} w_{kp+1} \dots w_{kp+i-1} \Rightarrow^{kp+i}_{GT} w_{kp+i} \dots w_{kp+r-1} \Rightarrow^{kp+r}_{GT} w_{kp+r} = w$$

Concentrar-nos-emos agora nos últimos r passos da derivação acima. Pela natureza de v em cada um desses passos, devem ter sido escolhidas produções pertencentes, respectivamente, aos conjuntos $v(1) = P_1, v(2) = P_2, v(3) = P_3, ..., v(r) = P_r$. Sejam $L_1 \to R_1, L_2 \to R_2, L_3 \to R_3, ..., L_r \to R_r$ as produções assim escolhidas. Vamos agora mostrar que podemos construir uma derivação em GA tal que

$$w_{kp} \Rightarrow^*_{GA} w$$

Com efeito, a produção $E_1 \to D_1$, corresponde à produção adaptativa $E_1 \to \{\mathcal{A}_1()\}D_1$ da gramática adaptativa GA. Adicionalmente, ao aplicá-la obtemos um novo conjunto de produções, entre as quais se conta a produção adaptativa $E_2 \to \{\mathcal{A}_2()\}D_2$ correspondente à produção $E_2 \to D_2$; por sua vez, esta última produção adaptativa nos fornece, quando aplicada, um conjunto de produções adaptativas entre as quais se conta a produção adaptativa $E_3 \to \{\mathcal{A}_3()\}D_3$ correspondente à produção $E_3 \to D_3$. Continuando deste modo, chegamos – após r passos – à produção adaptativa $E_r \to \{\mathcal{A}_r()\}D_r$ correspondente à produção $E_r \to D_r$; resumimos estas observações na seguinte tabela

L.E.	Produção	Prod. adap.	L.D.	Conjunto de produções adaptativas
w_{kp}	$E_1 \to D_1$	$E_1 \to \{\mathcal{A}_1()\}D_1$	w_{kp+1}	$[P_1] = \{E \to \{\mathcal{A}_1()\}D : E \to D \in P_1\}$
w_{kp+1}	$E_2 \to D_2$	$E_2 \to \{\mathcal{A}_2()\}D_2$	w_{kp+2}	$[P_2] = \{E \to \{\mathcal{A}_2()\}D : E \to D \in P_2\}$
w_{kp+2}	$E_3 \to D_3$	$E_3 \to \{\mathcal{A}_3()\}D_3$	w_{kp+3}	$[P_3] = \{E \to \{\mathcal{A}_3()\}D : E \to D \in P_3\}$
...	...	...	...	...
w_{kp+i-1}	...	...	w_{kp+i}	$[P_i] = \{E \to \{\mathcal{A}_i()\}D : E \to D \in P_i\}$
...	...	...	...	...
w_{kp+r-1}	$E_r \to D_r$	$E_r \to \{\mathcal{A}_r()\}D_r$	w	$[P_r] = \{E \to \{\mathcal{A}_r()\}D : E \to D \in P_r\}$

Dessa forma, podemos re-escrever os últimos r passos da derivação original como:

$$w_{kp} \Rightarrow^{kp+1}_{E_1 \to D_1} w_{kp+1} \dots w_{kp+i-1} \Rightarrow^{kp+i}_{E_i \to D_i} w_{kp+i} \dots w_{kp+r-1} \Rightarrow^{kp+r}_{E_r \to D_r} w,$$

aos quais corresponde a seguinte derivação adaptativa de r passos:

$$w_{kp} \Rightarrow^{kp+1}_{E_1 \to \{\mathcal{A}_1()\}D_1} w_{kp+1} \dots w_{kp+i-1} \Rightarrow^{kp+i}_{E_i \to \{\mathcal{A}_i()\}D_i} w_{kp+i} \dots w_{kp+r-1} \Rightarrow^{kp+r}_{E_r \to \{\mathcal{A}_r()\}D_r} w.$$

Resta agora provar que para toda derivação de kp passos segundo GT:

$$S \Rightarrow_{GT}^{kp} w_{kp},$$

existe uma derivação adaptativa de kp passos segundo GA:

$$S \Rightarrow_{GA}^{kp} w_{kp}.$$

Vamos provar uma afirmativa mais geral e obteremos o resultado acima como um caso particular: Dadas duas formas sentenciais $w_0, w_{kp} \in (N \cup T)^*$ então para toda derivação de kp passos desde w_0 até w_{kp} segundo GT:

$$w_0 \Rightarrow_{GT}^{kp} w_{kp},$$

existe uma derivação adaptativa de kp passos desde w_0 até w_{kp} segundo GA:

$$w_0 \Rightarrow_{GA}^{kp} w_{kp}.$$

Fazemos a demonstração desta afirmativa por indução matemática em k.
Base $k = 1$**.** A afirmativa a ser demonstrada, no caso $k = 1$, pode ser escrita, esquematicamente, da seguinte forma:

$$(\forall)(w_0 \Rightarrow_{GT}^{p} w_p) \Longrightarrow (\exists)(w_0 \Rightarrow_{GA}^{p} w_p)$$

Consideremos uma derivação, $w_0 \Rightarrow_{GT}^{p} w_p$, de p passos desde w_0 até w_p segundo GT; ela pode ser decomposta da seguinte forma:

$$w_0 \Rightarrow_{GT}^{1} w_1 \ldots w_{i-1} \Rightarrow_{GT}^{i} w_i \ldots w_{p-1} \Rightarrow_{GT}^{p} w_p.$$

explicitando as produções aplicadas em cada passo da derivação obtemos:

$$w_0 \Rightarrow_{E_1 \to D_1}^{1} w_1 \ldots w_{i-1} \Rightarrow_{E_i \to D_i}^{i} w_i \ldots w_{p-1} \Rightarrow_{E_p \to D_p}^{p} w_p,$$

a essa derivação corresponde a seguinte derivação adaptativa de p passos desde w_0 até w_{kp} segundo GA:

$$w_0 \Rightarrow_{E_1 \to \{\mathcal{A}_1()\} D_1}^{1} w_1 \ldots w_{i-1} \Rightarrow_{E_i \to \{\mathcal{A}_i()\} D_i}^{i} w_i \ldots w_{p-1} \Rightarrow_{E_p \to \{\mathcal{A}_p()\} D_p}^{p} w_p,$$

isto é

$$w_0 \Rightarrow_{GA}^{1} w_1 \ldots w_{i-1} \Rightarrow_{GA}^{i} w_i \ldots \Rightarrow_{GA}^{p-1} w_{p-1} \Rightarrow_{GA}^{p} w_p.$$

Ou seja, obtivemos uma derivação, $S \Rightarrow_{GA}^{p} w_p$, de p passos desde w_0 até w_{kp} segundo GA.

Hipótese Indutiva: Seja $k > 1$. Assumimos, como hipótese indutiva, que a afirmativa é válida para $h = k-1$. Podemos escrever essa hipótese indutiva da seguinte forma:

$$(\forall)(w_0 \Rightarrow_{GT}^{(k-1)p} w_{(k-1)p}) \Longrightarrow (\exists)(w_0 \Rightarrow_{GA}^{(k-1)p} w_{(k-1)p})$$

Consideramos agora uma derivação, $w_0 \Rightarrow_{GT}^{kp} w_{kp}$, de kp passos desde w_0 até w_{kp} segundo GT e a decompomos da seguinte forma:

$$w_0 \Rightarrow_{GT}^{(k-1)p} w_{(k-1)p} \Rightarrow_{GT}^{kp} w_{kp}$$

Neste caso, pela hipótese indutiva, temos uma derivação de $(k-1)p$ passos desde w_0 até w_{kp} segundo GA:

$$w_0 \Rightarrow_{GA}^{(k-1)p} w_{(k-1)p},$$

correspondente a

$$w_0 \Rightarrow_{GT}^{(k-1)p} w_{(k-1)p}$$

e, pelo caso base, temos uma derivação de p passos desde $w_{(k-1)p}$ até w_{kp} segundo GA:

$$w_{(k-1)p} \Rightarrow_{GA}^{p} w_{kp},$$

correspondente a

$$w_{(k-1)p} \Rightarrow_{GT}^{p} w_{kp}.$$

Assim podemos juntar essas duas derivações da seguinte forma:

$$w_0 \Rightarrow_{GA}^{(k-1)p} w_{(k-1)p} \Rightarrow_{GA}^{p} w_{kp}$$

e obtemos a derivação $w_0 \Rightarrow_{GA}^{kp} w_{kp}$ de kp passos desde w_0 até w_{kp} segundo GA correspondente a $w_0 \Rightarrow_{GT}^{kp} w_{kp}$.
Isto completa a indução e termina a prova. O resultado segue escolhendo $w_0 = S$ e $w_p \in (N \cup T)^*$.
□

Para generalizar o lema anterior para o caso das gramáticas livres de contexto adaptativas com verificação de aparência, basta modificar a sua prova para considerar esta forma mais abrangente de aplicação de produção; temos assim o seguinte:

Corolário 3.2.3 *A classe $\mathcal{LA}_{ac}^{\lambda}$ das linguagens geradas pela classe das gramáticas livres de contexto adaptativas com verificação de aparência contém a classe $\mathcal{T}_{ac}^{\lambda}$ das linguagens geradas pela classe das gramáticas periodicamente variantes no tempo com verificação de aparência.*

Em lugar de repetir os detalhes do lema 3.2.2, para provar o corolário 3.2.3, vamos apresentar outro resultado que permite traduzir gramáticas livres de contexto contexto com linguagem de controle regular com verificação de aparência para gramáticas livres de contexto adaptativas com verificação de aparência; novamente a prova é construtiva.

Lema 3.2.4 *A classe $\mathcal{LA}_{ac}^{\lambda}$ das linguagens geradas pela classe das gramáticas livres de contexto adaptativas com verificação de aparência contém a classe $\mathcal{R}_{ac}^{\lambda}$ das linguagens geradas pela classe das gramáticas livres de contexto com linguagem de controle regular com verificação de aparência.*

Prova
Com efeito seja $GR = (G, C, F)$ uma gramática com linguagem de controle regular sendo $G = (N, T, P, S)$ uma gramática livre de contexto, C um conjunto regular sobre o alphabeto P e $F \subset P$ um subconjunto das produções de G que se aplicam em modo de verificação de aparência. Seja $L = L(GR)$ a linguagem gerada por GR. Vamos construir uma gramática adaptativa GA tal que $L(GA) = L(GR)$.

Dividimos a prova em duas partes. Na primeira, mostramos que para toda expressão regular er sobre F podemos construir uma função adaptativa $\mathcal{A}_{er}$ que age como a função sucessor das letras que aparecem nas palavras da linguagem definida pela expressão regular er. Na segunda, aplicamos este resultado a uma expressão regular que define a linguagem de controle da gramática GR e mostramos que a toda derivação de uma palavra w segundo GR corresponde uma derivação da palavra w segundo $GA = (G, [P_0], [F], \mathcal{A}_{er})$, com $P_0 \subset P$.

Primeira parte. Vamos mostrar que, para toda expressão regular er sobre P, existe uma função adaptativa $\mathcal{A}_{er}$ tal que se $w \in L(er)$, onde $w = p_1 \dots p_n$, $p_i \in P$, $1 \leq i \leq n$, então $p_{i+1} \in \mathcal{A}_{er}(p_i)$, $1 \leq i < n$.

Base $|er| = 1$. Neste caso a $er = p$, onde $p \in P$ é uma produção da gramática G e isto quer dizer que as únicas produções de G que produzem palavras de $L(GR)$ devem ser aquelas que produzem palavras terminais ou a cadeia vazia λ; seja $p = E \to D \in P$ uma tal produção, isto é, $E \in N$, $D \in (N \cup T)^*$. Basta definir a função adaptativa $\mathcal{A}$ para estas produções, obtendo as correspondentes $[p] = E \to \{\mathcal{A}\}D \in [P]$.

Hipótese Indutiva: Seja er uma expressão regular sobre P com $|er| = k > 1$ e suponhamos que, para $(\forall)(0 \leq h < k)$ a seguinte afirmativa é válida: Para toda expressão regular er_h sobre P, tal que $|er_h| = h$ existe uma função adaptativa $\mathcal{A}_{er_h}$ tal que se $w \in L(er_h)$, onde $w = p_1 \dots p_n$, $p_i \in P$, $1 \leq i \leq n$, então $[p_{i+1}] \in \mathcal{A}_{er_h}([p_i])$,

$1 \leq i < n$.

Temos três casos a considerar:

1. $er = er_1^*$: Neste caso, pela hipótese indutiva, existe uma função adaptativa $\mathcal{A}_{er_1}$ tal que se $w \in L(er_1)$, com $w = p_1 \dots p_{|w|}$, $p_i \in P$, $1 \leq i \leq |w|$, então $[p_{i+1}] \in \mathcal{A}_{er_1}([p_i])$, $1 \leq i < |w|$.

   Seja $P_u^1 \subseteq P$ o conjunto de todas as produções $p \in P$ que são os últimos símbolos de alguma palavra definida pela expressão regular er_1. Seja $P_0^1 \subseteq P$ o conjunto de todas as produções $p \in P$ que são os primeiros símbolos de alguma palavra definida pela expressão regular er_1. Os conjuntos P_0^1 e P_u^1 são finitos pois P é finito. Definimos a função adaptativa

$$\mathcal{A}_{er}([p]) = \begin{cases} \mathcal{A}_{er_1}([p]) & \text{se } p \in P \setminus P_u^1 \\ \mathcal{A}_{er_1}([p]) \cup [P_0^1] & \text{se } p \in P_u^1 \end{cases}$$

2. $er = er_1 er_2$: Neste caso, pela hipótese indutiva, existem duas funções adaptativas $\mathcal{A}_{er_1}$ e $\mathcal{A}_{er_2}$ tais que

   (a) se $w_1 \in L(er_1)$, com $w_1 = p_1^1 \dots p_{|w_1|}^1$, $p_i \in P$, $1 \leq i \leq |w_1|$, então $[p_{i+1}^1] \in \mathcal{A}_{er_1}([p_i^1])$, $1 \leq i < |w_1|$,

   (b) se $w_2 \in L(er_2)$, com $w_2 = p_1^2 \dots p_{|w_2|}^2$, $p_j \in P$, $1 \leq j \leq |w_2|$, então $]p_{j+1}^2] \in \mathcal{A}_{er_2}([p_j^2])$, $1 \leq j < |w_2|$.

   Definimos os conjuntos P_0^1 e P_u^1 para a função adaptativa $\mathcal{A}_{er_1}$ como no caso 1 e os correspondentes conjuntos P_0^2 e P_u^2 para a função adaptativa $\mathcal{A}_{er_2}$. Além disso sejam $P_{w_1} = \{p_i^1 \in P : 1 \leq i \leq |w_1|\}$ e $P_{w_2} = \{p_j^2 \in P : 1 \leq j \leq |w_2|\}$ Definimos agora a função adaptativa

$$\mathcal{A}_{er}([p]) = \begin{cases} \mathcal{A}_{er_1}([p]) & \text{se } p \in P_{w_1} \setminus P_u^1 \\ \mathcal{A}_{er_1}([p]) \cup [P_0^2] & \text{se } p \in P_u^1 \\ \mathcal{A}_{er_2}([p]) & \text{se } p \in P_{w_2} \end{cases}$$

3. $er = (er_1|er_2)$: Neste caso, pela hipótese indutiva, existem duas funções adaptativas $\mathcal{A}_{er_1}$ e $\mathcal{A}_{er_2}$ tais que

   (a) se $w_1 \in L(er_1)$, com $w_1 = p_1^1 \dots p_{|w_1|}^1$, $p_i \in P$, $1 \leq i \leq |w_1|$, então $p_{i+1}^1 \in \mathcal{A}_{er_1}(p_i^1)$, $1 \leq i < |w_1|$,

   (b) se $w_2 \in L(er_2)$, com $w_2 = p_1^2 \dots p_{|w_2|}^2$, $p_j \in P$, $1 \leq j \leq |w_2|$, então $p_{j+1}^2 \in \mathcal{A}_{er_2}(p_j^2)$, $1 \leq j < |w_2|$.

Definimos os conjuntos P_0^1 e P_u^1 para a função adaptativa $\mathcal{A}_{er_1}$ como no caso 1 e os correspondentes conjuntos P_0^2 e P_u^2 para a função adaptativa $\mathcal{A}_{er_2}$. Além disso sejam $P_{w_1} = \{p_i^1 \in P : 1 \leq i \leq |w_1|\}$ e $P_{w_2} = \{p_i^2 \in P : 1 \leq i \leq |w_2|\}$ Definimos agora a função adaptativa

$$\mathcal{A}_{er}([p]) = \begin{cases} \mathcal{A}_{er_1}([p]) & \text{se } p \in P_0^1 \\ \mathcal{A}_{er_1}([p]) & \text{se } p \in P_{w_1} \setminus P_0^1 \\ \mathcal{A}_{er_2}([p]) & \text{se } p \in P_0^2 \\ \mathcal{A}_{er_2}([p]) & \text{se } p \in P_{w_2} \setminus P_0^2 \end{cases}$$

Segunda parte. Seja agora $er(C)$ uma expressão regular sobre P que define a linguagem de controle C da gramática $GR = (G, F, C)$ e consideremos uma palavra $w \in L(GR)$. Pela definição de GR existe uma derivação segundo GR, $S \Rightarrow_{GR}^{ac^*} w$; essa derivação é obtida pela aplicação de um número finito de produções de GR em modo de verificação de aparência:

(i) $w_0 = S$,

(ii) $w_{i-1} \Rightarrow^{ac} w_i$, para $0 \leq i \leq m$, w_{i-1} deriva diretamente w_i, em modo de verificação de aparência

(iii) $w_m = w$.

Assumimos que foram aplicadas as m produções $p_1, p_2 \ldots, p_m$, isto é: a produção $p_i = E \to D \in P$ é tal que se $p_i \in P \setminus F$ então $w_{i-1} = r_{i-1}Es_{i-1}$ e $w_i = r_iDs_i$ ou então se $p_i \in F$ então E não aparece em w_{i-1} e, neste caso, $w_i = w_{i-1}$.

Por ser GR uma gramática com linguagem de controle regular com verificação de aparência, a concatenação $p_1p_2 \ldots p_m$ dessas produções é uma palavra do conjunto C. Escolhemos como conjunto inicial de produções P_0 o conjunto formado pelas produções $p \in P$ as quais formam um prefixo em alguma palavra pertencente a $L(er(C))$.
Por outro lado, pela primeira parte, existe uma função adaptativa $\mathcal{A}_{er(C)}$ tal que $[p_i] \in \mathcal{A}_{er(C)}([p_{i+1}])$, $1 \leq i < m$; portanto à derivação $S \Rightarrow_{GR}^{ac} w$ da palavra w segundo a gramática GR podemos associar a seguinte derivação $S \Rightarrow_{GA}^{ac} w$ segundo a gramática $GA = (G, [P_0], [F], \mathcal{A})$

(i) $w_0 = S$,

(ii) $w_{i-1} \Rightarrow_{p_i}^{ac} w_i$, para $0 \leq i \leq m$, w_{i-1} deriva diretamente w_i, em modo de verificação de aparência

(iii) $w_m = w$.

□

Como conseqüência do lema 3.2.4 e do teorema 2.4.7, obtemos:

Teorema 3.2.5

$$\mathcal{LA}_{ac}^{\lambda} = \mathcal{R}_{ac}^{\lambda} = \mathcal{L}_0.$$

onde $\mathcal{L}_0$ é a classe das linguagens de tipo 0.

Vamos agora apresentar um resultado que permite simular as gramáticas matriciais com verificação de aparência através de gramáticas livres de contexto adaptativas.

Lema 3.2.6 *A classe $\mathcal{LA}_{ac}^{\lambda}$ das linguagens geradas pela classe das gramáticas livres de contexto adaptativas com verificação de aparência contém a classe $\mathcal{M}_{ac}^{\lambda}$ das linguagens geradas pela classe das gramáticas livres de contexto matriciais com verificação de aparência.*

Prova Consideremos uma gramática livre de contexto matricial com verificação de aparência $GM = (N, T, M, S, F)$, onde F é um subconjunto do conjunto P formado por todas as produções que aparecem nas matrizes de M.

O objetivo é obter uma função adaptativa que simule a aplicação de cada uma das matrizes da gramática GM; para simular a aplicação de uma matriz m_i basta definir uma função adaptativa que, depois de aplicar a primeira produção p_1^i de m_i, deixe disponível a produção seguinte p_2^i, e assim sucessivamente, até a penúltima produção da matriz m_i; depois de aplicar a última produção de m_i, deve poder ser aplicada qualquer matriz, portanto devem ficar disponíveis as primeiras produções de todas as matrizes: Se a matriz m_i possui $|m_i|$ produções e é definida como:

$$m_i = \left[p_1^i, \ldots, p_{|m_i|}^i\right]$$

então definimos a função adaptativa $\mathcal{A}$ como:

$$\mathcal{A}(p) = \begin{cases} -[P] \cup \{+[p_{j+1}^i]\} & \text{se } p = p_j^i, \text{ para } 1 \leq j < |m_i|, \\ -[P] \cup +[P_0] & \text{se } p = p_{|m_i|}^i \end{cases}$$

onde P_0 é o conjunto de produções iniciais para a gramática adaptativa, e é definido como:

$$P_0 = \{+[p_1^i] : 1 \leq i \leq |M|\},$$

ou seja, P_0 é formado pelas primeiras produções que aparecem nas matrizes de M.

Com essas definições, podemos associar a cada derivação de uma palavra $w \in$

$L(GM)$ segundo GM podemos associar uma derivação segundo a gramática livre de contexto adaptativa com verificação de aparência $GA = (G, [P_0], [F], \mathcal{A})$ da mesma forma como foi feito na prova do lema 3.2.4.
□

Como conseqüência do lema 3.2.6 e do teorema 2.4.7, obtemos:

Teorema 3.2.7

$$\mathcal{LA}^{\lambda}_{ac} = \mathcal{M}^{\lambda}_{ac} = \mathcal{L}_0$$

onde $\mathcal{L}_0$ é a classe das linguagens de tipo 0.

Lema 3.2.8 *A classe $\mathcal{LA}^{\lambda}_{ac}$ das linguagens geradas pela classe das gramáticas livres de contexto adaptativas com verificação de aparência contém a classe $\mathcal{P}^{\lambda}_{ac}$ das linguagens geradas pela classe das gramáticas livres de contexto programadas com verificação de aparência.*

Prova Seja $GP = (G, s, f)$ uma gramática livre de contexto programada, onde $G = (N, T, P, S)$ é uma gramática livre de contexto. Consideramos primeiro o caso das gramáticas programadas que possuem função falha igual ao conjunto vazio, isto é: $f(p) = \emptyset$, $\forall p \in P$. Neste caso, nenhuma produção é aplicada em modo de verificação de aparência, e basta definir a função adaptativa:

$$\mathcal{A}(p) = s(p), \forall p \in P$$

e definir P_0 como o conjunto formado pelas produções de P que têm o símbolo inicial S no lado esquerdo; assim a gramática livre de contexto adaptativa $GA = (G, [P_0], \emptyset, \mathcal{A})$, sem verificação de aparência, é tal que:

$$L(GA) = L(GP).$$

Consideramos agora o caso das gramáticas programadas que possuem função falha não vazia, isto é: $\exists p \in P$ tal que $f(p) \neq \emptyset$. Nesse caso, alguma produção é aplicada em modo de verificação de aparência, e é necessário identificar o conjunto formado por tais produções. Além disso, pode acontecer de alguma produção $p \in P$ ser tal que $s(p) \neq \emptyset$ e $f(p) \neq \emptyset$, o que levaria a uma indefinição da função adaptativa e, portanto, essa situação deve ser evitada. Procedendo como na prova do teorema 2.4.1 definimos uma outra gramática programada $GP_1 = (G_1, s_1, f_1)$ onde $G_1 = (N_1, T_1, P_1, S)$ é uma gramática livre de contexto tal que $L(GP_1) = L(GP)$. As funções sucesso, s_1, e falha, f_1, podem ser representadas esquematicamente como:

P_1	s_1	f_1
P	$s_1(P)$	$\emptyset$
$P_1 \setminus P$	$\emptyset$	$f_1(P_1 \setminus P)$

onde $s_1(P) \neq \emptyset$ e $f_1(P_1 \setminus P) \neq \emptyset$. Neste caso, as produções a serem aplicadas em modo de verificação de aparência são $F = P_1 \setminus P$, e basta definir a função adaptativa:

$$\mathcal{A}(p) = s(p), \forall p \in P \begin{cases} s_1(p), & \text{se } p \in P_1 \setminus F \\ f_1(p), & \text{se } p \in F \end{cases}$$

e também definir P_0 como o conjunto formado pelas produções de P que têm o símbolo inicial S no seu lado esquerdo; assim, a gramática livre de contexto adaptativa $GA = (G, [P_0], [F], \mathcal{A})$ com verificação de aparência é tal que $L(GA) = L(GP)$. □

Como conseqüência do lema 3.2.8 e do teorema 2.4.7 obtemos:

Teorema 3.2.9

$$\mathcal{LA}^{\lambda}_{ac} = \mathcal{P}^{\lambda}_{ac} = \mathcal{L}_0$$

onde $\mathcal{L}_0$ é a classe das linguagens de tipo 0.

O último resultado que apresentamos é um pequena extensão do teorema 2.4.1, que inclue a classe das linguagens geradas pelas gramáticas livres de contexto adaptativas com verificação de aparência na classe das linguagens geradas pelas gramáticas livres de contexto com linguagem de controle regular com verificação de aparência.

Teorema 3.2.10

$$\mathcal{LA}^{\lambda}_{ac} \subseteq \mathcal{L}(2,3,1) = \mathcal{R}^{\lambda}_{ac} = \mathcal{L}_0$$

onde $\mathcal{L}_0$ é a classe das linguagens de tipo 0.

Prova Seja $GA = (G, [P_0], [F], \mathcal{A})$ uma gramática livre de contexto adaptativa com verificação de aparência, onde $G = (N, T, P, S)$ é uma gramática livre de contexto e $P_0 \subset P$, $F \subset P$. Se $|P| = n$, digamos $p_1, \ldots, p_n$ então definimos a linguagem de controle C como a linguagem representada pela seguinte expressão regular:

$$C = P_0(p_1\mathcal{E}(p_1)| \ldots |p_n\mathcal{E}(p_n))^*$$

onde

$$\mathcal{E}(p_i) = \{p_j : [p_j] \in \mathcal{A}(p_i)\}.$$

Com argumentos similares aos usados nas provas dos lemas 2.3.2 e 3.2.2, pode-se provar que, a cada derivação de uma palavra segundo a gramática livre de contexto adaptativa com verificação de aparência GA, corresponde uma derivação da mesma palavra segundo a gramática livre de contexto com linguagem de controle regular e

verificação de aparência $GR = (G, C, F)$. Deste modo, a gramática livre de contexto com linguagem de controle C com verificação de aparência $GR = (G, C, F)$ é tal que

$$L(GR) = L(GA).$$

□

Este resultado encerra nosso estudo das gramáticas livres de contexto adaptativas com verificação de aparência, em relação às gramáticas livres de contexto com mecanismos de controle, estudados no capítulo 2.

3.2.1 Conversão de gramáticas variantes no tempo para a forma gramáticas livres de contexto adaptativas

Usamos agora a demonstração construtiva utilizada na prova do lema 3.2.2 para "traduzir" os exemplos de gramáticas livres de contexto periodicamente variantes no tempo para a forma de gramáticas livres de contexto adaptativas.

Exemplo 3.2.1 Gramática livre de contexto adaptativa para a linguagem $L = \{a^n b^n c^n : n \geq 0\}$**.** *Consideremos a seguinte gramática livre de contexto*

$$G = (\{S, A, B, C, D\}, \{a, b, c\}, P, S),$$

cujo o conjunto de produções é definido como segue:

$$\begin{array}{l} P = \{ \\ \quad p_0 = S \rightarrow ABC, \quad p_3 = C \rightarrow cC, \quad p_6 = C \rightarrow c, \\ \quad p_1 = A \rightarrow aA, \quad\;\; p_4 = A \rightarrow a, \quad\;\; p_7 = D \rightarrow b, \\ \quad p_2 = B \rightarrow bB, \quad\;\; p_5 = B \rightarrow D \\ \} \end{array}$$

Aplicando o método de demonstração do lema 3.2.2 à gramática periodicamente variante no tempo G_4 do exemplo 2.2.2 da secão 2.2, vamos decompor o conjunto de produções em três subconjuntos:

$$\begin{array}{rcl} P_0 & = & \{\ p_0, p_3, p_6\}, \\ P_1 & = & \{\ p_1, p_4, p_7\}, \\ P_2 & = & \{\ p_2, p_5\}. \end{array}$$

$$\mathcal{A}(p) = \begin{cases} \{-[p_0], -[p_3], -[p_6], +[p_1], +[p_4], +[p_7]\}, & \forall p \in P_0 \\ \{-[p_1], -[p_4], -[p_7], +[p_2], +[p_5]\}, & \forall p \in P_1 \\ \{-[p_2], -[p_5], +[p_0], +[p_3], +[p_6]\}, & \forall p \in P_2 \end{cases}$$

e o conjunto inicial de produções livres de contexto adaptativas é:

$$[P_0] = \{[p_0] = S \rightarrow \{\mathcal{A}()\}ABC, [p_3] = C \rightarrow \{\mathcal{A}()\}cC, [p_6] = C \rightarrow \{\mathcal{A}()\}c\}.$$

Alternativamente, podemos considerar disponíveis três funções adaptativas sem parâmetros

$$\mathcal{A} = \{\mathcal{A}_1(), \mathcal{A}_2(), \mathcal{A}_3()\}$$

especificadas por:

$\mathcal{A}_1()$
{
$-[S \to \{\mathcal{A}_1()\}ABC]$,
$-[C \to \{\mathcal{A}_1()\}cC]$,
$-[C \to \{\mathcal{A}_1()\}c]$,
$+[A \to \{\mathcal{A}_2()\}aA]$,
$+[A \to \{\mathcal{A}_2()\}a]$,
$+[D \to \{\mathcal{A}_2()\}b]$,
}
	$\mathcal{A}_2()$
	{
	$-[A \to \{\mathcal{A}_2()\}aA]$,
	$-[A \to \{\mathcal{A}_2()\}a]$,
	$-[D \to \{\mathcal{A}_2()\}b]$,
	$+[B \to \{\mathcal{A}_3()\}bB]$,
	$+[B \to \{\mathcal{A}_3()\}D]$,
	}
		$\mathcal{A}_3()$
		{
		$-[B \to \{\mathcal{A}_3()\}bB]$,
		$-[B \to \{\mathcal{A}_3()\}D]$,
		$+[S \to \{\mathcal{A}_1()\}ABC]$,
		$+[C \to \{\mathcal{A}_1()\}cC]$,
		$+[C \to \{\mathcal{A}_1()\}c]$,
		}

Derivação de $a^3b^3c^3$:

i	*Passo i da derivação*	*Produção aplicada no passo i*	$[P_i]$
0			$\{[p_0],[p_3],[p_6]\}$
1	$S \Rightarrow^{1}_{[p_0]} ABC$	$[p_0] = S \to \{\mathcal{A}_1()\}ABC$	$\{[p_1],[p_4],[p_7]\}$
2	$\Rightarrow^{2}_{[p_1]} aABC$	$[p_1] = A \to \{\mathcal{A}_2()\}aA$	$\{[p_2],[p_5]\}$
3	$\Rightarrow^{3}_{[p_2]} aAbBC$	$[p_2] = B \to \{\mathcal{A}_3()\}bB$	$\{[p_0],[p_3],[p_6]\}$
4	$\Rightarrow^{4}_{[p_3]} aAbBcC$	$[p_3] = C \to \{\mathcal{A}_1()\}cC$	$\{[p_1],[p_4],[p_7]\}$
5	$\Rightarrow^{5}_{[p_1]} aaAbBcC$	$[p_1] = A \to \{\mathcal{A}_2()\}aA$	$\{[p_2],[p_5]\}$
6	$\Rightarrow^{6}_{[p_2]} aaAbbBcC$	$[p_2] = B \to \{\mathcal{A}_3()\}bB$	$\{[p_0],[p_3],[p_6]\}$
7	$\Rightarrow^{7}_{[p_3]} aaAbbBccC$	$[p_3] = C \to \{\mathcal{A}_1()\}cC$	$\{[p_1],[p_4],[p_7]\}$
8	$\Rightarrow^{8}_{[p_4]} aaabbBccC$	$[p_4] = A \to \{\mathcal{A}_2()\}a$	$\{[p_2],[p_5]\}$
9	$\Rightarrow^{9}_{[p_5]} aaabbDccC$	$[p_5] = B \to \{\mathcal{A}_3()\}D$	$\{[p_0],[p_3],[p_6]\}$
10	$\Rightarrow^{10}_{[p_6]} aaabbDccc$	$[p_6] = C \to \{\mathcal{A}_1()\}c$	$\{[p_1],[p_4],[p_7]\}$
11	$\Rightarrow^{11}_{[p_7]} aaabbbccc$	$[p_7] = D \to \{\mathcal{A}_2()\}b$	$\{[p_2],[p_5]\}$

Exemplo 3.2.2 Gramática livre de contexto adaptativa para a linguagem $L = \{ww : w \in \{a,b\}^+\}$. *Consideremos a seguinte gramática livre de contexto* $G = (\{S,A,B,C,D\},\{a,b\},P,S)$, *cujo conjunto de produções é definido como segue:*

$$
\begin{array}{l}
P = \{ \\
\quad p_1 = S \to AB, \quad p_2 = A \to aA, \quad p_3 = B \to aB, \\
\quad p_4 = C \to C, \quad p_5 = A \to A, \quad p_6 = A \to bC, \\
\quad p_7 = B \to bD, \quad p_8 = B \to B, \quad p_9 = D \to D, \\
\quad p_{10} = A \to \lambda, \quad p_{11} = B \to \lambda, \quad p_{12} = C \to aA, \\
\quad p_{13} = D \to aB, \quad p_{14} = C \to bC, \quad p_{15} = D \to bD, \\
\quad p_{16} = C \to \lambda, \quad p_{17} = D \to \lambda. \\
\}
\end{array}
$$

Aplicando o método de demonstração do lema 3.2.2 à gramática periodicamente variante no tempo G_5 *do exemplo 2.2.3 da seção 2.2 vamos decompor o conjunto de produções em quatro subconjuntos:*

$$
\begin{array}{rcl}
P_0 & = & \{p_1, p_5, p_9\}, \\
P_1 & = & \{p_2, p_6, p_{10}, p_{12}, p_{14}, p_{16}\}, \\
P_2 & = & \{p_3, p_7, p_{11}, p_{13}, p_{15}, p_{17}\}, \\
P_3 & = & \{p_4, p_8\}.
\end{array}
$$

$$
\mathcal{A}(p) = \begin{cases}
-[P_0] \cup +[P_1], & \forall p \in P_0 \\
-[P_1] \cup +[P_2], & \forall p \in P_1 \\
-[P_2] \cup +[P_3], & \forall p \in P_2 \\
-[P_3] \cup +[P_0], & \forall p \in P_3
\end{cases}
$$

com $-[P_i] = \{-[p] : \forall p \in P_i\}$ *e* $+[P_i] = \{+[p] : \forall p \in P_i\}$, *como na prova do lema 3.2.2. O conjunto inicial de produções livres de contexto adaptativas é:*

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}()\}AB, [p_5] = A \to \{\mathcal{A}()\}A, [p_9] = D \to \{\mathcal{A}()\}D\}.$$

Escrita desta forma, a função adaptativa $\mathcal{A}$ *está pronta para ser decomposta em quatro funções adaptativas segundo a observação 3.2.1,* $\mathcal{A} = \{\mathcal{A}_1(), \mathcal{A}_2(), \mathcal{A}_3(), \mathcal{A}_4()\}$, *e neste caso, o conjunto inicial de produções adaptativas é:*

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}_1()\}AB, [p_5] = A \to \{\mathcal{A}_1()\}A, [p_9] = D \to \{\mathcal{A}_1()\}D\}$$

e as funções adaptativas são especificadas por:

$\mathcal{A}_1()$
{
$-[S \to \{\mathcal{A}_1()\}AB]$,
$-[A \to \{\mathcal{A}_1()\}A]$,
$-[D \to \{\mathcal{A}_1()\}D]$,
$+[A \to \{\mathcal{A}_2()\}aA]$,
$+[A \to \{\mathcal{A}_2()\}bC]$,
$+[A \to \{\mathcal{A}_2()\}\lambda]$,
$+[C \to \{\mathcal{A}_2()\}aA]$,
$+[C \to \{\mathcal{A}_2()\}bC]$,
$+[C \to \{\mathcal{A}_2()\}\lambda]$
}
	$\mathcal{A}_2()$
	{
	$-[A \to \{\mathcal{A}_2()\}aA]$,
	$-[A \to \{\mathcal{A}_2()\}bC]$,
	$-[A \to \{\mathcal{A}_2()\}\lambda]$,
	$-[C \to \{\mathcal{A}_2()\}aA]$,
	$-[C \to \{\mathcal{A}_2()\}bC]$,
	$-[C \to \{\mathcal{A}_2()\}\lambda]$,
	$+[B \to \{\mathcal{A}_3()\}aB]$,
	$+[B \to \{\mathcal{A}_3()\}bD]$,
	$+[B \to \{\mathcal{A}_3()\}\lambda]$,
	$+[D \to \{\mathcal{A}_3()\}aB]$,
	$+[D \to \{\mathcal{A}_3()\}bD]$,
	$+[D \to \{\mathcal{A}_3()\}\lambda]$
	}

$\mathcal{A}_3()$
{
$-[B \to \{\mathcal{A}_3()\}aB]$,
$-[B \to \{\mathcal{A}_3()\}bD]$,
$-[B \to \{\mathcal{A}_3()\}\lambda]$,
$-[D \to \{\mathcal{A}_3()\}aB]$,
$-[D \to \{\mathcal{A}_3()\}bD]$,
$-[D \to \{\mathcal{A}_3()\}\lambda]$,
$+[C \to \{\mathcal{A}_4()\}C]$,
$+[B \to \{\mathcal{A}_4()\}B]$
}
	$\mathcal{A}_4()$
	{
	$-[C \to \{\mathcal{A}_4()\}C]$,
	$-[B \to \{\mathcal{A}_4()\}B]$,
	$+[S \to \{\mathcal{A}_1()\}AB]$,
	$+[A \to \{\mathcal{A}_1()\}A]$,
	$+[D \to \{\mathcal{A}_1()\}D]$
	}

Nestas condições, a gramática livre de contexto adaptativa $(G, [P_0], \emptyset, \mathcal{A})$ *gera a linguagem dependente de contexto:* $L = \{ww : w \in \{a, b\}^+\}$.

Derivação de *bbaabbbaab:*

i	*Passo i da derivação*	*Produção aplicada*	$[P_i]$
0			$\{[p_1], [p_5], [p_9]\}$
1	$S \Rightarrow^{1}_{[p_1]} AB$	$[p_1] = S \rightarrow \{\mathcal{A}_1()\}AB$	$\{[p_2], [p_6], [p_{10}], [p_{12}], [p_{14}], [p_{16}]\}$
2	$\Rightarrow^{2}_{[p_6]} bCB$	$[p_6] = A \rightarrow \{\mathcal{A}_2()\}bC$	$\{[p_3], [p_7], [p_{11}], [p_{13}], [p_{15}], [p_{17}]\}$
3	$\Rightarrow^{3}_{[p_7]} bCbD$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}bD$	$\{[p_4], [p_8]\}$
4	$\Rightarrow^{4}_{[p_4]} bCbD$	$[p_4] = C \rightarrow \{\mathcal{A}_4()\}C$	$\{[p_1], [p_5], [p_9]\}$
5	$\Rightarrow^{5}_{[p_9]} bCbD$	$[p_9] = D \rightarrow \{\mathcal{A}_1()\}D$	$\{[p_2], [p_6], [p_{10}], [p_{12}], [p_{14}], [p_{16}]\}$
6	$\Rightarrow^{6}_{[p_{14}]} bbCbD$	$[p_{14}] = C \rightarrow \{\mathcal{A}_2()\}bC$	$\{[p_3], [p_7], [p_{11}], [p_{13}], [p_{15}], [p_{17}]\}$
7	$\Rightarrow^{7}_{[p_{15}]} bbCbbD$	$[p_{15}] = D \rightarrow \{\mathcal{A}_3()\}bD$	$\{[p_4], [p_8]\}$
8	$\Rightarrow^{8}_{[p_4]} bbCbbD$	$[p_4] = C \rightarrow \{\mathcal{A}_4()\}C$	$\{[p_1], [p_5], [p_9]\}$
9	$\Rightarrow^{9}_{[p_9]} bbCbbD$	$[p_9] = D \rightarrow \{\mathcal{A}_1()\}D$	$\{[p_2], [p_6], [p_{10}], [p_{12}], [p_{14}], [p_{16}]\}$
10	$\Rightarrow^{10}_{[p_{12}]} bbaAbD$	$[p_{12}] = C \rightarrow \{\mathcal{A}_2()\}aA$	$\{[p_3], [p_7], [p_{11}], [p_{13}], [p_{15}], [p_{17}]\}$
11	$\Rightarrow^{11}_{[p_{13}]} bbaAbbaB$	$[p_{13}] = D \rightarrow \{\mathcal{A}_3()\}aB$	$\{[p_4], [p_8]\}$
12	$\Rightarrow^{12}_{[p_8]} bbaAbbaB$	$[p_8] = B \rightarrow \{\mathcal{A}_4()\}B$	$\{[p_1], [p_5], [p_9]\}$
13	$\Rightarrow^{13}_{[p_5]} bbaAbbaB$	$[p_5] = A \rightarrow \{\mathcal{A}_1()\}A$	$\{[p_2], [p_6], [p_{10}], [p_{12}], [p_{14}], [p_{16}]\}$
14	$\Rightarrow^{14}_{[p_2]} bbaaAbbaaB$	$[p_2] = A \rightarrow \{\mathcal{A}_2()\}aA$	$\{[p_3], [p_7], [p_{11}], [p_{13}], [p_{15}], [p_{17}]\}$
15	$\Rightarrow^{15}_{[p_3]} bbaaAbbaaB$	$[p_3] = B \rightarrow \{\mathcal{A}_3()\}aB$	$\{[p_4], [p_8]\}$
16	$\Rightarrow^{16}_{[p_8]} bbaaAbbaaB$	$[p_8] = B \rightarrow \{\mathcal{A}_4()\}B$	$\{[p_1], [p_5], [p_9]\}$
17	$\Rightarrow^{17}_{[p_5]} bbaaAbbaaB$	$[p_5] = A \rightarrow \{\mathcal{A}_1()\}A$	$\{[p_2], [p_6], [p_{10}], [p_{12}], [p_{14}], [p_{16}]\}$
18	$\Rightarrow^{18}_{[p_6]} bbaabCbbaaB$	$[p_6] = A \rightarrow \{\mathcal{A}_2()\}bC$	$\{[p_3], [p_7], [p_{11}], [p_{13}], [p_{15}], [p_{17}]\}$
19	$\Rightarrow^{19}_{[p_7]} bbaabCbbaabD$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}bD$	$\{[p_4], [p_8]\}$
20	$\Rightarrow^{20}_{[p_4]} bbaabCbbaabD$	$[p_4] = C \rightarrow \{\mathcal{A}_4()\}C$	$\{[p_1], [p_5], [p_9]\}$
21	$\Rightarrow^{21}_{[p_9]} bbaabCbbaabD$	$[p_9] = D \rightarrow \{\mathcal{A}_1()\}D$	$\{[p_2], [p_6], [p_{10}], [p_{12}], [p_{14}], [p_{16}]\}$
22	$\Rightarrow^{22}_{[p_{16}]} bbaabbbaab$	$[p_{16}] = C \rightarrow \{\mathcal{A}_2()\}\lambda$	$\{[p_3], [p_7], [p_{11}], [p_{13}], [p_{15}], [p_{17}]\}$
23	$\Rightarrow^{23}_{[p_{17}]} bbaabbbaab$	$[p_{17}] = D \rightarrow \{\mathcal{A}_3()\}\lambda$	$\{[p_4], [p_8]\}$

3.2.2 Conversão de gramáticas com linguagem de controle regular para gramáticas livres de contexto adaptativas

Aplicamos agora a demonstração construtiva utilizada na prova do lema 3.2.4 para "traduzir" os exemplos de gramáticas livres de contexto com conjunto de controle regular com verificação de aparência para gramáticas livres de contexto adaptativas com verificação de aparência.

Exemplo 3.2.3 Gramática livre de contexto adaptativa para a linguagem $L = \{a^n b^n c^n : n \geq 1\}$**.** *Consideremos a seguinte gramática livre de contexto:*

$$G = (\{S, A, C\}, \{a, b, c\}, \{p_1, p_2, p_3, p_4, p_5\}, S)$$

com

$$p_1 = S \to AC, p_2 = A \to aAb, p_3 = C \to cC, p_4 = A \to ab, p_5 = C \to c.$$

Aplicando o método de demonstração do lema 3.2.4 à gramática livre de contexto com linguagem de controle regular G_{10} do exemplo 2.4.1 da seção 2.4, obtemos a seguinte função adaptativa:

$$\mathcal{A}(p) = \begin{cases} \{p_2, p_4\}, & p = p_1, \\ \{p_3\}, & p = p_2, \\ \{p_2, p_4\}, & p = p_3, \\ \{p_5\}, & p = p_4, \\ \emptyset, & p = p_5. \end{cases}$$

O conjunto inicial de produções livres de contexto adaptativas é:

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}()\}AC\}.$$

Escrita desta forma, a função adaptativa $\mathcal{A}$ está pronta para ser decomposta em quatro funções adaptativas segundo a observação 3.2.1, $\mathcal{A} = \{\mathcal{A}_1(), \mathcal{A}_2(), \mathcal{A}_3(), \mathcal{A}_4()\}$, e neste caso, o conjunto inicial de produções adaptativas é:

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}_1()\}AC, \}$$

e as funções adaptativas são especificadas por:

$\mathcal{A}_1()$
{
$-\,[S \to \{\mathcal{A}_1()\}AC]\,,$
$-\,[C \to \{\mathcal{A}_1()\}cC]\,,$
$-\,[C \to \{\mathcal{A}_5()\}c]\,,$
$+\,[A \to \{\mathcal{A}_2()\}aAb]\,,$
$+\,[A \to \{\mathcal{A}_3()\}ab]\,,$
}
	$\mathcal{A}_2()$
	{
	$-\,[S \to \{\mathcal{A}_1()\}AC]\,,$
	$-\,[A \to \{\mathcal{A}_2()\}aAb]\,,$
	$-\,[A \to \{\mathcal{A}_3()\}ab]\,,$
	$-\,[C \to \{\mathcal{A}_5()\}c]\,,$
	$+\,[C \to \{\mathcal{A}_1()\}cC]\,,$
	}
		$\mathcal{A}_3()$
		{
		$-\,[S \to \{\mathcal{A}_1()\}AC]\,,$
		$-\,[A \to \{\mathcal{A}_2()\}aAb]\,,$
		$-\,[C \to \{\mathcal{A}_1()\}cC]\,,$
		$-\,[A \to \{\mathcal{A}_3()\}ab]\,,$
		$+\,[C \to \{\mathcal{A}_5()\}c]\,.$
		}
			$\mathcal{A}_4()$
			{
			$-\,[S \to \{\mathcal{A}_1()\}AC]\,,$
			$-\,[A \to \{\mathcal{A}_2()\}aAb]\,,$
			$-\,[C \to \{\mathcal{A}_1()\}cC]\,,$
			$-\,[A \to \{\mathcal{A}_3()\}ab]\,,$
			$-\,[C \to \{\mathcal{A}_5()\}c]\,.$
			}

Nessas condições, a gramática livre de contexto adaptativa $GA = (G, [P_0], \emptyset, \mathcal{A})$*, sem verificação de aparência, gera a linguagem* $L = \{a^n b^n c^n : n \geq 1\}$.

Derivação de $a^3b^3c^3$*:*
Com efeito:

i	*Passo* i *da derivação*	*Produção aplicada no passo* i	$[P_i]$
0			$\{[p_1]\}$
1	$S \Rightarrow^{1}_{[p_1]} AC$	$[p_1] = S \to \{\mathcal{A}_1()\}AC$	$\{[p_2], [p_4]\}$
2	$\Rightarrow^{2}_{[p_2]} aAbC$	$[p_2] = A \to \{\mathcal{A}_2()\}aAb$	$\{[p_3]\}$
3	$\Rightarrow^{3}_{[p_3]} aAbcC$	$[p_3] = C \to \{\mathcal{A}_1()\}cC$	$\{[p_2], [p_4]\}$
4	$\Rightarrow^{4}_{[p_2]} aaAbbcC$	$[p_2] = A \to \{\mathcal{A}_2()\}aAb$	$\{[p_3]\}$
5	$\Rightarrow^{5}_{[p_3]} aaAbbccC$	$[p_3] = C \to \{\mathcal{A}_1()\}cC$	$\{[p_2], [p_4]\}$
6	$\Rightarrow^{6}_{[p_4[} aaabbbccC$	$[p_4] = A \to \{\mathcal{A}_2()\}ab$	$\{[p_5]\}$
7	$\Rightarrow^{7}_{[p_5[} aaabbbccc$	$[p_5] = C \to \{\mathcal{A}_2()\}c$	$\emptyset$

Exemplo 3.2.4 Gramática livre de contexto adaptativa para a linguagem $L = \{ww : w \in \{a,b\}^+\}$. *Consideremos a gramática com linguagem de controle regular sem verificação de aparência:*

$$G_{11} = (\{S,A,B\},\{a,b,c\},P,S,\{p_1(p_2p_6|p_3p_7)^n(p_5p_9|p_4p_8) : n \geq 0\},\emptyset)$$

com

$$P = \{p_1 = S \to AB, p_2 = A \to aA, p_3 = A \to bA, p_4 = A \to a, p_5 = A \to b,$$
$$p_6 = B \to aB, p_7 = B \to bB, p_8 = B \to a, p_9 = B \to b.\}$$

Aplicando o método de demonstração do lema 3.2.4 à gramática livre de contexto com linguagem de controle regular G_{11} do exemplo 2.4.2 da seção 2.4 obtemos a função adaptativa:

$$\mathcal{A}(p) = \begin{cases} \{p_2,p_3,p_4,p_5\}, & p = p_1, \\ \{p_6\}, & p = p_2, \\ \{p_7\}, & p = p_3, \\ \{p_8\}, & p = p_4, \\ \{p_9\}, & p = p_5. \\ \{p_2,p_3,p_4,p_5\}, & p = p_6, \\ \{p_2,p_3,p_4,p_5\}, & p = p_7, \\ \emptyset, & p = p_8, \\ \emptyset, & p = p_9, \end{cases}$$

e o conjunto inicial de produções livres de contexto adaptativas é:

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}()\}AB\}.$$

Escrita desta forma, a função adaptativa $\mathcal{A}$ está pronta para ser decomposta em seis funções adaptativas, segundo a observação 3.2.1, $\mathcal{A} = \{\mathcal{A}_1(),\mathcal{A}_2(),\mathcal{A}_3(),\mathcal{A}_4(),\mathcal{A}_5(),\mathcal{A}_6()\}$, e, neste caso, o conjunto inicial de produções adaptativas é:

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}_1()\}AC,\}$$

e as funções adaptativas são especificadas por:

$\mathcal{A}_1()$	$\mathcal{A}_2()$	$\mathcal{A}_3()$
{	{	{
$-[S \to \{\mathcal{A}_1()\}AB]$,	$-[S \to \{\mathcal{A}_1()\}AB]$,	$-[S \to \{\mathcal{A}_1()\}AB]$,
$-[B \to \{\mathcal{A}_1()\}aB]$,	$-[A \to \{\mathcal{A}_2()\}aA]$,	$-[A \to \{\mathcal{A}_2()\}aA]$,
$-[B \to \{\mathcal{A}_1()\}bB]$,	$-[A \to \{\mathcal{A}_3()\}bA]$,	$-[A \to \{\mathcal{A}_3()\}bA]$,
$-[B \to \{\mathcal{A}_6()\}a]$,	$-[A \to \{\mathcal{A}_4()\}a]$,	$-[A \to \{\mathcal{A}_4()\}a]$,
$-[B \to \{\mathcal{A}_6()\}b]$,	$-[A \to \{\mathcal{A}_5()\}b]$,	$-[A \to \{\mathcal{A}_5()\}b]$,
$+[A \to \{\mathcal{A}_2()\}aA]$,	$-[B \to \{\mathcal{A}_1()\}bB]$,	$-[B \to \{\mathcal{A}_1()\}aB]$,
$+[A \to \{\mathcal{A}_3()\}bA]$,	$-[B \to \{\mathcal{A}_6()\}a]$,	$-[B \to \{\mathcal{A}_6()\}a]$,
$+[A \to \{\mathcal{A}_4()\}a]$,	$-[B \to \{\mathcal{A}_6()\}b]$,	$-[B \to \{\mathcal{A}_6()\}b]$,
$+[A \to \{\mathcal{A}_5()\}b]$	$+[B \to \{\mathcal{A}_1()\}aB]$	$+[B \to \{\mathcal{A}_1()\}bB]$
}	}	}

$\mathcal{A}_4()$
{
$-[S \to \{\mathcal{A}_1()\}AB]$,
$-[A \to \{\mathcal{A}_2()\}aB]$,
$-[A \to \{\mathcal{A}_3()\}bB]$,
$-[A \to \{\mathcal{A}_4()\}a]$,
$-[A \to \{\mathcal{A}_5()\}b]$,
$-[B \to \{\mathcal{A}_1()\}aB]$,
$-[B \to \{\mathcal{A}_1()\}bB]$,
$-[B \to \{\mathcal{A}_6()\}b]$,
$+[B \to \{\mathcal{A}_6()\}a]$
}
	$\mathcal{A}_5()$
	{
	$-[S \to \{\mathcal{A}_1()\}AB]$,
	$-[A \to \{\mathcal{A}_2()\}aA]$,
	$-[A \to \{\mathcal{A}_3()\}bA]$,
	$-[A \to \{\mathcal{A}_4()\}a]$,
	$-[A \to \{\mathcal{A}_5()\}b]$,
	$-[B \to \{\mathcal{A}_1()\}aB]$,
	$-[B \to \{\mathcal{A}_1()\}bB]$,
	$-[B \to \{\mathcal{A}_6()\}a]$,
	$+[B \to \{\mathcal{A}_6()\}b]$
	}
		$\mathcal{A}_6()$
		{
		$-[S \to \{\mathcal{A}_1()\}AB]$,
		$-[A \to \{\mathcal{A}_2()\}aA]$,
		$-[A \to \{\mathcal{A}_3()\}bA]$,
		$-[A \to \{\mathcal{A}_4()\}a]$,
		$-[A \to \{\mathcal{A}_5()\}b]$,
		$-[B \to \{\mathcal{A}_1()\}aB]$,
		$-[B \to \{\mathcal{A}_1()\}bB]$,
		$-[B \to \{\mathcal{A}_6()\}a]$,
		$-[B \to \{\mathcal{A}_6()\}b]$
		}

Nestas condições a gramática livre de contexto adaptativa $GA = (G, [P_0], \emptyset, \mathcal{A})$ *sem verificação de aparência gera a linguagem* $L = \{ww : \{a, b\}^*\}$.

Derivação de bbaabbbaab*:*
Com efeito:

i	*Passo* i *da derivação*	*Produção aplicada no passo* i	$[P_i]$
0			$\{[p_1]\}$
1	$S \Rightarrow^1_{[p_1]} AB$	$[p_1] = S \to \{\mathcal{A}_1()\}AB$	$\{[p_2], [p_3], [p_4], [p_5]\}$
2	$\Rightarrow^2_{[p_3]} bAB$	$[p_3] = A \to \{\mathcal{A}_1()\}bA$	$\{[p_7]\}$
3	$\Rightarrow^3_{[p_7]} bAbB$	$[p_7] = B \to \{\mathcal{A}_1()\}bB$	$\{[p_2], [p_3], [p_4], [p_5]\}$
4	$\Rightarrow^4_{[p_3]} bbAbB$	$[p_3] = A \to \{\mathcal{A}_1()\}bA$	$\{[p_7]\}$
5	$\Rightarrow^5_{[p_7]} bbAbbB$	$[p_7] = B \to \{\mathcal{A}_1()\}bB$	$\{[p_2], [p_3], [p_4], [p_5]\}$
6	$\Rightarrow^6_{[p_2]} bbaAbbB$	$[p_2] = A \to \{\mathcal{A}_4()\}aA$	$\{[p_6]\}$
7	$\Rightarrow^7_{[p_6]} bbaAbbaB$	$[p_6] = B \to \{\mathcal{A}_1()\}aB$	$\{[p_2], [p_3], [p_4], [p_5]\}$
8	$\Rightarrow^8_{[p_2]} bbaaAbbaB$	$[p_2] = A \to \{\mathcal{A}_4()\}aA$	$\{[p_6]\}$
9	$\Rightarrow^9_{[p_6]} bbaaAbbaaB$	$[p_6] = B \to \{\mathcal{A}_1()\}aB$	$\{[p_2], [p_3], [p_4], [p_5]\}$
10	$\Rightarrow^{10}_{[p_5]} bbaabbbaaB$	$[p_5] = A \to \{\mathcal{A}_4()\}b$	$\{[p_6]\}$
11	$\Rightarrow^{11}_{[p_9]} bbaabbbaab.$	$[p_5] = A \to \{\mathcal{A}_4()\}b$	$\emptyset$

Exemplo 3.2.5 Gramática livre de contexto adaptativa com verificação de aparência para a linguagem $L = \{a^{2^n} : n \geq 0\}$. *Consideremos a gramática livre de contexto:*

$$G = (\{S, A, X\}, \{a\}, \{p_1, p_2, p_3, p_4, p_5\}, S,)$$

com

$$p_1 = S \to AA, p_2 = S \to X, p_3 = A \to S, p_4 = A \to X, p_5 = S \to a.$$

Definimos o conjunto de produções a serem aplicadas em modo de verificação de aparência como $F = \{p_2, p_4\}$. *Aplicando o método de demonstração do lema 3.2.4 à gramática livre de contexto com linguagem de controle regular* G_{12} *do exemplo 2.4.3 da seção 2.4 obtemos a função adaptativa:*

$$\mathcal{A}(p) = \begin{cases} \{p_1, p_2, p_5\}, & p = p_1, \\ \{p_3, p_4\}, & p = p_2, \\ \{p_3, p_4\}, & p = p_3, \\ \{p_1, p_2, p_5\}, & p = p_4, \\ \{p_5\}, & p = p_5. \end{cases}$$

e o conjunto inicial de produções livres de contexto adaptativas é:

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}()\}AA, [p_2] = S \to \{\mathcal{A}()\}X, [p_5] = S \to \{\mathcal{A}()\}a\}.$$

Escrita desta forma, a função adaptativa $\mathcal{A}$ *está pronta para ser decomposta em três funções adaptativas, segundo a observação 3.2.1,* $\mathcal{A} = \{\mathcal{A}_1(), \mathcal{A}_2(), \mathcal{A}_3()\}$. *Neste caso, o conjunto inicial de produções adaptativas é:*

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}_1()\}AA, [p_2] = S \to \{\mathcal{A}_2()\}X, [p_5] = S \to \{\mathcal{A}_3()\}a\}$$

e as funções adaptativas são especificadas por:

$\mathcal{A}_1()$
{
$-\,[S \to \{\mathcal{A}_1()\}AA]\,,$
$-\,[A \to \{\mathcal{A}_1()\}X]\,,$
$+\,[S \to \{\mathcal{A}_1()\}AA]\,,$
$+\,[S \to \{\mathcal{A}_2()\}X]\,,$
$+\,[S \to \{\mathcal{A}_3()\}a]\,,$
}
	$\mathcal{A}_2()$
	{
	$-\,[S \to \{\mathcal{A}_2()\}AA]\,,$
	$-\,[S \to \{\mathcal{A}_2()\}X]\,,$
	$-\,[S \to \{\mathcal{A}_3()\}a]\,,$
	$+\,[A \to \{\mathcal{A}_2()\}S]\,,$
	$+\,[A \to \{\mathcal{A}_1()\}X]\,,$
	}
		$\mathcal{A}_3()$
		{
		$-\,[S \to \{\mathcal{A}_2()\}AA]\,,$
		$-\,[S \to \{\mathcal{A}_2()\}X]\,,$
		$-\,[A \to \{\mathcal{A}_2()\}S]\,,$
		$-\,[A \to \{\mathcal{A}_2()\}X]\,,$
		$+\,[S \to \{\mathcal{A}_3()\}a]\,.$
		}

Nestas condições, a gramática livre de contexto adaptativa $GA = (G, [P_0], [F], \mathcal{A})$ *gera a linguagem* $L = \{a^{2^n} : n \geq 0\}$.

Derivação de a^{2^3}

i	*Passo i da derivação*	*Produção aplicada no passo i*	$[P_i]$
0			$\{[p_1],[p_2],[p_5]\}$
1	$S \Rightarrow^{1}_{[p_1]} AA$	$[p_1] = S \rightarrow \{\mathcal{A}_1()\}AA$	”
2	$\Rightarrow^{ac^2}_{[p_2]} AA$	$[p_2] = S \rightarrow \{\mathcal{A}_2()\}X$	$\{[p_3],[p_4]\}$
3	$\Rightarrow^{3}_{[p_3]} SA$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
4	$\Rightarrow^{4}_{[p_3]} SS$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
5	$\Rightarrow^{ac^5}_{[p_4]} SS$	$[p_4] = A \rightarrow \{\mathcal{A}_1()\}X$	$\{[p_1],[p_2],[p_5]\}$
6	$\Rightarrow^{6}_{[p_1]} AAS$	$[p_1] = S \rightarrow \{\mathcal{A}_1()\}AA$	”
7	$\Rightarrow^{7}_{[p_1]} AAAA$	$[p_1] = S \rightarrow \{\mathcal{A}_1()\}AA$	”
8	$\Rightarrow^{ac^8}_{[p_2]} AAAA$	$[p_2] = S \rightarrow \{\mathcal{A}_2()\}X$	$\{[p_3],[p_4]\}$
9	$\Rightarrow^{9}_{[p_3]} SAAA$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
10	$\Rightarrow^{10}_{[p_3]} SSAA$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
11	$\Rightarrow^{11}_{[p_3]} SSSA$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
12	$\Rightarrow^{12}_{[p_3]} SSSS$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
13	$\Rightarrow^{ac^{13}}_{[p_4]} SSSS$	$[p_4] = A \rightarrow \{\mathcal{A}_2()\}X$	$\{[p_1],[p_2],[p_5]\}$
14	$\Rightarrow^{14}_{[p_1]} SSSAA$	$[p_1] = S \rightarrow \{\mathcal{A}_2()\}AA$	”
15	$\Rightarrow^{15}_{[p_1]} SAASAA$	$[p_1] = S \rightarrow \{\mathcal{A}_2()\}AA$	”
16	$\Rightarrow^{16}_{[p_1]} AAAASAA$	$[p_1] = S \rightarrow \{\mathcal{A}_2()\}AA$	”
17	$\Rightarrow^{17}_{[p_1]} AAAAAAAA$	$[p_1] = S \rightarrow \{\mathcal{A}_2()\}AA$	”
18	$\Rightarrow^{ac^{18}}_{[p_2]} AAAAAAAA$	$[p_2] = S \rightarrow \{\mathcal{A}_2()\}X$	$\{[p_3],[p_4]\}$
19	$\Rightarrow^{19}_{[p_3]} SAAAAAAA$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
20	$\Rightarrow^{20}_{[p_3]} SAAAAAAS$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
21	$\Rightarrow^{21}_{[p_3]} SAAAAASS$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
22	$\Rightarrow^{22}_{[p_3]} SSAAAASS$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
23	$\Rightarrow^{23}_{[p_3]} SSSAAASS$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
24	$\Rightarrow^{24}_{[p_3]} SSSSAASS$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
25	$\Rightarrow^{25}_{[p_3]} SSSSASSS$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
26	$\Rightarrow^{26}_{[p_3]} SSSSSSSS$	$[p_3] = A \rightarrow \{\mathcal{A}_2()\}S$	”
27	$\Rightarrow^{ac^{27}}_{[p_4]} SSSSSSSS$	$[p_4] = A \rightarrow \{\mathcal{A}_2()\}X$	$\{[p_1],[p_2],[p_5]\}$
28	$\Rightarrow^{28}_{[p_5]} aSSSSSSS$	$[p_5] = S \rightarrow \{\mathcal{A}_3()\}a$	$\{[p_5]\}$
29	$\Rightarrow^{29}_{[p_5]} aSSSSSSa$	$[p_5] = S \rightarrow \{\mathcal{A}_3()\}a$	”
30	$\Rightarrow^{30}_{[p_5]} aaSSSSSa$	$[p_5] = S \rightarrow \{\mathcal{A}_3()\}a$	”
31	$\Rightarrow^{31}_{[p_5]} aaSSSSaa$	$[p_5] = S \rightarrow \{\mathcal{A}_3()\}a$	”
32	$\Rightarrow^{32}_{[p_5]} aaSSaSaa$	$[p_5] = S \rightarrow \{\mathcal{A}_3()\}a$	”
33	$\Rightarrow^{33}_{[p_5]} aaSSaaaa$	$[p_5] = S \rightarrow \{\mathcal{A}_3()\}a$	”
34	$\Rightarrow^{34}_{[p_5]} aaaSaaaa$	$[p_5] = S \rightarrow \{\mathcal{A}_3()\}a$	”
35	$\Rightarrow^{35}_{[p_5]} aaaaaaaa$	$[p_5] = S \rightarrow \{\mathcal{A}_3()\}a$	”

3.2.3 Conversão de gramáticas matriciais para gramáticas livres de contexto adaptativas

Aplicamos agora a demonstração construtiva utilizada na prova do lema 3.2.6 para "traduzir" os exemplos de gramáticas livres de contexto matriciais com verificação de aparência para gramáticas livres de contexto adaptativas com verificação de aparência.

Exemplo 3.2.6 Gramática livre de contexto adaptativa para a linguagem $L = \{a^n b^n c^n : n \geq 1\}$. *Consideremos a gramática matricial:*

$$G_1 = (\{S, C\}, \{a, b, c\}, M, S)$$

com

$$M = \{m_1, m_2, m_3\},$$

sendo

$$m_1 = [S \to SC], \quad m_2 = [S \to aSb, C \to cC], \quad m_3 = [S \to ab, C \to c].$$

Definimos o conjunto P como o conjunto de todas as produções que compõem as matrizes de M

$$P = \{p_1 = S \to SC, p_2 = S \to aSb, p_3 = C \to cC, p_4 = S \to ab, p_5 = C \to c\}$$

Aplicando o método de demonstração do lema 3.2.6 à gramática livre de contexto matricial G_1 do exemplo 2.1.1 da seção 2.1, obtemos a função adaptativa:

$$\mathcal{A}(p) = \begin{cases} \{p_1, p_2, p_4\}, & p = p_1, \\ \{p_3\}, & p = p_2, \\ \{p_1, p_2, p_4\}, & p = p_3, \\ \{p_5\}, & p = p_4, \\ \{p_1, p_2, p_4\}, & p = p_5. \end{cases}$$

O conjunto inicial de produções livres de contexto adaptativas é:

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}()\}SC, [p_2] = S \to \{\mathcal{A}()\}aSb, [p_4] = S \to \{\mathcal{A}()\}ab\}$$

Escrita desta forma, a função adaptativa $\mathcal{A}$ está pronta para ser decomposta em três funções adaptativas segundo a observação 3.2.1, $\mathcal{A} = \{\mathcal{A}_1(), \mathcal{A}_2(), \mathcal{A}_3()\}$. Nesse caso, o conjunto inicial de produções adaptativas é:

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}_1()\}SC, [p_2] = S \to \{\mathcal{A}_2()\}aSb, [p_4] = S \to \{\mathcal{A}_3()\}ab\}$$

e as funções adaptativas são especificadas por:

$\mathcal{A}_1()$
{
$-[C \to \{\mathcal{A}_1()\}cC]$,
$-[C \to \{\mathcal{A}_1()\}c]$,
$+[S \to \{\mathcal{A}_1()\}SC]$,
$+[S \to \{\mathcal{A}_2()\}aSb]$,
$+[S \to \{\mathcal{A}_3()\}ab]$,
}
	$\mathcal{A}_2()$
	{
	$-[S \to \{\mathcal{A}_1()\}SC]$,
	$-[S \to \{\mathcal{A}_2()\}aSb]$,
	$-[C \to \{\mathcal{A}_3()\}ab]$,
	$-[S \to \{\mathcal{A}_1()\}c]$,
	$+[C \to \{\mathcal{A}_1()\}cC]$,
	}
		$\mathcal{A}_3()$
		{
		$-[S \to \{\mathcal{A}_1()\}SC]$,
		$-[S \to \{\mathcal{A}_2()\}aSb]$,
		$-[C \to \{\mathcal{A}_1()\}cC]$,
		$-[S \to \{\mathcal{A}_3()\}ab]$,
		$+[C \to \{\mathcal{A}_1()\}c]$.
		}

Observe-se que em $\mathcal{A}_2$ foi retirada e adicionada a produção p_3. Como as remoções são sempre efetuadas antes das inserções, essas duas operações tem o mesmo efeito de, simplesmente, adicionar a produção p_3. Isto sugere uma forma mais econômica de escrever várias funções adaptativas que evita essa redundância: listar, em cada função adaptativa, todas as produções da gramática subjacente incluindo em ações elementares de inserção todas aquelas que devem fazer parte do conjunto de produções aplicáveis e incluindo todas as outras em ações elementares de remoção.

Nessas condições a gramática livre de contexto adaptativa $GA = (G, [P_0], \emptyset, \mathcal{A})$ sem verificação de aparência gera a linguagem $L = \{a^n b^n c^n : n \geq 1\}$.

Derivação de $a^3b^3c^3$:
Com efeito:

i	*Passo i da derivação*	*Produção aplicada no passo i*	$[P_i]$
0			$\{[p_1], [p_2], [p_4]\}$
1	$S \Rightarrow^1_{[p_1]} AC$	$[p_1] = S \to \{\mathcal{A}_1()\}SC$	$\{[p_1], [p_2], [p_4]\}$
2	$\Rightarrow^2_{[p_2]} aAbC$	$[p_2] = A \to \{\mathcal{A}_2()\}aSb$	$\{[p_3]\}$
3	$\Rightarrow^3_{[p_3]} aAbcC$	$[p_3] = C \to \{\mathcal{A}_3()\}cC$	$\{[p_1], [p_2], [p_4]\}$
4	$\Rightarrow^4_{[p_2]} aaAbbcC$	$[p_2] = A \to \{\mathcal{A}_2()\}aSb$	$\{[p_3]\}$
5	$\Rightarrow^5_{[p_3]} aaAbbccC$	$[p_3] = C \to \{\mathcal{A}_3()\}cC$	$\{[p_1], [p_2], [p_4]\}$
6	$\Rightarrow^6_{[p_4]} aaabbbccC$	$[p_4] = A \to \{\mathcal{A}_4()\}ab$	$\{[p_5]\}$
7	$\Rightarrow^7_{[p_5]} aaabbbccc$	$[p_5] = C \to \{\mathcal{A}_5()\}c$	$\{[p_1], [p_2], [p_4]\}$

Exemplo 3.2.7 Gramática livre de contexto adaptativa para a linguagem $L = \{ww : w \in \{a,b\}^+\}$. *Consideremos a gramática matricial:*

$$G_2 = (\{S, A, B\}, \{a, b\}, M, S)$$

com

$$M = \{m_1, m_2, m_3, m_4, m_5\},$$

sendo

$$\begin{array}{lll} m_1 = [S \to AB], & m_2 = [A \to aA, B \to aB], & m_3 = [A \to bA, B \to bB], \\ m_4 = [A \to a, B \to a], & m_5 = [A \to b, B \to b]. & \end{array}$$

Definimos o conjunto P, formado por todas as produções que compõem as matrizes de M:

$$P = \{p_1 = S \to AB, p_2 = A \to aA, p_3 = B \to aB, p_4 = A \to bA, p_5 = B \to bB, \\ p_6 = A \to a, p_7 = B \to a, p_8 = A \to b, p_9 = B \to b.\}$$

Aplicando o método de demonstração do lema 3.2.6 à gramática livre de contexto matricial G_2 do exemplo 2.1.2 da seção 2.1, obtemos a função adaptativa:

$$\mathcal{A}(p) = \begin{cases} \{p_1, p_2, p_4, p_6, p_8\}, & p = p_1, \\ \{p_3\}, & p = p_2, \\ \{p_1, p_2, p_4, p_6, p_8\}, & p = p_3, \\ \{p_5\}, & p = p_4, \\ \{p_1, p_2, p_4, p_6, p_8\}, & p = p_5. \\ \{p_7, \}, & p = p_6, \\ \{p_1, p_2, p_4, p_6, p_8\}, & p = p_7, \\ \{p_9\}, & p = p_8, \\ \{p_1, p_2, p_4, p_6, p_8\}, & p = p_9. \end{cases}$$

Observamos que, na demonstração do lema 3.2.6, o conjunto inicial de produções livres de contexto adaptativas é formado pelas primeiras produções de todas as matrizes; foi assim que definimos P_0 no exemplo anterior; essa definição contempla a possibilidade de o símbolo inicial aparecer em todas as primeiras produções de todas as matrizes. Mas, neste caso particular, o símbolo inicial aparece somente na matriz m_1 e podemos definir o conjunto inicial de produções livres de contexto adaptativas como:

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}()\}AB\}.$$

Escrita desta forma, a função adaptativa $\mathcal{A}$ está pronta para ser decomposta em cinco funções adaptativas, segundo a observação 3.2.1, $\mathcal{A} = \{\mathcal{A}_1(), \mathcal{A}_2(), \mathcal{A}_3(), \mathcal{A}_4(), \mathcal{A}_5()\}$, e neste caso, o conjunto inicial de produções adaptativas é:

$$[P_0] = \{[p_1] = S \to \{\mathcal{A}_1()\}AB\}$$

e as funções adaptativas são especificadas por:

$\mathcal{A}_1()$	$\mathcal{A}_2()$	$\mathcal{A}_3()$
$\{$	$\{$	$\{$
$-[B \to \{\mathcal{A}_1()\}aB]$,	$-[S \to \{\mathcal{A}_1()\}AB]$,	$-[S \to \{\mathcal{A}_1()\}AB]$,
$-[B \to \{\mathcal{A}_1()\}bB]$,	$-[A \to \{\mathcal{A}_2()\}aA]$,	$-[A \to \{\mathcal{A}_2()\}aA]$,
$-[B \to \{\mathcal{A}_1()\}a]$,	$-[A \to \{\mathcal{A}_3()\}bA]$,	$-[B \to \{\mathcal{A}_1()\}bB]$,
$-[B \to \{\mathcal{A}_1()\}b]$,	$-[B \to \{\mathcal{A}_1()\}bB]$,	$-[A \to \{\mathcal{A}_3()\}bA]$,
$+[S \to \{\mathcal{A}_1()\}AB]$,	$-[A \to \{\mathcal{A}_4()\}a]$,	$-[A \to \{\mathcal{A}_4()\}a]$,
$+[A \to \{\mathcal{A}_2()\}aA]$,	$-[B \to \{\mathcal{A}_1()\}a]$,	$-[B \to \{\mathcal{A}_1()\}a]$,
$+[A \to \{\mathcal{A}_3()\}bA]$,	$-[A \to \{\mathcal{A}_5()\}b]$,	$-[A \to \{\mathcal{A}_5()\}b]$,
$+[A \to \{\mathcal{A}_4()\}a]$,	$-[B \to \{\mathcal{A}_1()\}b]$,	$-[B \to \{\mathcal{A}_1()\}b]$,
$+[A \to \{\mathcal{A}_5()\}b]$,	$+[B \to \{\mathcal{A}_1()\}aB]$,	$+[B \to \{\mathcal{A}_1()\}bB]$.
$\}$	$\}$	$\}$

$\mathcal{A}_4()$	$\mathcal{A}_5()$
$\{$	$\{$
$-[S \to \{\mathcal{A}_1()\}AB]$,	$-[S \to \{\mathcal{A}_1()\}AB]$,
$-[A \to \{\mathcal{A}_2()\}aA]$,	$-[A \to \{\mathcal{A}_2()\}aA]$,
$-[B \to \{\mathcal{A}_1()\}aB]$,	$-[B \to \{\mathcal{A}_1()\}aB]$,
$-[A \to \{\mathcal{A}_3()\}bA]$,	$-[A \to \{\mathcal{A}_3()\}bA]$,
$-[B \to \{\mathcal{A}_1()\}bB]$,	$-[B \to \{\mathcal{A}_1()\}bB]$,
$-[A \to \{\mathcal{A}_4()\}a]$,	$-[A \to \{\mathcal{A}_4()\}a]$,
$-[A \to \{\mathcal{A}_1()\}b]$,	$-[B \to \{\mathcal{A}_1()\}a]$,
$-[B \to \{\mathcal{A}_5()\}b]$,	$-[A \to \{\mathcal{A}_5()\}b]$,
$+[B \to \{\mathcal{A}_1()\}a]$	$+[B \to \{\mathcal{A}_1()\}b]$
$\}$	$\}$

Nestas condições, a gramática livre de contexto adaptativa $GA = (G, [P_0], \emptyset, \mathcal{A})$ *sem verificação de aparência gera a linguagem* $L = \{a^n b^n c^n : n \geq 1\}$.

Derivação de bbaabbbaab*:*
Com efeito:

i	*Passo i da derivação*	*Produção aplicada*	$[P_i]$
0			$\{[p_1], [p_2], [p_4], [p_6], [p_8]\}$
1	$S \Rightarrow^{1}_{[p_1]} AB$	$[p_1] = S \rightarrow \{\mathcal{A}_1()\}AB$	$\{[p_1], [p_2], [p_4], [p_6], [p_8]\}$
2	$\Rightarrow^{2}_{[p_4]} bAB$	$[p_4] = A \rightarrow \{\mathcal{A}_3()\}bA$	$\{[p_5]\}$
3	$\Rightarrow^{3}_{[p_5]} bAbB$	$[p_5] = B \rightarrow \{\mathcal{A}_1()\}bB$	$\{[p_1], [p_2], [p_4], [p_6], [p_8]\}$
4	$\Rightarrow^{4}_{[p_4]} bbAbB$	$[p_4] = A \rightarrow \{\mathcal{A}_3()\}bA$	$\{[p_5]\}$
5	$\Rightarrow^{5}_{[p_5]} bbAbbB$	$[p_5] = B \rightarrow \{\mathcal{A}_1()\}bB$	$\{[p_1], [p_2], [p_4], [p_6], [p_8]\}$
6	$\Rightarrow^{6}_{[p_2]} bbaAbbB$	$[p_2] = A \rightarrow \{\mathcal{A}_2()\}aA$	$\{[p_3]\}$
7	$\Rightarrow^{7}_{[p_3]} bbaAbbaB$	$[p_3] = B \rightarrow \{\mathcal{A}_1()\}aB$	$\{[p_1], [p_2], [p_4], [p_6], [p_8]\}$
8	$\Rightarrow^{8}_{[p_2]} bbaaAbbaB$	$[p_2] = A \rightarrow \{\mathcal{A}_2()\}aA$	$\{[p_3]\}$
9	$\Rightarrow^{9}_{[p_3]} bbaaAbbaaB$	$[p_3] = B \rightarrow \{\mathcal{A}_1()\}aB$	$\{[p_1], [p_2], [p_4], [p_6], [p_8]\}$
10	$\Rightarrow^{10}_{[p_8]} bbaabbbaaB$	$[p_8] = A \rightarrow \{\mathcal{A}_5()\}b$	$\{[p_9]\}$
11	$\Rightarrow^{11}_{[p_9]} bbaabbbaab.$	$[p_9] = A \rightarrow \{\mathcal{A}_1()\}b$	$\{[p_1], [p_2], [p_4], [p_6], [p_8]\}$

Exemplo 3.2.8 Gramática livre de contexto adaptativa para a linguagem $L = \{a^{2^n} : n \geq 0\}$. *Consideremos a gramática matricial com verificação de aparência:*

$$G_3 = (\{S, A, B, C, U\}, \{a\}, M, S, F)$$

com

$$M = \{m_1, m_2, m_3, m_4, m_5\},$$

sendo

$$m_1 = [B \to U, A \to U, S \to CC], \quad m_2 = [S \to U, C \to B], \quad m_3 = [C \to U, B \to S],$$
$$m_4 = [B \to U, C \to U, S \to A], \quad m_5 = [S \to U, A \to a].$$

$$F = \{S \to U, A \to U, B \to U, C \to U\}.$$

Aqui se apresenta o problema de distinguir as ocorrências de uma mesma produção em duas matrizes diferentes; por exemplo, não é o mesmo aplicar a produção $S \to U$ *à matriz* m_5 *do que aplicá-la à matriz* m_2 *pois, no primeiro caso, essa aplicação deveria ser seguida pela aplicação da produção* $A \to a$ *e, no segundo caso, pela produção* $C \to B$. *Em outros termos, o valor da função adaptativa* $\mathcal{A}$ *não estaria bem definido para a produção* $S \to U$.

Para evitar esse problema, devemos identificar univocamente cada produção em cada matriz: devemos distinguir as diferentes ocorrências de cada produção. Pode-se fazer isso atribuindo um nome diferente a cada ocorrência de cada uma das produções.

Definimos o conjunto P *como a coleção de todas as produções que compõem as matrizes de* M

$$\begin{aligned} P = \{ \quad & p_1 = B \to U, \quad p_2 = A \to U, \quad p_3 = S \to CC, \quad p_4 = S \to U, \\ & p_5 = C \to B, \quad p_6 = C \to U, \quad p_7 = B \to S, \quad p_8 = B \to U, \\ & p_9 = C \to U, \quad p_{10} = S \to A, \quad p_{11} = S \to U, \quad p_{12} = A \to a \ \} \end{aligned}$$

$$\begin{aligned} F = \{ \quad & p_1 = B \to U, \quad p_2 = A \to U, \quad p_4 = S \to U, \quad p_6 = C \to U, \\ & p_8 = B \to U, \quad p_9 = C \to U, \quad p_{11} = S \to U \ \} \end{aligned}$$

Aplicando o método de demonstração do lema 3.2.4 à gramática livre de contexto matricial G_3 do exemplo 2.1.3 da seção 2.1, obtemos a função adaptativa:

$$\mathcal{A}(p) = \begin{cases} \{p_2\}, & p = p_1, \\ \{p_3\}, & p = p_2, \\ \{p_1, p_4, p_6, p_8, p_{11}\}, & p = p_3, \\ \{p_5\}, & p = p_4, \\ \{p_1, p_4, p_6, p_8, p_{11}\}, & p = p_5, \\ \{p_7\}, & p = p_6, \\ \{p_1, p_4, p_6, p_8, p_{11}\}, & p = p_7, \\ \{p_9\}, & p = p_8. \\ \{p_{10}\}, & p = p_9. \\ \{p_1, p_4, p_6, p_8, p_{11}\}, & p = p_{10}, \\ \{p_{12}\}, & p = p_{11}. \\ \{p_1, p_4, p_6, p_8, p_{11}\}, & p = p_{12}, \end{cases}$$

e o conjunto inicial de produções livres de contexto adaptativas é:

$$[P_0] = \{ \quad [p_1] = B \to \{\mathcal{A}()\}U, \quad [p_4] = S \to \{\mathcal{A}()\}U, \qquad [p_6] = C \to \{\mathcal{A}()\}U, \\ [p_8] = B \to \{\mathcal{A}()\}U, \quad [p_{11}] = S \to \{\mathcal{A}()\}U \ \}$$

Escrita desta forma, a função adaptativa $\mathcal{A}$ está pronta para ser decomposta em oito funções adaptativas, segundo a observação 3.2.1: $\mathcal{A} = \{\mathcal{A}_1(), \mathcal{A}_2(), \mathcal{A}_3(), \mathcal{A}_4(), \mathcal{A}_5(), \mathcal{A}_6(), \mathcal{A}_7(), \mathcal{A}_8()\}$. Neste caso, o conjunto inicial de produções adaptativas é:

$$[P_0] = \{ \quad [p_1] = B \to \{\mathcal{A}_1()\}U, \quad [p_4] = S \to \{\mathcal{A}_4()\}U, \qquad [p_6] = C \to \{\mathcal{A}_5()\}U, \\ [p_8] = B \to \{\mathcal{A}_6()\}U, \quad [p_{11}] = S \to \{\mathcal{A}_8()\}U \ \}$$

e as funções adaptativas são especificadas por:

$\mathcal{A}_1()$
{
$-[B \to \{\mathcal{A}_1()\}U]$, $+[A \to \{\mathcal{A}_2()\}U]$,
$-[S \to \{\mathcal{A}_3()\}CC]$, $-[S \to \{\mathcal{A}_4()\}U]$,
$-[C \to \{\mathcal{A}_3()\}B]$, $-[C \to \{\mathcal{A}_5()\}U]$,
$-[B \to \{\mathcal{A}_3()\}S]$, $-[B \to \{\mathcal{A}_6()\}U]$,
$-[C \to \{\mathcal{A}_7()\}U]$, $-[S \to \{\mathcal{A}_3()\}A]$,
$-[S \to \{\mathcal{A}_8()\}U]$, $-[A \to \{\mathcal{A}_3()\}a]$
}
	$\mathcal{A}_2()$
	{
	$-[B \to \{\mathcal{A}_1()\}U]$, $-[A \to \{\mathcal{A}_2()\}U]$,
	$+[S \to \{\mathcal{A}_3()\}CC]$, $-[S \to \{\mathcal{A}_4()\}U]$,
	$-[C \to \{\mathcal{A}_3()\}B]$, $-[C \to \{\mathcal{A}_5()\}U]$,
	$-[B \to \{\mathcal{A}_3()\}S]$, $-[B \to \{\mathcal{A}_6()\}U]$,
	$-[C \to \{\mathcal{A}_7()\}U]$, $-[S \to \{\mathcal{A}_3()\}A]$,
	$-[S \to \{\mathcal{A}_8()\}U]$, $-[A \to \{\mathcal{A}_3()\}a]$
	}

$\mathcal{A}_3()$
{
$+[B \to \{\mathcal{A}_1()\}U]$, $-[A \to \{\mathcal{A}_2()\}U]$,
$-[S \to \{\mathcal{A}_3()\}CC]$, $+[S \to \{\mathcal{A}_4()\}U]$,
$-[C \to \{\mathcal{A}_3()\}B]$, $+[C \to \{\mathcal{A}_5()\}U]$,
$-[B \to \{\mathcal{A}_3()\}S]$, $+[B \to \{\mathcal{A}_6()\}U]$,
$-[C \to \{\mathcal{A}_7()\}U]$, $-[S \to \{\mathcal{A}_3()\}A]$,
$+[S \to \{\mathcal{A}_8()\}U]$, $-[A \to \{\mathcal{A}_3()\}a]$
}
	$\mathcal{A}_4()$
	{
	$-[B \to \{\mathcal{A}_1()\}U]$, $-[A \to \{\mathcal{A}_2()\}U]$,
	$-[S \to \{\mathcal{A}_3()\}CC]$, $-[S \to \{\mathcal{A}_4()\}U]$,
	$+[C \to \{\mathcal{A}_3()\}B]$, $-[C \to \{\mathcal{A}_5()\}U]$,
	$-[B \to \{\mathcal{A}_3()\}S]$, $-[B \to \{\mathcal{A}_6()\}U]$,
	$-[C \to \{\mathcal{A}_7()\}U]$, $-[S \to \{\mathcal{A}_3()\}A]$,
	$-[S \to \{\mathcal{A}_8()\}U]$, $-[A \to \{\mathcal{A}_3()\}a]$
	}

$\mathcal{A}_5()$
{
$-[B \to \{\mathcal{A}_1()\}U]$, $-[A \to \{\mathcal{A}_2()\}U]$,
$-[S \to \{\mathcal{A}_3()\}CC]$, $-[S \to \{\mathcal{A}_4()\}U]$,
$-[C \to \{\mathcal{A}_3()\}B]$, $-[C \to \{\mathcal{A}_5()\}U]$,
$+[B \to \{\mathcal{A}_3()\}S]$, $-[B \to \{\mathcal{A}_6()\}U]$,
$-[C \to \{\mathcal{A}_7()\}U]$, $-[S \to \{\mathcal{A}_3()\}A]$,
$-[S \to \{\mathcal{A}_8()\}U]$, $-[A \to \{\mathcal{A}_3()\}a]$
}
	$\mathcal{A}_6()$
	{
	$-[B \to \{\mathcal{A}_1()\}U]$, $-[A \to \{\mathcal{A}_2()\}U]$,
	$-[S \to \{\mathcal{A}_3()\}CC]$, $-[S \to \{\mathcal{A}_4()\}U]$,
	$-[C \to \{\mathcal{A}_3()\}B]$, $-[C \to \{\mathcal{A}_5()\}U]$,
	$-[B \to \{\mathcal{A}_3()\}S]$, $-[B \to \{\mathcal{A}_6()\}U]$,
	$+[C \to \{\mathcal{A}_7()\}U]$, $-[S \to \{\mathcal{A}_3()\}A]$,
	$-[S \to \{\mathcal{A}_8()\}U]$, $-[A \to \{\mathcal{A}_3()\}a]$
	}

$\mathcal{A}_7()$
{
$-[B \to \{\mathcal{A}_1()\}U]$, $-[A \to \{\mathcal{A}_2()\}U]$,
$-[S \to \{\mathcal{A}_3()\}CC]$, $-[S \to \{\mathcal{A}_4()\}U]$,
$-[C \to \{\mathcal{A}_3()\}B]$, $-[C \to \{\mathcal{A}_5()\}U]$,
$-[B \to \{\mathcal{A}_3()\}S]$, $-[B \to \{\mathcal{A}_6()\}U]$,
$-[C \to \{\mathcal{A}_7()\}U]$, $+[S \to \{\mathcal{A}_3()\}A]$,
$-[S \to \{\mathcal{A}_8()\}U]$, $-[A \to \{\mathcal{A}_3()\}a]$
}
	$\mathcal{A}_8()$
	{
	$-[B \to \{\mathcal{A}_1()\}U]$, $-[A \to \{\mathcal{A}_2()\}U]$,
	$-[S \to \{\mathcal{A}_3()\}CC]$, $-[S \to \{\mathcal{A}_4()\}U]$,
	$-[C \to \{\mathcal{A}_3()\}B]$, $-[C \to \{\mathcal{A}_5()\}U]$,
	$-[B \to \{\mathcal{A}_3()\}S]$, $-[B \to \{\mathcal{A}_6()\}U]$,
	$-[C \to \{\mathcal{A}_7()\}U]$, $-[S \to \{\mathcal{A}_3()\}A]$,
	$-[S \to \{\mathcal{A}_8()\}U]$, $+[A \to \{\mathcal{A}_3()\}a]$
	}

Nessas condições, a gramática livre de contexto adaptativa $GA = (G, [P_0], [F], \mathcal{A})$ *gera a linguagem* $L = \{a^{2^n} : n \geq 0\}$.

Derivação de a^{2^3}

i	*Passo i da derivação*	*Produção aplicada*	$[P_i]$
0			$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
1	$S \Rightarrow^{ac^1}_{[p_1]} S$	$[p_1] = B \rightarrow \{\mathcal{A}()\}U$	”
2	$\Rightarrow^{ac^2}_{[p_2]} S$	$[p_2] = A \rightarrow \{\mathcal{A}()\}U$	$\{[p_3]\}$
3	$\Rightarrow^{3}_{[p_3]} CC$	$[p_3] = S \rightarrow \{\mathcal{A}()\}CC$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
4	$\Rightarrow^{ac^4}_{[p_4]} CC$	$[p_4] = S \rightarrow \{\mathcal{A}()\}U$	$\{[p_5]\}$
5	$\Rightarrow^{5}_{[p_5]} BC$	$[p_5] = C \rightarrow \{\mathcal{A}()\}B$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
6	$\Rightarrow^{ac^6}_{[p_4]} BC$	$[p_4] = S \rightarrow \{\mathcal{A}()\}U$	$\{[p_5]\}$
7	$\Rightarrow^{7}_{[p_5]} BB$	$[p_5] = C \rightarrow \{\mathcal{A}()\}B$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
8	$\Rightarrow^{ac^8}_{[p_6]} BB$	$[p_6] = C \rightarrow \{\mathcal{A}()\}U$	$\{[p_7]\}$
9	$\Rightarrow^{9}_{[p_7]} BS$	$[p_7] = B \rightarrow \{\mathcal{A}()\}S$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
9	$\Rightarrow^{ac^{10}}_{[p_6]} BB$	$[p_6] = C \rightarrow \{\mathcal{A}()\}U$	$\{[p_7]\}$
10	$\Rightarrow^{11}_{[p_7]} SS$	$[p_7] = B \rightarrow \{\mathcal{A}()\}S$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
11	$S \Rightarrow^{ac^{11}}_{[p_1]} SS$	$[p_1] = B \rightarrow \{\mathcal{A}()\}U$	”
12	$\Rightarrow^{ac^{12}}_{[p_2]} SS$	$[p_2] = A \rightarrow \{\mathcal{A}()\}U$	$\{[p_3]\}$
13	$\Rightarrow^{13}_{[p_3]} SCC$	$[p_3] = S \rightarrow \{\mathcal{A}()\}CC$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
14	$S \Rightarrow^{ac^{14}}_{[p_1]} SCC$	$[p_1] = B \rightarrow \{\mathcal{A}()\}U$	”
15	$\Rightarrow^{ac^{15}}_{[p_2]} SCC$	$[p_2] = A \rightarrow \{\mathcal{A}()\}U$	$\{[p_3]\}$
16	$\Rightarrow^{16}_{[p_3]} CCCC$	$[p_3] = S \rightarrow \{\mathcal{A}()\}CC$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
17	$\Rightarrow^{ac^{17}}_{[p_4]} CCCC$	$[p_4] = S \rightarrow \{\mathcal{A}()\}U$	$\{[p_5]\}$
18	$\Rightarrow^{18}_{[p_5]} CCBC$	$[p_5] = C \rightarrow \{\mathcal{A}()\}B$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
19	$\Rightarrow^{ac^{19}}_{[p_4]} CCBC$	$[p_4] = S \rightarrow \{\mathcal{A}()\}U$	$\{[p_5]\}$
20	$\Rightarrow^{20}_{[p_5]} BCBC$	$[p_5] = C \rightarrow \{\mathcal{A}()\}B$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
21	$\Rightarrow^{ac^{21}}_{[p_4]} BCBC$	$[p_4] = S \rightarrow \{\mathcal{A}()\}U$	$\{[p_5]\}$
22	$\Rightarrow^{22}_{[p_5]} BCBB$	$[p_5] = C \rightarrow \{\mathcal{A}()\}B$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
23	$\Rightarrow^{ac^{23}}_{[p_4]} BCBB$	$[p_4] = S \rightarrow \{\mathcal{A}()\}U$	$\{[p_5]\}$
24	$\Rightarrow^{24}_{[p_5]} BBBB$	$[p_5] = C \rightarrow \{\mathcal{A}()\}B$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
25	$\Rightarrow^{ac^{25}}_{[p_6]} BBBB$	$[p_6] = C \rightarrow \{\mathcal{A}()\}U$	$\{[p_7]\}$
26	$\Rightarrow^{26}_{[p_7]} SBBB$	$[p_7] = B \rightarrow \{\mathcal{A}()\}S$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
27	$\Rightarrow^{ac^{27}}_{[p_6]} SBBB$	$[p_6] = C \rightarrow \{\mathcal{A}()\}U$	$\{[p_7]\}$
28	$\Rightarrow^{28}_{[p_7]} SBSB$	$[p_7] = B \rightarrow \{\mathcal{A}()\}S$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
29	$\Rightarrow^{ac^{29}}_{[p_6]} SBSB$	$[p_6] = C \rightarrow \{\mathcal{A}()\}U$	$\{[p_7]\}$
30	$\Rightarrow^{30}_{[p_7]} SBSS$	$[p_7] = B \rightarrow \{\mathcal{A}()\}S$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$
31	$\Rightarrow^{ac^{31}}_{[p_6]} SBSS$	$[p_6] = C \rightarrow \{\mathcal{A}()\}U$	$\{[p_7]\}$
32	$\Rightarrow^{32}_{[p_7]} SSSS$	$[p_7] = B \rightarrow \{\mathcal{A}()\}S$	$\{[p_1],[p_4],[p_6],[p_8],[p_{11}]\}$

i	*Passo i da derivação*	*Produção aplicada*	$[P_i]$
33	$S \Rightarrow^{ac^{33}}_{[p_1]} SSSS$	$[p_1] = B \rightarrow \{\mathcal{A}()\}U$	”
34	$\Rightarrow^{ac^{34}}_{[p_2]} SSSS$	$[p_2] = A \rightarrow \{\mathcal{A}()\}U$	$\{[p_3]\}$
35	$\Rightarrow^{35}_{[p_3]} SSSCC$	$[p_3] = S \rightarrow \{\mathcal{A}()\}CC$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
36	$S \Rightarrow^{ac^{33}}_{[p_1]} SSSCC$	$[p_1] = B \rightarrow \{\mathcal{A}()\}U$	”
37	$\Rightarrow^{ac^{34}}_{[p_2]} SSSCC$	$[p_2] = A \rightarrow \{\mathcal{A}()\}U$	$\{[p_3]\}$
38	$\Rightarrow^{35}_{[p_3]} CCSSCC$	$[p_3] = S \rightarrow \{\mathcal{A}()\}CC$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
39	$S \Rightarrow^{ac^{33}}_{[p_1]} CCSSCC$	$[p_1] = B \rightarrow \{\mathcal{A}()\}U$	”
40	$\Rightarrow^{ac^{34}}_{[p_2]} CCSSCC$	$[p_2] = A \rightarrow \{\mathcal{A}()\}U$	$\{[p_3]\}$
41	$\Rightarrow^{35}_{[p_3]} CCCCSCC$	$[p_3] = S \rightarrow \{\mathcal{A}()\}CC$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
42	$S \Rightarrow^{ac^{33}}_{[p_1]} CCCCSCC$	$[p_1] = B \rightarrow \{\mathcal{A}()\}U$	”
43	$\Rightarrow^{ac^{34}}_{[p_2]} CCCCSCC$	$[p_2] = A \rightarrow \{\mathcal{A}()\}U$	$\{[p_3]\}$
44	$\Rightarrow^{35}_{[p_3]} CCCCCCCC$	$[p_3] = S \rightarrow \{\mathcal{A}()\}CC$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
45	$\Rightarrow^{36}_{[p_4]} CCCCCCCC$	$[p_4] = S \rightarrow \{\mathcal{A}_4()\}U$	$\{[p_5]\}$
46	$\Rightarrow^{37}_{[p_5]} CCCCCCCB$	$[p_5] = C \rightarrow \{\mathcal{A}_3()\}B$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
47	$\Rightarrow^{38}_{[p_4]} CCCCCCCB$	$[p_4] = S \rightarrow \{\mathcal{A}_4()\}U$	$\{[p_5]\}$
48	$\Rightarrow^{39}_{[p_5]} CCCCCCBB$	$[p_5] = C \rightarrow \{\mathcal{A}_3()\}B$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
49	$\Rightarrow^{40}_{[p_4]} CCCCCCBB$	$[p_4] = S \rightarrow \{\mathcal{A}_4()\}U$	$\{[p_5]\}$
50	$\Rightarrow^{41}_{[p_5]} CCCCCBBB$	$[p_5] = C \rightarrow \{\mathcal{A}_3()\}B$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
51	$\Rightarrow^{42}_{[p_4]} CCCCCBBB$	$[p_4] = S \rightarrow \{\mathcal{A}_4()\}U$	$\{[p_5]\}$
52	$\Rightarrow^{43}_{[p_5]} CCCCBBBB$	$[p_5] = C \rightarrow \{\mathcal{A}_3()\}B$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
53	$\Rightarrow^{44}_{[p_4]} CCCCBBBB$	$[p_4] = S \rightarrow \{\mathcal{A}_4()\}U$	$\{[p_5]\}$
54	$\Rightarrow^{45}_{[p_5]} CCCBBBBB$	$[p_5] = C \rightarrow \{\mathcal{A}_3()\}B$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
55	$\Rightarrow^{46}_{[p_4]} CCCBBBBB$	$[p_4] = S \rightarrow \{\mathcal{A}_4()\}U$	$\{[p_5]\}$
56	$\Rightarrow^{47}_{[p_5]} CCBBBBBB$	$[p_5] = C \rightarrow \{\mathcal{A}_3()\}B$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
58	$\Rightarrow^{48}_{[p_4]} CCBBBBBB$	$[p_4] = S \rightarrow \{\mathcal{A}_4()\}U$	$\{[p_5]\}$
59	$\Rightarrow^{49}_{[p_5]} CBBBBBBB$	$[p_5] = C \rightarrow \{\mathcal{A}_3()\}B$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
60	$\Rightarrow^{50}_{[p_4]} CBBBBBBB$	$[p_4] = S \rightarrow \{\mathcal{A}_4()\}U$	$\{[p_5]\}$
61	$\Rightarrow^{51}_{[p_5]} BBBBBBBB$	$[p_5] = C \rightarrow \{\mathcal{A}_3()\}B$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
62	$\Rightarrow^{52}_{[p_6]} BBBBBBBB$	$[p_6] = C \rightarrow \{\mathcal{A}_5()\}U$	$\{[p_7]\}$
63	$\Rightarrow^{53}_{[p_7]} SBBBBBBB$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}S$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
64	$\Rightarrow^{54}_{[p_6]} SBBBBBBB$	$[p_6] = C \rightarrow \{\mathcal{A}_5()\}U$	$\{[p_7]\}$
65	$\Rightarrow^{55}_{[p_7]} SSBBBBBB$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}S$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
66	$\Rightarrow^{56}_{[p_6]} SSBBBBBB$	$[p_6] = C \rightarrow \{\mathcal{A}_5()\}U$	$\{[p_7]\}$
67	$\Rightarrow^{57}_{[p_7]} SSSBBBBB$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}S$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
68	$\Rightarrow^{58}_{[p_6]} SSSBBBBB$	$[p_6] = C \rightarrow \{\mathcal{A}_5()\}U$	$\{[p_7]\}$

i	*Passo i da derivação*	*Produção aplicada*	$[P_i]$
69	$\Rightarrow^{59}_{[p_7]} SSSSBBBB$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}S$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
70	$\Rightarrow^{60}_{[p_6]} SSSSBBBB$	$[p_6] = C \rightarrow \{\mathcal{A}_5()\}U$	$\{[p_7]\}$
71	$\Rightarrow^{61}_{[p_7]} SSSSSBBB$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}S$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
72	$\Rightarrow^{62}_{[p_6]} SSSSSBBB$	$[p_6] = C \rightarrow \{\mathcal{A}_5()\}U$	$\{[p_7]\}$
73	$\Rightarrow^{63}_{[p_7]} SSSSSSBB$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}S$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
74	$\Rightarrow^{64}_{[p_6]} SSSSSSBB$	$[p_6] = C \rightarrow \{\mathcal{A}_5()\}U$	$\{[p_7]\}$
75	$\Rightarrow^{65}_{[p_7]} SSSSSSSB$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}S$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
76	$\Rightarrow^{66}_{[p_6]} SSSSSSSB$	$[p_6] = C \rightarrow \{\mathcal{A}_5()\}U$	$\{[p_7]\}$
77	$\Rightarrow^{67}_{[p_7]} SSSSSSSS$	$[p_7] = B \rightarrow \{\mathcal{A}_3()\}S$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
78	$\Rightarrow^{ac^{68}}_{[p_8]} SSSSSSSS$	$[p_8] = B \rightarrow \{\mathcal{A}_6()\}U$	$\{[p_7]\}$
79	$\Rightarrow^{ac^{69}}_{[p_9]} SSSSSSSS$	$[p_9] = C \rightarrow \{\mathcal{A}_7()\}U$	$\{[p_7]\}$
80	$\Rightarrow^{70}_{[p_{10}]} ASSSSSSS$	$[p_{10}] = S \rightarrow \{\mathcal{A}_3()\}A$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
81	$\Rightarrow^{ac^{71}}_{[p_8]} ASSSSSSS$	$[p_8] = B \rightarrow \{\mathcal{A}_6()\}U$	$\{[p_7]\}$
82	$\Rightarrow^{ac^{72}}_{[p_9]} ASSSSSSS$	$[p_9] = C \rightarrow \{\mathcal{A}_7()\}U$	$\{[p_7]\}$
83	$\Rightarrow^{73}_{[p_{10}]} AASSSSSS$	$[p_{10}] = S \rightarrow \{\mathcal{A}_3()\}A$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
84	$\Rightarrow^{ac^{74}}_{[p_8]} AASSSSSS$	$[p_8] = B \rightarrow \{\mathcal{A}_6()\}U$	$\{[p_7]\}$
85	$\Rightarrow^{ac^{75}}_{[p_9]} AASSSSSS$	$[p_9] = C \rightarrow \{\mathcal{A}_7()\}U$	$\{[p_7]\}$
86	$\Rightarrow^{76}_{[p_{10}]} AAASSSSS$	$[p_{10}] = S \rightarrow \{\mathcal{A}_3()\}A$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
87	$\Rightarrow^{ac^{77}}_{[p_8]} AAASSSSS$	$[p_8] = B \rightarrow \{\mathcal{A}_6()\}U$	$\{[p_7]\}$
88	$\Rightarrow^{ac^{78}}_{[p_9]} AAASSSSS$	$[p_9] = C \rightarrow \{\mathcal{A}_7()\}U$	$\{[p_7]\}$
89	$\Rightarrow^{79}_{[p_{10}]} AAAASSSS$	$[p_{10}] = S \rightarrow \{\mathcal{A}_3()\}A$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
90	$\Rightarrow^{ac^{80}}_{[p_8]} AAAASSSS$	$[p_8] = B \rightarrow \{\mathcal{A}_6()\}U$	$\{[p_7]\}$
91	$\Rightarrow^{ac^{81}}_{[p_9]} AAAASSSS$	$[p_9] = C \rightarrow \{\mathcal{A}_7()\}U$	$\{[p_7]\}$
92	$\Rightarrow^{82}_{[p_{10}]} AAAAASSS$	$[p_{10}] = S \rightarrow \{\mathcal{A}_3()\}A$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
93	$\Rightarrow^{ac^{83}}_{[p_8]} AAAAASSS$	$[p_8] = B \rightarrow \{\mathcal{A}_6()\}U$	$\{[p_7]\}$
94	$\Rightarrow^{ac^{84}}_{[p_9]} AAAAASSS$	$[p_9] = C \rightarrow \{\mathcal{A}_7()\}U$	$\{[p_7]\}$
95	$\Rightarrow^{85}_{[p_{10}]} AAAAAASS$	$[p_{10}] = S \rightarrow \{\mathcal{A}_3()\}A$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
96	$\Rightarrow^{ac^{86}}_{[p_8]} AAAAAASS$	$[p_8] = B \rightarrow \{\mathcal{A}_6()\}U$	$\{[p_7]\}$
97	$\Rightarrow^{ac^{87}}_{[p_9]} AAAAAASS$	$[p_9] = C \rightarrow \{\mathcal{A}_7()\}U$	$\{[p_7]\}$
98	$\Rightarrow^{88}_{[p_{10}]} AAAAAAAS$	$[p_{10}] = S \rightarrow \{\mathcal{A}_3()\}A$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
99	$\Rightarrow^{ac^{89}}_{[p_8]} AAAAAAAS$	$[p_8] = B \rightarrow \{\mathcal{A}_6()\}U$	$\{[p_7]\}$
100	$\Rightarrow^{ac^{90}}_{[p_9]} AAAAAAAS$	$[p_9] = C \rightarrow \{\mathcal{A}_7()\}U$	$\{[p_7]\}$
101	$\Rightarrow^{91}_{[p_{10}]} AAAAAAAA$	$[p_{10}] = S \rightarrow \{\mathcal{A}_3()\}A$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
102	$\Rightarrow^{ac^{92}}_{[p_{11}]} AAAAAAAA$	$[p_{11}] = S \rightarrow \{\mathcal{A}_8()\}U$	$\{[p_{12}]\}$
103	$\Rightarrow^{ac^{93}}_{[p_{12}]} aAAAAAAA$	$[p_{12}] = A \rightarrow \{\mathcal{A}_3()\}a$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$

i	*Passo i da derivação*	*Produção aplicada*	$[P_i]$
104	$\Rightarrow^{ac^{92}}_{[p_{11}]} aAAAAAAA$	$[p_{11}] = S \rightarrow \{\mathcal{A}_8()\}U$	$\{[p_{12}]\}$
105	$\Rightarrow^{ac^{93}}_{[p_{12}]} aaAAAAAA$	$[p_{12}] = A \rightarrow \{\mathcal{A}_3()\}a$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
106	$\Rightarrow^{ac^{92}}_{[p_{11}]} aaAAAAAA$	$[p_{11}] = S \rightarrow \{\mathcal{A}_8()\}U$	$\{[p_{12}]\}$
107	$\Rightarrow^{ac^{93}}_{[p_{12}]} aaaAAAAA$	$[p_{12}] = A \rightarrow \{\mathcal{A}_3()\}a$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
108	$\Rightarrow^{ac^{92}}_{[p_{11}]} aaaAAAAA$	$[p_{11}] = S \rightarrow \{\mathcal{A}_8()\}U$	$\{[p_{12}]\}$
109	$\Rightarrow^{ac^{93}}_{[p_{12}]} aaaaAAAA$	$[p_{12}] = A \rightarrow \{\mathcal{A}_3()\}a$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
110	$\Rightarrow^{ac^{92}}_{[p_{11}]} aaaaAAAA$	$[p_{11}] = S \rightarrow \{\mathcal{A}_8()\}U$	$\{[p_{12}]\}$
111	$\Rightarrow^{ac^{93}}_{[p_{12}]} aaaaaAAA$	$[p_{12}] = A \rightarrow \{\mathcal{A}_3()\}a$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
112	$\Rightarrow^{ac^{92}}_{[p_{11}]} aaaaaAAA$	$[p_{11}] = S \rightarrow \{\mathcal{A}_8()\}U$	$\{[p_{12}]\}$
113	$\Rightarrow^{ac^{93}}_{[p_{12}]} aaaaaaAA$	$[p_{12}] = A \rightarrow \{\mathcal{A}_3()\}a$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
114	$\Rightarrow^{ac^{92}}_{[p_{11}]} aaaaaaAA$	$[p_{11}] = S \rightarrow \{\mathcal{A}_8()\}U$	$\{[p_{12}]\}$
115	$\Rightarrow^{ac^{93}}_{[p_{12}]} aaaaaaaA$	$[p_{12}] = A \rightarrow \{\mathcal{A}_3()\}a$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$
112	$\Rightarrow^{ac^{92}}_{[p_{11}]} aaaaaaaA$	$[p_{11}] = S \rightarrow \{\mathcal{A}_8()\}U$	$\{[p_{12}]\}$
113	$\Rightarrow^{ac^{93}}_{[p_{12}]} aaaaaaaa$	$[p_{12}] = A \rightarrow \{\mathcal{A}_3()\}a$	$\{[p_1], [p_4], [p_6], [p_8], [p_{11}]\}$

3.2.4 Conversão de gramáticas programadas para gramáticas livres de contexto adaptativas

Aplicando aqui a demonstração construtiva utilizada na prova do lema 3.2.6 para "traduzir" os exemplos de gramáticas livres de contexto matriciais com verificação de aparência para gramáticas livres de contexto adaptativas com verificação de aparência.

Exemplo 3.2.9 Gramática programada $G_7 = (G, s, f)$ para a linguagem $L = \{a^n b^n c^n : n \geq 1\}$. *Consideremos a seguinte gramática livre de contexto $G_7 = (\{S, A, B, C\}, \{a, b, c\}, P, S)$, cujo conjunto de produções e funções de sucesso e falha são definidos como segue:*

P	$s(p)$	$f(p)$
$p_1 = S \to ABC$	$\{p_2, p_5\}$	$\emptyset$
$p_2 = A \to aA$	$\{p_3\}$	$\emptyset$
$p_3 = B \to bB$	$\{p_4\}$	$\emptyset$
$p_4 = C \to cC$	$\{p_2, p_5\}$	$\emptyset$
$p_5 = A \to a$	$\{p_6\}$	$\emptyset$
$p_6 = B \to b$	$\{p_7\}$	$\emptyset$
$p_7 = C \to c$	$\emptyset$	$\emptyset$

Aplicando o método de demonstração do lema 3.2.8 à gramática livre de contexto programada G_7 do exemplo 2.3.1 da seção 2.2, obtemos a função adaptativa:

$$\mathcal{A}(p) = \begin{cases} \{p_2, p_5\} & se\ p = p_1, \\ \{p_3\} & se\ p = p_2, \\ \{p_4\} & se\ p = p_3, \\ \{p_2, p_5\} & se\ p = p_4, \\ \{p_6\} & se\ p = p_5, \\ \{p_7\} & se\ p = p_6, \\ \emptyset & se\ p = p_7. \end{cases}$$

Definimos o conjunto inicial de produções como:

$$P_0 = \{[p_1] = S \to \{\mathcal{A}()\}ABC\}$$

Nestas condições, a gramática livre de contexto adaptativa $GA = (G, [P_0], \emptyset, \mathcal{A})$ gera a linguagem dependente de contexto: $L = \{a^n b^n c^n : n \geq 1\}$.

Derivação de $a^3b^3c^3$:

Com efeito:

i	*Passo i da derivação*	*Produção aplicada*	$[P_i]$
0	S		$\{[p_1]\}$
1	$\Rightarrow_{p_1} ABC$	$p_1 = S \rightarrow \{\mathcal{A}()\}ABC$	$\{[p_2], [p_5]\}$
2	$\Rightarrow_{p_2} aABC$	$p_2 = A \rightarrow \{\mathcal{A}()\}aA$	$\{[p_3]\}$
3	$\Rightarrow_{p_3} aAbBC$	$p_3 = B \rightarrow \{\mathcal{A}()\}bB$	$\{[p_4]\}$
4	$\Rightarrow_{p_4} aAbBcC$	$p_4 = C \rightarrow \{\mathcal{A}()\}cC$	$\{[p_2], [p_5]\}$
5	$\Rightarrow_{p_2} aaAbBcC$	$p_2 = A \rightarrow \{\mathcal{A}()\}aA$	$\{[p_3]\}$
6	$\Rightarrow_{p_3} aaAbbBcC$	$p_3 = B \rightarrow \{\mathcal{A}()\}bB$	$\{[p_4]\}$
7	$\Rightarrow_{p_4} aaAbbBccC$	$p_4 = C \rightarrow \{\mathcal{A}()\}cC$	$\{[p_2], [p_5]\}$
8	$\Rightarrow_{p_5} a^3bbBccC$	$p_5 = A \rightarrow \{\mathcal{A}()\}a$	$\{[p_6]\}$
9	$\Rightarrow_{p_6} a^3b^3ccC$	$p_6 = B \rightarrow \{\mathcal{A}()\}b$	$\{[p_7]\}$
10	$\Rightarrow_{p_7} a^3b^3c^3$	$p_7 = C \rightarrow \{\mathcal{A}()\}c$	$\emptyset$

Exemplo 3.2.10 Gramática programada $G_8 = (G, s, f)$ para a linguagem $L = \{ww : w \in \{a, b\}^+\}$. *Consideremos a seguinte gramática livre de contexto programada $G_8 = (\{S, A, B\}, \{a, b\}, P, S)$, cujo conjunto de produções e funções de sucesso e falha são definidos como segue:*

P	$s(p)$	$f(p)$
$p_1 = S \to AB$	$\{p_2, p_3, p_6, p_7\}$	$\emptyset$
$p_2 = A \to aA$	$\{p_4\}$	$\emptyset$
$p_3 = A \to bA$	$\{p_5\}$	$\emptyset$
$p_4 = B \to aB$	$\{p_2, p_3, p_6, p_7\}$	$\emptyset$
$p_5 = B \to bB$	$\{p_2, p_3, p_6, p_7\}$	$\emptyset$
$p_6 = A \to a$	$\{p_8\}$	$\emptyset$
$p_7 = A \to b$	$\{p_9\}$	$\emptyset$
$p_8 = B \to a$	$\{p_1\}$	$\emptyset$
$p_9 = B \to b$	$\{p_1\}$	$\emptyset$

Neste caso, como no exemplo anterior, não existem produções a serem aplicadas em modo de verificação de aparência.

Aplicando o método de demonstração do lema 3.2.8 à gramática livre de contexto programada G_8 do exemplo 2.3.2 da seção 2.3, obtemos a função adaptativa seguinte:

$$\mathcal{A}(p) = \begin{cases} \{p_2, p_3, p_6, p_7\} & se\ p = p_1 \\ \{p_4\} & se\ p = p_2 \\ \{p_5\} & se\ p = p_3 \\ \{p_2, p_3, p_6, p_7\} & se\ p = p_4 \\ \{p_2, p_3, p_6, p_7\} & se\ p = p_5 \\ \{p_8\} & se\ p = p_6 \\ \{p_9\} & se\ p = p_7 \\ \{p_1\} & se\ p = p_8 \\ \{p_1\} & se\ p = p_9 \end{cases}$$

Definimos o conjunto inicial de produções como:

$$P_0 = \{[p_1] = S \to \{\mathcal{A}()\}AB\}$$

Nessas condições, a gramática livre de contexto adaptativa $GA = (G, [P_0], \emptyset, \mathcal{A})$ gera a linguagem dependente de contexto: $L = \{ww : w \in \{a, b\}^*\}$.

Derivação de bbaabbbaab:

Com efeito:

i	*Passo i da derivação*	*Produção aplicada no passo i*	$[P_i]$
0	S		$\{[p_1]\}$
1	$\Rightarrow_{p_1} AB$	$p_1 = S \rightarrow \{\mathcal{A}()\}AB$	$\{[p_2], [p_3], [p_6], [p_7]\}$
2	$\Rightarrow_{p_3} bAB$	$p_3 = A \rightarrow \{\mathcal{A}()\}bA$	$\{[p_5]\}$
3	$\Rightarrow_{p_5} bAbB$	$p_5 = B \rightarrow \{\mathcal{A}()\}bB$	$\{[p_2], [p_3], [p_6], [p_7]\}$
4	$\Rightarrow_{p_3} bbAbB$	$p_3 = A \rightarrow \{\mathcal{A}()\}bA$	$\{[p_5]\}$
5	$\Rightarrow_{p_5} bbAbbB$	$p_5 = B \rightarrow \{\mathcal{A}()\}bB$	$\{[p_2], [p_3], [p_6], [p_7]\}$
6	$\Rightarrow_{p_2} bbaAbB$	$p_2 = A \rightarrow \{\mathcal{A}()\}aA$	$\{[p_4]\}$
7	$\Rightarrow_{p_4} bbaAbbaB$	$p_4 = B \rightarrow \{\mathcal{A}()\}aB$	$\{[p_2], [p_3], [p_6], [p_7]\}$
8	$\Rightarrow_{p_2} bbaaAbbaaB$	$p_2 = A \rightarrow \{\mathcal{A}()\}aA$	$\{[p_4]\}$
9	$\Rightarrow_{p_4} bbaaAbbaaB$	$p_4 = B \rightarrow \{\mathcal{A}()\}aB$	$\{[p_2], [p_3], [p_6], [p_7]\}$
10	$\Rightarrow_{p_7} bbaabbbaaB$	$p_7 = A \rightarrow \{\mathcal{A}()\}b$	$\{[p_9]\}$
11	$\Rightarrow_{p_9} bbaabbbaab$	$p_9 = B \rightarrow \{\mathcal{A}()\}b$	$\{[p_1]\}$

Exemplo 3.2.11 Gramática programada $G_9 = (G, s, f)$ para a linguagem $L = \{a^{2^n} : n \geq 0\}$. *Consideremos a seguinte gramática livre de contexto programada $G_9 = (\{S, A\}, \{a\}, P, S)$, cujo o conjunto de produções e funções de sucesso e falha são definidos como segue:*

P	$s(p)$	$f(p)$
$p_1 = S \to AA$	$\{p_1\}$	$\{p_2\}$
$p_2 = A \to S$	$\{p_2\}$	$\{p_1, p_3\}$
$p_3 = S \to a$	$\{p_3\}$	$\emptyset$.

Aplicando o método de demonstração do lema 3.2.8 à gramática livre de contexto programada G_9 do exemplo 2.3.3 da seção 2.3, obtemos as novas funções de sucesso e falha:

P	$s_1(p)$	$f_1(p)$
p_1	$\{p_1, q_1\}$	$\emptyset$
p_2	$\{p_2, q_2\}$	$\emptyset$
p_3	$\{p_3, q_3\}$	$\emptyset$.
q_1	$\emptyset$	$\{p_2, q_2\}$
q_2	$\emptyset$	$\{p_1, p_3, q_1, q_3\}$
q_3	$\emptyset$	$\emptyset$.

O conjunto de produções aplicáveis em modo de verificação de aparência é

$$F = \{q_1 = S \to AA, q_2 = A \to S, q_3 = S \to a\}$$

Seguindo o método utilizado na demonstração do lema 3.2.8, definimos a função adaptativa:

$$\mathcal{A}(p) = \begin{cases} \{p_1, q_1\}, & p = p_1, \\ \{p_2, q_2\}, & p = p_2, \\ \{p_3, q_3\}, & p = p_3, \\ \{p_2, q_2\}, & p = q_1, \\ \{p_1, p_3, q_1, q_3\}, & p = q_2, \\ \emptyset & p = q_3. \end{cases}$$

O conjunto de produções iniciais é:

$$P_0 = \{[p_1] = S \to \{\mathcal{A}()\}AA, [p_3] = S \to \{\mathcal{A}()\}a\}$$

Nessas condições, a gramática livre de contexto adaptativa $GA = (G, [P_0], \emptyset, \mathcal{A})$ gera a linguagem $L = \{a^{2^n} : n \geq 0\}$.

Derivação de a^{2^3}:

i	*Passo i da derivação*	*Produção aplicada no passo i*	$[P_i]$
0	S		$\{[p_1]\}$
1	$\Rightarrow^{1}_{p_1} AA$	$p_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_1], [q_1]\}$
2	$\Rightarrow^{ac^{2}}_{q_1} AA$	$q_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_2], [q_2]\}$
3	$\Rightarrow^{3}_{p_2} SA$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
4	$\Rightarrow^{4}_{p_2} SS$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
5	$\Rightarrow^{ac^{5}}_{q_2} SS$	$q_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_1], [p_3], [q_1], [q_3]\}$
6	$\Rightarrow^{6}_{p_1} AAS$	$p_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_1], [q_1]\}$
7	$\Rightarrow^{7}_{p_1} AAAA$	$p_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_1], [q_1]\}$
8	$\Rightarrow^{ac^{8}}_{q_1} AAAA$	$q_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_2], [q_2]\}$
9	$\Rightarrow^{9}_{p_2} ASAA$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
10	$\Rightarrow^{10}_{p_2} ASAS$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
11	$\Rightarrow^{11}_{p_2} ASSS$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
12	$\Rightarrow^{12}_{p_2} SSSS$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
13	$\Rightarrow^{ac^{13}}_{q_2} SSSS$	$q_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_1], [p_3], [q_1], [q_3]\}$
14	$\Rightarrow^{14}_{p_1} AASSS$	$p_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_1], [q_1]\}$
15	$\Rightarrow^{15}_{p_1} AAAASS$	$p_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_1], [q_1]\}$
16	$\Rightarrow^{16}_{p_1} AAAAAAS$	$p_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_1], [q_1]\}$
17	$\Rightarrow^{17}_{p_1} AAAAAAAA$	$p_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_1], [q_1]\}$
18	$\Rightarrow^{ac^{18}}_{q_1} AAAAAAAA$	$q_1 = S \rightarrow \{\mathcal{A}()\}AA$	$\{[p_2], [q_2]\}$
19	$\Rightarrow^{18}_{p_2} SAAAAAAA$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
20	$\Rightarrow^{18}_{p_2} SSAAAAAA$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
21	$\Rightarrow^{18}_{p_2} SSSAAAAA$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
22	$\Rightarrow^{18}_{p_2} SSSSAAAA$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
23	$\Rightarrow^{18}_{p_2} SSSSSAAA$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
24	$\Rightarrow^{18}_{p_2} SSSSSSAA$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
25	$\Rightarrow^{18}_{p_2} SSSSSSSA$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
26	$\Rightarrow^{18}_{p_2} SSSSSSSS$	$p_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_2], [q_2]\}$
27	$\Rightarrow^{ac^{19}}_{q_2} SSSSSSSS$	$q_2 = A \rightarrow \{\mathcal{A}()\}S$	$\{[p_1], [p_3], [q_1], [q_3]\}$
28	$\Rightarrow^{19}_{p_3} aSSSSSSS$	$p_3 = S \rightarrow \{\mathcal{A}()\}a$	$\{[p_1], [p_3], [q_1], [q_3]\}$
29	$\Rightarrow^{14}_{p_3} aaSSSSSS$	$p_3 = S \rightarrow \{\mathcal{A}()\}a$	$\{[p_3], [q_3]\}$
30	$\Rightarrow^{15}_{p_3} aaaSSSSS$	$p_3 = S \rightarrow \{\mathcal{A}()\}a$	$\{[p_3], [q_3]\}$
31	$\Rightarrow^{16}_{p_3} aaaaSSSS$	$p_3 = S \rightarrow \{\mathcal{A}()\}a$	$\{[p_3], [q_3]\}$
32	$\Rightarrow^{17}_{p_3} aaaaaSSS$	$p_3 = S \rightarrow \{\mathcal{A}()\}a$	$\{[p_3], [q_3]\}$
33	$\Rightarrow^{15}_{p_3} aaaaaaSS$	$p_3 = S \rightarrow \{\mathcal{A}()\}a$	$\{[p_3], [q_3]\}$
34	$\Rightarrow^{15}_{p_3} aaaaaaaS$	$p_3 = S \rightarrow \{\mathcal{A}()\}a$	$\{[p_3], [q_3]\}$
35	$\Rightarrow^{16}_{p_3} aaaaaaaa$	$p_3 = S \rightarrow \{\mathcal{A}()\}a$	$\{[p_3], [q_3]\}$

3.3 Resumo

Neste capítulo provamos, por construção, que as gramáticas livres de contexto adaptativas com verificação de aparência permitem simular a operação das gramáticas livres de contexto com mecanismos de controle e verificação de aparência apresentadas no capítulo 2. Desse modo, provamos que o formalismo proposto neste capítulo tem poder de máquina de Turing, e determinamos formas para sua conversão entre os diversos formalismos.

Mais ainda, provamos, também por construção, que as gramáticas livres de contexto adaptativas com verificação de aparência podem ser simuladas pelas gramáticas livres de contexto com linguagem de controle regular e com verificação de aparência e, assim, podemos intercambiar qualquer par destes cinco formalismos na representação de uma linguagem.

Vamos fazer uso dessa equivalência no capítulo 4, em que apresentamos um método de construção de um analisador sintático não-determinístico para linguagens dependente de contexto obtido a partir gramáticas livres de contexto com linguagem de controle regular e com verificação de aparência.

Capítulo 4

Analisadores para gramáticas controladas

Neste capítulo, discutimos a infraestrutura necessária para obter reconhecedores *ascendentes* e *descendentes* para as gramáticas com mecanismos de controle estudadas nos capítulos 2 e para as gramáticas livres de contexto adaptativas, estudadas no capítulo 3. Para a geração desses analisadores, baseamos-nos nos teoremas 2.4.1 e 3.2.10 que garantem que as gramáticas livres de contexto com linguagem de controle regular e verificação de aparência podem simular qualquer um dos outros quatro dispositivos estudados nesta tese. Por este motivo, nos limitados a estudar a obteção de analisadores para essas gramáticas.

Na seção 4.1 apresentamos, um analisador *ascendente baseado em autômatos-pilha*, que opera em forma não-determinística, para as gramáticas livres de contexto com linguagem de controle regular[5]. Para a implementação dessa ferramenta, é necessário obter primeiro um autômato finito a partir da expressão regular que descreve a linguagem de controle de uma gramática livre de contexto com linguagem de controle regular e verificação de aparência. Para esse fim usamos, na seção 4.2, o algoritmo *wirth2ape*[29], (descrito no apêndice e que serve para obter autômatos de pilha estruturados a partir de gramáticas livres de contexto expressadas em notação de Wirth) para obter autômatos finitos, requeridos para a construção do analisador, a partir de expressões regulares escritas em notação de Wirth.

Finalmente, na seção 4.3 discutimos, como obter *analisadores descendentes determinísticos baseados em autômatos finitos adaptativos* a partir de gramáticas livres de contexto com linguagem de controle regular e verificação de aparência. A construção desses analisadores somente considera o caso em das gramáticas livres de contexto com linguagem de controle regular com conjunto de produções tal que o único não-terminal que produz uma seqüência de não-terminais é o símbolo inicial

da gramática. Novamente, pelo teorema 3.2.10, essa discussão abre o caminho para obter analisadores baseados em autômatos adaptativos a partir de gramáticas livres de contexto adaptativas com verificação de aparência.

4.1 Analisadores ascendentes controlados

No final dos anos sessenta, alguns esforços foram feitos para usar autômatos-pilha (**SA**, pelas suas siglas em inglês *Stack Automata*) como fundamento teórico e prático para projeto e implementação de compiladores.

Diferentemente dos autômatos de pilha (em inglês, *pushdown automata*) essas máquinas possuem uma pilha que pode ser acessada e alterada aleatoriamente. Os **SA** foram introduzidos por Ginsburg e Greibach[30], e os autores mostram como usá-los para realizar verificação de tipos. Knuth e Bigelow[31] projetaram uma linguagem de programação para **SA** e Harrison e Schkolnik[32] desenvolveram uma caraterização gramatical dos **SA**.

Nesta seção, derivamos um autômato-pilha para gramáticas com conjunto de controle regular; modificando ligeramente a operação dessa máquina obtemos um analisador ascendente que opera fazendo reduções na forma sentencial corrente, trocando o lado direito das produções da gramática pelo correspondente lado esquerdo. Como esses autômatos reconhecem linguagens dependentes ao contexto[33], obtemos analisadores dependentes de contexto.

4.1.1 Introdução

Construímos analisadores para linguagens dependentes ao contexto seguindo os três passos seguintes:

1. Projetar uma gramática livre de contexto com conjunto de controle regular R para a linguagem-objetivo.
2. Construir o autômato finito FA correspondente a uma expressão regular que define o conjunto R.
3. Obter, a partir do autômato finito FA, o correspondente 1-**SA** definido nesta seção.

4.1.2 Os analisadores

Autômatos-pilha somente de leitura 1-**SA**, os quais podem escrever em qualquer lugar da pilha mas somente podem ler a entrada uma vez, são definidos por Ginsburg

e Greibach[34], página 391; nós vamos modificar sua operação impondo as seguintes restrições:

1. **pré-processamento:** Empilhar toda a cadeia de entrada,
2. **processamento:** Examinar, de forma não-determinística, o conteúdo da pilha, desde o topo até o fundo, para determinar uma produção a ser aplicada; se uma tal produção for encontrada, realizar a transição e, depois disso, posicionar o apontador da pilha para o topo da pilha,
3. **condição de parada:** Parar aceitando caso a pilha estiver vazia e o 1-**SA** se encontre num estado final.

Definição 4.1.1 *Um 1-***SA** *é uma sétupla* $SA = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$*, em que:*

1. Q *é um conjunto finito não-vazio de estados,*
2. Σ *é o alfabeto de entrada,*
3. $\Gamma \cup \Sigma$ *é o alfabeto da pilha,*
4. δ *é uma função de* $Q \times \Sigma \times \Gamma$ *em* $2^{Q \times \{-1,0,1\} \times \Gamma^*}$*, que satisfaz a seguinte propriedade: Se* $(p, e, w) \in \delta(q, a, Z)$ *e* $w \neq Z$*, então* $e = 0$.
5. $q_0 \in Q$ *é o estado inicial,*
6. $Z_0 \in \Gamma \setminus \Sigma$ *é o símbolo inicial da pilha,*
7. $F \subseteq Q$ *é o conjunto de estados finais,*

A interpretação de $(p, e, w) \in \delta(q, a, Z)$ é a seguinte: Suponha que um 1-**SA** SA se encontra no estado q, lendo a na fita de entrada e tendo Z na posição apontada pelo apontador da pilha. Embora possa haver outras escolhas não-determinísticas, o autômato SA pode executar a seguinte seqüência de operações:

(i) ir ao estado p,

(ii) mover o cabeçote de leitura da cadeia de entrada para direita se $a \neq \lambda$, ou mantê-lo na mesma posição, em caso contrário,

(iii) deslocar o cabeçote de leitura da pilha de uma posição em direção ao fundo da pilha se $e = -1$, deslocá-lo de uma posição em direção ao topo se $e = 1$ e mantê-lo na mesma posição se $e = 0$.

A interpretação de $(p, 0, w) \in \delta(q, a, Z)$ é a seguinte: Se SA estiver na configuração (q, a, Z), então SA pode executar a seguinte seqüência de operações:

(i) deslocar a cabeça de leitura na fita de entrada, consumindo a se $a \neq \lambda$,

(ii) ir ao estado p,

(iii) substituir Z por w na pilha.

Para outras definições e descrições das operações do 1-**SA** remetemos o leitor à referência original[34].

4.1.3 Notação abreviada

A fim de de poupar espaço nos exemplos seguintes vamos escrever $(p, w) \in \delta(q, a, Z)$ sempre e quando se verifique a propriedade no item *4.* da definição 4.3.1, i.e. $(p, 0, w) \in \delta(q, a, Z)$. Veja a Fig. 4.1:

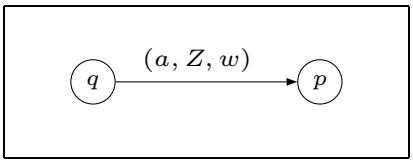


Figura 4.1: Representação gráfica para $(p, 0, w) \in \delta(q, a, Z)$.

4.1.4 Construindo os analisadores

Seja $GR = (G, R, F)$ uma gramática livre de contexto com conjunto de controle regular, em que $G = (N, T, P, S)$ é uma gramática livre de contexto e $F \subset P$ e R é um conjunto regular sobre P. Pode-se observar que o conteúdo da pilha do 1-**SA** será formado pelas formas sentenciais geradas por G.

1. Obter, usando o algoritmo *wirth2ape*, um autômato determinístico completamente especificado correspondente a uma expressão regular que define R. Recordemos, da seção 2.4, que o alfabeto de entrada para esse autômato será $\{E/D : E \to D \in P\}$.

2. Inverter a orientação de todas as transições e inverter os papéis entre os estados finais e iniciais. Converter o autômato resultante em um autômato finito determinístico.

3. Para cada transição $p \in \delta(q, E/D)$ no autômato, incluir uma transição $(p, 0, E)$ ao conjunto que define $\delta(q, \lambda, (D)^R)$ no 1-**SA**,

4. Fazer que todos os estados iniciais do autômato finito determinístico (item "2" acima) sejam estados finais do 1-**SA**,

5. Acrescentar um novo estado ao 1-**SA** e criar uma transição-λ partindo deste estado para o estado inicial do autômato finito determinístico.

4.1.5 Navegação na pilha

Até agora, o 1-**SA** construído ainda não pode movimentar o cabeçote de de leitura da pilha. Isso pode ser obtido adicionando-se um par de laços para cada transição já existente no 1-**SA**. Essa manipulação pode certamente introduzir não-determinismos e também polui a notação gráfica; portanto, cada vez que que a Fig. 4.1 aparecer em nossa representação gráfica, ela deverá ser interpretada como mostra a Fig 4.2. Assim, duas transições consecutivas subentendem um estado intermediario "oculto", que faz ao mesmo tempo o papel dos estados "s" e "r". As quádruplas $(\lambda, -1, W, W)$ e $(\lambda, 1, W, W)$ significam $(r, -1, W) \in \delta(r, \lambda, W)$, e $(p, 1, W) \in \delta(p, \lambda, W)$ respectivamente. Por outro lado, em em outras transições, as referências aos estados q, p devem ser trocadas por referências aos estados r, s respectivamente:

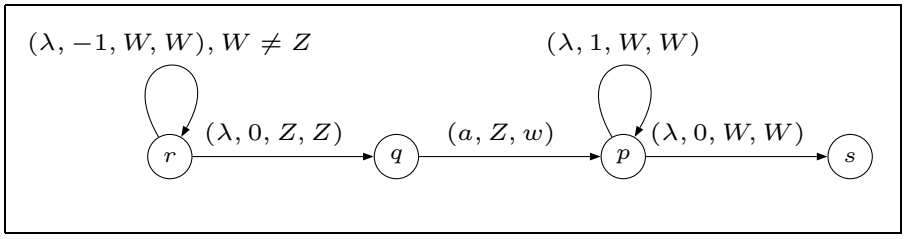


Figura 4.2: Interpretação para $(p, 0, w) \in \delta(q, a, Z)$ da Fig. 4.1.

4.1.6 Construção das árvores de análise

A construção da árvore de derivação para uma palavra reconhecida pelas máquinas descritas na seção 4.3 baseia-se nas seguintes observações: Já que a cadeia de entrada é completamente empilhada antes do início do análise, seu comprimento é conhecido; adicionalmente, o comprimento do lado direito de cada produção é conhecido; levando-se em consideração essas duas informações, é possível conhecer o tamanho da pilha em qualquer momento da análise e, como a busca de uma produção aplicável sempre tem início no topo da pilha, é sempre possível determinar o ponto exato de aplicação da produção correspondente.

Vamos incluir essas informações nas tabelas de análise dos três exemplos a seguir; nesses exemplos usamos a mesma estratégia empregada na subseção 4.3.4 para obter os analisadores correspondentes aos exemplos 2.4.1, 2.4.2 e 2.4.3. Em todos os casos, apresentamos o autômato-pilha, que aceita a linguagem gerada pela gramática do exemplo correspondente da seção 2.4, uma tabela descrevendo a análise de uma palavra dessa linguagem, e a correspondente árvore de análise para essa palavra. Nas árvores de análise, etiquetamos as arestas com as produções reduzidas, indicando com superindices o instante de cada redução.

Exemplo 4.1.1 Analisador para a linguagem do triplo balanceamento, reconhecendo a palavra $a^3b^3c^3$.

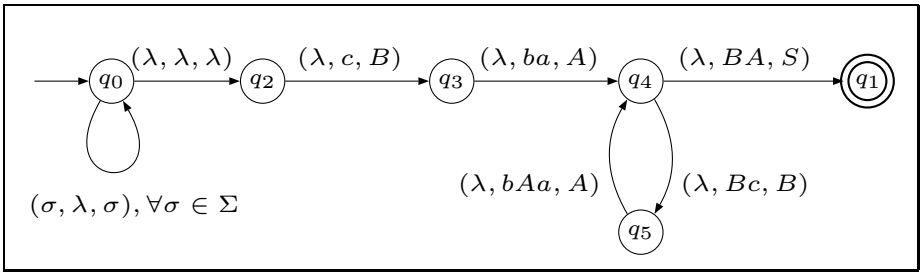


Figura 4.3: Autômato-Pilha que aceita $a^nb^nc^n$

Acima, o analisador que aceita $a^nb^nc^n$; à direita a tabela de análise para $a^3b^3c^3$, e embaixo, a correspondente árvore de análise.

time	0	1	2	3	4	5	6	7
	c	B						
	c	c	B					
	c	c	c	B				
	b	b	c	c				
	b	b	b	b	B			
	b	b	b	b	c	B		
	a	a	A	A	b	b		
	a	a	a	a	A	A	B	
	a	a	a	a	a	a	A	S
estado anterior	q_0	q_2	q_3	q_4	q_5	q_4	q_5	q_4
produção reduzida	$-$	p_5	p_4	p_3	p_2	p_3	p_2	p_1
estado corrente	q_2	q_3	q_4	q_5	q_4	q_5	q_4	q_1
tamanho da pilha	9	9	4	3	2	3	2	1
ponto de aplicação	$-$	9	4	3	2	3	2	1

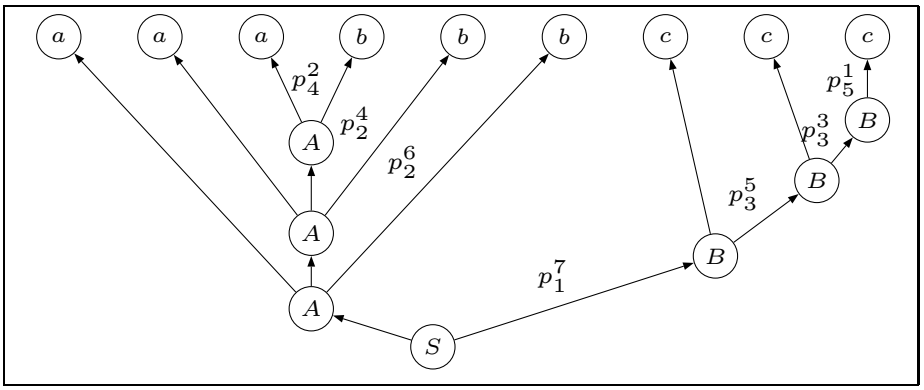


Figura 4.4: Árvore de derivação para $a^3b^3c^3$

Exemplo 4.1.2 Analisador para a linguagem cópia, reconhecendo *abbabb.*

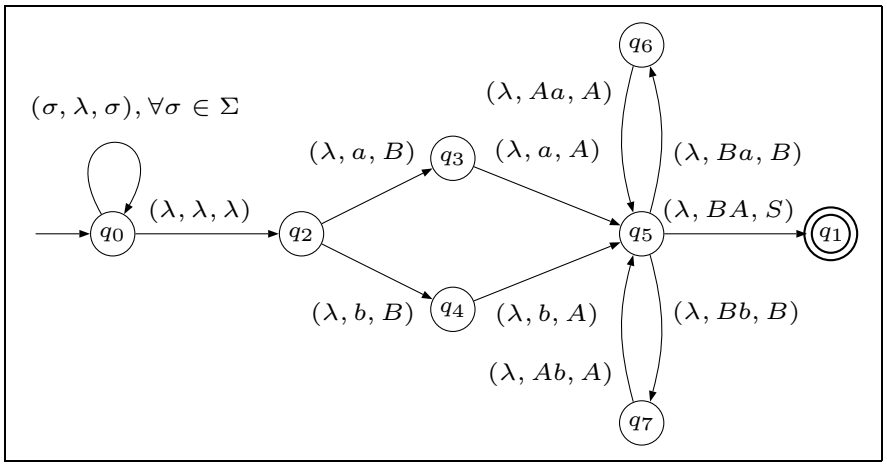


Figura 4.5: 1-*SA* que aceita ww

Acima, o analisador que aceita $ww, w \in \{a, b\}^*$. À direita, a tabela de de análise para abbabb; embaixo, a árvore de derivação. Observar que acontecem escolhas não-determinísticas na pilha nos instantes $t = 1$ e $t = 2$ da análise.

time	0	1	2	3	4	5	6	7
	b	B	B					
	b	b	b	B				
	a	a	a	a	B			
	b	b	A	A	a	B		
	b	b	b	b	A	A	B	
	a	a	a	a	a	a	A	S
estado anterior	q_0	q_2	q_4	q_5	q_7	q_5	q_6	q_5
produção reduzida	$-$	p_9	p_5	p_7	p_3	p_6	p_2	p_1
estado corrente	q_2	q_4	q_5	q_7	q_5	q_6	q_5	q_1
não-determinismo		$*$	$*$					
tamanho da pilha	6	6	6	5	4	3	2	1
ponto de aplicação	$-$	6	3	5	3	2	1	

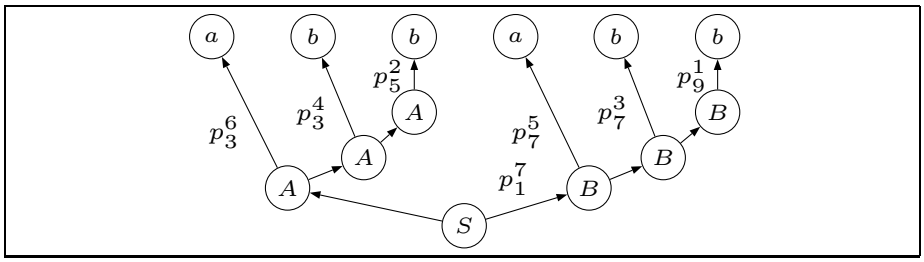


Figura 4.6: Árvore de derivação para *abbabb*

Exemplo 4.1.3 Analisador para a linguagem potência, reconhecendo a palavra a^{2^3}. *Este exemplo apresenta a operação do analisador correspondente à gramática livre de contexto com linguagem de controle regular com verificação de aparência do exemplo* 2.4.3*, e o correspondente autômato.*

O analisador resultante é altamente não-determinístico: quase todas as produções são reduzidas por escolhas não determinísticas na pilha. Para não poluir as tabelas que descrevem a análise, optamos neste caso por marcar apenas as escolhas determinísticas na pilha. Existem apenas 7 dessas escolhas na análise da palavra a^{2^3}, que tem 35 passos.

Neste exemplo, quando se realiza uma redução segundo alguma produção marcada para ser aplicada em modo de verificação de aparência, a forma sentencial não se modifica. Isso faz com que sejam duplicados alguns nós na árvore de análise; entre as duplicatas, na árvore de análise, colocamos uma aresta etiquetada com a produção aplicada, com um superíndice que indica o passo de aplicação e um asterisco que denota que essa produção foi aplicada em modo de verificação de aparência.

As produções p_4 e p_2 são reduzidas em modo de verificação de aparência nos instantes $t = 9$, $t = 18$, $t = 23$, $t = 28$, $t = 31$, $t = 34$, da análise.

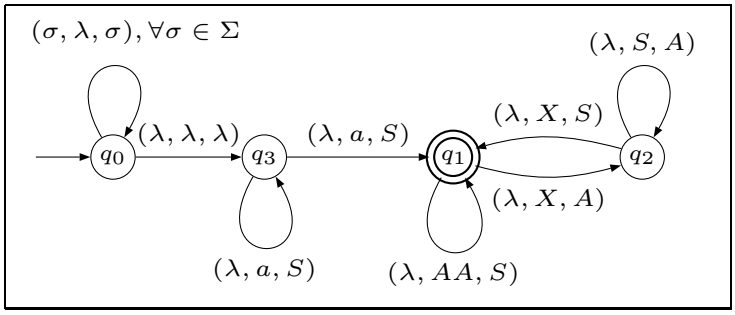


Figura 4.7: 1-SA que aceita a^{2^n}

Na tabela a seguir, temos os primeiros doze passos da análise e a primeira redução executada na pilha em modo de verificação de aparência, segundo a produção $p_4 = A \to X$, no instante $t = 9$ da análise:

tempo	0	1	2	3	4	5	6	7	8	9	10	11	12
	a	S	S	S	S	S	S	S	S	S	A	A	A
	a	a	S	S	S	S	S	S	S	S	S	A	A
	a	a	a	S	S	S	S	S	S	S	S	S	A
	a	a	a	a	S	S	S	S	S	S	S	S	S
	a	a	a	a	a	S	S	S	S	S	S	S	S
	a	a	a	a	a	a	S	S	S	S	S	S	S
	a	a	a	a	a	a	a	S	S	S	S	S	S
	a	a	a	a	a	a	a	a	S	S	S	S	S
verificação de aparência										$*$			
estado anterior	q_0	q_2	q_2	q_2	q_2	q_2	q_2	q_2	q_2	q_2	q_1	q_1	q_1
produção reduzida	$-$	p_5	p_5	p_5	p_5	p_5	p_5	p_5	p_5	p_4	p_3	p_3	p_3
estado corrente	q_2	q_2	q_2	q_2	q_2	q_2	q_2	q_2	q_2	q_1	q_1	q_1	q_1
determinismo									$*$				
tamanho atual da pilha	$-$	8	8	8	8	8	8	8	8	8	8	8	8
ponto de aplicação	$-$	8	7	6	5	4	3	2	1	$-$	8	7	6

Na tabela a seguir temos os doze passos seguintes da análise e e duas reduções feitas na pilha em modo de verificação de aparência, segundo as produções $p_2 = S \to X$ *e* $p_4 = A \to X$ *nos instantes* $t = 18$ *e* $t = 23$ *da análise, respectivamente:*

tempo	13	14	15	16	17	18	19	20	21	22	23	24
	A	A	A	A	A	A						
	A	A	A	A	A	A	S					
	A	A	A	A	A	A	A	S				
	A	A	A	A	A	A	A	S	S			
	S	A	A	A	A	A	A	A	S	S	S	A
	S	S	A	A	A	A	A	A	S	S	S	S
	S	S	S	A	A	A	A	A	A	S	S	S
	S	S	S	S	A	A	A	A	A	S	S	S
verificação de aparência						$*$					$*$	
estado anterior	q_1	q_1	q_1	q_1	q_1	q_1	q_2	q_2	q_2	q_2	q_2	q_1
produção reduzida	p_3	p_3	p_3	p_3	p_3	p_2	p_1	p_1	p_1	p_1	p_4	p_3
estado corrente	q_1	q_1	q_1	q_1	q_1	q_2	q_2	q_2	q_2	q_2	q_1	q_1
determinismo					$*$					$*$		
tamanho atual da pilha	8	8	8	8	8	8	7	7	5	4	4	4
ponto de aplicação	5	4	3	2	1	$-$	7	5	3	1	$-$	4

Na tabela a seguir temos três reduções das produções p_2, p_4 *e* p_2 *em modo de verificação de aparência nos instantes* $t = 28$, $t = 31$ *e* $t = 34$, *respectivamente:*

tempo	25	26	27	28	29	30	31	32	33	34	35
	A	A	A	A							
	A	A	A	A	S						
	S	A	A	A	A	S	S	A	A	A	
	S	S	A	A	A	S	S	S	A	A	S
verificação de aparência				$*$			$*$			$*$	
estado anterior	q_1	q_1	q_1	q_1	q_1	q_1	q_1	q_2	q_2	q_2	q_2
produção reduzida	p_3	p_3	p_3	p_2	p_1	p_1	p_4	p_3	p_3	p_2	p_1
estado corrente	q_1	q_1	q_1	q_2	q_1	q_1	q_2	q_2	q_2	q_2	q_2
determinismo			$*$			$*$			$*$		$*$
tamanho atual da pilha	4	4	4	4	3	2	2	2	2	2	1
ponto de aplicação	3	2	1	–	3	1	–	2	1	1	1

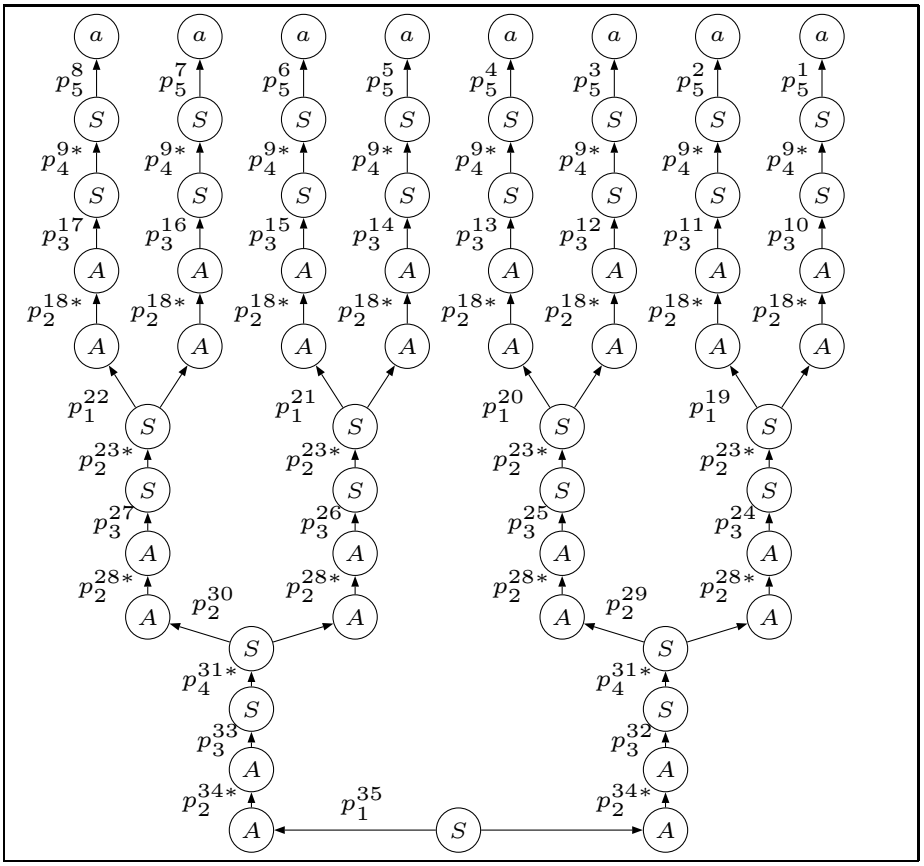


Figura 4.8: Árvore de derivação para a^{2^3}

4.1.7 Considerações sobre complexidade

Complexidade de tempo e espaço para obter o 1-SA

Construir um autômato finito não determinístico para uma expressão regular dada gasta tempo e espaço lineares em relação ao comprimento da expressão regular, mas transformá-lo em um autômato finito determinístico pode ocupar, no pior caso, tempo e espaço exponenciais em relação ao comprimento da expressão regular.

Complexidade de espaço da operação do 1-SA

Na sua versão não-determinística, os 1-**SA**, conforme especificados na seção 4.3.2 ocupam uma quantidade de espaço linear em relação ao comprimento n da cadeia de entrada: n posições para armazenar a cadeia na fita de leitura e n posições para empilhar a cadeia na pilha; dessa forma os 1-**SA** coincidem com os *autômatos linearmente limitados*, no sentido de utilizar somente quantidade de espaço que é um múltiplo do comprimento da cadeia de entrada, e esse resultado é coerente com o fato de os 1-**SA** aceitarem linguagens dependentes de contexto[33]. Assim, a complexidade de espaço dos analisadores baseados em 1-**SA** não-determinísticos é linear, isto é $O(n)$, em relação ao comprimento da entrada de dados.

Complexidade de tempo da operação do 1-SA

Cada vez que realiza uma redução, o 1-**SA** percorre a pilha, em busca de uma posição para executar alguma transição, realizar a redução correspondente e retornar o cabeçote de leitura da pilha para que aponte novamente para o topo da pilha. No pior caso, cada vez que realiza uma redução, o cabeçote de leitura da pilha pode atingir o fundo da pilha, realizando um total de $2n$ movimentos, para uma pilha de tamanho n. Asumindo que o analisador não têm produções a serem reduzidas em *modo de verificação de aparência*, podemos obter um limitante inferior para a complexidade de tempo da operação do 1-**SA** da seguinte forma: Desconsideramos as reduções que trocam um símbolo por outro, pois elas incrementam o tempo de operação e estamos interesados num limitante inferior; portanto, assumimos que, a cada redução, é removido um símbolo da forma sentencial; se começãmos com uma palavra de comprimento n a que a complexidade de tempo é limitada inferiormente por: $2(n + (n-1) + (n-2) + \ldots + 2 + 1) = \Omega(n^2)$. Porém, no caso de existirem produções a serem reduzidas em modo de verificação de aparência (exemplo 4.3.3), pode acontecer que muitas das reduções do lado direito de uma produção para o co-rrespondente lado esquerdo *não diminuam o tamanho da forma sentencial*.

4.2 O algoritmo *wirth2ape*

No apêndice A é descrito o algoritmo *wirth2ape*[29], que usaremos nesta seção para obter os autômatos finitos que são a base dos analisadores dependentes de contexto descritos na seção 4.3. Trata-se de um algoritmo para transformar gramáticas livres de contexto, expressas em notação de Wirth[35], em *Autômatos de Pilha Estruturados*[9]. Em particular, se a grámatica em questão for uma gramática regular que não seja livre de contexto, então o algoritmo *wirth2ape* a transforma em um autômato finito não-determinístico. Nas subseções a seguir discutiremos como retirar, do autômato obtido, algumas transições-λ, eliminando, assim, alguns desses não-determinismos, e diminuindo a quantidade de estados do autômato.

4.2.1 Colapsando estados

O algoritmo *wirth2ape* produz diversas transições-λ, algumas das quais, que chamaremos *colapsadores*, podem ser eliminadas numa forma que vamos descrever nesta seção, proporcionando um autômato finito mais compacto que o original.

Para tanto vamos considerar o autômato finito como um *grafo dirigido*; assim sendo, dizemos que um subconjunto C, do conjunto de estados Q do autômato, é uma *componente fortemente conexo* se, e somente se, entre dois quaisquer estados q e p de C existem um caminho de q a p e outro caminho de p a q e todos os estados desses caminhos pertencem a C. Vamos descrever um algoritmo que elimina colapsadores dentro dos componentes fortemente conexos do autômato finito.

Algumas das transições-λ que restam depois da eliminação de colapsadores agem como *pontes orientadas* entre *componentes fortemente conexos* do autômato finito (visto como um grafo orientado); denominaremos essas pontes de *pontes-λ*. Vamos descrever também um algoritmo de eliminação de pontes-λ entre componentes fortemente conexos, obtendo assim um autômato finito ainda mais compacto do que aquele obtido depois de eliminar os colapsadores.

Esses dois algoritmos serão empregados em um algoritmo principal para eliminação de colapsadores e pontes-λ. O esquema do algoritmo baseia-se em uma análise recursiva e o correspondente processo de síntese, para reorganizar os componentes fortemente conexos, deve ser feito em sentido reverso, ou seja, do estado final para o inicial, segundo uma ordenação topológica das componentes fortemente conexas que começa na componente fortemente conexa que contém o estado q_0 e termina na componente fortemente conexa que contém o estado q_1.

Definição 4.2.1 *Seja $Trans$ o conjunto de todas as transições do autômato finito*

gerado pelo algoritmo wirth2ape. Se $t \in Trans$ *então* t *é da forma:*

$$t = (q, a, p)$$

onde q, p *são estados pertencentes ao conjunto* Q *dos estados do autômato gerado pelo algoritmo wirth2ape. Nessas condições, dizemos que* q *é o estado de partida da transição* t *e* p *é o estado de chegada da transição* t*.*

Definição 4.2.2 *Seja* q *um dos estados gerados pelo algoritmo wirth2ape e* a *um símbolo do alfabeto. Seja:*

$$\Gamma^+(q) = \{p \in Q : \exists \text{ uma transição } t \text{ tal que } t = (q, a, p) \text{ para algum } a \in \Sigma.\}$$

Definimos o grau positivo (ou de saída) do estado q *como:*

$$d^+(q) = |\Gamma^+(q)|.$$

Seja:

$$\Gamma^-(q) = \{p \in Q : \exists \text{ uma transição } t \text{ tal que } t = (p, a, q) \text{ para algum } a \in \Sigma.\}$$

Definimos o grau negativo (ou de entrada) do estado q *como:*

$$d^-(q) = |\Gamma^-(q)|.$$

Definição 4.2.3 *Seja* $Trans$ *o conjunto de todas as transições do autômato finito gerado pelo algoritmo wirth2ape. Denominamos "colapsadores" todas as transições* $t \in Trans$ *tais que*

$$\delta(q, \lambda) = p, \text{ e } d^+(q) = 1.$$

O conjunto de todas transições que são colapsadores de $Trans$ *é denotado como* $Colapsadores(Trans)$*.*

Definição 4.2.4 *Uma transição* $t = (q, a, p)$*, com* $q, p \in Q$ *e* $a \in \Sigma$ *é uma "ponte" se ela une duas componentes fortemente conexas difererentes, isto é, se existem* C_i, C_j*, componentes fortemente conexas tais que* $q \in C_i$ *e* $p \in C_j$*. Se* $a = \lambda$ *dizemos que* t *é uma ponte-*λ*.*

4.2.2 O algoritmo EliminaColapsadores

Algoritmo 4.2.1 EliminaColapsadores

Entrada: *O conjunto $Trans$ das transições geradas pelo algoritmo wirth2ape.*

Saída: *Um novo conjunto de transições $Trans_1$, sem colapsadores, que representa um autômato finito $FA(Trans_1)$ equivalente ao autômato finito representado por $Trans$, isto é, tal que $L(FA(Trans_1)) = L(FA(Trans))$.*

Método

1. *Para cada $t \in Colapsadores(Trans)$, com $t = (q, a, p)$ sejam*

$$min = mínimo\{q, p\}$$

$$max = máximo\{q, p\}$$

   (a) *Substituir cada ocorrência de max por min em todas as trasições de T.*

   (b) *Eliminar a transição*

$$\delta(min, \lambda) = min.$$

   gerada no passo anterior.

Fim-do-Método

Observação 4.2.1 *A substituição em (1.a) preserva o papel do estado final q_1.*

Observação 4.2.2 *A substituição em (1.a) transforma circuitos de comprimento 2:*

$$\delta(q, \sigma) = p, \delta(p, \lambda) = q, \text{ e } d^+(p) \equiv d^-(p) = 1.$$

no laço correspondente

$$\delta(q, \sigma) = q.$$

4.2.3 O algoritmo EliminaPontes-λ

Algoritmo 4.2.2 EliminaPontes-λ

Entrada: *O conjunto $Trans$ das transições geradas pelo algoritmo wirth2ape, particionado em duas componentes fortemente conexas C_i, C_j, e uma ponte-λ entre elas $\delta(q, \lambda) = p$, onde $q \in C_i$ e $p \in C_j$*

Saída: *Um novo conjunto de transições $Trans_1$, sem pontes, que representa um autômato finito $FA(Trans_1)$ equivalente ao representado por $Trans$, isto é, $L(FA(Trans_1)) = L(FA(Trans))$.*

Método

1. *Se $d^+(p) = 0$ então substituir todas as referências a q por referências a p.*
2. *Se $d^+(p) > 0$ então*
   (a) *Para cada $r \in \Gamma^+(a)$, tal que $\delta(p, a) = r$, com $a \in \Sigma$ acrescentar a $Trans$ a transição $\delta(q, a) = r$.*
   (b) *Se p é estado final, então marcar q como estado final.*

Fim-do-Método

4.2.4 O algoritmo ComponentesConexos

O algoritmo apresentado aqui é uma variação do algoritmo de busca em profundeza[48] (DFS, pelas suas siglas em inglês: *Depth First Search*) em grafos dirigidos.

O algoritmo esta dividido em duas partes; a primeira serve como programa principal; a segunda, denominada "Visitar", é uma variação sobre a forma de visitar os vértices de um grafo dirigido do algoritmo de busca em profundeza e ela que faz a identificação dos vértices numa mesma componente fortemente conexa. Para simplificar questões sobre sobre passagens de parâmetros admitimos que essa segunda parte tem acesso, em forma global aos parâmetros e variáveis do programa principal.

Definição 4.2.5 *Seja $G = (V, E)$ um grafo dirigido e $v \in V$ um vértice. definimos $dfsnum(v)$ como o número de vértices visitados antes de visitar v no algoritmo de busca em profundenza.*

Algoritmo 4.2.3 ComponentesFortementeConexasDFS(G)

Entrada: *Um grafo dirigido $G = (V, E)$.*

Saída: *A lista dos vértices que pertencem a cada componente fortemente conexa.*

Método

1. *Crie um novo vértice x com arestas $x \to v$, $\forall v \in V$*
2. *Inicializar o contador $N \leftarrow 0$*
3. *Inicializar a lista $L \leftarrow \emptyset$*
4. *Construir uma árvore dirigida T, inicialmente com o vértice x*
5. *Visitar(x)*

Fim-do-Método

Algoritmo 4.2.4 Visitar(p)

Entrada: *Um vértice de uma árvore dirigida T.*

Saída: *A lista dos vértices que pertencem a cada componenete fortemente conexo.*

Método

1. *Adicione p a L*

2. $dfsnum(p) = N$
3. $N = N + 1$
4. $low(p) = dfsnum(p)$
5. *Para cada aresta* $p \rightarrow q$ *faça*
   (a) *Se* q *não pertence a* T *então*
      i. *adicione* $p \rightarrow q$ *a* T
      ii. $visitar(q)$
      iii. $low(p) = min(low(p), low(q))$
   (b) *Fim-então*
   (c) *CasoContrario* $\{low(p) = min(low(p), dfsnum(q))\}$
6. *Fim-Para*
7. *Se* $low(p) = dfsnum(p)$ *então*
   (a) *output "componente fortemente conexa:"*
   (b) *repetir*
      i. *remover o último elemento* v *da lista* L
      ii. *imprimir* v
      iii. *remover* v *do grafo* G
   (c) *até* v=p
8. *Fim-Se*

Fim-do-Método

4.2.5 O algoritmo EliminaColapsadoresEPontes-λ

Algoritmo 4.2.5 EliminaTransiçõesEPontes-λ

Entrada: *O conjunto $Trans$ das transições geradas pelo algoritmo wirth2ape.*

Saída: *Um novo conjunto de transições $Trans_1$, sem colapsadores nem pontes-λ, que representa um autômato finito $FA(Trans_1)$ equivalente ao autômato finito representado por $Trans$, isto é, $L(FA(Trans_1)) = L(FA(Trans))$.*

Método

1. $Trans_1 = EliminaColapsadores(Trans)$
2. *Sejam $C_0, \ldots, C_m$ as componentes fortemente conexas do autômato representado por $Trans_1$, numa ordenação topológica onde $q_0 \in C_0$ e $q_1 \in C_m$*
   (a) *Para cada $i = 0 \ldots m$*

   aplicar o algoritmo EliminaColapsadores ao componente C_i:

   (b) *Para cada $i = m, m-1, \ldots, 1$*

   Se existe uma ponte-λ entre C_i e C_{i-1} eliminá-las usando o algoritmo EliminaPontes-λ.

Fim-do-Método

4.2.6 Exemplo 1 de eliminação de colapsadores

Neste exemplo vamos detalhar a execução do algoritmo *EliminaColapsadores* quando executado ao receber como dado a expressão regular, em notação de Wirth:

$$FA = p_1\{p_2p_3\}p_4p_5.$$

Essa expressão regular define a linguagem de controle da gramática G_{10} do exemplo 2.4.1. A Fig. 4.9 mostra a saída do algoritmo *wirth2ape*

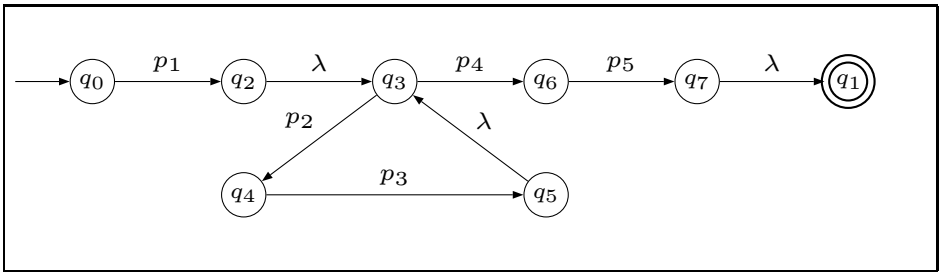


Figura 4.9: FA original. Três colapsadores: $\delta(q_2, \lambda) = q_3$, $\delta(q_5, \lambda) = q_3$, $\delta(q_7, \lambda) = q_1$.

O primeiro colapsador, $\delta(q_2, \lambda) = q_3$, é eliminado colapsando-se o estado q_3 no estado q_2; o resultado da eliminação é mostrado na Fig. 4.10. O estado q_3 não é mostrado na Fig. 4.10, pois todas as transições que faziam referência a ele, diferentes do colapsador eliminado, fazem agora referência a q_2.:

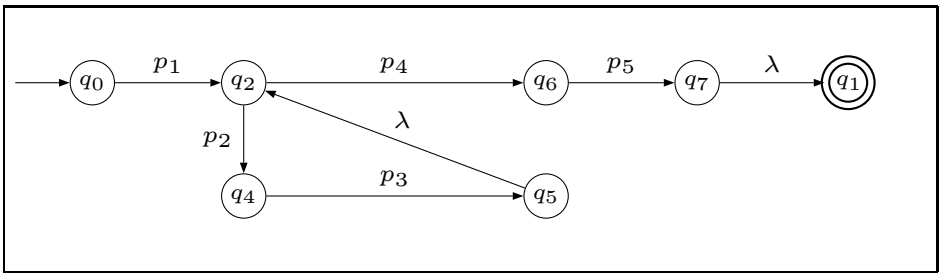


Figura 4.10: FA após um passo. 2 colapsadores: $\delta(q_5, \lambda) = q_2$, $\delta(q_7, \lambda) = q_1$

O segundo colapsador, $\delta(q_5, \lambda) = q_2$, é eliminado colapsando-se o estado q_5 no estado q_3; o resultado da eliminação é mostrado na figura seguinte. O estado q_5 não é mostrado na figura, pois a referência que a transição $\delta(q_4, p_3) = q_5$ fazia ao estado q_5 foi substituída por uma referência ao estado q_4 e não existem mais transições, diferentes do colapsador eliminado, chegando ao estado q_5 ou dele saindo. O resultado está mostrado na Fig. 4.11:

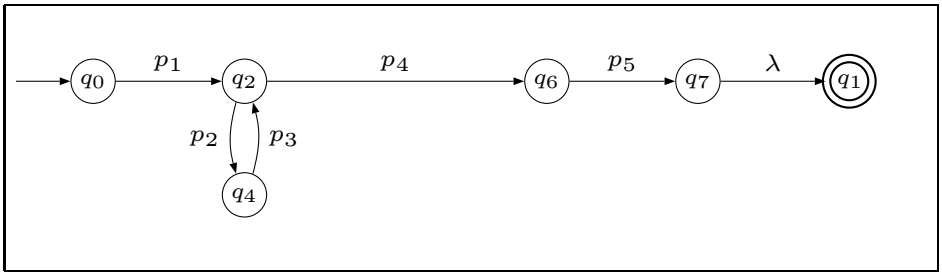


Figura 4.11: FA após dois passos. 1 colapsadores: $\delta(q_7, \lambda) = q_1$

Finalmente, o terceiro e último colapsador, $\delta(q_7, \lambda) = q_1$, é eliminado colapsando o estado q_7 no estado q_1; o resultado da eliminação é mostrado na Fig. 4.12. O estado q_7 desapareceu pois a referência que a ele fazia a transição $\delta(q_5, p_5) = q_7$ foi substituída por outra alusiva ao estado q_1:

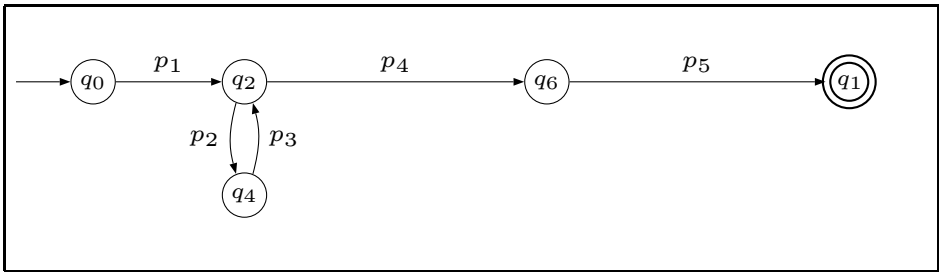


Figura 4.12: FA após três passos. zero colapsadores

O autômato acima é, salvo isomorfismo de estados, exatamente o mesmo que o autômato da Fig. 2.8 que define a linguagem de controle para a gramática G_{10} do exemplo 2.4.1.

O isomorfismo ϕ, mencionado, estabelece uma relação bijetora entre os estados do autômato na Fig. 4.12 e os estados do autômato na Fig. 2.8, da seguinte forma:

$$
\begin{aligned}
q_0 &\rightarrow q_0,\\
q_1 &\rightarrow q_1,\\
q_2 &\rightarrow q_2,\\
q_4 &\rightarrow q_3,\\
q_6 &\rightarrow q_4.
\end{aligned}
$$

Este isomorfismo *conserva transições*; isto quer dizer que se q, p são estados do autômato da Fig. 4.12, entre os quais existe uma transição $\delta(q, a) = p$ então $\delta(\phi(q), a) = \phi(p)$ é a transição correspondente no autômato da Fig. 2.8.

4.2.7 Exemplo 2 de eliminação de colapsadores

Neste exemplo vamos detalhar a execução do algoritmo *EliminaColapsadores* quando executado ao receber como dado a expressão regular em notação de Wirth:

$$FA = p_1\{p_2p_6|p_3p_7\}(p_5p_9|p_4p_8).$$

Essa expressão regular define a linguagem de controle da gramática G_{11} do exemplo 2.4.2. A Fig. 4.13 mostra a saída do algoritmo *wirth2ape*.

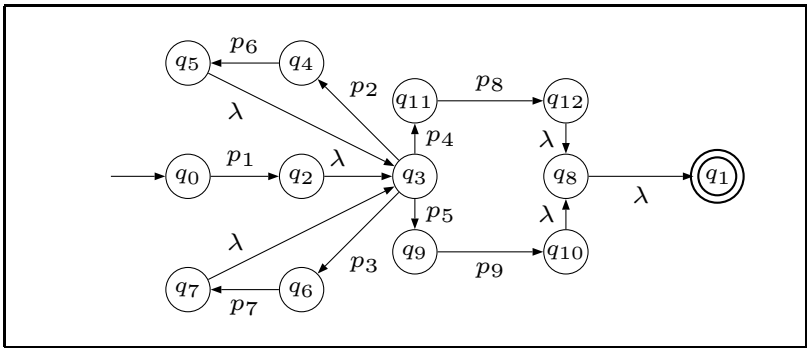


Figura 4.13: FA original. 6 colapsadores: $\delta(q_2, \lambda) = q_3$, $\delta(q_5, \lambda) = q_3$, $\delta(q_7, \lambda) = q_3$, $\delta(q_{12}, \lambda) = q_8$, $\delta(q_{10}, \lambda) = q_8$, $\delta(q_8, \lambda) = q_1$

A eliminação do primeiro colapsador $\delta(q_2, \lambda) = q_3$ colapsa o estado q_3 no estado q_2; as únicas transições, diferentes do colapsador eliminado, que faziam referência ao estado q_3, eram o segundo e o terceiro colapsadores, e tiveram essas referências substituídas por referências ao estado q_2. Todas essas mudanças são mostradas na Fig. 4.14:

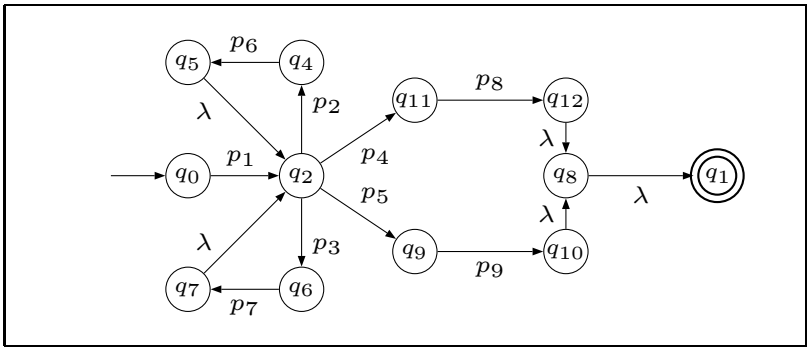


Figura 4.14: FA após um passo. 5 colapsadores: $\delta(q_5, \lambda) = q_2$, $\delta(q_7, \lambda) = q_2$, $\delta(q_{12}, \lambda) = q_8$, $\delta(q_{10}, \lambda) = q_8$, $\delta(q_8, \lambda) = q_1$

A eliminação do segundo colapsador $\delta(q_5, \lambda) = q_2$ colapsa o estado q_5 no estado q_2; a única transição, diferente do colapsador eliminado, que fazia referência ao estado q_5, era $\delta(q_4, p_6) = q_5$, e ela teve essa referência substituída por uma referência ao estado q_2:

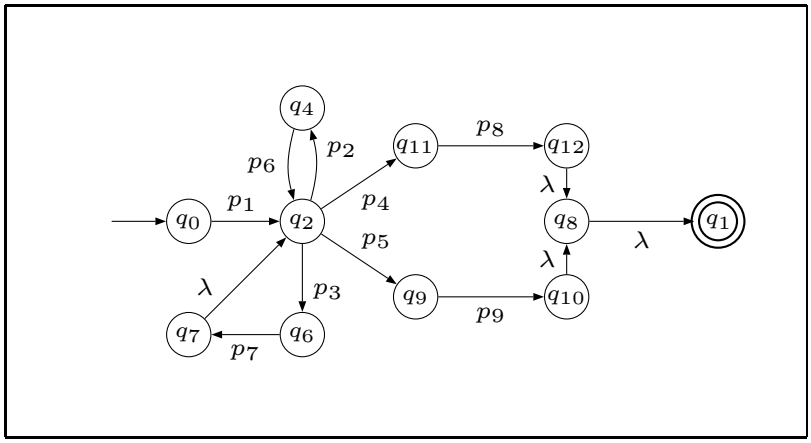


Figura 4.15: FA após dois passos. 4 colapsadores: $\delta(q_7, \lambda) = q_2$, $\delta(q_{12}, \lambda) = q_8$, $\delta(q_{10}, \lambda) = q_8$, $\delta(q_8, \lambda) = q_1$

A eliminação do terceiro colapsador, $\delta(q_7, \lambda) = q_2$, colapsa o estado q_7 no estado q_2; a única transição, diferente do colapsador eliminado, que fazia referência ao estado q_5, era $\delta(q_6, p_7) = q_7$, e ela teve essa referência substituída por uma referência ao estado q_2:

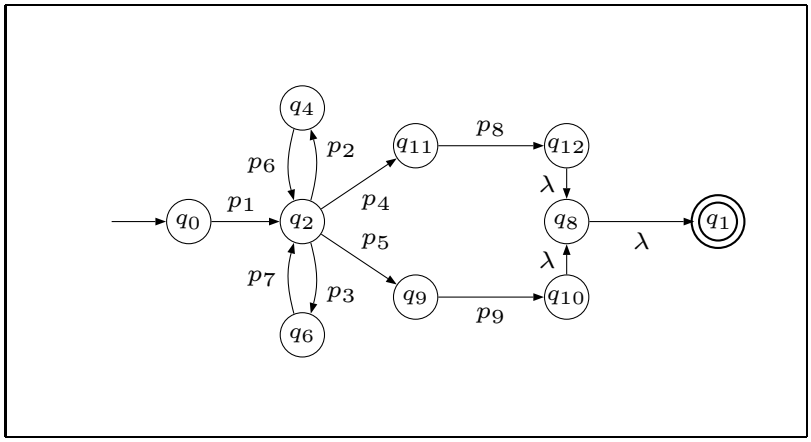


Figura 4.16: FA após três passos. 3 colapsadores: $\delta(q_{12}, \lambda) = q_8$, $\delta(q_{10}, \lambda) = q_8$, $\delta(q_8, \lambda) = q_1$

A eliminação do quarto colapsador, $\delta(q_{12}, \lambda) = q_8$, colapsa o estado q_{12} no estado q_8; a única transição, diferente do colapsador eliminado, que fazia referência ao estado q_{12}, era $\delta(q_{11}, p_8) = q_{12}$, e ela teve essa referência substituída por uma referência ao estado q_8, conforme a Fig. 4.17

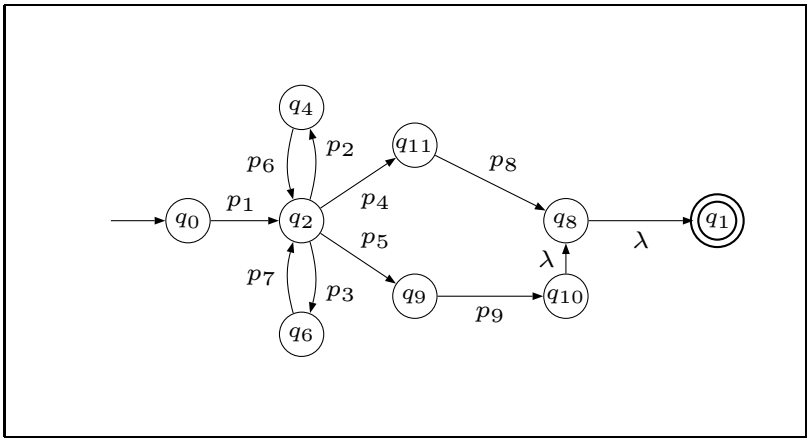


Figura 4.17: FA após quatro passos. 2 colapsadores: $\delta(q_{10}, \lambda) = q_8$, $\delta(q_8, \lambda) = q_1$

A eliminação do quinto colapsador, $\delta(q_{10}, \lambda) = q_8$, colapsa o estado q_{10} no estado q_8; a única transição, diferente do colapsador eliminado, que fazia referência ao estado q_{10}, era $\delta(q_9, p_9) = q_{10}$, e ela teve essa referência substituída por uma referência ao estado q_8, como se pode observar na Fig. 4.18

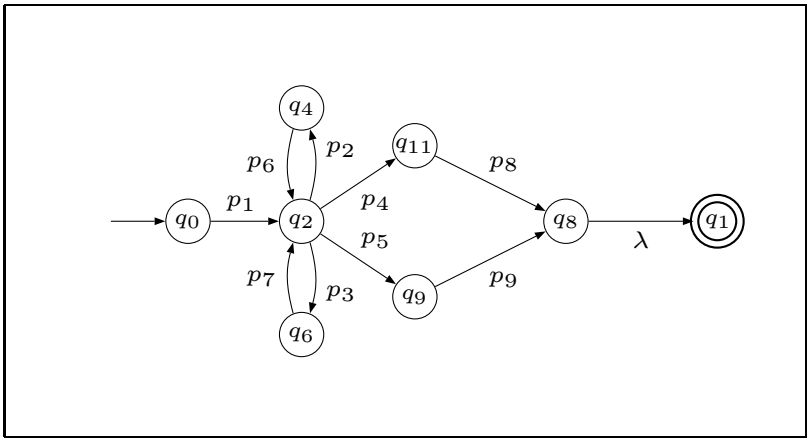


Figura 4.18: FA após cinco passos. 1 colapsador: $\delta(q_8, \lambda) = q_1$

.

Vamos agora eliminar o último colapsador.

A eliminação do último colapsador, $\delta(q_8, \lambda) = q_1$, colapsa o estado q_8 no estado q_1; as referências ao estado q_8, na transições $\delta(q_9, p_9) = q_8$, $\delta(q_{11}, p_8) = q_8$ foram substituídas por referências ao estado q_1. Veja a Fig. 4.19.

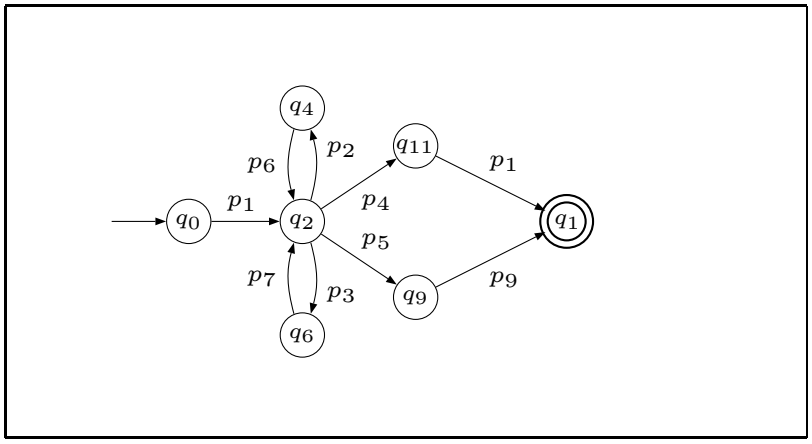


Figura 4.19: FA após seis passos. zero colapsadores

.

O autômato acima é, salvo isomorfismo de estados, exatamente o mesmo que o autômato da Fig. 2.9, que define a linguagem de controle para a gramática G_{11} do exemplo 2.4.2.

O isomorfismo ϕ, mencionado, estabelece uma relação bijetora entre os estados do autômato na Fig. 4.19 e os estados do autômato na Fig. 2.9, da seguinte forma:

$$
\begin{aligned}
q_0 &\longrightarrow q_0,\\
q_1 &\longrightarrow q_1,\\
q_2 &\longrightarrow q_2,\\
q_4 &\longrightarrow q_3,\\
q_6 &\longrightarrow q_4,\\
q_{11} &\longrightarrow q_5,\\
q_9 &\longrightarrow q_6.
\end{aligned}
$$

Esse isomorfismo ϕ *conserva transições*; isto quer dizer que se q, p são estados do autômato da Fig. 4.12, entre os quais existe uma transição $\delta(q, a) = p$ então $\delta(\phi(q), a) = \phi(p)$ é a transição correspondente no autômato da Fig. 2.8.

4.2.8 Exemplo 3 de eliminação de colapsadores e pontes-λ

Neste exemplo vamos detalhar a execução do algoritmo *EliminaColapsadores* quando executado ao receber como dado a expressão regular em notação de Wirth:

$$FA = \{\{p_1\}p_2\{p_3\}p_4\}p_5\{p_5\}.$$

Essa expressão regular define a linguagem de controle da gramática G_{10} do exemplo 2.4.1. A Fig. 4.20 mostra a saída do algoritmo *wirth2ape*:

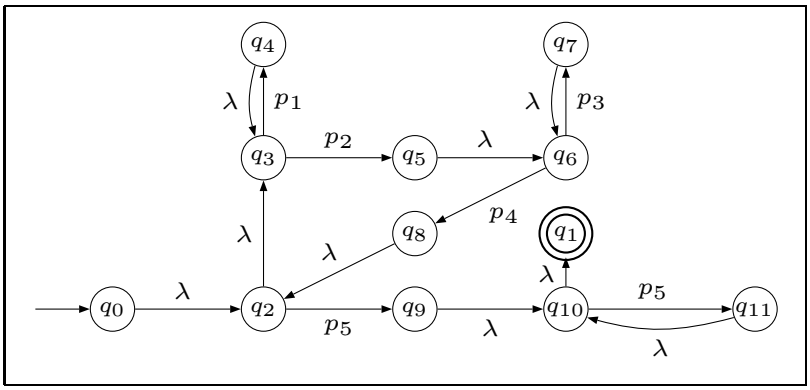


Figura 4.20: FA original: 7 colapsadores

O autômato possui 7 colapsadores: $\delta(q_0, \lambda) = q_2$, $\delta(q_4, \lambda) = q_3$, $\delta(q_5, \lambda) = q_6$, $\delta(q_7, \lambda) = q_6$, $\delta(q_8, \lambda) = q_2$, $\delta(q_9, \lambda) = q_{10}$, e $\delta(q_{11}, \lambda) = q_{10}$. A eliminação do primeiro colapsador, colapsa o estado q_2 no estado q_0. Veja a Fig. 4.21:

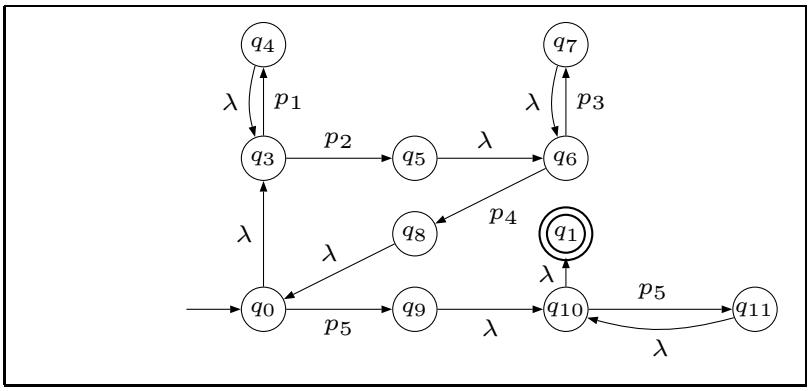


Figura 4.21: FA após um passo: 6 colapsadores

A eliminação do segundo colapsador, $\delta(q_4, \lambda) = q_3$, colapsa o estado q_4 no estado q_3; a única transição que referenciava o estado q_4 era $\delta(q_3, p_1) = q_4$ e essa

referência foi substituída para referenciar o estado q_3, produzindo um laço no estado q_3, como mostra a Fig. 4.22.

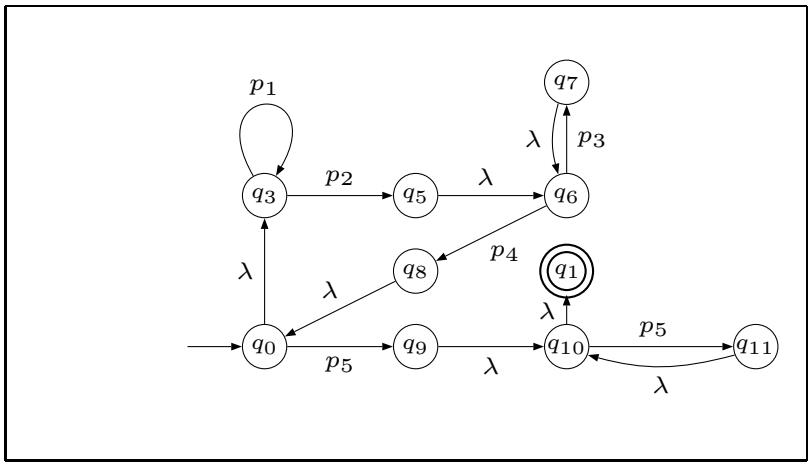


Figura 4.22: FA após dois passos: 5 colapsadores

A eliminação do terceiro colapsador, $\delta(q_5, \lambda) = q_6$, colapsa o estado q_6 no estado q_5; as duas produções que referenciavam o estado q_6, $\delta(q_6, p_3) = q_7$, e $\delta(q_7, \lambda) = q_6$, tiveram suas referências substituídas para referenciarem o estado q_5. A Fig. 4.23 apresenta todas essas alterações:

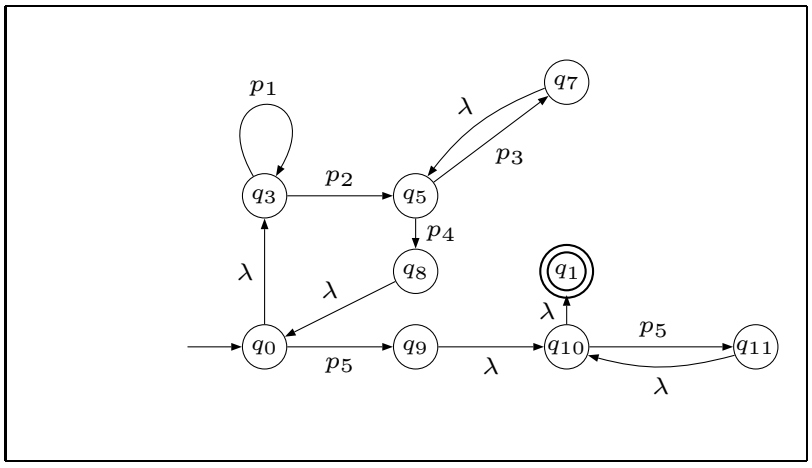


Figura 4.23: FA após três passos: 4 colapsadores

Vamos agora eliminar o quarto colapsador.

A eliminação do quarto colapsador, $\delta(q_7, \lambda) = q_5$, colapsa o estado q_7 no estado q_5; a única transição que referenciava o estado q_7 era $\delta(q_5, p_3) = q_7$ e teve essa

referência substituída para referenciar q_5, produzindo um laço em q_5. Todas essas alterações estão apresentadas na Fig. 4.24:

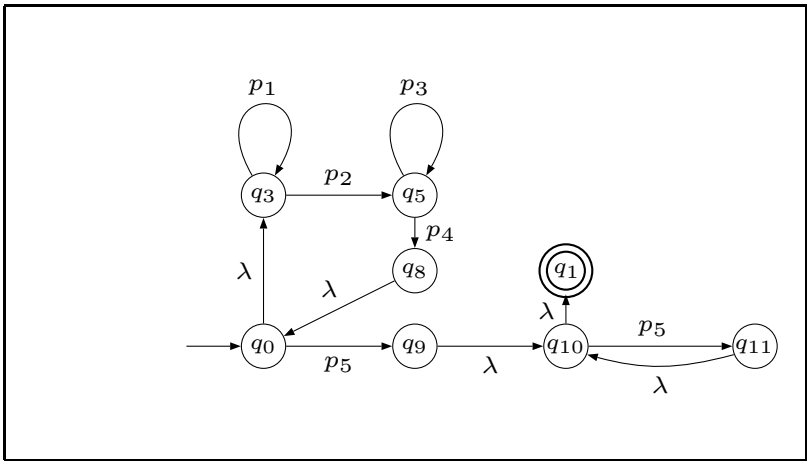


Figura 4.24: FA após quatro passos: 3 colapsadores

A eliminação do quinto colapsador, $\delta(q_8, \lambda) = q_2$, colapsa o estado q_8 no estado q_5; a única transição que referenciava q_8 era $\delta(q_5, p_4) = q_8$, e teve a referência substituída para referenciar q_2. Veja a Fig. 4.25.

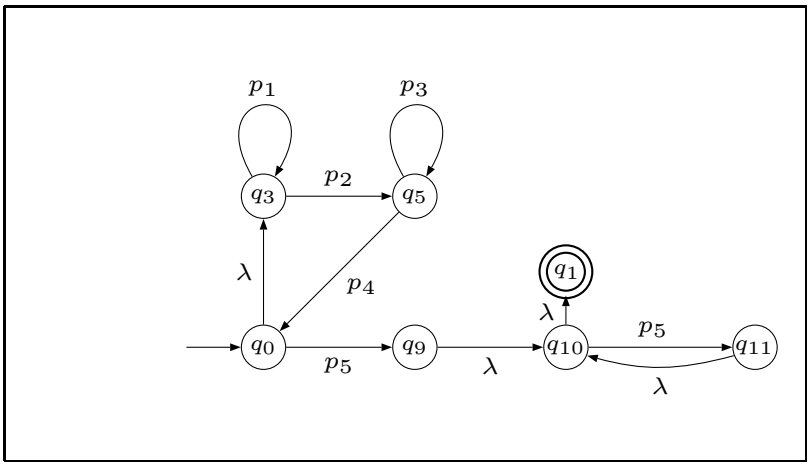


Figura 4.25: FA após cinco passos: 2 colapsadores

Vamos agora eliminar o sexto colapsador.

A eliminação do sexto colapsador, $\delta(q_9, \lambda) = q_{10}$, colapsa o estado q_{10} no estado q_9; as duas produções que referenciavam o estado q_{10}, $\delta(q_{10}, p_5) = q_{11}$, e

$\delta(q_{11}, \lambda) = q_{10}$, tiveram suas referências substituídas para referenciar o estado q_9. A Fig. 4.26 apresenta todas essas mudanças:

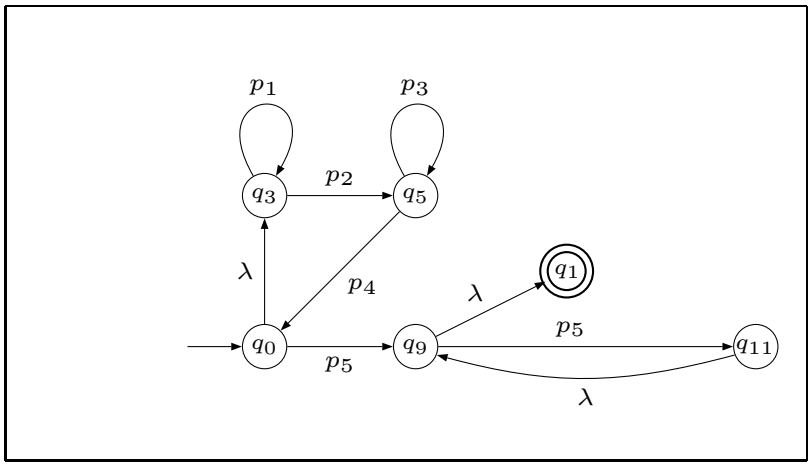


Figura 4.26: FA após seis passos: 1 colapsador

A eliminação do sétimo colapsador, $\delta(q_{11}, \lambda) = q_{10}$, colapsa o estado q_{11} no estado q_9. A única transição que referenciava o estado q_{11} era $\delta(q_9, p_5) = q_{11}$, e teve essa referência substituída para referenciar q_9, produzindo um laço em q_9, como mostra a Fig. 4.27:

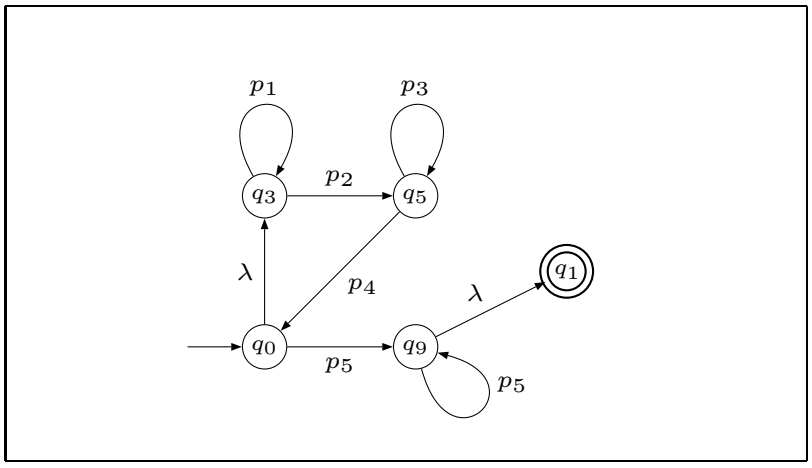


Figura 4.27: FA após sete passos: 0 colapsadores

É o momento de aplicar o algoritmo principal *EliminaColapsadoresEPontes-*λ.

O autômato, na figura anterior, possui três componentes fortemente conexas: $C_1 = \{q_0, q_3, q_5\}$, $C_2 = \{q_9\}$ $C_3 = \{q_1\}$. A transição $\delta(q_9, \lambda) = q_1$ é uma ponte-λ

entre as componentes C_2 e C_3. Veja a Fig. 4.28.

Executamos o algoritmo *EliminaColapsadores* na componente C_1:

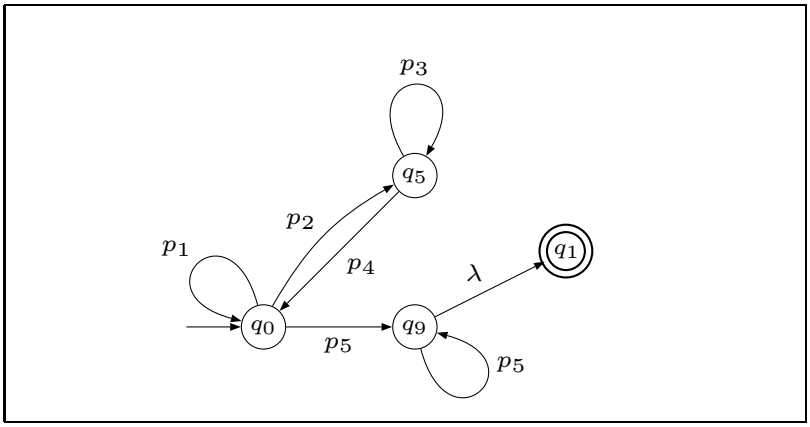


Figura 4.28: FA após eliminar colapsador no componente C_1

As componentes C_2 e C_3, não possuem colapsadores, e, portanto, essa parte do algoritmo é encerrada.

Agora devemos executar a eliminação de pontes-λ, começando pelos componentes C_3 e C_2; neste caso o algoritmo *EliminaPontes-*λ elimina a ponte-λ $\delta(q_9, \lambda) = q_1$, transformando os componentes C_3 e C_2 em apenas um componente fortemente conexo como se pode observar na Fig. 4.29.

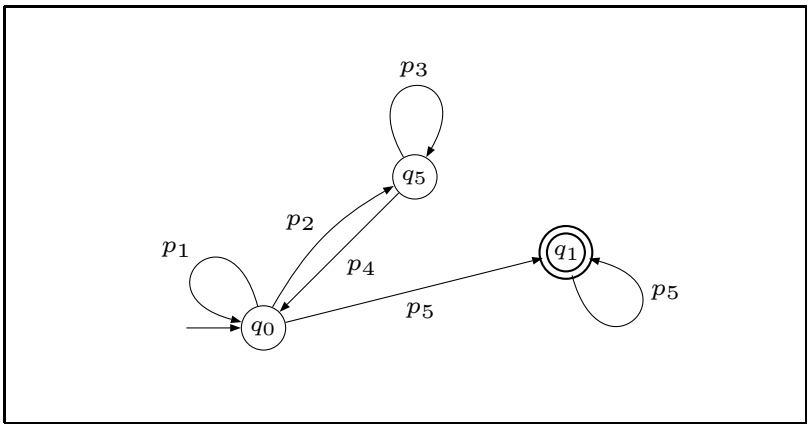


Figura 4.29: FA após eliminar ponte-λ entre as componentes C_2 e C_3

O autômato acima é, salvo isomorfismo de estados, exatamente o mesmo que o autômato da Fig. 2.10 que define a linguagem de controle para a gramática G_{12} do exemplo 2.4.3.

O isomorfismo ϕ, mencionado, estabelece uma relação bijetora entre os estados do autômato na Fig. 4.29 e os estados do autômato na Fig. 2.10, da seguinte forma:

$$q_0 \rightarrow q_0,$$
$$q_1 \rightarrow q_1,$$
$$q_5 \rightarrow q_2.$$

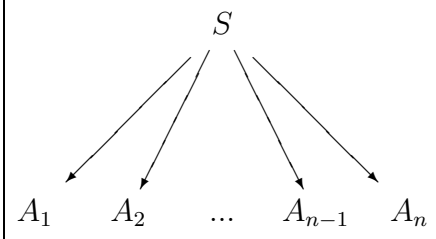
Esse isomorfismo ϕ *conserva transições*; isto quer dizer que se q, p são estados do autômato da Fig. 4.29, entre os quais existe uma transição $\delta(q, a) = p$ então $\delta(\phi(q), a) = \phi(p)$ é a transição correspondente no autômato da Fig. 2.8.

4.3 Analisadores Adaptativos descendentes controlados

As gramáticas livres de contexto com linguagem de controle regular com verificação de aparência podem servir como "pedra de Roseta" entre "algumas" gramáticas livres de contexto adaptativas com verificação de aparência e autômatos finitos adaptativos, que por sua vez são analisadores para as linguagens que essas gramáticas geram. Consideramos apenas gramáticas para as quais o símbolo inicial aparece em produções cujo lado direito é uma seqüência de símbolos não-terminais e nenhum outro terminal se duplica, como em $A \to AA$.

A árvore de análise é codificada, na estrutura do autômato adaptativo, da seguinte forma: O símbolo inicial da gramática é representado por um estado especial chamado q_t; a partir desse estado, é construído um autômato finito que serve como base para fazer um reconhecimento das palavras da linguagem, acompanhado da construção da árvore de análise correspondente.

A construção do autômato finito é feita em três etapas. Consideremos a produção $p = S \to A_1 A_2 \ldots A_{n-1} A_n$ cuja árvore de derivação é mostrada na tabela a seguir:

produção inicial	árvore de análise
$p = S \to A_1 A_2 \ldots A_{n-1} A_n$	

Arvore de análise para $p = S \to A_1, \ldots, A_n$.

Na primeira etapa, de construção do autômato finito adaptativo, são criados estados $q_2, \ldots, q_{n+1}$ e transições $\delta(q_t, A_i) \to q_{i+1}$, com $i = 1, \ldots, n$; esses estados e transições representam a codificação, no autômato finito adaptativo, *da aplicação da produção p na árvore de derivação* da palavra que estiver sendo reconhecida pelo autômato.

Também são criados estados $r_1, \ldots, r_n$ e transições $\delta(r_i, \lambda) \to q_{i+1}$, com $i = 1, \ldots, n$; esses estados são chamados *registradores* e servem como memória para armazenar a posição onde devem ser aplicadas as producões que têm em seus lados esquerdos os não-terminais $A_1, \ldots, A_n$.

O correspondente autômato finito (ainda não adaptativo) para essa árvore de

análise é apresentado na figura 4.30:

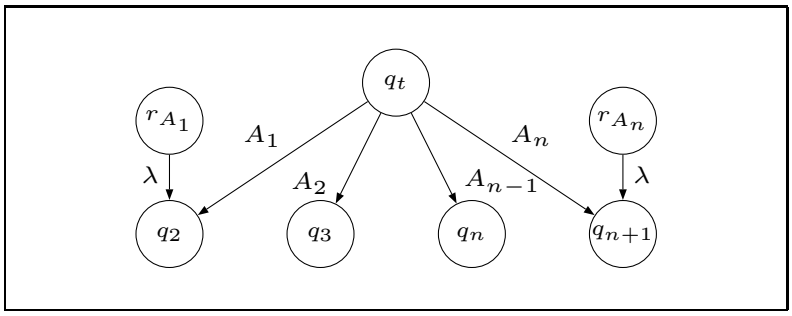


Figura 4.30: Início. Omitimos os registradores entre r_{A_2} e r_{A_n}.

A árvore de análise está codificada, nesse autômato finito, da seguinte forma: O símbolo inicial S da gramática é representado pelo estado q_t; os registradores $r_{A_1}, \ldots,$ r_{A_m} servem para armazenar o estado em que devem ser aplicadas as produções que têm, no seu lado esquerdo, os não-terminais $A_1, \ldots, A_m$.

A segunda etapa, da construção do autômato finito adaptativo, cria um *caminho viável* desde o estado inicial q_0 ao estado final q_1 do autômato finito adaptativo. Esse caminho viável é representado por um caminho, formado por transições-λ, que partindo do estado q_o, atravessa os estados $q_2, \ldots, q_{n+1}$ e termina no estado q_1. A Fig. 4.31 mostra o resultado dessa etapa, no exemplo anterior:

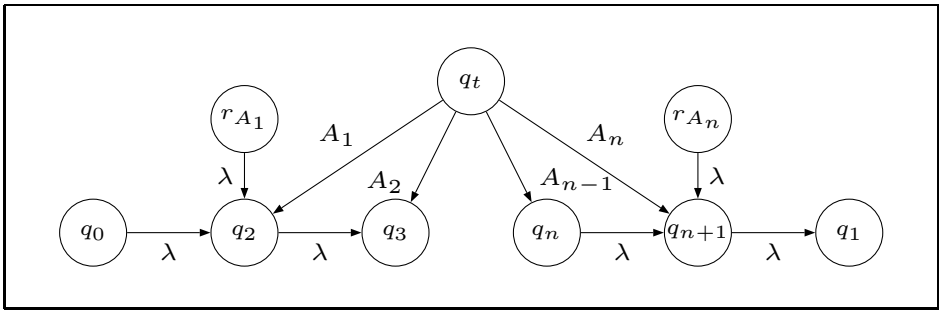


Figura 4.31: Criação de um caminho "viável" entre q_0 e q_1.

Na terceira etapa, da construção do autômato finito adaptativo, devem-se trocar as transicões λ do caminho que vai do estado q_0 até o estado q_1, por transições que consomem símbolos do alfabeto da linguagem gerada pela gramática livre de contexto com linguagem de controle regular; O critério para fazer isso é examinar a expressão regular que controla a derivação das palavras da linguagem associando, a cada transição no caminho viável, um símbolo do alfabeto "produzido" por alguma produção na derivação mais curta possível (pode ser que fiquem transições vazias

no final do caminho viável); deste modo deve-se criar um *prefixo* de alguma palavra da linguagem em questão.

A Fig. 4.32 mostra o resultado dessa etapa no exemplo anterior.

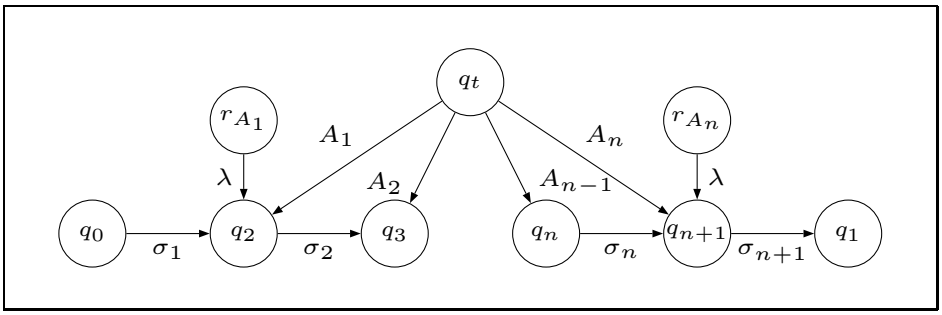


Figura 4.32: Prefixo "viável" entre q_0 e q_1

A árvore de análise vai sendo construída por funções adaptativas à medida que o autômato finito adaptativo vai consumindo símbolos da cadeia de entrada e transitando pelas transições que compõem o prefixo viável. Para construir a árvore de análise é necessário:

1. *Marcar* cada terminal, no prefixo viável, com a produção que foi usada para gerá-lo, codificando assim uma parte da árvore de análise da palavra que estiver sendo consumida, sem realizar mudanças estruturais, ou então,

2. Realizar mudanças estruturais no autômato finito adaptativo, que codifiquem, a cada símbolo consumido pelo mesmo, a aplicação da produção responsável pela geração desse símbolo.

O código indicador de que um símbolo σ foi produzido pela aplicação de uma produção p_σ, a partir de um estado q_A apontado pelo registrador r_A é uma transição que vai de q_A ao estado de partida da transição que consome σ. A Fig. 4.33 mostra o efeito de tais produções *sem realizar mudanças estruturais* no autômato finito adaptativo:

Grosso modo, tais mudanças estruturais requerem, em geral:

1. Remover algumas transições, para incluir um nova transição entre outras duas, já existentes,

2. Criar novos estados e transições, para conservar a sintaxe descrita pelas transições removidas e incluir a sintaxe dos símbolos que estiverem sendo gerados pela produção que está sendo aplicada à árvore de análise e que está sendo codificada na estrutura do autômato finito adaptativo.

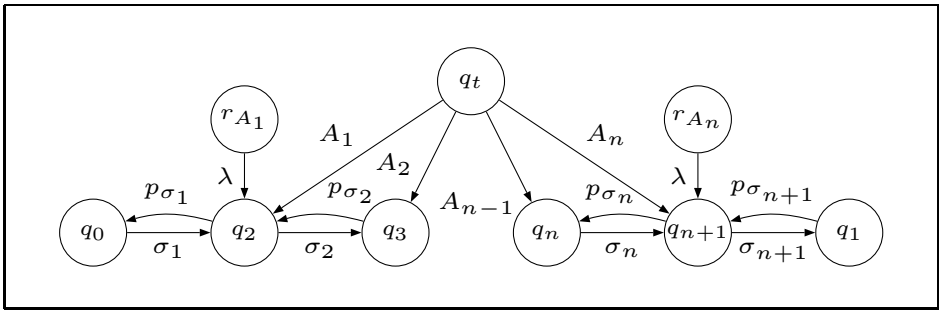
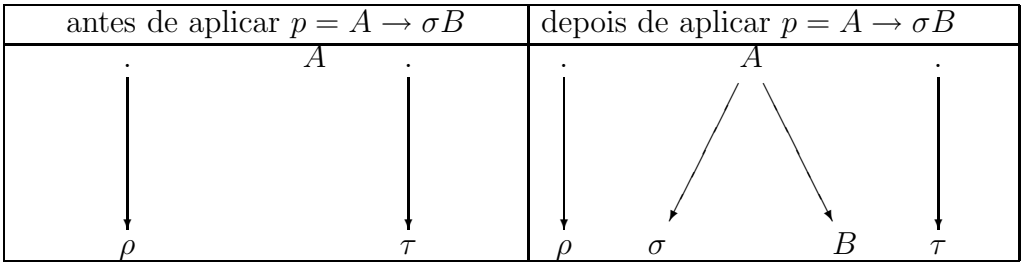


Figura 4.33: Marcação dos terminais do prefixo "viável" do caminho entre q_0 e q_1.

3. Atualizar os registradores.

Agora vamos detalhar como realizar essas mundanças estruturais, discutindo o caso de aplicação de uma produção entre dois símbolos num caminho viável entre q_0 e q_1. Suponhamos que o estado seja q_A e que ele seja apontado pelo registrador r_A. Consideramos apenas o caso de produções com recursão à esquerda; as discussões a seguir podem ser extendidas para o caso de produções com recursão central.

Sejam os símbolos terminais $\rho, \sigma, \tau \in \Sigma$, os símbolos não-terminais $A, B \in N$, e a produção $p = A \rightarrow \sigma B$. A seguinte tabela mostra uma árvore de derivação antes e depois da aplicação da produção p ao nó A:



O terminal τ pode (mas não precisa) ter sido gerado pela mesma produção que gerou o terminal ρ, mas vamos considerar o caso mais geral, em que cada um desses terminais é gerado pelas produções p_τ e p_ρ respectivamente. A Fig. 4.34 mostra a codificação da árvore de análise antes da aplicação da produção $p = A \rightarrow \sigma B$:

São criados dois novos estados, q_r, q_s, para codificar a aplicação da produção p_σ; observe que não é alterada a transição indicativa de que o símbolo ρ foi gerado pela aplicação da produção p_ρ; em troca, o estado de chegada da transição que consome ρ foi atualizada para incluir o símbolo σ no prefixo viável; o símbolo σ, consumido por essa transição, é marcado como tendo sido gerado pela produção p_σ; as transições entre os estados q_A, q_y, são atualizadas, substituindo o estado q_A pelo estado q_s; finalmente o registrador r_A é atualizado para apontar para q_s.

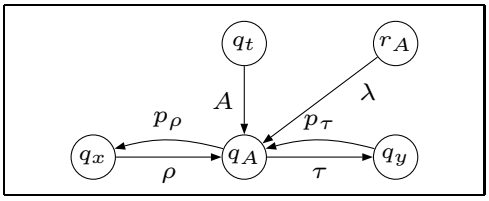


Figura 4.34: Antes de aplicar a produção $p_\sigma = A \to \sigma B$

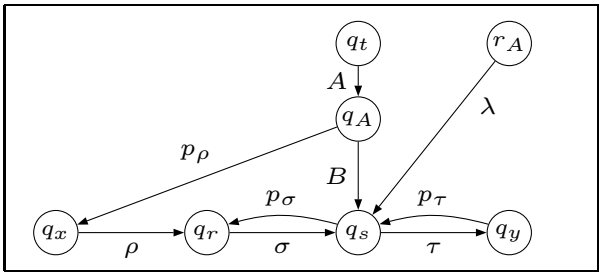


Figura 4.35: Depois de aplicar a produção $p_{i_1} = p_\sigma = A \to \sigma B$.

Para realizar estas transformações estruturais, o único dado que temos é o estado apontado pelo registrador r_A (o único dado *é o próprio registrador*); vamos mostrar uma seqüência de ações adaptativas elementares que realizam essas transformações estruturais; para sua operação, definimos três variáveis: q_A, q_x, q_y, e dois geradores q_r^* e q_s^*, para os novos estados; as ações elementares listadas a seguir devem de ser executadas na seqüência em que aparecem, de esquerda para a direita e de cima para baixo; tal seqüencialidade pode ser obtida por uso de ações iniciais e finais, omitidas para maior clareza:

/*calcula estados aos quais aplicar a transformação referente ao estado q_A*/
$?[(r_A, \lambda) \to q_A]$, $?[(q_x, \rho) \to q_A]$, $?[(q_y, p_\tau) \to q_A]$,
/*remoção de transições ao redor do estado q_A**********/
$-[(r_A, \lambda) \to q_A]$, $-[(q_x, \rho) \to q_A]$, $-[(q_y, p_\tau) \to q_A]$, $-[(q_A, \tau) \to q_y]$,
/*inserção de transições ao redor do estado q_A**********/
$+[(r_A, \lambda) \to q_s]$, $+[(q_x, \rho) \to q_r]$, $+[(q_r, \sigma) \to q_s]$, $+[(q_s, p_\sigma) \to q_r]$,
$+[(q_s, \tau) \to q_y]$, $+[(q_y, p_\tau) \to q_s]$, $+[(q_A, B) \to q_s]$

Para terminar de descrever as transformações estruturais, devemos adotar algum critério, segundo o qual se possa determinar em quais transições devem ser colocadas funções adaptativas, e a como essas funções adaptativas devem operar.

Se a expressão regular que descreve as derivações das palavras das linguagen não apresenta nenhuma estrela de Kleene, ou nenhum fecho transitivo "+", então a linguagem é formada por um conjunto finito de palavras, e a determinação de

uma árvore de análise para cada uma delas é trivial, podendo ser feita por busca exaustiva. Portanto, vamos supor que a expressão regular contenha, pelo menos, uma estrela de Kleene; o processo descrito a seguir pode ser adaptado ao caso do fecho transitivo "+".

Em particular, a aplicação da primeira produção de uma (sub) expressão regular que expressa repetições por estrela de Kleene, por exemplo, $(p_{i_1} \dots p_{i_n})^*$, deve fazer modificações estruturais, de modo que fique previsto, na nova estrutura, a construção da árvore de análise correspondente às demais produções que aparecem nessa expressão regular.

Assim, a mudança estrutural, para cada uma das produções $p_{i_2}, \dots, p_{i_n}$, é similar à descrita pelas figuras 4.34 e 4.35 e a transição adaptativa deve ser "acoplada" ao estado que é apontado pelo registrador referente ao não-terminal que consta no lado esquerdo de p_{i_1}; esse estado deve fazer parte do prefixo viável, no caminho que vai do estado q_0 ao estado q_1, e a transição adaptativa deve consumir algum símbolo terminal; vamos assumir que o símbolo a ser consumido é μ, e que a função adaptativa é $\mathcal{A}_\mu$. A Fig. 4.36 resume essas considerações:

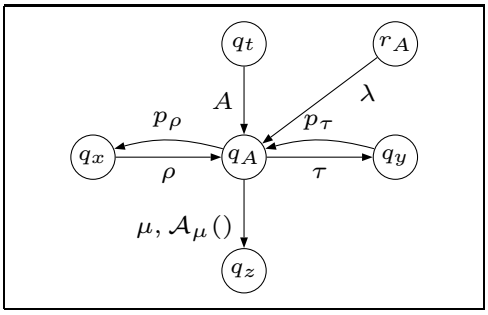


Figura 4.36: Antes de aplicar a produção $p_{i_1} = p_\sigma = A \to \sigma B$

Nesse caso, as transformações estruturais são similares ao caso descrito nas figuras 4.34 e 4.35, com a diferença que o estado q_z deve ser intercalado entre os estados q_A e q_s; desta vez, definimos quatro variáveis q_A, q_x, q_y, q_z, e dois geradores q_r^* e q_s^*:

/*calcula estados aos quais aplicar a transformação referente ao estado q_A****/
$?[(r_A, \lambda) \to q_A], \quad ?[(q_x, \rho) \to q_A], \quad ?[(q_y, p_\tau) \to q_A], \quad ?[(q_A, \mu) \to \{\mathcal{A}_\mu()\} q_z],$
/*remoção de transições ao redor do estado q_A**********/
$-[(r_A, \lambda) \to q_A], \quad -[(q_x, \rho) \to q_A], \quad -[(q_y, p_\tau) \to q_A], \quad -[(q_A, \tau) \to q_y],$
$-[(q_A, \mu) \to \{\mathcal{A}_\mu()\} q_z],$
/*inserção de transições ao redor do estado q_A**********/
$+[(r_A, \lambda) \to q_z], \quad +[(q_x, \rho) \to q_r], \quad +[(q_r, \sigma) \to q_z], \quad +[(q_z, p_\sigma) \to q_r],$
$+[(q_z, \tau) \to q_y], \quad +[(q_y, p_\tau) \to q_z], \quad +[(q_A, B) \to q_z], \quad +[(q_A, \mu) \to \{\mathcal{A}_\mu()\} q_s]$

A Fig. 4.37 a seguir mostra o efeito das transformações elementares anteriores; o símbolo σ foi intercalado, no prefixo viável, entre os símbolos ρ e τ; o registrador foi atualizado e a função adaptativa foi re-colocada, para poder realizar alterações em uma próxima aplicação da produção p_{i_1} no estado apontado pelo registrador r_A:

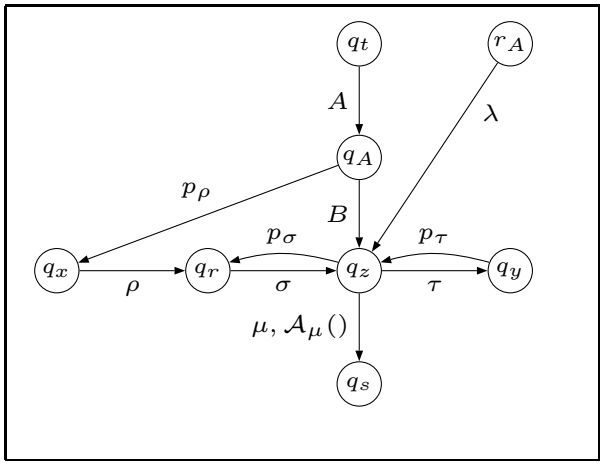


Figura 4.37: Depois de aplicar a produção $p_{i_1} = p_\sigma = A \to \sigma B$

Caso existam conflitos (por exemplo, se em um estado, apontado por r_{A_i}, existem duas produções que podem ser aplicadas, gerando o mesmo terminal), eles devem ser resolvidos anexando, em transições *que partem do estado em que acontece o conflito*, funções adaptativas que se encarregam de cada uma das alternativas possíveis. Essa estratégia de resolução de conflitos pode implicar em uma reescrita maciça de várias transições, que indicam qual produção foi aplicada, para produzir um dado símbolo terminal.

Vamos ilustrar todas as considerações anteriores em um exemplo único, na próxima seção, mas antes vamos fazer alguns comentários adicionais.

Os espaços de memória que servem para armazenar os estados em que devem ser aplicadas as produções, foram chamados de "registradores" numa alusão ao modelo de Shutt[7], pois a ideia básica é simular a memória do modelo *r-SMFA*, onde r é um número natural. Esses registradores não são incluídos nas figuras do exemplo da próxima seção para não poluir os desenhos, mas estão presentes nas funções adaptativas.

As operações descritas nesta seção não estão suficientemente detalhadas para constituir uma prova de que sempre é possível obter um analisador como o descrito. No capítulo 5 descrevemos alguns detalhes que devem ser levados em conta para desenvolver tal prova, bem como alguns problemas relacionados.

4.3.1 Analisador Adaptativo para G'_{10} da observação 2.4.2

Para iniciar a construção do analisador baseado em AA, fazemos a construção correspondente à primeira produção da gramática, criando o caminho viável entre q_0 e q_1. Obtém-se o autômato da Fig. 4.38.

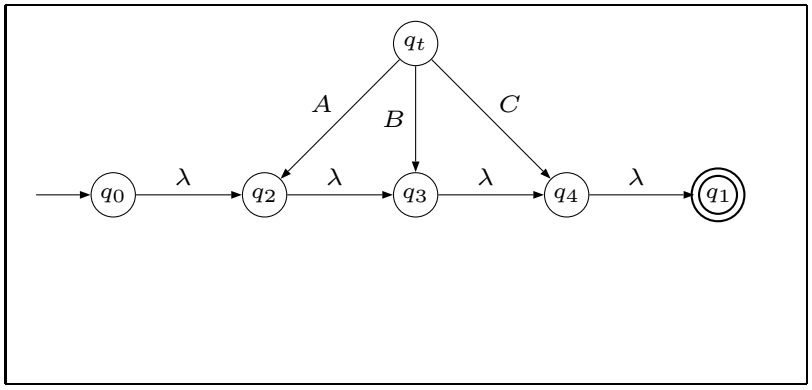


Figura 4.38: Início

Construímos, em seguida, as transições adaptativas que fazem a *análise* das palavras representadas pela expressão regular que define a linguagem de controle; os índices de cada função adaptativa provêm da produção à qual estão associadas; $\mathcal{A}_2$ encarrega-se de gerar a subárvore correspondente às repetições das aplicações das produções p_2, p_3 e p_4; $\mathcal{A}_5$ faz com que as últimas a, b, c reconhecidas sejam associadas às produções p_5, p_6 e p_7, respectivamente; ambas as funções adaptativas dependem dos registradores r_A, r_B e r_C, para determinar a qual nó da árvore aplicar as transformações:

$\mathcal{A}_5()\{$
/*variáveis para a manipulação das transições*/
$q_{Ai}, q_{Bi}, q_{Ci}, q_{Af}, q_{Bf}, q_{Cf},$
/*usa os registradores para determinar os estados aos quais aplicar a transf.*/
$?[(r_A, \lambda) \rightarrow q_{Ai}]$, $?[(r_B, \lambda) \rightarrow q_{Bi}]$, $?[(r_C, \lambda) \rightarrow q_{Ci}]$
/*elimina as transições que consomem os últimos símbolos a, b e c*/
$-[(q_{Ai}, p_2) \rightarrow q_{Af}]$, $-[(q_{Bi}, p_3) \rightarrow q_{Bf}]$, $-[(q_{Ci}, p_4) \rightarrow q_{Cf}]$,
/*marca os últimos símbolos a, b e c como consumidos por p_5, p_6, e p_7*/
$+[(q_{Ai}, p_5) \rightarrow q_{Af}]$, $+[(q_{Bi}, p_6) \rightarrow q_{Bf}]$, $+[(q_{Ci}, p_7) \rightarrow q_{Cf}]$

$\}$

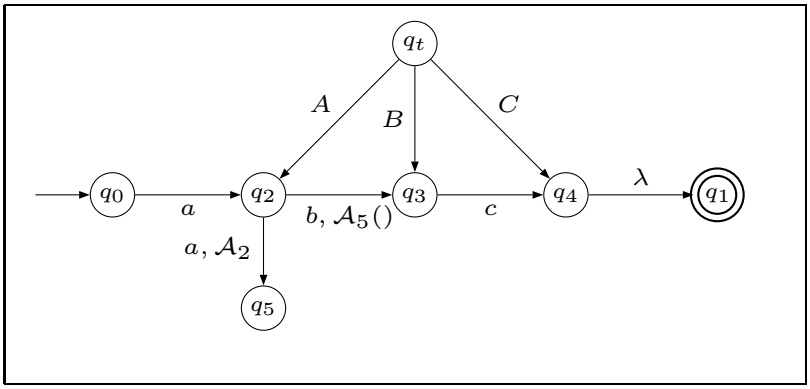


Figura 4.39: Autômato Finito Adaptativo inicial

Na Fig 4.39, vemos o analisador pronto para iniciar sua execução; como primeiro experimento, vamos efetuar o reconhecimento da palavra *abc*; deve-se notar que os registradores r_A, r_B e r_C (omitidos na figura) apontam para q_2, q_3 e q_4 respectivamente; a Fig. 4.40, mostra o efeito da aplicação da função adaptativa $\mathcal{A}_5$; note-se que não foram realizadas mudanças estruturais, apenas a semântica dos três últimos passos do reconhecimento foi *estabelecida* indicando quais produções foram aplicadas na árvore de análise.

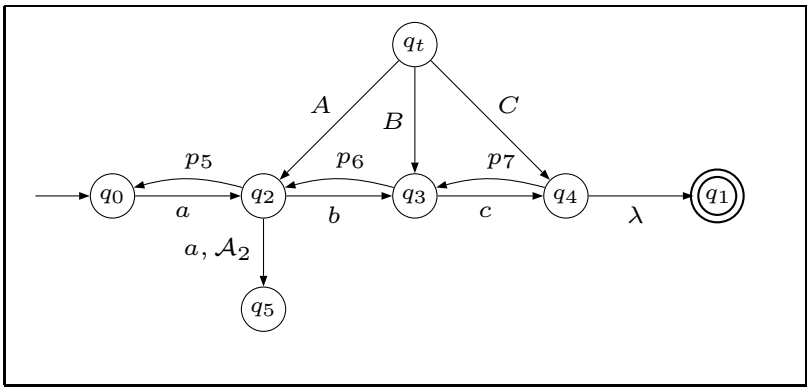


Figura 4.40: Após aceitar *abc*

Precisamos agora descrever a função adaptativa $\mathcal{A}_2$; como esta função adaptativa realiza mudanças estruturais complexas, vamos ilustrar seu funcionamento (que, para o programador de autômatos adaptativos, deve ser atômico) em vários passos, a fim de ilustrar melhor a estratégia adaptativa usada. Observamos primeiramente que, se o segundo símbolo a foi consumido, deve ser porque os primeiros símbolos a, b e c estão associados ao uso das produções p_2, p_3 e p_4 na árvore de análise. Isto implica também que deve ser consumido um segundo símbolo b e um segundo símbolo

c. Tudo isto deve ocorrer mantendo o caminho viável entre q_0 e q_1. Consideramos agora que a palavra a ser consumida é *aabbcc*.

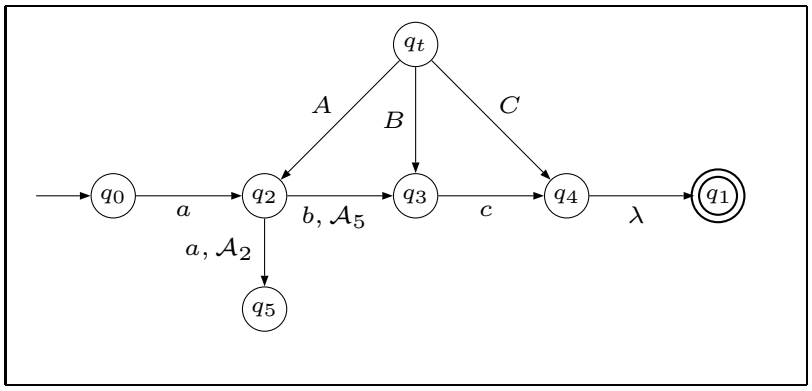


Figura 4.41: Depois de ter consumido o prefixo a de *aabbcc*. Controle em q_2.

Na Fig. 4.41 vemos a árvore de análise depois de ter consumido o primeiro símbolo a da palavra *aabbcc*; nada é modificado até esse ponto. Na Fig. 4.42 vemos a primeira alteração a ser realizada pela função adaptativa $\mathcal{A}_2$: estabelecer que os primeiros símbolos a, b e c são consumidos pela aplicação, na árvore de análise das produções p_2, p_3 e p_4 respectivamente.

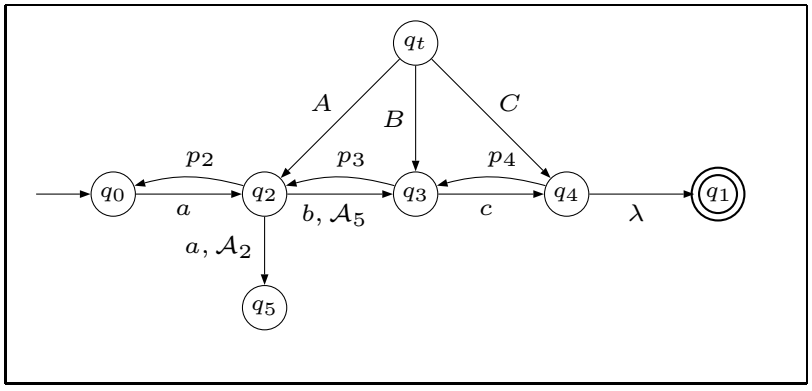


Figura 4.42: Consumindo o prefixo *aa* de *aabbcc* (1). Controle em q_5.

A função adaptativa $\mathcal{A}_2$ agora deve estabelecer que a segunda letra a consumida corresponde também a uma aplicação da produção p_2; para isto é criado o estado q_6; deve ser criada também uma nova transição adaptativa (que partindo de q_5 produz a criação do estado de chegada q_7) adaptativa para levar em conta de um terceiro a, e o registrador r_A deve ser atualizado para apontar a q_5. O resultado é mostrado na Fig. 4.43.

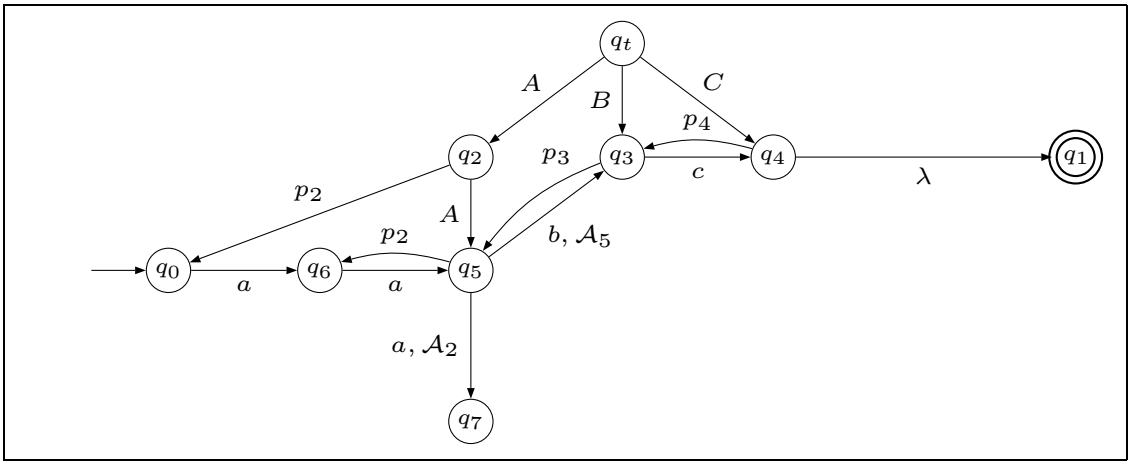


Figura 4.43: Consumindo o prefixo aa de $aabbcc(2)$. Controle em q_5.

Agora, a função adaptativa $\mathcal{A}_2$ deve criar a transição que consome o segundo símbolo b da palavra $aabbcc$, e que fica associada à aplicação da produção p_3. O registrador r_B deve ser atualizado para apontar o estado q_9. O caminho viável é mantido. Veja Fig. 4.44.

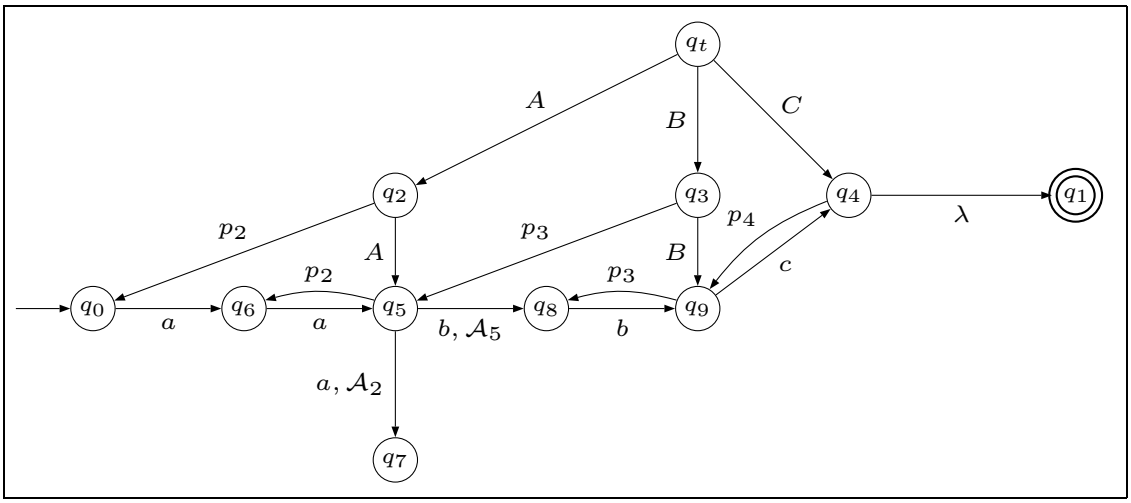


Figura 4.44: Consumindo o prefixo aa de $aabbcc(3)$. Controle em q_5.

A função adaptativa $\mathcal{A}_2$ agora deve estabelecer que a segunda letra c consumida corresponde também a uma aplicação da produção p_4; para isto, foram criados os estados q_{10} e q_{11}; e o registrador r_A é atualizado para apontar q_{11}; o caminho viável é mantido trocando-se a transição vazia $\delta(q_4, \lambda) = q_1$ pela transição vazia $\delta(q_{11}, \lambda) = q_1$. Ver Fig. 4.45.

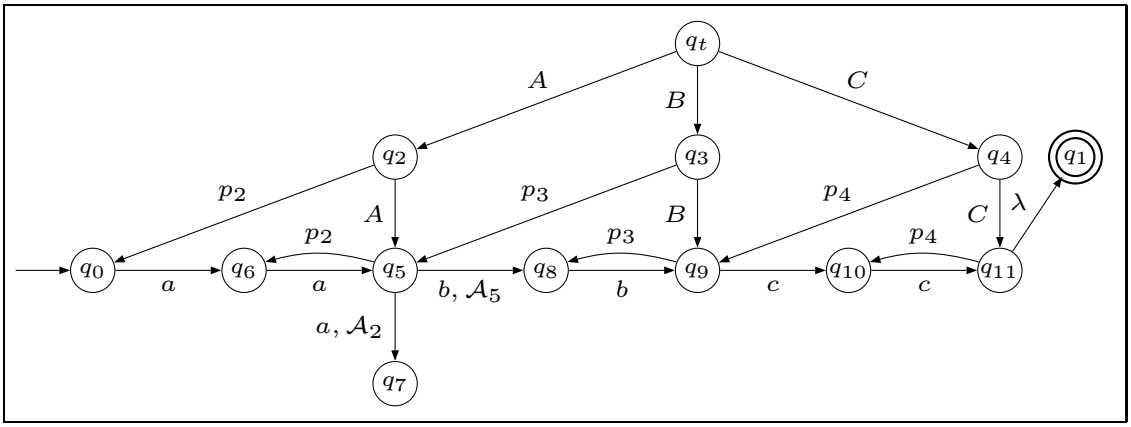


Figura 4.45: Consumindo o prefixo *aa* de *aabbcc*(4). Controle em q_5.

A Fig. 4.46 mostra o analisador depois de consumir o primeiro símbolo b. A função adaptativa $\mathcal{A}_5$ foi ativada, e substitui as transições $\delta(q_5, p_2) = q_6$, $\delta(q_9, p_3) = q_8$ e $\delta(q_{11}, p_4) = q_{10}$ pelas transições $\delta(q_5, p_5) = q_6$, $\delta(q_9, p_6) = q_8$ e $\delta(q_{11}, p_7) = q_{10}$; o controle fica em q_8.

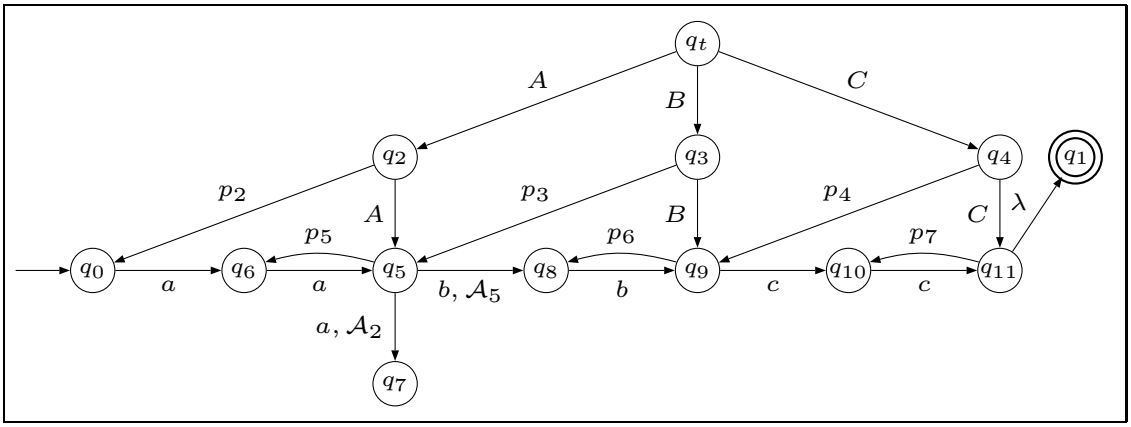


Figura 4.46: Depois de consumir o prefixo *aab* de *aabbcc*(5). Controle em q_8.

A partir daqui, o autômato não sofre mais modificações, consumindo o sufixo *bcc* da cadeia e deixando, codificada na sua estrutura, a árvore de análise para a cadeia reconhecida. Com base na observação do comportamento do analisador, pode-se agora definir, finalmente, a função adaptativa $\mathcal{A}_2()$, na qual incluímos números de linha para poder ilustrar como separá-la para garantir a seqüencidalidade das ações elementares.

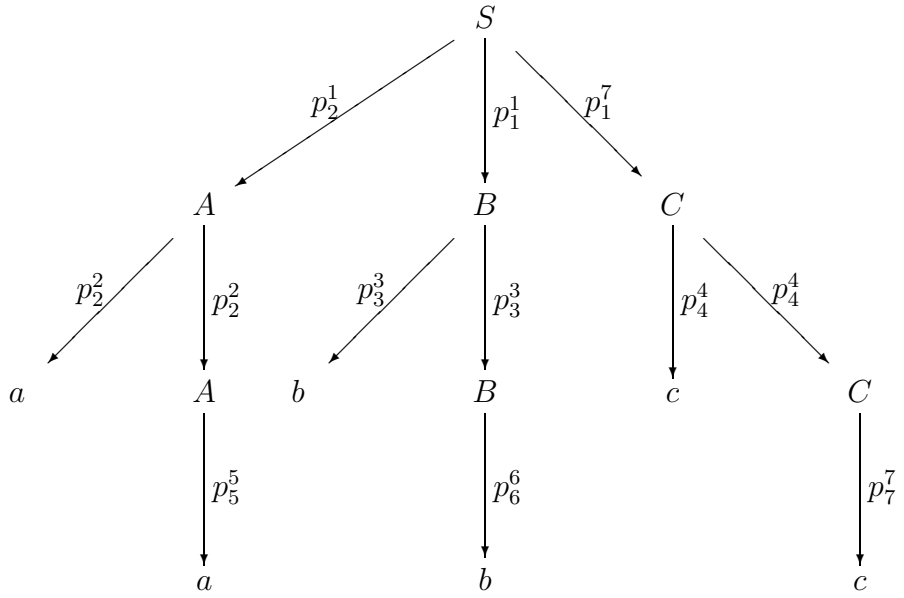
$\mathcal{A}_2()\{$

1 /*variáveis para a manipulação das transições*/
2 $q_A, q_B, q_C, q_x, q_y, q_z,$
3 /*geradores para mudanças estruturais*/
4 $q^*_{Ar}, q^*_{As}, q^*_{Br}, q^*_{Bs}, q^*_{Cr}, q^*_{Cs},$
5 /*usa os registradores para determinar os estados onde aplicar a transf.*/
6 $?[(r_A, \lambda) \rightarrow q_A], \quad ?[(r_B, \lambda) \rightarrow q_B], \quad ?[(r_C, \lambda) \rightarrow q_C]$
7 /*elimina a transições que consomem as últimas letras a, b e c*/
8 $-[(q_A, p_5) \rightarrow q_x], \quad -[(q_B, p_6) \rightarrow q_y], \quad -[(q_C, p_7) \rightarrow q_z],$
9 /*marca as últimas letras a, b e c como consumidas por p_2, p_3, e p_4*/
10 $+[(q_A, p_2) \rightarrow q_x], \quad +[(q_B, p_3) \rightarrow q_y], \quad +[(q_C, p_4) \rightarrow q_z]$
11 /*calcula estados onde aplicar a transformação referente ao estado q_A*/
12 $?[(q_x, a) \rightarrow q_A], \quad ?[(q_y, p_3) \rightarrow q_A], \quad ?[(q_A, a) \rightarrow \{\mathcal{A}_2\}q_z],$
13 /*eliminação de transições ao redor do estado q_A*/
14 $-[(q_x, a) \rightarrow q_A], \quad -[(q_y, p_3) \rightarrow q_A], \quad -[(q_A, b) \rightarrow \{\mathcal{A}_6\}q_y],$
15 $-[(q_A, a) \rightarrow \{\mathcal{A}_2\}q_z],$
16 /*adição de transições ao redor do estado q_A*/
17 $+[(q_x, a) \rightarrow q_{Ar}], \quad +[(q_{Ar}, a) \rightarrow q_z], \quad +[(q_z, p_2) \rightarrow q_{Ar}],$
18 $+[(q_z, a) \rightarrow \{\mathcal{A}_2\}q_{As}], +[(q_z, b) \rightarrow \{\mathcal{A}_6\}q_y], +[(q_y, p_3) \rightarrow q_z],$
19 $+[(q_A, A) \rightarrow q_z]$
20 /*calcula estados onde aplicar a transformação referente ao estado q_B*/
21 $?[(q_x, a) \rightarrow q_B], \quad ?[(q_y, p_3) \rightarrow q_B], \quad ?[(q_B, a) \rightarrow \{\mathcal{B}_2\}q_z],$
22 /*eliminação de transições ao redor do estado q_B*/
23 $-[(q_x, a) \rightarrow q_B], \quad -[(q_y, p_3) \rightarrow q_B], \quad -[(q_B, b) \rightarrow \{\mathcal{B}_6\}q_y],$
24 $-[(q_B, a) \rightarrow \{\mathcal{B}_2\}q_z],$
25 /*adição de transições ao redor do estado q_B*/
26 $+[(q_x, a) \rightarrow q_{Br}], \quad +[(q_{Br}, a) \rightarrow q_z], \quad +[(q_z, p_2) \rightarrow q_{Br}],$
27 $+[(q_z, a) \rightarrow \{\mathcal{B}_2\}q_{Bs}], +[(q_z, b) \rightarrow \{\mathcal{B}_6\}q_y], +[(q_y, p_3) \rightarrow q_z],$
28 $+[(q_B, B) \rightarrow q_z]$
29 /*calcula estados onde aplicar a transformação referente ao estado q_C*/
30 $?[(q_x, a) \rightarrow q_C], \quad ?[(q_y, p_3) \rightarrow q_C], \quad ?[(q_C, a) \rightarrow \{\mathcal{C}_2\}q_z],$
31 /*eliminação de transições ao redor do estado q_C*/
32 $-[(q_x, a) \rightarrow q_C], \quad -[(q_y, p_3) \rightarrow q_C], \quad -[(q_C, b) \rightarrow \{\mathcal{C}_6\}q_y],$
33 $-[(q_C, a) \rightarrow \{\mathcal{C}_2\}q_z],$
34 /*adição de transições ao redor do estado q_C*/
35 $+[(q_x, a) \rightarrow q_{Cr}], \quad +[(q_{Cr}, a) \rightarrow q_z], \quad +[(q_z, p_2) \rightarrow q_{Cr}],$
36 $+[(q_z, a) \rightarrow \{\mathcal{C}_2\}q_{Cs}], +[(q_z, b) \rightarrow \{\mathcal{C}_6\}q_y], +[(q_y, p_3) \rightarrow q_z],$
37 $+[(q_C, C) \rightarrow q_z]$
}

Para impor a seqüencialidade da ações adaptativas elementares introduzimos uma função adaptativa inicial $\mathcal{A}_{2inicial}()$ e uma função adaptativa final $\mathcal{A}_{2final}()$; a função adaptativa $\mathcal{A}_2$ executa as linhas 20-28; a função adaptativa inicial $\mathcal{A}_{2inicial}$ executa, usando uma função adaptativa inicial, as linhas 6-10 e executa em seu próprio corpo as linhas 12-19; a função adaptativa final $\mathcal{A}_{2final}$ executa as linhas 29-37.

$\mathcal{A}_2()\{$
$\mathcal{A}_{2inicial}(\ldots)$
$/ * linhas 20 - 28 * /$
$\mathcal{A}_{2final}(\ldots)$
$\}$

Para concluir, apresentamos a árvore de análise decodificada a partir da estrutura do autômato finito adaptativo obtida com a ajuda do analisador. As arestas foram codificadas com as produções aplicadas; os superíndices das arestas denotam o *instante do análise* no qual a produção foi aplicada.



Árvore de análise, decodificada da estutura do autômato finito adaptativo.

4.4 Resumo

Neste capítulo:

1. Descrevemos como obter um *analisador ascendente baseado em autômatos-pilha* para as gramáticas livres de contexto com linguagem de controle regular e verificação de aparência.

2. Descrevemos os algoritmos *EliminaColapsadores*, *EliminaPontes-*λ, que servem como infraestrutura básica para obter o analisador ascendente mencionado,

3. Discutimos, de forma geral, uma estratégia que pode ser usada para obter um *analisador descendente baseado em autômatos finitos adaptativos* para algumas (como aquelas descritas na seção 4.5) gramáticas livres de contexto com linguagem de controle regular e verificação de aparência.

O analisadores mencionados e os algoritmos de eliminação de colapsadores e pontes-λ são de nossa autoria e foram desenvolvidos no decurso de nossa pesquisa. Cronologicamente, foi desenvolvido primeiro o analisador *ascendente*, depois os algoritmos (na ordem em que os mencionamos) e finalmente os analisadores descendentes.

Capítulo 5

Contribuições e temas para futuras pesquisas

Neste capítulo apresentamos um resumo dos principais resultados obtidos nesta pesquisa, e fazemos um balanço da contribuição que esses resultados trazem para a área. Comentamos também alguns temas que podem ser motivo de pesquisas futuras, assim como alguns problemas que ficaram em aberto durante esta pesquisa e fazemos conjecturas sobre as possíveis soluções destes problemas.

5.1 Principais contribuições.

No presente trabalho, nós resolvemos, em relação às gramáticas livres de contexto adaptativas com verificação de aparência (veja seção 3.2), as questões da equivalência com máquina de Turing, da eliminação de geradores, variáveis, ações adaptativas iniciais e finais, produções dependentes de contexto, e do fato de ser desnecessário gerar novos não-terminais ou novas produções.

Outras contribuições que este trabalho oferece são:

1. Especificação estritamente gramatical de condições dependentes de contexto (seção 3.2) e obtenção automática de analisadores capazes de reconhecer tais dependências de contexto (ascendentes na seção 4.3 e descendentes, com algumas restrições, na seção 4.5)

2. A análise, em modo adaptativo, de uma linguagem (seção 4.5), sem a necessidade de gerar, a priori, o reconhecedor ou analisador completo dessa linguagem.

5.2 Contribuições didáticas.

Como contribuições menos significativas, e também como subprodutos didáticos, produzidos durante esta pesquisa, podemos citar, entre outros, os seguintes:

1. Os diagramas de fluxo aqui utilizados para gramáticas programadas, que foram baseados e adaptados dos diagramas de Rosenkrantz[14],

2. O uso de autômatos finitos para representar a linguagem de controle das gramáticas com linguagem de controle regular.

3. Os analisadores ascendentes (não-determinísticos e dependentes de contexto), baseados em autômatos-pilha e obtidos a partir de gramáticas livres de contexto com linguagem de controle regular e verificação de aparência.

4. Os algoritmos *EliminaColapsadores* e *eliminaPontes-*λ para a remoção de transições-λ do conjunto de transições fornecido como saída pelo algoritmo *wirth2ape*.

5. Os analisadores descendentes baseados em autômatos finitos adaptativos e obtidos a partir de gramáticas livres de contexto com linguagem de controle regular e verificação de aparência.

5.3 Temas para futuras pesquisas.

5.3.1 Projeto e implementação de linguagens de programação.

Considerando que os dispositivos estudados nesta tese têm poder computacional de máquina de Turing, eles representam diferentes formas alternativas para ensaiar a sintaxe de uma linguagem de programação; portanto, eles podem também servir como a base para um sistema, mais complexo, que inclua um ambiente de execução de modo que permita fazer definição estritamente sintática de linguagens de programação, de forma similar ao que foi feito, historicamente, para a linguagem de programação ALGOL68[10].

Com efeito, a sintaxe e a semântica da linguagem de programação ALGOL68 foram totalmente descritas, no documento citado, em forma sintática, usando o mode-lo formal das gramáticas-W (citada na subseção 1.5.1), que possui poder computacional de máquina de Turing; em particular, para fazer a representação da semântica, foi usado o paradigma da semântica operacional[22]; isto significa que uma parte das construções básicas da linguagem foi implementada, de modo tal que todas as outras construções fossem executadas acima dessas construções básicas.

Qual a vantagem de nossa proposta sobre a estratégia descrita no parágrafo anterior? A vantagem é a simplicidade: embora gramáticas-W e gramáticas livres de contexto adaptativas possuam o mesmo poder computacional, as gramáticas-W precisam prever, nas suas produções, uma possível geração de uma quantidade infinita de produções livres de contexto, sendo que as gramáticas livres de contexto adaptativas trabalham sempre, não apenas com uma quantidade finita, mas fixa, de produções livres de contexto.

Para descrever um tema de pesquisa de uma possível aplicação dos analisadores descendentes adaptativos, descritos na seção 4.5, na área de projeto de linguagens de programação, vamos relembrar uma discussão que aconteceu durante o decorrer da disciplina "PCS 730[1] Projetos e Técnicas de Construção de Compiladores" no segundo período de aulas de 1998[2] e que pode ser resumida na seguinte pergunta:

> Por que, ao invés de fazer compiladores completos para linguagens de programação, não fazemos *"um sistema"* que gere, para cada programa a ser compilado, um programa executável que inclui "apenas" as partes da linguagem que o dado programa requer para sua execução?

ou, para ser mais preciso:

> Por que, ao invés de construir analisadores (completos) para linguagens de programação, baseados em reconhecedores livres de contexto para a sintaxe dessas linguagens, não criamos *"um sistema"* que gere, para cada programa a ser compilado, um programa executável que inclui "apenas" as partes da estrutura sintática da linguagem (ou seja: apenas uma *parte* do analisador) que o dado programa requer para seu funcionamento?

Por essa caraterística (*lazy*) de utilizar apenas a parte "necessária" da sintaxe da linguagem reconhecida para analisar um programa denotado em tal linguagem, vamos chamar tal sistema de *subcompilador*.

Os analisadores discutidos na seção 4.5 são uma primeira aproximação para a construção de *subcompiladores adaptativos*; de fato, no exemplo apresentado na seção 4.5.1, examinamos o resultado da execução do analisador adaptativo ao tratar as palavras *abc* e *aabbcc*; nesses casos, as estruturas sintáticas registradas nas árvores de análise codificadas no autômato finito adaptativo são diferentes; e cada uma delas indica estritamente *o que a palavra, necessita da linguagem*; generalizando, para o caso em que as "palavras" são programas de uma linguagem de alto nível, vamos obter um analisador, que é um autômato finito adaptativo pAA, que constrói a

[1] O código mudou depois para PCS 5730.

[2] Participei como ouvinte dessa edição da disciplina e obtive uma nota "A-moral".

árvore de análise de um dado programa fonte P, exercitando "apenas", a parte da sintaxe da linguagem reconhecida pelo pAA que se mostra estritamente necessária para fazer a análise sintática de P.

Nestas condições, pode-se colocar a seguinte pergunta:

Em que casos se mostra útil aplicar uma linguagem que seja analisada adaptativamente dessa forma?

Considerando que o tempo poupado é tempo de compilação do compilador da linguagem, é essa a economia que essa análise adaptativa pode oferecer a um usuário. Ou seja, pode-se reformular a pergunta:

Em que situações é custoso compilar um compilador?

A resposta a essa pergunta é: *quando a linguagem, que compilador deve aceitar, é mostruosamente grande com respeito aos recursos computacionais disponíveis.* Nesse contexto, podem-se procurar aplicações de subcompiladores em áreas onde os recursos de execução são limitados, por exemplo, execução remota, ou em ambientes de execução de dispositivos móveis.

5.3.2 Síntese do paradigma Adaptativo

Nossa principal preocupação durante o desenvolvimento desta pesquisa foi a de obter uma melhor compreensão dos dispositivos adaptativos e das técnicas adaptativas; essa preocupação é dirigida pelo objetivo de vir a formular uma síntese do paradigma adaptativo; isto é, explicar o que significa *paradigma adaptativo*. Um tal paradigma deve servir como modelo para explicar o comportamento de qualquer dispositivo que, durante sua operação, faça modificações na estrutura que o define, adicionando ou retirando elementos na mesma.

Em nosso caso, do ponto de vista teórico, nossos dispositivos adaptativos de estudo têm sido os autômatos e as gramáticas adaptativas que, embora possuam a vantagem de oferecer um ponto de vista unificado para o estudo de linguagens formais, têm contra si a enorme quantidade de detalhes associados a suas representações e respectivas notações. Em Neto[26] foi feita uma primeira tentativa sistemática para simplificar a notação do autômato adaptativo; essa simplificação, embora tenha conseguido atingir quase todos os seus objetivos, não conseguiu cobrir algumas questões fundamentais: por exemplo, não respondeu à pergunta sobre a real necessidade de parâmetros nas funções adaptativas.

É no contexto da simplificação das gramáticas adaptativas, descrita na seção 5.1, que esta tese representa uma significativa contribuição à síntese do paradigma adaptativo.

Por outro lado, a conversão canônica de gramáticas adaptativas em autômatos adaptativos, descrita na seção 4.1 de Iwai[1], não pode ser também aplicada às gramáticas livres de contexto adaptativas com verificação de aparência, pois elas não dispõem de produções dependentes de contexto. Contornamos essa dificuldade, parcialmente, com a construção dos analisadores descendentes baseados em autômatos adaptativos, discutida na seção 4.5 desta tese. Essa construção mostra que, para as gramáticas livres de contexto adaptativas nas quais somente o símbolo inicial da gramática somente produz uma cadeia de não-terminais e, é possível construir um parser adaptativo descendente. Entretanto, essa construção inclui variáveis e geradores, que gostariamos de ter evitado, para obtermos assim uma simplificação do modelo dos autômatos adaptativos similar à que foi alcançada para a simplificação das gramáticas adaptativas, que descrevemos na seção 5.1. Fica então, como tema de pesquisa, determinar se essa construção pode ser generalizada para qualquer gramática livres de contexto adaptativa ou então identificar a classe das gramáticas para as quais tal construção sempre é possível.

5.3.3 Paradigma adaptativo vs linguagens de programação

Embora, neste trabalho, não tenhamos logrado formular concretamente uma caracterização do Paradigma Adaptativo, ou do que poderia significar uma linguagem de programação de Paradigma Adaptativo, fizemos algumas contribuições que vinculam os dispositivos adaptativos estudados neste trabalho ao projeto de linguagens de programação.

Para projetar uma linguagem de paradigma adaptativo, faltaria experimentar diferentes sintaxes até conseguir uma que representasse adequadamente o paradigma adaptativo em alto nível.

5.4 Problemas em aberto

Nesta seção apresentamos um conjunto de problemas para os quais não tivemos a oportunidade de investigar solução ou para cujas soluções não nos foi possível ainda obter um resultado satisfatório.

Gostaríamos de iniciar essa discussão com duas "expressões de desejo" sobre a construção de subcompiladores adaptativos, descritos na subseção 5.3.1:

1. Seria conveniente que implementar um subcompilador adaptativo para uma linguagem de programação L não fosse mais complexo que implementar um compilador completo para a mesma linguagem de programação L.
2. Seria conveniente que a execução de um subcompilador adaptativo para uma linguagem de programação L fosse computacionalmente menos exigente que a execução de um compilador completo para a mesma linguagem de programação L.

Consideramos, a seguir, questões referentes às gramáticas livres de contexto adaptativas, com verificação de aparência.

Definição 5.4.1 *Seja $\mathcal{LA}(i,j)$, onde $i = 0, 1, 2, 2-\lambda, 3$, e onde $0 \leq j \leq 1$ a classe das linguagens geradas pelas gramáticas adaptativas baseadas em gramáticas de tipo-i, com $i = 0, 1, 2, 2-\lambda, 3$, segundo a hierarquia de Chomsky, e onde $j = 0$ denota que a gramática não possui produções a serem aplicadas em modo de verificação de aparência e $j = 1$ denota que a gramática possui produções a serem aplicadas em modo de verificação de aparência.*

Problema em aberto 5.4.1 *Gramáticas adaptativas baseadas em gramáticas regulares com verificação de aparência têm poder de máquina de Turing, isto é, é válida a seguinte equação:*

$$\mathcal{LA}(3,1) = \mathcal{L}_0?$$

Mais geralmente podemos listar 8 problemas em aberto, incluindo o anterior:

Problema em aberto 5.4.2 *Completar a tabela, na qual cada entrada representa a posição de $\mathcal{LA}(i,j)$ com respeito às linguagens na hierarquia de Chomsky:*

$i \backslash j$	0	1
0	$\mathcal{L}_0$	
1		
2		$\mathcal{L}_0$
$2-\lambda$		
3		

Sobre os analisadores baseados em autômatos finitos adaptativos, queremos assinalar alguns problemas em aberto:

Problema em aberto 5.4.3 (Teórico) *Não temos um teorema que garanta que a transformação descrita na seção 4.5 sempre se aplica, ou seja, essa transformação é mais uma heurística que um algoritmo.*

A discussão feita na seção 4.5 pode servir como ponto de partida para um tal teorema, levando em consideração os casos não tratados; por exemplo, deve-se admitir, numa prova geral, que a primeira produção a ser aplicada tenha no lado direito, terminais e não-terminais. Parece razoável que a aplicação de uma produção auto-recursiva central possa ser simulada pela aplicação de várias produções recursivas à esquerda. A verificação de aparência deve ter o efeito de transições em vazio no analisador; portanto deve-se investigar se isso pode introduzir não-determinismos ou não.

Como os analisadores descendentes adaptativos obtidos na seção 4.5 usam variáveis e geradores, é natural colocar o seguinte problema:

Problema em aberto 5.4.4 (Teórico) *Será possível obter uma subclasse dos autômatos adaptativos, que não requera o uso de geradores nem de variáveis, e que seja equivalente à classe das gramáticas livres de contexto adaptativas com verificação de aparência?*

Parte da dificuldade em formalizar a construção dos analisadores adaptativos descritos na seção 4.5 é devida ao seguinte problema, que surgiu ao se tentar usar essa construção para obter um analisador adaptativo a partir da gramática G_{12} do exemplo 2.4.3:

Problema em aberto 5.4.5 (Prático) *Em caso de produções que duplicam um não-terminal, é necessário criar on-line novos registradores, e conservar "registro" na memória do AA, dos nomes desses novos registradores.*

Mesmo que isso seja resolvido (alterando o modelo dos AA para obter acesso aos nomes dos geradores, por exemplo), isso *não é uma simplificação do modelo AA.*

Outros problemas que devem ser pesquisados são:

Problema em aberto 5.4.6 (Prático) *Obter analisadores, como os apresentados na seção 4.5, para linguagens geradas pelas:*

1. *gramáticas programadas,*

2. *gramáticas matriciais, e*

3. *gramáticas periodicamente variantes no tempo.*

Supomos que esses últimos deveriam mapear diretamente a $TVPDA$[6], mas não queremos arriscar ainda uma conjectura nesse caso.

Em particular, sabemos que as gramáticas matriciais são equivalentes às redes de Petri: veja Dassow[3], páginas 267-270, para a simulação de redes de Petri pelas gramáticas matriciais e Hauschildt[36], para a simulação de gramáticas matriciais por redes de Petri. Essa equivalência deveria poder ser explorarada para criar um analisador para as linguagens geradas pelas gramáticas matriciais.

5.5 Conjecturas

As tabelas 3 e 4 na página 184 de Salomaa[2] (veja o final da seção $A.1$ do apêndice) sugere os seguintes valores para os problemas abertos acima.

Conjectura 5.5.1 *Para os casos sem verificação de aparência:*

	$\mathcal{LA}(i,0), i = 0,1,2,2-\lambda,3$
0	$\mathcal{L}_0$
1	$\mathcal{L}_0$ *ou* $\mathcal{L}_1 \subset \mathcal{LA}(1,0) \subset \mathcal{REC}$ *ou* $\mathcal{L}_1$
2	$\mathcal{L}_0$
$2-\lambda$	$\mathcal{L}_0^{'}$ *ou* $\mathcal{L}_1^{'} \subset \mathcal{LA}(2-\lambda,0) \subset \mathcal{REC}^{'}$ *ou* $\mathcal{L}_2^{'} \subset \mathcal{LA}(2-\lambda,0) \subset \mathcal{L}_1^{'}$
3	$\mathcal{L}_0$ *ou* $\mathcal{L}_2$ *ou* $\mathcal{L}_3$

onde $\mathcal{REC}$ é a classe das linguagens recursivas e o apóstrofe simple indica que foram excluídas da classe as linguagens que contêm a palavra vazia λ. Para os casos com verificação de aparência,

	$\mathcal{LA}(i,1), i = 0,1,2,2-\lambda,3$
0	$\mathcal{L}_0$
1	$\mathcal{L}_0$ *ou* $\mathcal{L}_1 \subset \mathcal{LA}(1,1) \subset \mathcal{REC}$ *ou* $\mathcal{L}_1$
2	$\mathcal{L}_0$
$2-\lambda$	$\mathcal{L}_0^{'}$ *ou* $\mathcal{L}_1^{'} \subset \mathcal{LA}(2-\lambda,1) \subset \mathcal{REC}^{'}$ *ou* $\mathcal{L}_2^{'} \subset \mathcal{LA}(2-\lambda,1) \subset \mathcal{L}_1^{'}$
3	$\mathcal{L}_0$ *ou* $\mathcal{L}_2$ *ou* $\mathcal{L}_3$

O teorema 2.1.2, sobre gramáticas matriciais sob derivações mais à esquerda, sugere a seguinte:

Conjectura 5.5.2 *Gramáticas livres de contexto adaptativas sob "alguma noção" de derivação mais à esquerda, sem verificação de aparência, identificam a classe $\mathcal{L}_1$ das linguagens dependentes de contexto.*

Uma sugestão para tentar provar essa conjectura é desenvolver uma simulação das gramáticas matriciais sob derivações mais à esquerda usando Gramáticas livres de contexto adaptativas. Deve-se notar que a noção de derivação mais à esquerda a ser usada não pode ser, simplesmente, a noção de derivação mais à esquerda das gramáticas na hierarquia de Chomsky, pois, no caso das gramáticas matriciais sob derivações mais à esquerda, é apenas a primeira produção da matriz que é aplicada mais à esquerda como na hierarquia de Chomsky; as outras produções da matriz podem (ou não) ser aplicadas mais à esquerda, como na hierarquia de Chomsky.

A simulação dos *SMFA*, feita pelos autômatos finitos adaptativos descritos na seção 4.5, pode ser usada como base para um estudo posterior das classes das linguagens aceitas pelos analisadores baseados em autômatos adaptativos definidos nessa seção. A ideia básica é utilizar os resultados de Shutt e Rubinstein[37], [38], e Wang[39], sobre o poder computacional dos *r-SMFA*.

Essa mesma idéia pode ser usada para classificar o poder computacional de subclasses de autômatos adaptativos, por exemplo, projetando uma subclasse dos autômatos adaptativos para simular os autômatos auto-montáveis *k-NFA*, ganha-se, para a classe dos autômatos adaptativos, a classificação do poder computacional dos *k-NFA* descrita por Klein[21].

Em particular, se $\mathcal{L}_{pAA}$ é classe das linguagens aceitas pelo analisadores baseados em autômatos finitos adaptativos descritos na seção 4.5, então a tabela 3, na página 184 de Salomaa[2] (veja o final da seção $A.1$ do apêndice A), sugere a seguinte:

Conjectura 5.5.3

$$\mathcal{L}_2^{'} \subset \mathcal{L}_{pAA} \subset \mathcal{L}_1^{'}$$

onde $\mathcal{L}_2^{'}$ é a classe das linguagens livres de contexto que não contêm a palavra vazia λ e $\mathcal{L}_1^{'}$ é a classe das linguagens dependentes de contexto que não contêm a palavra vazia λ.

A conjectura é falsa se i) existe uma linguagem livre de contexto, que não contém a palavra vazia λ, que não pode ser aceita e analisada por nenhum autômato finito adaptativo como os descritos na seção 4.5, ou então se ii) a classe $\mathcal{L}_{pAA}$ inclui alguma linguagem recursivamente enumerável que não contém a palavra vazia λ e que não é dependente de contexto.

Como os analisadores baseados em autômatos finitos adaptativos descritos na seção 4.5, operam em forma descendente eles poderiam ser adequados para aceitar as linguagens geradas pelas gramáticas $LL[k]$; essas linguagens formam uma hierarquia infinita, propriamente contida na classe das linguagens geradas pelas gramáticas $LR[k]$ (proposições 2.12 e 2.14, páginas 229 e 230 de Salomaa[2]); isso sugere a nossa última conjectura:

Conjectura 5.5.4 *Se $\mathcal{L}2_{pAA}$ é a classe das linguagens aceitas pelo analisadores baseados nos autômatos finitos adaptativos descritos na seção 4.5, construidos a partir de gramáticas com linguagem de controle regular que geram linguagens livres de contexto, então*

1. *A classe das linguagens geradas pelas gramáticas LL[k] é aceita por uma subclasse de $\mathcal{L}2_{pAA}$, que denominaremos $\mathcal{LL}_{pAA}[k]$.*

2. *A classe das linguagens geradas pelas gramáticas LR[k] é aceita por uma superclasse de* $\mathcal{L}2_{pAA}$ *que denominaremos* $\mathcal{LR}_{pAA}[k]$.

Se a conjectura 5.5.3 for verdadeira então é falsa a afirmativa do ponto 2 da conjectura 5.5.4.

Referências Bibliográficas

[1] Iwai, Margarete Keiko. **Um formalismo gramátical adaptativo para linguagens dependentes de contexto.** Tese de Doutorado.Escola Politécnica da USP, 2000.

[2] Salomaa, Arto. **Formal Languages.** ACM Monographs Series, Academic Press, (1973).

[3] Dassow, Jürgen; Paun, Gheorghe. **Regulated Rewriting in Formal Language Theory.** EATCS Monographs on Theoretical Computer Science, volume 18, 308 páginas, 1989. Springer-Verlag. ISBN 3-540-51414-7. ISBN 0-387-51414-7.

[4] Dassow, Jürgen; Paun, Gheorghe; Salomaa, Arto. **Grammars with controled derivations.** in Rozemberg, G. Salomaa, Arto (Ed). Handbook of formal languages. 1997.

[5] Bravo, César; Neto, João, José. **Building Context-Sensitive Analisadores from CF Grammars with Regular Control Language.** Lecture Notes in Computer Science, Volume 2759, pp 306-308. Springer-Verlag Heidelberg. ISSN: 0302-9743. 2003.

[6] Krithivasan, Kamala. **Time Variant Pushdown Automata.** International Journal of Computer Mathematics, Vol 24, pp. 223-236. 1988.

[7] Shutt, John. **Self-Modifying Finite Automata.** Computer Science technical Report WPI-CS-TR-93-11, Worcester Polytechnic Institute, December 1993.

[8] Neto, João J. **Contribuições à metodologia da construção de compiladores.** Tese de Livre Docência, Escola Politécnica da Universidade de São Paulo, 1993.

[9] Neto, João J. **Adaptive automata for context-dependent languages.** ACM SIGPLAN Notices Volume 29, No 9,115-124, September 1994.

[10] van Wijngaarden et al. **Revised Report on the Algorithmic Language ALGOL 68.** Berlin: Springer-Verlag, 1976.

[11] Shutt, John. **Recursive Adaptable Grammars.** Master thesis, Worcester Polytechnic Institute, (1993).

[12] Jackson, Quinn Tyler. **Disambiguation as a Quantifiable Computational Process**, Perfection, vol. 1, n. 3, 2000. Disponível em: citeseer.nj.nec.com/jackson00disambiguation.html. Acesso em 22 mar. 2003

[13] Ábrahám, S. **Some questions of language theory.** International Conference on Computational Linguistics, 1965. pp 1-11

[14] Rosenkrantz, Daniel J. **Programmed Grammars and Classes of Formal Languages.** Journal of the Association for Computer Machinery, Vol 16, No 1, January 1969, pp 107-131.

[15] Salomaa, Arto. **Periodically Time-Variant Context-Free Grammars.** Information and Control, 17, (1970), 294-311.

[16] Ibarra, O **Simple matrix languages.** Information and Control, 17, 1970. pp 359-394

[17] Rozenberg, G.; Salomaa, A. (Eds.) **Handbook of Formal Languages.** Springer-Verlag, Vol. 1, 2, 3, 1997

[18] Salomaa, Arto. **On Finite Automata with a Time-Variant structure.** Information and Control, 13, pp 85-98, (1968).

[19] Krithivasan, Kamala. **Time Variant Finite Automata.** International Journal of Computer Mathematics, Vol 19, pp. 103-123. 1986.

[20] Shutt, John. **Self-Modifying Finite Automata : Power and Limitations.** Computer Science technical Report WPI-CS-TR-95-4, Worcester Polytechnic Institute, (1995).

[21] Klein, Andreas; Kutrib, Martin. **Self-Assembling Finite Automata.** IFIG Research Report 0201, January 2002. Universität Giessen.

[22] Marcotty, Michael; Ledgard, Henry F.; Bochmann, Gregor V. **A Sampler of Formal Definitions.** ACM Computing Surveys, Vol. 8, No. 2, June 1976.

[23] Jackson, Quinn Tyler. **Some Theoretical and Practical Results in Context-Sensitive and Adaptive Parsing**, Progress in Complexity, Information, and Design, vol 1, n. 4, pp 1-11. International Society for Complexity, 2002. Editor: William A. Dembski.

[24] Jackson, Quinn Tyler. **The §-Calculus. The Death of "Ad Hockery"**, Comunicação pessoal. pp 1-40, 2003.

[25] Shutt, John. **Self-Modifying Finite Automata : Basic Definitions and Results.** Computer Science technical Report WPI-CS-TR-95-2, Worcester Polytechnic Institute, (1995).

[26] Neto, João J.; Bravo, César. **Adaptive automata - a revisited proposal.** CIAA2002: Seventh International Conference on Implementation and Application of Automata, 03-06 juillet 2002. Tours, France Lecture Notes in Computer Science Vol. 2608. Editor Jean-Marc Champarnaud. ISBN: 3-540-40391-4

[27] Boullier, Pierre. **Dynamic grammars and semantic analysis**, Rapport de Recherche de l'INRIA- Rocquencourt. France, 1994,

[28] Fowler, M.; Scott K. **UML Distilled.** Addison-Wesley, September 1999.

[29] Neto, João J. **Introdução à compilação.**, Série "Engenharia de Computação" Livros Técnicos e Científicos Editora S.A., São Paulo, 1987.

[30] Ginsburg, Seymour; Greibach, Sheila and Harrison Michael A. **Stack Automata and Compiling.** Journal of the ACM Vol.14, No. 1, January, p. 172-201, 1967.

[31] Knuth, Donald E.; and Bigelow, Richard H. **Programming Languages for Automata.** Journal of the ACM Vol.14, No. 4, October, p. 148-172, 1967.

[32] Harrison, Michael A; and Schkolnik, Mario . **A Grammatical Characterization of One-Way Nondeterministic Stack Languages.** Journal of the ACM Vol.18, No. 2, April, p. 148-172, 1971.

[33] Hopcroft, J. E.; and Ullman, J. D. **Sets Accepted by One-Way Stack Automata are Context Sensitive.** Information and Control 13, 2, August. p 113-133, 1968.

[34] Ginsburg, Seymour; Greibach, Sheila and Harrison Michael A. **One-Way Stack Automata".** Journal of the ACM Vol.14, No. 2, January, p.389-418, 1967.

[35] Wirth, Niklaus. **What can we do about the unnecessary diversity of notation for syntactic definitions?.** Communications of the ACM, Volume 20, Issue 11 (November 1977), pp 822-823 Publisher ACM Press, New York, NY, USA, ISSN:0001-0782

[36] Hauschildt, Dirk; Jantzen, Matthias *Petri Net Algorithms in the Theory of Matrix Grammars* Acta Informatica, Vol. 31, No. 8, pages 719-728. 1994.

[37] Rubinstein, R.S.; Shutt, John. **Self-Modifying Finite Automata** Proceedings of the 13th IFIP World Computer Congress, Amsterdam, Volume 1, pp 483-498 B. Pehrson and I.Simon (Editores) North-Holland, 1994, IFIP

[38] Rubinstein, R.S.; Shutt, John. **Self-Modifying Finite Automata: an introduction** Information Processing Letters, volume 56, issue 4, 1995, pp. 185-190. B. Pehrson and I.Simon (Editores) Elsevier Science B. V. (North-Holland), 1994, IFIP

[39] Y. Wang, K. Inoue, A. Ito, and T. Okazaki, **A note on self-modifying finite automata**, Information Processing Letters 72 nos. 1-2 (29 October 1999), pp. 19-24.

[40] Jackson, Quinn Tyler. **Adaptive Predicates in Natural Language Parsing**, Perfection, vol. 1, n. 4, 2000. Disponível em: citeseer.nj.nec.com/jackson00adaptive.html. Acesso em 22/03/2003.

[41] Jackson, Quinn Tyler; and Langan, Christopher Michael. **Adaptive Predicates in Empty-Start Natural Language Parsing**, Noesis-E, vol 1, n. 6, 2001. Disponível em: citeseer.nj.nec.com/jackson01adaptive.html. Acesso em 22/03/2003.

[42] Jackson, Quinn Tyler. **Efficient formalism-only parsing of XML/HTML using the §-calculus** ACM SIGPLAN Notices. Volume 38, Issue 2 (February 2003). pp 29-35. ISSN:0362-1340. Disponível em: http://portal.acm.org/citation.cfm?doid=772970.772974. Acesso em 22/03/2003.

[43] Lewis, Harry L.; and Papadimitriou, Christos H. **Elements of the Theory of Computation.** Prentice-Hall, Inc, 1981. ISBN 0-13-273417-6

[44] Michell, John C. **Foundations for Programming Languages.** The MIT Press, 1996.

[45] Milos, Don; Pleban, Uwe; Loegel, George. **Direct Implementation of Compiler Specifications or The Pascal P-code Compiler Revisited.** Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages January 1984.

[46] Pagan, Frank. **Formal Specification of Programming Languages : a panoramic Primer.** Prentice Hall. 1981.

[47] P. Pleban, U; Lee , P. **An Automatically Generated, Realistic Compiler for an Imperative Programming Language.** Proceedings SIGPLAN 1988 Conference on Programming Languages Design and Implementation. Atlanta, Georgia, June 22-24, 1988.

[48] Cormen, C. Leisserson, e R. Rivest, **Introduction to Algorithms**, MIT Press/McGraw-Hill, 1990

[49] Salomaa, Arto. **On grammars with restricted use of productions.** Ann. Acad. Sci. Fennicae, Ser. AI, 454, (1969).

[50] Altman E.; Banerji, R. **Some problems of finite representability** Information and Control 8, (1965), 251-263.

[51] Ginsburg S.; Spanier E. **Control set on grammars** Mathematics Systems Theory 2 (1968), pp 159-177.

[52] Friant J. **Grammaires ordonnees-grammaires matricielles** Univ. de Montréal MA-101 (1968)

Apêndice A

Este apêndice está dividido em três seções. Na primeira, são listados resultados usados no capítulo 2 desta tese. Na segunda seção é descrita a *notação de Wirth*[35], para gramáticas livres de contexto. Na terceira seção é descrito o algoritmo *wirth2ape*[29], o qual, a partir de uma gramática livre de contexto G descrita em notação de Wirth, obtem um autômato de pilha estruturado que aceita a linguagem gerada pela gramática G[29].

A.1 Definições e resultados básicos

Os resultados listados neste apêndice foram incluidos para fornecer enunciados completos de todas afirmativas utilizadas no texto. A maioria delas é citada diretamente por diferentes teoremas do capítulo 2 desta tese. Os dois últimos resultados, porém, são utilizados apenas na prova de resultados deste apêndice. As provas podem ser consultadas na obra citada. Conservamos a numeração original entre parênteses imediatamente após da nova numeração deste apêndice.

Definição A.1.1 *Um sistema de reescrita é um par ordenado $RW = (\Sigma, P)$ onde Σ é um alfabeto e P é um conjunto finito de pares ordenados de palavras sobre Σ. Os elementos (A, B) de P se denominam "regras de reescrita" ou "produções" e são denotadas $A \to B$. Uma palavra A sobre V "gera" diretamente uma palavra B sobre V e denotamos $A \Rightarrow B$ se, e somente se, existem palavras α, β, C e D sobre V e uma produção $C \to D \in P$ tais que $A = \alpha C \beta$ e $B = \alpha D \beta$. O fechamento reflexivo-transitivo de $\Rightarrow$ é denotado $\Rightarrow^*$.*

Definição A.1.2 *Uma "gramática gerativa" $G = (N, T, P, S)$ é uma gramática no sentido de Chomsky; uma gramática gerativa induz o sistema de reescrita (N, P). Uma "gramática analitica" $G = (N, T, P, S)$ é uma gramática cujas produções $A \to B$ são tais que $A \in (N \cup T)^*$ e $B \in (N \cup T)^*$ com a restrição de que B deve*

conter algum não-terminal de N. A gramática analitica também induz o sistema de reescrita (N, P) e suas relações associadas $\Rightarrow$ e $\Rightarrow^*$. A linguagem "reconhecida" ou "aceita" pela gramática analitca G é:

$$L(G) = \{w : w \in T^*, w \Rightarrow^* S\}$$

Teorema A.1.1 (Theorem 2.1, Part I, pag 10) *Seja $G = (N, T, P, S)$ uma gramática gerativa ou analítica. Seja P_1 o conjunto das produções $A \to B$ tal que $B \to A \in P$. Então:*

$$L(G_1) = L(G), \text{ onde } G_1 = (N, T, P_1, S).$$

Teorema A.1.2 (Theorem 3.1, Part I, pag 19) *Para toda gramática $G = (N, T, P, S)$, pode-se construir uma gramática equivalente $G_1 = (N_1, T, P_1, S)$ tal que toda produção contendo terminais de T é da forma $A \to a$, com $A \in N_1$ e $a \in T$. Além disso, se G é de tipo 0, tipo 1, tipo 2. ou de comprimento crescente, então G_1 é também de tipo 0, tipo 1, tipo 2. ou de comprimento crescente, respectivamente.*

Teorema A.1.3 (Theorem 9.1, Part I, pag 82) *Existe um algoritmo para decidir se uma palavra w pertence ou não à linguagem $L(G)$ gerada por uma gramática tipo-1 $G = (N, T, P, S)$.*

Teorema A.1.4 (Theorem 9.9, Part I, pag 89) *Seja L uma linguagem de tipo 0 sobre o alfabeto Σ tal que $a, b \notin \Sigma$. Então existe uma linguagem de tipo 1 L_1 tal que*

(i) L_1 é formado pelas palavras da forma $a^i bw$ onde $i \geq 0$ e $w \in L$,

(ii) Para cada palavra $w \in L$, existe um $i \geq 0$, tal que $a^i bw \in L_1$

Definição A.1.3 *Suponhamos que*

$$\mathcal{D} : S = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n = w$$

seja uma derivação segundo a gramática $G = (N, T, P, S)$. Então a "área de trabalho"[1] da palavra w pela derivação" $\mathcal{D}$ é definida por

$$Workspace_G(w, \mathcal{D}) = max\{|w_i| : 0 \leq i \leq n\}.$$

O área de trabalho da palavra $w \in L(G)$ é definida por

$$Workspace_G(w) = min\{Workspace_G(w, \mathcal{D}) : \mathcal{D} \text{ é uma derivação de } w\}.$$

[1] *workspace* em inglês

Definição A.1.4 *Seja L uma linguagem sobre o alfabeto Σ. Um homomorfismo h de Σ é chamado um "apagamento k-linear"*[2] *em relação a L se, e somente se, para cada $w \in L$*

$$|w| \leq k|h(w)|.$$

Uma família $\mathcal{L}$ se diz fechada em relação à operação de apagamento linear se, e somente se, $h(L) \in \mathcal{L}$ onde $L \in \mathcal{L}_1$, k é um inteiro, e h é um apagamento k-linear relativo a L.

Teorema A.1.5 (Theorem 10.4, Part I, pag 98) *A família $\mathcal{L}_1$ é fechada em relação ao apagamento linear.*

Definição A.1.5 *Um sistema de reescrita (Σ, P) é chamado "máquina seqüencial geralizada" se e somente se satisfaz as seguintes condições:*

(i) Σ é dividido em dois alfabetos disjuntos Q e $\Sigma_i \cup \Sigma_o$. Os conjuntos Q, Σ_i e Σ_o são denominados alfabetos de estados, de entrada, e de saída, respectivamente. Os conjuntos Σ_i e Σ_o são não vazios mas não necessariamente disjuntos.

(ii) Um elemento $q_0 \in Q$ e um conjunto $F \subset Q$ são escolhidos como o "estado inicial" e o "conjunto de estados finais"

(iii) As produções em P são da forma:

$$q_i a \to w q_j, \qquad q_i, q_j \in Q, a \in \Sigma_i, w \in \Sigma_o \Sigma_o^*.$$

Dada uma máquina seqüencial geralizada GSM e uma palavra w sobre seu alfabeto de entrada Σ_i, denotamos

$$GSM(w) = \{z : q_0 w \Rightarrow^* z q_1, \text{ para algum } q_1 \in F\}$$

Seja L uma linguagem sobre Σ_i. Então GSM mapeia L na linguagem

$$GSM(L) = \{z : z \in GSM(w) \text{ para alguma palavra } w \in L\}$$

e dizemos que essa linguagem é uma "imagem gsm". Por outro lado, se $w \in \Sigma_o^*$, definimos

$$GSM^{-1}(w) = \{z : w \in GSM(z)\}.$$

Para uma linguagem $L \subset \Sigma_o^*$, definimos

$$GSM^{-1}(L) = \{z : z \in GSM^{-1}(w) \text{ para alguma palavra } w \in L\}$$

e dizemos que essa linguagem é uma "imagem gsm inversa".

[2] *k-linear-erasing* em inglês

Definição A.1.6 *Dizemos que uma máquina de estados geralizada GSM é "livre de λ" se, e somente se, todas as palavras w que aparecem nas suas produções são diferentes da palavra vazia λ. Uma imagem gsm é livre de λ se, e somente se, é obtida por meio de uma máquina de estados geralizada GSM livre de λ.*

Definição A.1.7 *Uma família de linguagens é denominada "família abstrata de linguagens", e é denotada AFL (pelas suas siglas em inglês: abstract family of languages) se, e somente se, ela contém uma linguagem não-vazia e fechada em relação a cada uma das seguintes operações: união, fechamento de concatenação livre de λ (de fato $L^+ = \bigcup_{i=1}^{\infty} L^i$), homomorfismo livres de λ, homomorfismo inverso, e intersecção com linguagens regulares. Uma AFL se diz "completa" se é fechada por homomorfismos arbitrários.*

Teorema A.1.6 (Theorem 2.2, Part II, pag 131) *Toda AFL é fechada em relação a imagens gsm sem a palavra vazia λ. Toda AFL completa é fechada em relação a imagens gsm.*

Teorema A.1.7 (Theorem 9.2, Part I, pag 82) *Para toda gramática de comprimento crescente existe uma gramática de tipo 1 equivalente.*

Teorema A.1.8 (Theorem 10.1, Part I, pag 93) *Se $G = (N, T, P, S)$ é um gramática tipo 0 e existe um número natural k tal que:*

$$Workspace_G(w) \leq k|w|$$

para todas as palavras não-vazias $w \in L(G)$, então $L(G)$ é de tipo 1.

Tabela 3 de Salomaa[2], página 184, a posição das classes das linguagens $\mathcal{L}(i, j, 0)$ geradas por gramáticas de tipo-i com linguagem de controle de tipo-j sem verificação de aparência:

$i \backslash j$	0	1	2	3
0	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_0$
1	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_1 \subseteq \mathcal{L}(1,2,0) \subset \mathcal{REC}$	$\mathcal{L}_1$
2	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_2 \subset \mathcal{L}(1,2,0)$	$\mathcal{L}_2 \subset \mathcal{L}(2,3,0)$
$2-\lambda$	$\mathcal{L}_0^{'}$	$\mathcal{L}_0^{'}$	$\mathcal{L}_2^{'} \subset \mathcal{L}(1,2,0) \subset \mathcal{REC}^{'}$	$\mathcal{L}_2^{'} \subset \mathcal{L}(2-\lambda,3,0) \subset \mathcal{L}_1^{'}$
3	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_2$	$\mathcal{L}_3$

onde $\mathcal{REC}$ denota a classe das linguagens recursivas e $\mathcal{REC}^{'}$ é formada pelas linguagens de $\mathcal{REC}$ que não contêm a palavra vazia λ.

Tabela 4 de Salomaa[2], página 184, a posição das classes das linguagens $\mathcal{L}(i,j,1)$ geradas por gramáticas de tipo-i com linguagem de controle de tipo-j com verificação de aparência:

$i\backslash j$	0	1	2	3
0	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_0$
1	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_1 \subseteq \mathcal{L}(1,2,1) \subset \mathcal{RECC}$	$\mathcal{L}_1$
2	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_0$
$2-\lambda$	$\mathcal{L}_0^{'}$	$\mathcal{L}_0^{'}$	$\mathcal{L}_2^{'} \subset \mathcal{L}(2-\lambda,2,1) \subset \mathcal{RECC}^{'}$	$\mathcal{L}_2^{'} \subset \mathcal{L}(2-\lambda,3,1) \subset \mathcal{L}_1^{'}$
3	$\mathcal{L}_0$	$\mathcal{L}_0$	$\mathcal{L}_2$	$\mathcal{L}_3$

A.2 Notação de Wirth

Em 1977, Niklaus Wirth propus uma notação[35], para unificar a descrição de sintaxe de linguages de programação. Essa notação, que nós chamamos de notação de Wirth, têm as seguintes propriedades:

1. Distingue claramente entre meta-terminais e não-terminais,
2. Não exclue meta-símbolos de ser usados como símbolos da linguagem (por exemplo "|" en BNF),
3. Possue uma construção para denotar iteração, evitando assim o uso de recursão para expressar repetições,
4. Evita usar um símbolo explicito para a cadeia vazia λ,
5. Está baseado no conjunto de carateres ASCII.

A palavra *identifier* é usada para denotar *não-terminal* e *literal* é usada para denotar *terminal*. As repetições são denotadas por chaves, i.e., $\{a\}$ significa $\lambda|a|aa|aaa|\ldots$. Opções são expressadas por colchetes, i.e., $[a]$ significa $a|\lambda$. Parenteses servem para agrupar, por exemplo, $(a|b)c$ significa $ac|bc$. Símbolos terminais, i.e, literais, são delimitados por aspas duplas; se uma aspa dupla deve aparecer como um terminal ela é escrita duas vezes.

Como exemplo, é apresentado a seguir a definição da sintaxe da notação de Wirth na própria notação de Wirth:

$$
\begin{array}{lcl}
syntax & = & \{production\}. \\
production & = & identifier\ ''=''\ expression\ ''.''. \\
expression & = & term\{\ ''|''\ term\}. \\
term & = & factor\{factor\}. \\
factor & = & identifier \mid literal \mid ''(''expression'')'' \mid \\
 & & ''[''expression'']'' \mid ''\{''expression''\}''. \\
literal & = & ''''''''character\{character\}''''''''.
\end{array}
$$

A.3 O algoritmo *wirth2ape*

Dada uma gramáticas livres de contexto G expressada em notação de Wirth o algoritmo *wirth2ape* constroi[29] um autômato de pilha estruturado que aceita a linguagem gerada pela gramática G.

Nossa discussão do algoritmo não é detalhada, mas apenas um resumo, para oferecer ao leitor uma primeira visão desse algoritmo.

O algoritmo pode ser dividido em duas partes, denominadas, respectivamente, *Atribuição de estados* e *Obtenção das produções*. Vamos apresentar cada uma dessas partes separadamente. O algoritmo requer certo pré-processamento que descrevemos a seguir

1. Se a gramática estiver em outra notação, deve-se inicialmente converté-la para a notação de Wirth.
2. Determinar a forma geral da linguagem, resolvendo o sistema de equações representado pela gramática, interpretando os não-terminais como variáveis e os terminais como constantes:
   - Agrupar as várias opções do mesmo não-terminal
   - Fatorar as auto-recursões à esquerda e à direita
   - Transformar em iterações as auto-recursõess à esquerda e à direita

     $A = Ax|b$ torna-se $A = b\{x\}$
     $A = b|Ax$ torna-se $A = \{x\}b$
   - Determinar os não-terminais essenciais, pesquisando-os a partir da raíz da gramática, de acordo com as dependências impostas pela gramática

     A raiz da gramática é sempre essencial

     São essenciais todos os não-terminais direita ou indiretamente *auto-recurivos centrais*, que não possam ser estritamente expressos em função de terminais e/ou não-terminais essenciais.
   - Eliminar por substituições sucessivas, os demais não-terminais,
   - Após cada substituição, defatorar a expressão e reaplicar o procedimento.

Algoritmo A.3.1 *wirth2ape: Atribuição de estados*

Entrada: Uma expressão de Wirth E denotando uma gramática livre de contexto G

Saída: Um conjunto de estados atribuídos a posições da expressão de Wirth que reconhece a linguagem gerada pela gramática G.

1. *Iniciar em zero uma variável j de contagem de estados já atribuídos,*

   (a) *Para cada regra da forma:*

   $$A_i = \Theta_{i_1}|\Theta_{i_2}|\ldots|\Theta_{i_n}.$$

   definir um estado inicial $q_{A_{i_0}}$ único para a correspondente sub-máquina:

   $$A_i = \underset{q_{A_{i_0}}}{.}\Theta_{i_1}|\underset{q_{A_{i_0}}}{.}\Theta_{i_2}|\ldots|\underset{q_{A_{i_0}}}{.}\Theta_{i_n}.$$

   (b) *Para cada agrupamento entre parênteses*

   $$A_i = \underset{q_x}{.}(\Theta_{i_1}|\Theta_{i_2}|\ldots|\Theta_{i_n})$$

   ou colchetes

   $$A_i = \underset{q_x}{.}[\Theta_{i_1}|\Theta_{i_2}|\ldots|\Theta_{i_n}]$$

   associar o mesmo estado q_x, atribuído previamente à sus esquerda, aos pontos iniciais de cada uma das suas alternativas. Atribuir também o novo estado j à sua direita, e incrementar j.

   $$A_i = \underset{q_x}{.}(\underset{q_x}{.}\Theta_{i_1}|\underset{q_x}{.}\Theta_{i_2}|\ldots|\underset{q_x}{.}\Theta_{i_n})\underset{j}{.}$$

   ou

   $$A_i = \underset{q_x}{.}[\underset{q_x}{.}\Theta_{i_1}|\underset{q_x}{.}\Theta_{i_2}|\ldots|\underset{q_x}{.}\Theta_{i_n}]\underset{j}{.}$$

   (c) *Para cada agrupamento entre chaves*

   $$A_i = \underset{q_x}{.}\{\Theta_{i_1}|\Theta_{i_2}|\ldots|\Theta_{i_n}\}$$

   atribuir o novo estado j à sua direita e também aos pontos iniciais de cada uma das alternativas do agrupamento

   $$A_i = \underset{q_x}{.}\{\underset{j}{.}\Theta_{i_1}|\underset{j}{.}\Theta_{i_2}|\ldots|\underset{j}{.}\Theta_{i_n}\}\underset{j}{.}$$

(d) Para cada ocorrência de terminal, não-terminal ou vazio λ

$$\overset{\bullet}{q_x}\ \sigma$$

atribuir o novo estado j *à sua direita*

$$\overset{\bullet}{q_x}\ \sigma\ \overset{\bullet}{j}$$

e incrementar o contador j*.*

Algoritmo A.3.2 *wirth2ape: Obtenção das produções*

Entrada: Um conjunto de estados atribuídos às posições da expressão de Wirth

Saída: Um conjunto de transições que representam o autômato de pilha estruturado que reconhece a linguagem gerada pela gramática G.

1. *Para cada regra associada a um não-terminal* A_i

$$A_i = \underset{q_0}{\cdot}\,\Theta_{i_1}\,\underset{r_{i_1}}{\cdot} \mid \underset{q_0}{\cdot}\,\Theta_{i_2}\,\underset{r_{i_2}}{\cdot} \mid \ldots \mid \underset{q_0}{\cdot}\,\Theta_{i_n}\,\underset{r_{i_n}}{\cdot}$$

2. *Para cada ocorrência de terminal*

$$\underset{r_{i_k}}{\cdot}\,\sigma\,\underset{r_{i_j}}{\cdot}$$

criar uma produção de consumo do átomo σ*, transitando entre os estados* q_{i_k} *e* q_{i_j}*, correspondentes às atribuições* r_{i_k} *e* r_{i_j}*.*

$$\gamma q_{i_k}\sigma\alpha \rightarrow \gamma q_{i_j}\alpha,$$

3. *Todas as demais transições internas são em vazio*

   (a) *Agrupamentos entre parênteses*

$$\underset{r_{i_k}}{\cdot}\,(\Theta_{i_1}\,\underset{r_{i_1}}{\cdot} \mid \Theta_{i_2}\,\underset{r_{i_2}}{\cdot} \mid \ldots \mid \Theta_{i_n}\,\underset{r_{i_n}}{\cdot}\,)\,\underset{r_{i_j}}{\cdot}$$

   ou colchetes

$$\underset{r_{i_k}}{\cdot}\,[\Theta_{i_1}\,\underset{r_{i_1}}{\cdot} \mid \Theta_{i_2}\,\underset{r_{i_2}}{\cdot} \mid \ldots \mid \Theta_{i_n}\,\underset{r_{i_n}}{\cdot}\,]\,\underset{r_{i_j}}{\cdot}$$

   Para cada final de opção criar uma transição em vazio fechando o laço:

$$\gamma q_{i_m}\alpha \rightarrow \gamma q_{i_j}\alpha \text{ para } m = 1, \ldots, n$$

   (b) *Agrupamentos entre chaves:*

$$\{\underset{r_{i_j}}{\cdot}\,\Theta_{i_1}\,\underset{r_{i_1}}{\cdot} \mid \underset{r_{i_j}}{\cdot}\,\Theta_{i_2}\,\underset{r_{i_2}}{\cdot} \mid \ldots \mid \underset{r_{i_j}}{\cdot}\,\Theta_{i_n}\,\underset{r_{i_n}}{\cdot}\,\}\,\underset{r_{i_j}}{\cdot}$$

   Para cada final de opção criar uma transição em vazio fechando o laço:

$$\gamma q_{i_m}\alpha \rightarrow \gamma q_{i_j}\alpha \text{ para } m = 1, \ldots, n$$

   (c) *Apenas para agrupamentos entre colchetes ou chaves, criar ainda*

$$\gamma q_{i_k}\alpha \rightarrow \gamma q_{i_j}\alpha,$$

4. *Finalizar construindo as transições entre sub-máquinas*

   (a) *Chamadas de sub-máquina para cada ocorrência de não-terminal* A_m

   $$\overset{\cdot}{r_{i_k}} A_m \overset{\cdot}{r_{i_j}}$$

   criar uma transição de chamada da sub-máquina a_m *associada ao não-terminal* A_m *(o estado inicial dessa sub-máquina é* q_{m_0}*)*

   $$\gamma q_{i_k} \alpha \rightarrow \gamma r_{i_j} q_{m_0} \alpha$$

   (b) *Retornos de de sub-máquina ao fim de cada alternativa de um não-terminal*

   $$A_i = \Theta_{i_1} \underset{q_{r_{i_1}}}{\cdot} |\Theta_{i_2} \underset{q_{r_{i_2}}}{\cdot} | \ldots | \Theta_{i_n} \underset{q_{r_{i_n}}}{\cdot}$$

   para cada final de opção (estados q_{j_k}*, associados às marcações* r_{j_k}*) criar uma transição de retorno do tipo*

   $$\gamma r_{uv} q_{i_k} \alpha \rightarrow \gamma q_{uv} \alpha$$