ASPECTOS COMPUTACIONAIS ASSOCIADOS À IMPLEMENTAÇÃO DE ALGORITMOS PARA SISTEMAS A EVENTOS DISCRETOS

Leonardo Enrique Bermeo Clavijo

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Elétrica.

Orientador: João Carlos dos Santos Basilio

Rio de Janeiro
Abril de 2014

ASPECTOS COMPUTACIONAIS ASSOCIADOS À IMPLEMENTAÇÃO DE ALGORITMOS PARA SISTEMAS A EVENTOS DISCRETOS

Leonardo Enrique Bermeo Clavijo

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. João Carlos dos Santos Basilio, Ph.D.

Prof. Amit Bhaya, Ph.D.

Prof. Gilberto Oliveira Corrêa, Ph.D.

Prof. Antonio Eduardo Carrilho da Cunha, Dr.Eng.

Prof. Marcos Vicente de Brito Moreira, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2014

Clavijo, Leonardo Enrique Bermeo

Aspectos computacionais associados à implementação de algoritmos para sistemas a eventos discretos/Leonardo Enrique Bermeo Clavijo. – Rio de Janeiro: UFRJ/COPPE, 2014.

XX, 150 p.: il.; 29, 7cm.

Orientador: João Carlos dos Santos Basilio

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2014.

Referências Bibliográficas: p. 139 – 150.

1. Sistemas a eventos discretos. 2. Computação científica. 3. Complexidade. I. Basilio, João Carlos dos Santos. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

"Os métodos são as verdadeiras riquezas."
Nietzsche

Agradecimentos

Agradeço aos meus filhos Daniela e Esteban pela esperança, a tranquilidade e a coragem para seguir adiante cada dia da minha vida. A minha mãe Atala Clavijo pela ajuda ao longo desse grande percurso chamado vida e a minha esposa Jazmin Gutierrez pelo sacrifício e compreensão ao longo desses quatro anos.

Agradeço ao meu orientador João Carlos Basilio pelo acompanhamento constante e próximo nesse processo de estudar doutorado, pela ajuda acadêmica e pessoal e, sobretudo, pela grande amizade oferecida em todo momento.

Agradeço às pessoas excelentes que conheci no Brasil, que agora posso me orgulhar de chamar amigos: Mario Rodríguez, Lilian Kawakami Carvalho, Mayara Alexandre Costa, Oscar Godoy, William Pinto, Tatiana Vargas e Santiago Toro.

Agradeço aos meus colegas do LCA, em especial a Gustavo da Silva Viana, Bernardo Bouzan, Eduardo Nunes e Angela Arana pela cooperação recebida em todo esse tempo.

Agradeço ao povo brasileiro por essa grande oportunidade acadêmica, cultural e vital que tem brindado aos estudantes estrangeiros de pós-graduação.

Agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico do Brasil (CNPq).

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

ASPECTOS COMPUTACIONAIS ASSOCIADOS À IMPLEMENTAÇÃO DE ALGORITMOS PARA SISTEMAS A EVENTOS DISCRETOS

Leonardo Enrique Bermeo Clavijo

Abril/2014

Orientador: João Carlos dos Santos Basilio

Programa: Engenharia Elétrica

De uma forma geral, a análise de complexidade de algoritmos que envolvem sistemas a eventos discretos (SEDs) é realizada considerando-se unicamente o desempenho no pior caso. No entanto, a complexidade média também deve ser analisada uma vez que é possível que o pior caso do desempenho do algoritmo estudado não seja atingido pela maior parte das instâncias que aparecem na prática. Na área de diagnose de falhas em SEDs existem dois problemas centrais: a diagnose em tempo real de falhas (para a qual deve ser construído um autômato diagnosticador) e a verificação de diagnosticabilidade (que implica a construção de um autômato verificador ou de um autômato diagnosticador). Esta tese estuda a complexidade média dos algoritmos de construção de diagnosticadores e verificadores para diagnose de falhas em SEDs usando, para tanto, uma abordagem experimental. Para que esse objetivo possa ser alcançado foi desenvolvida uma ferramenta de computação científica chamada DESLAB, para facilitar o uso dos métodos experimentais no estudo de algoritmos para sistemas a eventos discretos. Para analisar complexidade média dos algoritmos de diagnose de falhas em SEDs foram realizados dois experimentos usando geradores de autômatos reportados recentemente na literatura. Partindo dos resultados experimentais, foram propostos modelos para caracterizar a complexidade média dos algoritmos estudados. Uma contribuição marginal da tese é a abordagem do problema da diagnose de falhas intermitentes na operação de sensores. Nesse contexto foram obtidas condições necessárias e suficientes para a diagnosticabilidade de falhas intermitentes na operação de sensores em sistemas a eventos discretos descritos por autômatos.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

COMPUTATIONAL ISSUES ASSOCIATED WITH THE IMPLEMENTATION OF ALGORITHMS FOR DISCRETE EVENT SYSTEMS

Leonardo Enrique Bermeo Clavijo

April/2014

Advisor: João Carlos dos Santos Basilio

Department: Electrical Engineering

Generally speaking, the complexity analysis of algorithms involving discrete-event systems (DES) modeled as automata is carried out considering only the worst case performance. In spite of that, the average case complexity must also be taken into account since it is possible that the worst case performance of the algorithm being studied is not often reached for most practical instances. The field of fault diagnosis of DES has two main issues: online fault diagnosis (which requires to build a diagnoser automaton) and verification of diagnosability (which implies the construction of either a verifier automaton or a diagnoser automaton). In this thesis, we study the average case complexity in the construction of diagnosers and verifiers for fault diagnosability verification of DES using, for that purpose, an experimental approach. In order to be able to achieve the proposed goal, we first developed DESLAB, a scientific computing program whose purpose is to facilitate the use of experimental methods in the study of algorithms for DES described as automata. The average case complexity analysis of the algorithms for fault diagnosis of DES was carried out by performing two different experiments using automata generators, recently reported in literature. On the basis of the experimental results, we proposed models for characterizing the average case complexity of the algorithms being studied. In the final part of this thesis, we deal with the problem of diagnosability of intermittent failure in sensor operation. As a side contribution of this thesis, we present necessary and sufficient conditions for diagnosability of intermittent sensor faults and propose a test based on diagnoser automaton to verify intermittent sensor fault diagnosability.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Lista de Algoritmos

Lista de Símbolos

Σ	—	Conjunto de Eventos
Σ_o	—	Conjunto de eventos observáveis
Σ_{uo}	—	Conjunto de eventos não observáveis
Σ_c	—	Conjunto de eventos controláveis
Σ_{uc}	—	Conjunto de eventos não controláveis
$\Sigma_f = \sigma_f$	—	Conjunto de eventos de falha, $\Sigma_f \subset \Sigma$
L^*	—	Fecho de Kleene da linguagem L
$\overline{L}$	—	Fecho em prefixo da linguagem L
$P_{\Sigma_o} : \Sigma^* \to \Sigma_o^*$	—	Projeção do elemento de Σ^* em Σ_o^*
$D : \Sigma^* \to 2^{\Sigma_{dil}}$	—	Dilatação do elemento de Σ^* em $2^{\Sigma_{dil}}$
$\|w\|$	—	Comprimento de uma sequência w
L/w	—	Continuação da linguagem L após uma sequência w
2^Σ	—	Conjunto potência de Σ
$G_1 \times G_2$	—	Composição produto de G_1 e G_2
$G_1 \parallel G_2$	—	Composição paralela de G_1 e G_2

G_D	—	Diagnosticador centralizado associado a um autômato G
$I_{m:n}$	—	Conjunto de inteiros $\{m, m+1, \ldots, n\}$
$(s_i)_{i \in I_{0:kn-1}}$	—	Sequência representativa de um autômato completo com n estados e k eventos
$\mathcal{C}^{(n,k)}$	—	Conjunto de autômatos completos que possuem n estados e k eventos
$\mathcal{A}^{(n,k)}$	—	Conjunto de autômatos acessíveis (possivelmente incompletos) possuem n estados e k eventos
$G_i^{(n,k,u)}$	—	Autômato acessível com u eventos não observáveis que pertence ao conjunto $\mathcal{A}^{(n,k)}$, $i = 1, 2, \ldots, \lvert A^{(n,k)} \rvert$
$G_{D,i}^{(n,k,u)}$	—	Diagnosticador associado ao autômato $G_i^{(n,k,u)}$
$G_{V,i}^{(n,k,u)}$	—	Verificador associado ao autômato $G_i^{(n,k,u)}$
$A_v^{(n,k,u)}$	—	Conjunto de autômatos válidos, $A_v^{(n,k,u)} \subset A^{(n,k,u)}$
$\mathbf{D}^{(n,k,u)}$	—	Vetor $(D_1^{(n,k,u)}, D_2^{(n,k,u)}, \ldots)$ contendo o número de estados do diagnosticador $D_i^{(n,k,u)} = \lvert X_{D,i}^{(n,k,u)} \rvert$ calculado para cada autômato $G_i^{(n,k,u)}$ no conjunto de autômatos válidos $\mathcal{A}_v^{(n,k,u)}$
$\mathbf{V}^{(n,k,u)}$	—	Vetor $(V_1^{(n,k,u)}, V_2^{(n,k,u)}, \ldots)$ contendo o número de estados do verificador $V_i^{(n,k,u)} = \lvert X_{V,i}^{(n,k,u)} \rvert$ calculado para cada autômato $G_i^{(n,k,u)}$ no conjunto de autômatos válidos $\mathcal{A}_v^{(n,k,u)}$
m_D	—	Valor médio do número de estados dos diagnosticadores calculados para um conjunto completo de autômatos válidos
m_V	—	Valor médio do número de estados dos verificadores calculados para um conjunto completo de autômatos válidos
s_D	—	Desvio padrão do número de estados dos diagnosticadores calculados para um conjunto completo de autômatos válidos
s_V	—	Desvio padrão do número de estados dos verificadores calculados para um conjunto completo de autômatos válidos
$\hat{\theta}$	—	Estimativa de máxima verosimilhança baseada numa observação amostral

α	—	Nível de significância de um teste de qualidade de ajuste
$\hat{M}_{LN}$	—	Estimativa do valor esperado de uma variável aleatória com distribuição lognormal
$\hat{S}^2_{LN}$	—	Estimativa do desvio padrão de uma variável aleatória com distribuição lognormal
m_{LN}	—	Valor esperado de uma variável aleatória com distribuição lognormal
$\hat{s}^2_{LN}$	—	Desvio padrão de uma variável aleatória com distribuição lognormal
$D^{(n,k,u)}$	—	Variável aleatória que resulta ao calcular a cardinalidade do conjunto de estados do diagnosticador construído para um autômato do conjunto $\mathcal{A}_v^{(n,k,u)}$ selecionado aleatoriamente
$\mathcal{Q}^{(p,n)}$	—	Classe das partições do conjunto $I_{1:p}$ em n blocos
ρ_{LN}	—	Erro percentual na estimação de m_{LN}
$\hat{A}_v^{(n,k,u)}$	—	Conjunto de 10000 autômatos válidos gerados aleatoriamente com distribuição uniforme
$\hat{\mathbf{D}}^{(n,k,u)}$	—	vetor $(D_1^{(n,k,u)}, D_2^{(n,k,u)}, \ldots, D_{10000}^{(n,k,u)})$ de 10000 valores em que cada $D_i^{(n,k,u)} = \lvert X_{D,i}^{(n,k,u)} \rvert$ é o número de estados do diagnosticador calculado para cada autômato $G_i^{(n,k,u)}$ no subconjunto de autômatos válidos $\hat{\mathcal{A}}_v^{(n,k,u)}$
$\hat{\mathbf{V}}^{(n,k,u)}$	—	vetor $(V_1^{(n,k,u)}, V_2^{(n,k,u)}, \ldots, V_{10000}^{(n,k,u)})$ de 10000 valores em que cada $V_i^{(n,k,u)} = \lvert X_{V,i}^{(n,k,u)} \rvert$ é o número de estados do verificador calculado para cada autômato $G_i^{(n,k,u)}$ no subconjunto de autômatos válidos $\hat{\mathcal{A}}_v^{(n,k,u)}$
$\mathbf{M}_D(n,k,u)$	—	Função que associa um ponto $(n,k,u) \in I_U$ com o valor esperado da complexidade dos diagnosticadores construídos para o conjunto $\mathcal{A}_v^{(n,k,u)}$ de todos os autômatos válidos.
$\hat{\mathbf{M}}_D(n,k,u)$	—	Função que associa um ponto $(n,k,u) \in I_U$ com a média amostral da complexidade dos diagnosticadores construídos para o conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$ de autômatos válidos gerados aleatoriamente com distribuição uniforme.
$\mathbf{M}_V(n,k,u)$	—	Função que associa um ponto $(n,k,u) \in I_U$ com o valor esperado da complexidade dos verificadores construídos para o conjunto $\mathcal{A}_v^{(n,k,u)}$ de todos os autômatos válidos.

$\hat{\mathbf{M}}_V(n,k,u)$	—	Função que associa um ponto $(n,k,u) \in I_U$ com a média amostral da complexidade dos verificadores construídos para o conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$ de autômatos válidos gerados aleatoriamente com distribuição uniforme.
G_{dil}	—	Autômato que modela planta na qual existem falhas intermitentes na operação de sensores
$G_{dil.d}$	—	Diagnosticador de falhas intermitentes na operação de sensores

Capítulo 1

Introdução

O método científico considera que a análise teórica forma um ciclo que se alterna progressivamente com a experimentação, sendo ambas as abordagens fundamentais no desenvolvimento da ciência (McGeoch et al., 2002). Os passos fundamentais que definem o método científico são os seguintes: *(i)* formulação de um modelo para descrever o objeto de estudo; *(ii)* uso do modelo para a formulação de hipóteses; *(iii)* realização de experimentos para validar as hipóteses, e; *(iv)* refinamento do modelo inicial e repetição (Peisert and Bishop, 2007). A figura 1.1 representa graficamente os passos do método científico.

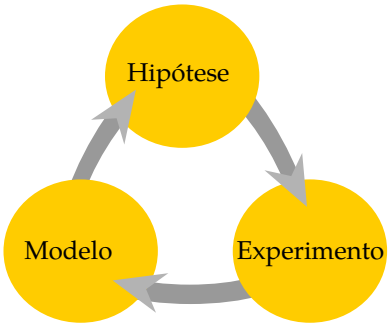


Figura 1.1: Ciclo do método científico.

No caso particular da área de sistemas a eventos discretos (SEDs), a teoria está fundamentada em modelos matemáticos provenientes da ciência da computação. Partindo desses modelos, usa-se uma metodologia dedutiva de abstração, prova e teorema (McGeoch, 2012) para, finalmente, projetar algoritmos que resolvam diferentes problemas associados com SEDs. A avaliação experimental em SEDs depende da implementação dos algoritmos em uma linguagem e uma plataforma de computação específica. É importante ressaltar que um algoritmo formulado corretamente na teoria ainda deve ser realizável na prática com um desempenho razoável. Conforme reconhecido por vários pesquisadores (Johnson, 2002; McGeoch, 2012; McGeoch et al., 2002; Sedgewick, 2013), a aplicação do método científico (que envolve o uso

de experimentos) tem um papel fundamental na compreensão do desempenho de algoritmos e, em consequência, no estudo de SEDs. Para aplicar o método científico no estudo de algoritmos é necessário percorrer os seguinte passos: projeto e análise, implementação e avaliação experimental.

A abordagem teórica clássica do desempenho de algoritmos está baseada na *teoria da complexidade computacional*[1], cujo objetivo é estabelecer o nível de dificuldade de um problema, classificando-o dentro de uma determinada *classe de complexidade*. De uma forma geral, as classes de complexidade podem ser classificadas da seguinte forma (Bovet and Crescenzi, 1994): a classe P inclui todos os problemas que podem ser solucionados em tempo polinomial (isto é, o limite do tempo $T(n)$ para resolver um problema, considerando uma instância de tamanho n, é de ordem[2] $\mathcal{O}(n^k)$ para algum inteiro fixo k) por meio de um dispositivo de computação determinístico; a classe NP contém os problemas que podem ser resolvidos em tempo polinomial por um dispositivo de computação não determinístico, sendo considerados, todavia, eficientemente verificáveis, isto é, se uma possível solução é apresentada, existe um algoritmo (chamado verificador) para verificar em tempo polinomial se essa solução é correta; a classe PSPACE inclui todos os problemas que podem ser resolvidos ocupando um montante de espaço (memoria) polinomial; um problema $\mathcal{D}$ é NP-difícil (PSPACE-difícil) se qualquer problema em NP (PSPACE) pode ser transformado em $\mathcal{D}$ por meio de um algoritmo em tempo polinomial; um problema é NP-completo (PSPACE-completo) se é NP-difícil (PSPACE-difícil) e, além disso, pertence à classe NP. A classe EXPTIME representa os problemas cujas soluções são de tempo exponencial ($T(n)$ é de ordem $\mathcal{O}(2^{n^k})$). Vale a pena ressaltar que um problema na classe P é considerado eficiente de ser resolvido e, de forma contrária, um problema dentro das classes NP-completo (PSPACE-completo) ou EXPTIME é considerado intratável, sendo EXPTIME a classe que contém os problemas mais complexos de serem resolvidos. É sabido que $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$ (Sipser, 2006), porém, determinar quais inclusões são próprias é um dos maiores problemas abertos da teoria da complexidade (em especial, a questão: é $\mathsf{P} = \mathsf{NP}$?). É importante ressaltar que a teoria clássica da complexidade estuda a dificuldade no pior caso de um problema, isto é, a complexidade computacional ao resolver o problema para a instância que produz o pior desempenho ao usar o melhor algoritmo conhecido na solução do problema.

No caso de SEDs modelados por autômatos, vários resultados já foram obtidos

[1] Um estudo formal contemporâneo da teoria da complexidade é realizado em Arora and Barak (2009)

[2] A notação do Grande-O ou notação assintótica é uma notação matemática utilizada para analisar o comportamento assintótico de funções. Diz-se que a função $f(n)$ é de ordem $\mathcal{O}(g(n))$ (ou, equivalentemente, $f(n) \in \mathcal{O}(g(n))$) se existirem $c, n_0 > 0$, tais que para todo $n > n_0$, tem-se que $f(n) \leq cg(n)$.

para caracterizar a dificuldade teórica esperada de problemas típicos, alguns desses resultados são citados a seguir:

- Problemas da classe P: verificação de diferentes propriedades de SEDs, por exemplo controlabilidade e observabilidade (Cassandras and Lafortune, 2008; Kumar and Garg, 1995), normalidade (Bouzan, 2013), diagnosticabilidade usando verificadores (Cassez et al., 2007; Jiang et al., 2001; Moreira et al., 2011a; Qiu and Kumar, 2006), propriedade do observador em abstrações (Pena et al., 2008).

- Problemas NP-completos: projeção diagnosticável (Cassez et al., 2007; Yoo and Lafortune, 2002a,b), projeção normal e projeção observável (Yoo and Lafortune, 2002a), seleção ótima de sensores para controle supervisório (Rohloff et al., 2006).

- Problemas NP-difíceis: linguagem controlável suprema com especificações modulares (Gohari and Wonham, 2000).

- Problemas PSPACE-completos: codiagnosticabilidade para SEDs temporizados usando verificadores (Cassez, 2012), co-observabilidade (Rohloff et al., 2003).

- Problemas EXPTIME: diagnosticabilidade e codiagnosticabilidade usando diagnosticadores (Basilio et al., 2012; Debouk et al., 2000; Sampath et al., 1995), busca de bases mínimas que preservam diagnosticabilidade (Basilio et al., 2012; Jiang et al., 2003).

O descobrimento de que um problema de grande relevância prática no estudo de SEDs pertence à classe de complexidade NP (ou, inclusive, a uma classe de complexidade ainda mais difícil) não implica que a sua possível solução deva ser desconsiderada e, pelo contrario, justifica pesquisar a forma de abordar o problema seguindo várias possíveis direções:

- *Aproveitamento da estrutura.* É possível que a estrutura específica das instâncias típicas do problema permitam formular um algoritmo que seja especificamente eficiente (possivelmente de tempo polinomial) para o tipo de instâncias que aparecem frequentemente (Gohari and Wonham, 2000).

- *Solução aproximada.* Se o problema de SEDs a ser resolvido é de otimização, pode ser obtida uma solução aproximada sub-ótima que possa ser obtida com um algoritmo cuja complexidade computacional seja tratável (Rohloff et al., 2006).

- *Complexidade média.* A ideia fundamental dessa abordagem é obter uma média do tempo de execução do algoritmo que soluciona o problema sobre todas as instâncias do mesmo tamanho. O algoritmo ainda pode ser considerado computacionalmente tratável se as instâncias para as quais possui grande complexidade computacional aparecem com pouca probabilidade na prática (Muller-Hannemann and Schirra, 2010). Um caso importante é o algoritmo simplex na programação linear, o qual funciona de forma muito eficiente na prática, apesar de pertencer à classe EXPTIME, funcionando, para a maior parte das instâncias, de forma superior a outros algoritmos conhecidos cuja complexidade é de tempo polinomial.

Assim como os sistemas tecnológicos tornam-se mais complexos, suas falhas tornam-se mais difíceis de se prever, entender e concertar. Por essa razão, um sistema ao ser projetado ou manufaturado deve garantir que as falhas que possam ocorrer sejam identificáveis. Essa propriedade intrínseca, chama-se *diagnosticabilidade* e, idealmente, deve ser garantida na etapa de projeto. Quando o sistema está em funcionamento, tem-se o problema da *diagnose online*, isto é, observar o sistema e detectar a ocorrência de uma falha após um número finito de observações depois da ocorrência da falha. Assim, a diagnose de falhas em SEDs possui uma grande relevância no projeto de sistemas complexos (Carvalho, 2011; Lin, 1994; Sampath et al., 1996) e, em consequência, neste trabalho será estudada a complexidade média de dois algoritmos usados em diagnose de falhas de SEDs modelados por autômatos, quais sejam: *(i)* o diagnosticador proposto por Sampath et al. (1995) e; *(ii)* o verificador da propriedade de diagnosticabilidade proposto por Moreira et al. (2011a). O problema da construção de diagnosticadores pertence à classe EXPTIME, de forma que é de grande interesse prático determinar se o algoritmo de síntese de um diagnosticador é computacionalmente tratável no caso médio.

Existem vários algoritmos conhecidos para a construção de verificadores (Cassez et al., 2007; Moreira et al., 2011a; Qiu and Kumar, 2006; Yoo and Lafortune, 2002b) cuja complexidade computacional é de ordem quadrática (e, por conseguinte, podem ser considerados algoritmos eficientes da classe P), porém, é de interesse determinar se, no caso médio, é possível obter uma eficiência superior à complexidade no pior caso. Para tanto, nesse trabalho iremos analisar a complexidade média do algoritmo proposto por Moreira et al. (2011a) para a construção de verificadores. A escolha desse algoritmo é feita pelas seguintes razões:

a) A implementação é simples e eficiente porque usa autômatos determinísticos e operações básicas entre estes.

b) A diferença de um fator linear entre as complexidades computacionais de algoritmos que resolvem o mesmo problema pode ser desconsiderada, de acordo com

o teorema da aceleração linear (Greenlaw and Hoover, 1998).

c) É de grande interesse para o nosso grupo de pesquisa completar o ciclo do método científico no estudo dos algoritmos propostos.

Na área de ciências da computação, os pesquisadores têm usado as seguintes metodologias para analisar a complexidade média em algoritmos que envolvem autômatos: *(i)* os métodos da *combinatória analítica*[3] e; *(ii)* os métodos da *algoritmia experimental*[4]. A primeira abordagem foi usada por Broda et al. (2011) para determinar a complexidade média de um autômato derivativo e, também, por Nicaud (2009) para estabelecer a complexidade média do autômato de Glushkov. A abordagem da algoritmia experimental foi usada por Almeida (2010) e Tabakov and Vardi (2005) para avaliar o desempenho de alguns algoritmos existentes para minimização de autômatos determinísticos.

No caso de algoritmos mais gerais (não necessariamente aqueles cujas instâncias são autômatos), a algoritmia experimental fornece metodologias testadas na derivação de modelos hipotéticos da complexidade de diversos algoritmos. A referência central do tema é o texto de McGeoch (2012) e, dentro dos propósitos deste trabalho, é também relevante mencionar o artigo de McGeoch et al. (2002) que apresenta os elementos fundamentais para derivar modelos de complexidade assintótica de algoritmos e, também, o artigo de Coffin and Saltzman (2000) que considera os métodos de análise estatística que devem ser usados na análise experimental de algoritmos.

Na contexto de SEDs modelados por autômatos, a avaliação experimental do algoritmos já foi utilizada em vários trabalhos, dentre os quais cabe mencionar os seguintes: na tese de Vahidi (2004) é utilizada para avaliar a eficiência dos diagramas de decisão binárias na síntese de supervisores para controle supervisório; no trabalho de Wang et al. (2007), para mostrar a eficacia dos métodos de controle supervisório de SEDs no desenvolvimento de programas concorrentes; no artigo de Grastien and Anbulagan (2013), para validar a metodologia de diagnose de falhas de SEDs usando algoritmos de satisfabilidade; e no trabalho de Pena et al. (2014) é usada para mostrar a eficiência prática do algoritmo de verificação da propriedade do observador proposto. Em todos os trabalhos revisados, a experimentação serve

[3] A *combinatória analítica* estuda as propriedades de estruturas combinatórias por meio de funções geradoras e cálculo de variável complexa. Um tratamento amplo do tema está em Flajolet and Sedgewick (2009) e uma panorâmica das aplicações da combinatória analítica em análise de complexidade de algoritmos que envolvem autômatos é apresentada em Broda et al. (2012).

[4] A algoritmia experimental (ou análise experimental de algoritmos) é uma área do estudo dos algoritmos usando meios experimentais. Assim, a algoritmia experimental estuda algoritmos integrando controle de parâmetros de entrada, construção de modelos e a análise estatística (McGeoch, 2012).

para realizar uma análise preliminar de desempenho dos algoritmos por meio de tempos de execução, com vistas a mostrar sua viabilidade prática.

Nesse ponto, é importante ressaltar que dentro do contexto de SEDs modelados por autômatos não existem resultados de análise de complexidade média de algoritmos (nem por métodos analíticos nem experimentais) e que os resultados de complexidade existentes para algoritmos em SEDs referem-se sempre à complexidade no pior caso. Uma vez que, no caso geral, a modelagem de um SED é realizada por composição paralela de sistemas mais simples e, por conseguinte o número de estados incrementa-se multiplicativamente, o tamanho típico de instâncias práticas de sistemas de eventos discretos pode atingir valores elevados. Por essa razão, considerar o problema da complexidade média pode ser de grande interesse inclusive para algoritmos considerados eficientes como, por exemplo, o algoritmo de Moreira et al. (2011b) (a ser estudado nesse trabalho) que possui complexidade quadrática em relação ao tamanho da instância.

Levando em consideração a importância prática do problema de diagnose de falhas, assim como a inexistência de resultados em complexidade média de algoritmos em SEDs (e, especificamente, em algoritmos para diagnose de falhas em SEDs), neste trabalho será estudada a complexidade média dos algoritmos para diagnose de falhas em SEDs mencionados acima usando, para tal efeito, métodos experimentais sobre instâncias geradas automaticamente por meio dos geradores de autômatos propostos nos trabalhos de Almeida et al. (2007), Bassino and Nicaud (2006, 2007), Bassino et al. (2008, 2009) e Héam et al. (2010).

Para poder usar métodos experimentais na análise de complexidade média de algoritmos em SEDs, torna-se necessária a utilização de uma ferramenta computacional com características muito específicas. A revisão das ferramentas de computação existentes para SEDs levou-nos ao desenvolvimento da biblioteca DESLAB (veja Clavijo et al. (2012) e referências) para facilitar o uso do método científico no estudo de algoritmos para SEDs descritos por autômatos. A proposta contida nessa ferramenta computacional será apresentada neste trabalho.

Como trabalho adicional desta tese de doutorado, será considerado o problema da diagnose de falhas intermitentes na operação de sensores (Carvalho et al., 2013a). Uma vez que a confiabilidade e segurança de um sistema que processa informação depende, em última instância, da operação adequada dos sensores que possui, é importante, do ponto de vista prático, detectar apropriadamante (e de forma imediata) o mau funcionamento de um sensor. Dentro da área SEDs, o problema de falhas na operação em sensores já foi abordado no contexto de controle supervisório (Rohloff, 2005; Sanchez and Montoya, 2006) e também no contexto de diagnose de falhas de SEDs (Carvalho et al., 2012, 2013b). Diferentemente das abordagens feitas nos

trabalhos de Rohloff (2005), Sanchez and Montoya (2006), Carvalho et al. (2012) e Carvalho et al. (2013b), na abordagem estudada aqui, não está sendo proposto um sistema que possa remediar o problema das falhas em sensores, mas um sistema que efetivamente detecta o mau funcionamento de um sensor. Para tanto, o modelo de perda de observações introduzido em Carvalho et al. (2012) será modificado limitando-o ao problema de detetar o mau funcionamento de um sensor e, dessa maneira, converter o problema de detetar falhas na operação de sensores em um problema equivalente de diagnosticabilidade de falhas intermitentes conforme proposto em Contant et al. (2004). Sob essa consideração, serão apresentadas condições necessárias e suficientes para a diagnosticabilidade de falhas de sensores intermitentes e, além disso, será proposto um teste para verificar a diagnosticabilidade de falhas intermitentes na operação de sensores.

Este trabalho está organizado da seguinte forma. No capítulo 2 é apresentada uma breve revisão dos conceitos fundamentais em SEDs modelados por autômatos incluindo os conceitos fundamentais de diagnose de falhas e controle supervisório. Além disso, é apresentado o algoritmo de construção do diagnosticador proposto por Sampath et al. (1995) e o algoritmo para construção do verificador proposto por Moreira et al. (2011a) cujas complexidades médias serão objeto de estudo nesse trabalho. No capítulo 3 é apresentada a ferramenta de computação científica DESLAB para análise e síntese de algoritmos para SEDs descritos por autômatos. No capítulo 4 é realizado um estudo preliminar da complexidade média do diagnosticador e do verificador por meio de um experimento exaustivo realizado sobre conjuntos de instâncias de cardinalidade moderada. Além disso, o capítulo inclui uma análise estatística dos dados experimentais obtidos para propor um possível modelo probabilístico da complexidade média do algoritmo de construção do diagnosticador. No capítulo 5 é realizado um experimento com amostragem sobre conjuntos de instâncias cujas cardinalidades são maiores que aquelas consideradas no capítulo 4. Com base na análise dos resultados experimentais são formuladas, nesse mesmo capítulo, as hipóteses sobre o crescimento da complexidade média dos algoritmos de construção do diagnosticador e do verificador. No capítulo 6 é abordado o problema da diagnose de falhas intermitentes na operação de sensores. Comentários finais e sugestões de tópicos futuros de pesquisa são apresentados no capítulo 7.

Capítulo 2

Sistemas a eventos discretos

O objetivo deste capítulo é apresentar uma breve revisão dos conceitos fundamentais em SEDs modelados por autômatos. Inicialmente, será apresentada uma breve revisão da teoria de sistemas a eventos discretos e, em seguida, a teoria de autômatos e linguagens formais. Finalmente serão revisadas as formulações centrais nas áreas de diagnose de falhas e controle supervisório de SEDs.

Sistemas a eventos discretos (SEDs) são sistemas dinâmicos que evoluem de acordo com as ocorrências abruptas de eventos físicos em intervalos possivelmente irregulares. Por exemplo, um evento poderia corresponder à chegada ou partida de um cliente numa fila atendida por um servidor, ao término de uma tarefa em um sistema de manufactura ou a ocorrência de um distúrbio ou mudança da referência num sistema de controle complexo. Os SEDs aparecem naturalmente na maioria das áreas da engenharia que tenham algum grau de processamento de informação.

A modelagem de sistemas a eventos discretos já foi utilizada numa grande variedade de áreas tais como: bases de dados (Lafortune, 1988), sistemas de manufactura (Cho and Lim, 1998), robótica (Wang et al., 2005), projeto de processadores e sistemas digitais (Balemi et al., 1993), trafego aéreo e veicular (Hsu et al., 1993), logística (transporte e armazenamento de bens, organização, e entrega de serviços) (Medrano and Enari, 2010), "workflows" e arquitetura de "software" (Liao et al., 2013; Wang et al., 2010) e redes de computação e comunicações (Rudie and Wonham, 1992). Novas aplicações em modelagem e controle de sistemas biológicos estão aparecendo, como, por exemplo, a modelagem de epidemias reportada em Brunsch and Rudie (2008) e a aplicação de diagnose de falhas em sistemas industriais (Nunes, 2012).

Este capítulo está dividido da seguinte forma. A seção 2.1 introduze os sistemas a eventos discretos e sua modelagem. As seções 2.2 e 2.3 tratam de linguagens formais e autômatos, respectivamente. Na seção 2.4 é formulado o problema da diagnose de falhas e na seção 2.5 é considerado o problema de controle supervisório e as metodologias para melhorar o problema da complexidade computacional.

Finalmente, na seção 2.6, são apresentadas as conclusões.

2.1 Modelagem de sistemas a eventos discretos

Em termos gerais, o problema da modelagem de sistemas de engenharia enuncia-se assim: dado um sistema de engenharia, formular um modelo conceitual e selecionar um modelo matemático de uma classe de modelos para serem usados com um propósito específico.

O uso de computadores é a maior motivação dos modelos escolhidos na teoria de sistemas discretos, sendo uma preocupação fundamental a correção dos programas de computador ou *software* que implementam boa parte da funcionalidade dos sistemas discretos usados em engenharia. Os modelos usados para SEDs têm os seguintes propósitos fundamentais:

1. Descrição e especificações. O modelo deve ser informativo e possibilitar que seja facilmente acrescido para incluir várias especificações.
2. Análise da operação de um sistema complexo. Essa capacidade do modelo inclui, atualmente, a possibilidade de simulação.
3. Diagnosticabilidade. Decidir rapidamente se o sistema considerado ainda está funcionando de acordo com as especificações iniciais.
4. Controle supervisório. Coordenar e controlar um fluxo adequado da informação.
5. Verificação formal. Determinar se um sistema de engenharia está bem projetado antes de passar para a etapa de produção.

Usam-se os seguintes critérios para a seleção da classe de modelos que melhor representam um sistema particular:

1. A concordância do modelo com a realidade em relação ao propósito do modelo.
2. A complexidade da classe de modelos.

A seleção do modelo é um compromisso entre *exatidão* e *complexidade*. O modelo deve aproximar o objeto "real" modelado tão exatamente quanto possível. Frequentemente, atingir esse alvo conduz a uma classe extensa de modelos. Por outro lado, o critério de complexidade impõe que o modelo deve ser maximamente simples. Para cada caso particular, devemos escolher o modelo mais adequado com base nos critérios de concordância e simplicidade.

Alguns dos modelos usualmente adotados para SEDs são:

1. Autômatos;
2. Redes de Petri;
3. Álgebra de processos;
4. Autômatos temporizados;
5. Autômatos híbridos.

A modelagem por autômatos possui as seguintes vantagens: uma coleção importante de problemas de engenharia podem ser modelados dentro dessa classe; a implementação computacional é simples e transparente; os algoritmos e operações entre autômatos são conhecidos; os autômatos podem ser transformados facilmente em semigrupos; existem ferramentas de software muito evoluídas para trabalhar com autômatos de grande porte; e, recentemente apareceram novas técnicas de geração aleatória e enumeração de autômatos. Por outro lado, operações simultâneas (paralelismo) são mais apropriadamente modeladas com redes de Petri.

Para ilustrar o uso de modelos de SEDs, a seguir vamos apresentar a modelagem de um processo de manufatura simplificado.

Exemplo 2.1. *(Cassandras and Lafortune (2008)) Considere uma célula de manufatura formada por duas máquinas (M_1 e M_2) e um robô que transporta as peças de M_1 para M_2. A máquina M_1 recebe peças brutas e quando as peças estão prontas, elas são recolhidas pelo robô. Caso o robô esteja ocupado, a máquina M_1 retém a peça até que o robô esteja completamente livre. Caso uma outra peça chegue enquanto a máquina M_1 estiver processando/retendo alguma peça, a máquina M_1 rejeita a peça recebida. Quando o robô recebe uma peça de M_1, inicia o transporte dessa peça até a máquina M_2. No momento em que chegar a M_2, o robô somente entregará a peça à máquina M_2 se estiver livre; caso contrário reterá a peça até M_2 ficar disponível. Após entregar a peça a M_2, o robô retorna à máquina M_1. A máquina M_2 recebe a peça do robô e a processa.*

A tabela 2.1 descreve os estados e os eventos das máquinas M_1 e M_2 e do robô. Note que os eventos e_1 (entrega de peça ao robô) e a_2 (entrega/chegada de peça em M_2) pertencem a dois subsistemas: máquina M_1 e robô, e robô e máquina M_2, respectivamente. É importante notar que para que o evento e_1 ocorra, a máquina M_1 deverá estar no estado H_1 e o robô no estado I; para que o evento a_2 ocorra, o robô deverá estar no estado H e a máquina M_2 deverá estar no estado I_2. Para os demais eventos do sistema, isto é, aqueles que estão presentes em somente um dos subsistemas, a ocorrência não dependerá do estado em que os demais subsistemas estiverem, sendo determinada somente pelo estado atual do subsistema; por exemplo, a ocorrência do evento t_1 (fim de processamento da peça em M_1) dependerá apenas

Tabela 2.1: Os estados e os eventos das máquinas M_1, M_2 e do robô.

Elemento	Estados	Eventos
Máquina M_1	M_1 disponível:I_1	Chegada de peça a M_1: a_1
	M_1 processando: P_1	Fim de processamento: t_1
	M_1 retendo peça pronta: H_1	Entrega de peça ao robô: e_1
	$X_1 = \{I_1, P_1, H_1\}$	$E_1 = \{a_1, t_1, e_1\}$
Robô	Robô disponível: I,	Entrega de peça ao robô: e_1
	Transportando M_1-M_2: T_{12}	Chegada a M_2: c_2
	Esperando em M_2: H	Entrega/chegada de peça a M_2: a_2
	Retornando para M_1: R	Chegada a M_1: r_1
	$X_r = \{I, T_{12}, H, R\}$	$E_r = \{e_1, c_2, a_2, r_1\}$
Máquina M_2	M_2 disponível: I_2	Entrega/chegada de peça em M_2: a_2
	M_2 processando: P_2	Fim de processamento: t_2
	$X_2 = \{I_2, P_2\}$	$E_2 = \{a_2, t_2\}$

da máquina M_1 estar no estado P_1, independentemente de quais estados estiverem o robô e a máquina M_2. □

2.2 Linguagens

Na atualidade, todos estamos familiarizados com a ideia da informação digitalizada. Por exemplo, é reconhecido que os computadores representam a informação com sequências contendo somente 0 ou 1. Embora um usuário interaja com um computador usando uma diversidade de formas de informação (gráficos, sons, voz etc.), a informação interna no computador sempre será representada por cadeias de 0s e 1s.

A informação, qualquer que seja seu conteúdo, é sempre representada por sequências de símbolos escolhidas de um conjunto fixo; por exemplo, a informação interna no computador sempre será representada por cadeias formadas por 0 e 1. Formalmente, o conjunto Σ de símbolos usados para representar a informação é chamado de *alfabeto*. Definimos como *sequência w sobre* Σ, à sequência formada por símbolos tomados de Σ (também referida como cadeia). Chamamos *eventos* a cada um dos elementos do conjunto Σ. A cardinalidade do alfabeto é denotada por $|\Sigma|$. Uma sequência vazia é denotada pelo simbolo ε. Para uma sequência w definida sobre Σ, seu comprimento é denotado por $\|w\|$. Por definição $\|\varepsilon\| = 0$.

Definimos Σ^* como o conjunto formado por todas as sequências de comprimento finito sobre o alfabeto Σ (incluindo a sequencia vazia ε). Uma *linguagem* L sobre Σ é qualquer subconjunto de Σ^*.

Exemplo 2.2. *Seja $\Sigma = \{a, b, c\}$. Pode-se definir, entre outras, as seguintes linguagens sobre Σ^*: $L_1 = \{\varepsilon, a, ab, abc\}$, $L_2 = \{a, aa, aabb\}$, $L_3 = \{$conjunto das sequências que contêm $ab\}$. Nesse exemplo, tem-se que $|L_1| = 4$, $|L_2| = 3$ e $|L_3| = \infty$.* ❑

Operações com Linguagens

Para um conjunto Y qualquer denotamos por 2^Y o conjunto formado por todos os subconjuntos de Y; também chamado *conjunto potência de* Y. Suponha que Σ seja um alfabeto. Como uma linguagem L sobre Σ é qualquer subconjunto de Σ^*, então o conjunto de todas as possíveis linguagens sobre Σ é 2^{Σ^*}. Se L_1 e L_2 forem linguagens sobre Σ, então $L_1 \cap L_2$, $L_1 \cup L_2$ e o complemento relativo de L_2 em relação a L_1 (denotado por $L_1 \setminus L_2$) também serão linguagens sobre Σ. Se L for uma linguagem sobre Σ, então $L^C = \Sigma^* \setminus L$ é a linguagem chamada *complementar de* L em relação a Σ^*.

Duas operações importantes, próprias da teoria de SEDs, são a *concatenação* e o *fecho de Kleene*. Se L_1 e L_2 forem linguagens então

$$L_1 L_2 = \{xy : x \in L_1 \text{ e } y \in L_2\}, \tag{2.1}$$

é a *concatenação* de L_1 e L_2. Para uma linguagem L, definimos $L^0 = \{\varepsilon\}$, e recursivamente, $L^{n+1} = L^n L$. O *fecho de Kleene* da linguagen L, denotado por L^*, é definido como:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots = \bigcup_{k=0}^{\infty} L^k \tag{2.2}$$

Definiremos, a seguir, operações que são importantes na teoria de sistemas a eventos discretos. Suponha que $u, v, w \in \Sigma^*$. Se $t = uvw$, então v é um *fator* de t, u é um *prefixo* de t e w é um sufixo de t. A relação u *é prefixo de* t define uma ordem parcial [1] na linguagem L , denotada por $u \leq t$. Para uma linguagem $L \in \Sigma^*$, o *fecho do prefixo de* L, denotado $\overline{L}$, é o conjunto

$$\overline{L} = \{v \in \Sigma^* : (\exists w \in L)[v \leq w]\},$$

isto é, L é formado por todos os prefixos das sequências de L. Assim, se, por exemplo $L = \{bb, abc\}$, então $\overline{L} = \{\varepsilon, b, bb, a, ab, abc\}$. Uma linguagem L tal que $L = \overline{L}$ é dita ser prefixo-fechada. Uma linguagem L é ser dita viva se para cada sequência w de L, existe pelo menos um $v \in L$ tal que $w \leq v$.

Para dois alfabetos Σ_o e Σ, tais que $\Sigma_o \subset \Sigma$, a *projeção* $P_{\Sigma_o} : \Sigma^* \to \Sigma_o^*$ (Ramadge and Wonham, 1987) é definida para cada sequência $w \in \Sigma^*$ da seguinte forma:

(P1) $P_{\Sigma_o}(\varepsilon) = \varepsilon$.

(P2) $P_{\Sigma_o}(\sigma) = \sigma$, se $\sigma \in \Sigma_o$.

(P3) $P_{\Sigma_o}(\sigma) = \varepsilon$, se $\sigma \in \Sigma \setminus \Sigma_o$.

(P4) $P_{\Sigma_o}(w\sigma) = P_{\Sigma_o}(w)P_{\Sigma_o}(\sigma), w \in \Sigma^*, \sigma \in \Sigma.$ (2.3)

[1] O leitor interessado em ordens em conjuntos pode revisar Holmes (1998)

Qualquer operação que seja definida para uma sequência pode ser estendida para uma linguagem se aplicarmos a operação em questão para cada uma das sequências da linguagem. Assim, por exemplo, para uma linguagem L, a projeção é definida como $P_{\Sigma_o}(L) = \bigcup_{w \in L} P_{\Sigma_o}(w)$.

O operador projeção inversa $P^{-1}_{\Sigma_o} : \Sigma^* \to 2^{\Sigma^*}$ é definido por

$$P^{-1}_{\Sigma_o}(v) = \{w \in \Sigma^* : P_{\Sigma_o}(w) = v\}. \tag{2.4}$$

Dada a linguagem L, define-se a pós-linguagem de L após uma sequência w (denotada por L/w), como

$$L/w = \{v \in \Sigma^* : wv \in L\}. \tag{2.5}$$

Exemplo 2.3. *Seja a linguagem* $L = \{a, c, ab, abbc, abcc\}$ *definida sobre* $\Sigma_1 = \{a, b, c\}$ *e, por sua vez, considere* $\Sigma_2 = \{a, b\}$.

- *A projeção de* L *sobre* Σ_2 *é* $M = P_{\Sigma_2}(L) = \{\varepsilon, a, ab, abb\}$. *A projeção inversa de* M *em* Σ_1 *é* $P^{-1}_{\Sigma_1}(M) = \{c^*\} \cup \{c^*ac^*\} \cup \{c^*ac^*bc^*\} \cup \{c^*ac^*bc^*bc^*\}$[2]. *Note que* $P^{-1}_{\Sigma_1}(M) = P^{-1}_{\Sigma_1}(P_{\Sigma_2}(L)) \supset L$.
- *A pós-linguagem de* L *após* ab *é* $L/(ab) = \{\varepsilon, bc, cc\}$. ❑

2.3 Autômatos

Na teoria de sistemas contínuos, definem-se as categorias de *sinais* e *sistemas*. Os sinais representam informação e os sistemas são os elementos que a processam. Um sistema é representado por um modelo matemático, por exemplo uma função de transferência. Na teoria de SEDs, o papel dos sinais é representado pelas sequências, sendo o autômato um sistema elementar de processamento de informação.

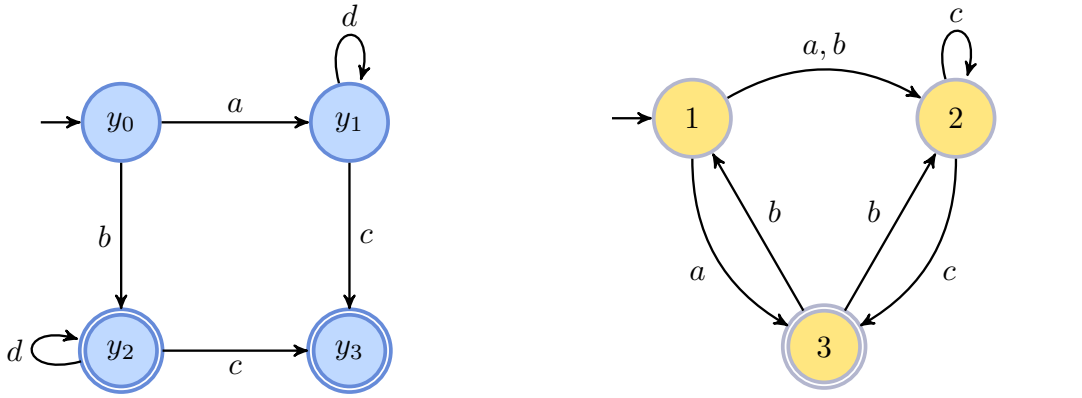
2.3.1 Autômatos determinísticos

Formalmente, definimos um autômato determinístico G sobre Σ como a sêxtupla

$$G = (X, \Sigma, f, \Gamma, x_0, X_m), \tag{2.6}$$

em que X denota o *espaço de estados*, Σ o conjunto de eventos, $f : X \times \Sigma \to X$ é a *função de transição de estados*, definida parcialmente no conjunto de eventos, $\Gamma : X \to 2^X$ é a *função dos eventos ativos*, x_0 é o *estado inicial do sistema* e

[2] Aqui a notação c^* foi usada para denotar $\{\varepsilon, c, cc, ccc, \dots\}$ e os colchetes da concatenação foram retirados para simplificar a notação



(a) Autômato determinístico G (b) Autômato não determinístico G_{nd}

Figura 2.1: Tipos de autômatos.

$X_m \subseteq X$ o conjunto dos *estados marcados*. Usam-se, de forma equivalente, as notações $f(x_a, \sigma) = x_b$, $x_a \xrightarrow{\sigma} x_b$ para a função de transição de estados.

Do ponto de vista da teoria de grafos, um autômato é um grafo rotulado (Sakarovitch, 2009). O rótulo de um estado inicial é uma seta sem estado de origem e o rótulo de um estado final é representado por duas circunferências concêntricas. Desconsiderando os rótulos dos estados iniciais e finais, a estrutura de dados subjacente ao autômato G é um *multidígrafo*.

Exemplo 2.4. *O grafo do autômato* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ *está representado na figura 2.1(a) . Para esse autômato,* $X = \{y_0, y_1, y_2, y_3\}$, $\Sigma = \{a, b, c, d\}$, $X_m = \{y_2, y_3\}$, $x_0 = y_0$. *A seta* $y_2 \xrightarrow{c} y_3$ *representa graficamente a função de transição de estados, denotada também como* $f(y_2, c) = y_3$. *A função* $\Gamma(x)$ *representa os eventos que são viáveis de acontecer no estado* x. *Por exemplo,* $\Gamma(y_1) = \{c, d\}$ *representa os eventos que podem ocorrer no estado* y_1. ❑

Por meio da função de transição de estado f fica completamente determinado o próximo estado de um autômato ao ser disparado um evento específico. Contudo, um autômato é uma máquina de cálculo com sequências. Um *caminho* p num autômato é uma sucessão de transições factíveis começando no estado inicial. Para o autômato da figura 2.1(a) um caminho válido é $\rho = y_0 \xrightarrow{a} y_1 \xrightarrow{d} y_1 \xrightarrow{c} y_3$. A sequência $w = adc \in \Sigma^*$, formada pelos eventos do caminho ρ, é chamada de *traço* e é denotada como $tr(\rho)$.

Para introduzir formalmente o cálculo com sequências, definimos a *função de transição de estados estendida*. Seja $f^* : X \times \Sigma^* \longrightarrow X$, com $\sigma \in \Sigma$ e $w \in \Sigma^*$,

definida pelas condições:

(FTE1) $f^*(x, \varepsilon) = \varepsilon$.

(FTE2) $f^*(x, \sigma) = \begin{cases} f(x, \sigma), \text{ se } f(x, \sigma) \text{ for definida.} \\ \text{não definida, caso contrário.} \end{cases}$

(FTE3) $f^*(x, \sigma w) = f^*(f(x, \sigma), w), \text{ se } f(x, \sigma) \text{ for definida.}$

Um autômato processa um subconjunto de sequências de Σ^*, isto é, uma linguagem sobre o alfabeto Σ. Do ponto de vista da teoria de SEDs, duas linguagens são importantes: a *linguagem gerada* pelo autômato, definida como

$$L(G) = \{w \in \Sigma^* : f^*(x_0, w) \text{ é definida}\},$$

e a *linguagem marcada*, que é definida como:

$$L_m(G) = \{w \in \Sigma^* : f^*(x_0, w) \in X_m\}.$$

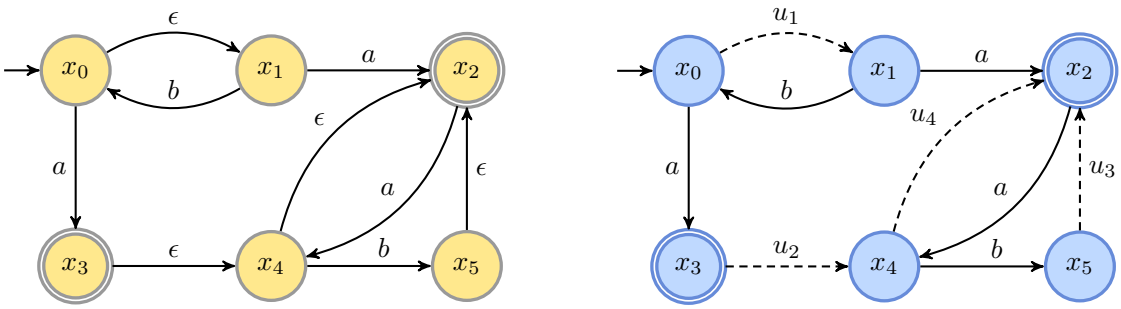
A linguagem gerada por um autômato é formada pelo conjunto dos traços que são definidos a partir de todos os caminhos possíveis começando no estado inicial, enquanto a linguagem marcada é o subconjunto da linguagem gerada formado por traços que terminam em estados marcados.

2.3.2 Autômatos não determinísticos

Um autômato não determinístico é definido pela sêxtupla $G_{nd} = (X, \Sigma, f, \Gamma, X_0, X_m)$, em que X é o conjunto dos estados, Σ o alfabeto de entrada, $X_0 \subset X$ o conjunto dos estados iniciais, Γ a função dos eventos ativos e $f : X \times \Sigma \longrightarrow 2^X$. Além da possibilidade de conter mais de um estado inicial, uma outra característica importante de um autômato não determinístico é que o contradomínio da função $f(x, \sigma)$ é agora um subconjunto de X e não um estado.

Exemplo 2.5. *Considere o autômato não determinístico G_{nd}, da figura 2.1(b). A função de transição de estados f assume valores em 2^X, para cada valor de $x \in X$. Por exemplo, $f(1, a) = \{2, 3\}$ e $f(3, b) = \{1, 2\}$. Essa correspondência a vários estados reflete incerteza no conhecimento da evolução dinâmica do sistema.* ❑

De forma similar ao caso dos autômatos determinísticos, estendemos a função de transição de estados f, para que o autômato G_{nd} processe cadeias. Sendo $\sigma \in \Sigma$, $x \in X$ e $w \in \Sigma^*$, a função de transição de estados estendida $f^* : X \times \Sigma^* \to 2^X$ é



(a) Autômato G_ϵ (b) Autômato determinístico G_{op}

Figura 2.2: Um autômato-ε e sua versão determinística com observação parcial.

definida pelas condições a seguir:

$$\begin{aligned}
&\text{(FTE1)} \quad f^*(x,\varepsilon) = \{x\}\,;\\
&\text{(FTE2)} \quad f^*(x,\sigma) = f(x,\sigma);\\
&\text{(FTE3)} \quad f^*(x,\sigma w) = \bigcup_{q\in f(x,\sigma)} f^*(q,w).
\end{aligned} \tag{2.7}$$

Suponha que $\sigma \in \Sigma$ e $w \in \Sigma^*$. Então, a linguagem gerada pelo autômato G_{nd} não determinístico é definida como

$$L(G_{nd}) = \{w \in \Sigma^* : \left(\bigcup_{x_0\in X_0} f^*(x_0,w)\right) \neq \emptyset\}.$$

e a *linguagem marcada* como

$$L_m(G_{nd}) = \{w \in \Sigma^* : \left(\bigcup_{x_0\in X_0} f^*(x_0,w)\right) \cap X_m \neq \emptyset\}.$$

2.3.3 Autômatos com transições ε

Nos casos anteriores de autômatos determinísticos e não determinísticos, a mudança de estado é a resposta a um evento detectado. Na classe de autômatos com transições ε (chamados de autômatos-ε), o estado do autômato pode mudar espontaneamente, sem que seja detectado qualquer evento[3] . Os autômatos-ε são importantes, por exemplo, para a obtenção de linguagens regulares como resultado de operações booleanas. Do ponto de vista da modelagem de sistemas a eventos discretos, são importantes para representar falhas ou ausência de sensores para alguns eventos.

[3]Alguns autores não fazem diferença entre autômatos não determinísticos e autômatos não determinísticos com transições ε. Seguindo o ponto de vista em Lawson (2004) aqui os apresentamos de forma separada porque, do ponto de vista computacional, a função de transição de estados é diferente.

Um autômato não determinístico com transições-ε ou, de forma mais simples, um autômato-ε é uma sêxtupla $G_\varepsilon = (X, \Sigma \cup \{\varepsilon\}, f_\varepsilon, \Gamma, X_0, X_m)$, em que cada símbolo tem a mesma definição que no autômato não determinístico da seção 2.3.2. A função $f_\varepsilon : X \times \Sigma \cup \{\varepsilon\} \longrightarrow 2^X$, é aumentada em seu domínio de definição pelo evento ε. Dito de outra forma, no autômato-ε são possíveis transições espontâneas e invisíveis do tipo $x_3 \xrightarrow{\varepsilon} x_4$ (ver figura 2.2(a)).

Exemplo 2.6. *Considere o autômato G_ε da figura 2.2(a). Para esse autômato a sequência ab é um traço válido. É importante ressaltar que nesse autômato não existe exatamente uma sequência de transições da forma $x_m = f^*(x_0, ab)$. No entanto, existem as sequências de transições $x_5 = f^*(x_0, a\varepsilon b)$ e $x_2 = f^*(x_0, a\varepsilon b\varepsilon)$ que são observadas equivalentemente à própria sequência ab.* ❑

Para definir as linguagens gerada e marcada por um autômato-ε é necessário introduzir o conceito de alcance-ε. O alcance-ε do estado x, denotado por $E_R(x)$, é definido como o conjunto de todos os estados que podem ser alcançados a partir do estado x seguindo unicamente transições rotuladas pelo evento ε. Por definição $x \in E_R(x)$. Para um conjunto Q de estados, estendemos naturalmente a definição de $E_R(x)$ da seguinte forma:

$$E_R(Q) = \bigcup_{q \in Q} E_R(q) \tag{2.8}$$

No autômato da figura 2.2(a) tem-se que $E_R(x_0) = \{x_0, x_1\}$, $E_R(x_1) = \{x_1\}$, $E_R(x_2) = \{x_2\}$, $U_R(x_3) = \{x_2, x_3, x_4\}$, $E_R(x_4) = \{x_2, x_4\}$ e $E_R(x_5) = \{x_2, x_5\}$.

Suponha que $Q \subseteq X$ seja um subconjunto dos estados do autômato G_ε e considere o evento $\sigma \in \Sigma$. Usamos a notação $F(Q, \sigma)$ para representar o conjunto $\bigcup_{q \in Q} f_\varepsilon(q, \sigma)$; isto é, um estado x pertence ao conjunto $F(Q, \sigma)$ quando existirem um estado $q \in Q$ e uma transição $q \xrightarrow{\sigma} x$ em G_ε rotulada por σ. Sejam $\sigma \in \Sigma$, $x \in X$ e $w \in \Sigma^*$. Então, a função de transição de estados estendida $f_\varepsilon^* : X \times \Sigma^* \to 2^X$ para um autômato-ε, é definida pelas seguintes condições:

$$\begin{aligned}
&\text{(FTE1)} \quad f_\varepsilon^*(x, \varepsilon) = E_R(x). \\
&\text{(FTE2)} \quad f_\varepsilon^*(x, \sigma) = E_R(F(E_R(x), \sigma)). \\
&\text{(FTE3)} \quad f_\varepsilon^*(x, \sigma w) = \bigcup_{q \in E_R(F(E_R(x), \sigma))} f_\varepsilon^*(q, w).
\end{aligned} \tag{2.9}$$

A linguagem gerada pelo autômato-ε é definida como

$$L(G_\varepsilon) = \{w \in \Sigma^* : [\exists x_0 \in X_0](f_o^*(x, w) \neq \emptyset)\}.$$

e a linguagem marcada como

$$L_m(G_\varepsilon) = \{w \in \Sigma^* : [\exists x_0 \in X_0](f_o^*(x, w) \cap X_m \neq \emptyset)\}.$$

Conforme dito anteriormente, os autômatos-ε servem para representar alguma perda da informação de sensores. Não obstante, essa perda de informação poderia ser modelada utilizando autômatos determinísticos introduzindo o conceito de *observação parcial*. Assim, o autômato G_{op} da figura 2.2(b), ao invés de possuir transições rotuladas pelo evento ε, possui transições de tipo $x_i \xrightarrow{u_i} x_j$ com $u_i \in \Sigma_{uo}$, sendo Σ_{uo} o conjunto de eventos não-observáveis do sistema modelado[4]. Representaremos essas transições com linha tracejada no diagrama de transição de estados do autômato (veja a transição $x_0 \xrightarrow{u_1} x_1$ da figura 2.2(b)). O alfabeto Σ do autômato determinístico G_{op} é particionado da forma $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. O conjunto Σ_o representa os *eventos observáveis*, isto é, dos quais possuímos informação relativa à sua ocorrência por meio de sensores. O conjunto Σ_{uo} contém *eventos não-observáveis*, isto é, dos quais temos unicamente informação estrutural na modelagem do sistema, isto é, informação de que eles produzirão mudanças de estado no sistema, sem poder registrar exatamente sua ocorrência.

Seja $G_{op} = (X, \Sigma, f, \Gamma, x_0, X_m)$ um autômato determinístico com observação parcial em que $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. Definimos o alcance não-observável do estado x, denotado por $U_R(x)$, como o conjunto de todos os estados que podem ser alcançados a partir do estado x seguindo unicamente transições rotuladas por eventos não-observáveis $u_i \in \Sigma_{uo}$. Por definição $x \in U_R(x)$. Usaremos a notação $F(Q, \sigma)$ para representar o conjunto $\bigcup_{q \in Q} f(q, \sigma)$; isto é, o estado x pertence ao conjunto $F(Q, \sigma)$, se existirem um estado $q \in Q$ e uma transição $q \xrightarrow{\sigma} x$ em G_{op} rotulada pelo evento σ. Assim, é possível definir a função de transição de estados observada

$$f_o \ : X \times \Sigma_o \to 2^X$$
$$f_o(x, \sigma) = U_R(F(U_R(x), \sigma)),$$

para denotar o conjunto de possíveis estados que pode ser atingido a partir do estado x após a ocorrência do evento $\sigma \in \Sigma_o$. Por exemplo, para o autômato da figura 2.2(b), $U_R(x_3) = \{x_2, x_3, x_4\}$ e $f_o(x_0, b) = \{x_0, x_1\}$.

A figura 2.3(a) ilustra equivalências entre diferentes tipos de autômatos. Partindo de um autômato G_ε é possível encontrar um autômato não determinístico G_{nd} e, a partir desse, um autômato determinístico G_{obs}, todos com as mesmas linguagens gerada e marcada. De forma análoga tem-se que a partir do modelo determinístico

[4]Eventos não-observáveis são aqueles cujas ocorrências não podem ser registradas por sensores ou são assim consideradas devido à natureza distribuída do sistema.

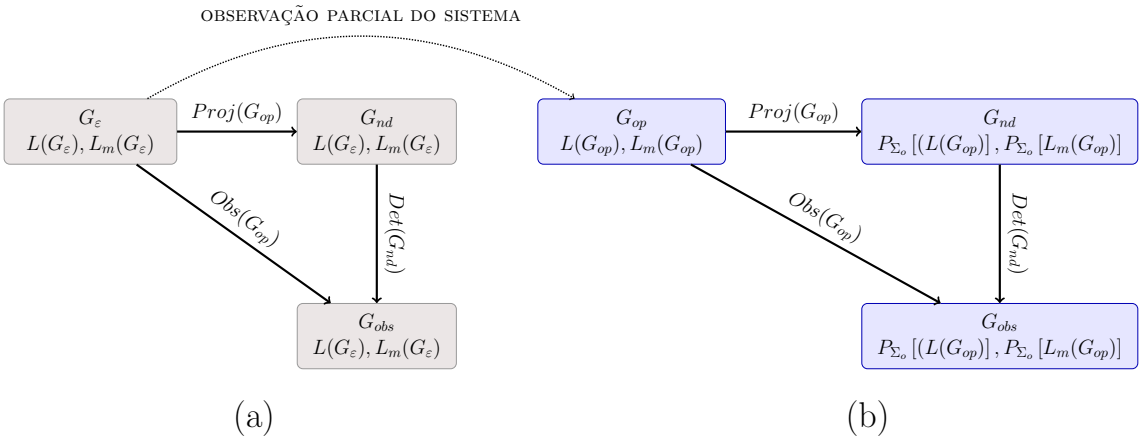


Figura 2.3: Conversão de autômatos-ε (a); Conversão de autômatos com observação parcial (b).

G_{op} que possui observação parcial, é possível calcular um modelo G_{nd} não determinístico e um modelo determinístico G_{obs}, cujas linguagens gerada e marcada são projeções sobre Σ_o de $L(G_{op})$ e $L_m(G_{op})$, conforme ilustrado na figura 2.3(b).

O algoritmo de conversão entre um autômato determinístico com observação parcial no autômato observador $Obs(G_{op})$ é apresentado no algoritmo 2.1. O mesmo algoritmo, com alterações menores, pode ser usado para o cálculo de $G_{obs} = Det(G_\varepsilon)$ (veja Cassandras and Lafortune (2008)). Algoritmos para o cálculo de $Proj(G_{op})$ e $Proj(G_\varepsilon)$ podem ser encontrados em Opitz (2006) e um algoritmo de cálculo de $Det(G_{nd})$ pode ser encontrado em van Glabbeek and Ploeger (2008).

Algoritmo 2.1: Construção de um observador para um autômato com observação parcial

---

Entradas: O autômato com observação parcial $G_{op} = (X, \Sigma, f, \Gamma, x_0, X_m)$ com conjunto de eventos observáveis Σ_o.
Saída: O autômato determinístico com observação total

$$G_{obs} = Obs(G) = (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, x_{0,obs}, X_{m,obs}).$$

1. Defina $x_{0,obs} = U_R(x_0)$. Inicialize $X_{obs} = \{x_{0,obs}\}$.
2. Para cada $Q \in X_{obs}$ e $\sigma \in \Sigma$ calcule $Q' = U_R(F(Q, \sigma))$.
3. Se $Q' \neq \emptyset$ defina $f_{obs}(Q, \sigma) = Q'$. Caso contrário $f_{obs}(Q, \sigma)$ é indefinida.
4. Repita os passos 2 e 3 até que toda a parte acessível de $G_{obs} = Obs(G_{op})$ tenha sido gerada.
5. Defina $X_{m,obs} = \{Q \in X_{obs} : Q \cap X_m \neq \emptyset\}$.

Observação 2.1. *Em relação à complexidade computacional, os algoritmos para o cálculo de $Obs(G_{op})$ e $Det(G_\varepsilon)$ possuem, no pior caso, complexidade exponencial na*

cardinalidade do espaço de estados do autômato de entrada. Em termos da notação $\mathcal{O}$, a complexidade no cálculo dessas operações é da ordem $\mathcal{O}(2^n)$ sendo $n = |X|$ a cardinalidade do conjunto de estados do autômato de entrada (Moore, 1971; Wong, 1998).

2.3.4 Operações com autômatos

Com o objetivo de se modelar SEDs, é necessário definir algumas operações sobre autômatos. Algumas das operações a seguir são válidas somente para autômatos determinísticos, enquanto outras são válidas para a classe mais geral de autômatos não determinísticos. Se a operação for definida tanto para autômatos determinísticos quanto para não determinísticos, usaremos como operando H, e caso esteja limitada aos autômatos determinísticos usaremos o operando G.

Parte acessível

Considere um autômato $H = (X, \Sigma, f, \Gamma, X_0, X_m)$. Definimos o conjunto dos estados accessíveis a partir do estado inicial da seguinte forma:

$$X_{ac} = \{x \in X : ((\exists w \in \Sigma^*) \wedge (\exists x_0 \in X_0))[f^*(x_0, w) = x]\}.$$

Seja $Ac(H) = (X_{ac}, \Sigma, f_{ac}, X_0, X_{ac,m})$ o autômato resultante da retirada dos estados não-accessíveis e suas transições. Para esse autômato $X_{ac,m} = X_{ac} \cap X_m$ e f é restrita a a X_{ac}, isto é, $f_{ac} = f|_{X_{ac} \times \Sigma \to X_{ac}}$.

Parte coacessível

Para um autômato $H = (X, \Sigma, f, \Gamma, X_0, X_m)$, definimos o conjunto dos estados coacessíveis da seguinte forma:

$$X_{coac} = \{x \in X : ((\exists w \in \Sigma^*) \wedge (\exists x_m \in X_m))[f^*(x, w) = x_m]\}$$

Seja $CoAc(H) = (X_{coac}, \Sigma, f_{coac}, X_{0,coac}, X_m)$ o autômato resultante da retirada dos estados não coaccessíveis. Nesse autômato, $X_{0,coac} = X_0 \cap X_{coac}$ e f é restrita a X_{coac}, isto é, $f_{coac} = f|_{X_{coac} \times \Sigma \to X_{coac}}$.

Autômato Trim

Um autômato é dito *Trim* se todos os seus estados são acessíveis e coacessíveis. Formalmente, a operação *Trim* é definida como:

$$Trim(H) = CoAc(Ac(H)) = Ac(CoAc(H)).$$

As operações $Ac(\cdot)$ e $CoAc(\cdot)$ determinam se a linguagem marcada por uma autômato é ou não vazia. Assim, uma linguagem marcada por um autômato é vazia se não existirem estados marcados accessíveis ou estados iniciais coacesíveis.

Complementar

Considere um autômato determinístico $G = (X, \Sigma, f, \Gamma, x_0, X_m)$, cuja linguagem marcada é $M = L_m(G)$. Denotamos G_C ao autômato cuja linguagem marcada é $L_m(G_C) = M^C = \Sigma^* \setminus M$, sendo $G_C = Comp(G)$. Denotaremos por $Comp(\cdot)$ à operação complementar, definida pelos passos a seguir:

1. Transforme o autômato G, em um autômato completo, cuja linguagem gerada seja Σ^*. Para esse efeito, defina $G_T = (X_T, \Sigma, f_T, \Gamma_T, x_0, X_m)$, em que o conjunto de estados é $X_T = X \cup \{x_d^*\}$, sendo x_d^* um estado acrescentado para completar o autômato . Defina a função de transição de estados $f_T : X_T \times \Sigma \to X_T$ de G_T como:

   - $f_T(x, \sigma) = x',$ se $f(x, \sigma) = x'$;
   - $f(x, \sigma) = x_d^*,$ se $\sigma \in \Sigma \setminus \Gamma(x)$ e $x \in X$;
   - $f(x_d^*, \sigma) = x_d^*,$ se $\sigma \in \Sigma$. .

2. Com o autômato G_T do passo 1, define-se G_C como o autômato obtido desmarcando-se os estados marcados e marcando-se os não marcados, isto é, $X_{m,C} = X_T \setminus X_m$. Com esta mudança obtém-se o autômato $G_C = (X_T, \Sigma, f_T, \Gamma_T, x_0, X_{m,c})$ que satisfaz $L_m(G_C) = M^C$.

Projeção inversa

Considere as linguagens $K_o = L(G) \subset \Sigma_o^*$ e $K_{m,o} = L_m(G)$. Seja Σ um conjunto tal que $\Sigma_o \subset \Sigma$. Seja a projeção $P_{\Sigma_o} : \Sigma^* \to \Sigma_o^*$. Um autômato que gera $P_{\Sigma_o}^{-1}(K_o)$ e marca $P_{\Sigma_o}^{-1}(K_{m,o})$ pode ser obtido adicionando-se autolaços rotulado pelos eventos em $\Sigma \setminus \Sigma_o$ em todos os estados de G.

Fecho de Kleene

Considere um autômato determinístico G, tal que $L_m(G) = L_1$. Construa o autômato-ε H^* tal que $L_m(H) = L_1^*$, da seguinte forma:

1. Acrescente um novo estado inicial, marque-o e conecte-o ao antigo estado inicial de G por meio de uma transição ε.

2. Coloque transições ε ligando cada estado marcado de G até o antigo estado inicial. Denote o autômato-ε resultante como H_ε.

3. Se precisar, obtenha um autômato determinístico por meio da operação $Obs(H_\varepsilon^*)$.

O autômato H^*, assim construído, marca a linguagem $L_m(H^*) = L_1^*$.

Produto ou interseção das linguagens

Considere os autômatos

$$H_1 = (X_1, \Sigma_1, f_1, \Gamma_1, X_{01}, X_{m1}) \text{ e } H_2 = (X_2, \Sigma_2, f_2, \Gamma_2, X_{02}, X_{m2}).$$

Formalmente o autômato produto é definido como

$$H_1 \times H_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\times 2}, \Gamma_{1\times 2}, X_{01} \times X_{02}, X_{m1} \times X_{m2}),$$

em que

- $f_{1\times 2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), \text{ se } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ \text{não definida, caso contrário.} \end{cases}$
- $\Gamma_{1\times 2}((x_1, x_2)) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$.

O produto dos autômatos H_1 e H_2 corresponde à interseção das linguagens gerada e marcada por eles. Dessa maneira, pode se facilmente provar que :

$$L(H_1 \times H_2) = L(H_1) \cap L(H_2),$$
$$L_m(H_1 \times H_2) = L_m(H_1) \cap L_m(H_2).$$

Composição paralela (interseção das projeções inversas)

Considere os autômatos

$$H_1 = (X_1, \Sigma_1, f_1, \Gamma_1, X_{01}, X_{m1}) \text{ e } H_2 = (X_2, \Sigma_2, f_2, \Gamma_2, X_{02}, X_{m2}).$$

A composição paralela dos autômatos H_1 e H_2 é definida pelo autômato

$$H_1 \parallel H_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\parallel 2}, \Gamma_{1\parallel 2}, X_{01} \times X_{02}, X_{m1} \times X_{m2})$$

em que

$$f_{1\|2}((x_1,x_2),\sigma) = \begin{cases} (f_1(x_1,\sigma), f_2(x_2,\sigma)), \text{ se } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ (f_1(x_1,\sigma), x_2), \text{ se } \sigma \in \Gamma_1(x_1) \setminus \Sigma_2, \\ (x_1, f_2(x_2,\sigma)), \text{ se } \sigma \in \Gamma_2(x_2) \setminus \Sigma_1, \\ \text{não definida, caso contrário.} \end{cases}$$

Definindo as projeções $P_1 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_1^*$ e $P_2 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_2^*$, as linguagens gerada e marcada pelo autômato $H_1 \parallel H_2$ são:

$$L(H_1 \parallel H_2) = P_1^{-1}(L(H_1)) \cap P_2^{-1}(L(H_2))$$
$$L_m(H_1 \parallel H_2) = P_1^{-1}(L_m(H_1)) \cap P_2^{-1}(L_m(H_2))$$

União

Sejam G_1 e G_2 autômatos determinísticos, tais que $L_1 = L_m(G_1)$ e $L_2 = L_m(G_2)$. O autômato-ε H_{1+2} cuja linguagem marcada é $L_m(H_{1+2}) = L_1 \cup L_2$ é obtido introduzindo-se um novo estado inicial que esteja conectado, por meio de duas transições ε, aos estados iniciais de G_1 e G_2.

Concatenação

Sejam G_1 e G_2 autômatos determinísticos, tais que $L_1 = L_m(G_1)$ e $L_2 = L_m(G_2)$. O autômato H_{conc} cuja linguagem marcada é $L_m(H_{conc}) = L_1L_2$ é obtido ligando-se com transições ε, os estados marcados de G_1 ao estado inicial de G_2 e, em seguida, desmarcando-se os estados os estados do autômato resultante que correspondiam aos estados marcados de G_1.

2.4 Diagnose de falhas em sistemas a eventos discretos

Assim como os sistemas tecnológicos tornam-se mais complexos, também suas falhas tornam-se mais difíceis de se prever, entender e concertar. Por essa razão, um sistema ao ser projetado, ou manufaturado, deve garantir que as falhas que possam ocorrer sejam identificáveis. Essa propriedade intrínseca, chama-se de *diagnosticabilidade* e, idealmente, deve ser garantida na etapa de projeto. Quando o sistema está em funcionamento, tem-se o problema da *diagnose*, isto é, observar o sistema e detectar a ocorrência de uma falha após um número finito de observações depois da ocorrência da falha.

A diagnosticabilidade baseada em modelo do sistema tem sido desenvolvida, paralelamente, nas áreas de controle automático e inteligência artificial. Um panorama completo do estado atual da teoria da diagnose e das abordagens existentes encontra-se em Carvalho (2011), Basilio et al. (2010) e Zaytoon and Lafortune (2013). De uma forma geral, a diagnosticabilidade basada em modelos é abordada por *métodos baseados em estados* ou *métodos baseados em eventos* (veja os artigos seminais dessas abordagens em Lin (1994) e Sampath et al. (1995), respectivamente). No primeiro caso, observações e falhas dependem do estado do sistema, sendo cada falha associada a um conjunto de estados.

Nos métodos baseados em eventos, considera-se que cada falha seja um evento não-observável, produzindo transições entre estados cujo conhecimento não acrescenta informação. O marco da teoria da diagnose de sistemas a eventos discretos com base em eventos é o trabalho de Sampath et al. (1995). Nesse trabalho, além de se estabelecer condições de diagnosticabilidade para um sistema, foi proposta a construção do "autômato diagnosticador" usado tanto para verificar a diagnosticabilidade de um sistema na fase de projeto, quanto a diagnose online, isto é, com o sistema funcionando. Partindo dessa abordagem, várias extensões foram propostas para sistemas distribuídos (Basilio and Lafortune, 2009; Debouk et al., 2000; Pencolé and Cordier, 2005), para verificar a diagnosticabilidade minimizando-se o custo computacional (Moreira et al., 2011a), para tipos de falhas intermitentes (Carvalho et al., 2012), entre outras. A seguir serão apresentados os conceitos básicos, de uma forma similar à apresentada em Cassez et al. (2007).

Suponha que um sistema modelado pelo autômato $G = (X, \Sigma, f, \Gamma, x_0, X_m)$, possua um único tipo de falha denotado pelo evento σ_f e gera a linguagem $L(G)$. Adicionalmente, suponha que o conjunto de eventos de G seja particionado em eventos observáveis e não-observáveis da forma $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ e que exista um *evento de falha* $\sigma_f \in \Sigma_{uo}$. Um traço $w \in L(G)$ da forma $w = w_1 \sigma_f w_2$ é chamado de errôneo-k se $\|w_2\| \geq k$ e será chamado traço errôneo se for um traço errôneo-k para algum $k \in \mathbb{N}$. Define-se $L_Y(G)$ como o conjunto dos traços errôneos em L e, complementarmente, $L_N(G) = L(G) \setminus L_Y(G)$, o conjunto de traços normais. Para $k \in \mathbb{N}$, define-se $L_{Y \geq k}(G)$ como o conjunto dos traços errôneos-k de $L(G)$. Verifica-se que $L_{Y \geq k+1}(G) \subseteq L_{Y \geq k}(G) \subseteq \cdots \subseteq L_{Y \geq 0}(G)$.

Um diagnosticador é um dispositivo que observa a planta e gera alarme quando detecta uma falha. Esse alarme pode ser imediato ou ser gerado depois de algum tempo. O tempo transcorrido até gerar o alarme é determinado pelo número k de eventos observáveis e não observáveis que acontecem depois da falha.

Definição 2.1. *(Cassez et al., 2007)* **(Diagnosticador-(Σ_o,k))** *Seja o SED* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$*, sendo* $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ *e* $\sigma_f \in \Sigma_{uo}$*. Ao mapeamento* $D : \Sigma_o^* \rightarrow \{N, Y\}$ *dá-se o nome de diagnosticador-*(Σ_o,k) *para* G *se:*

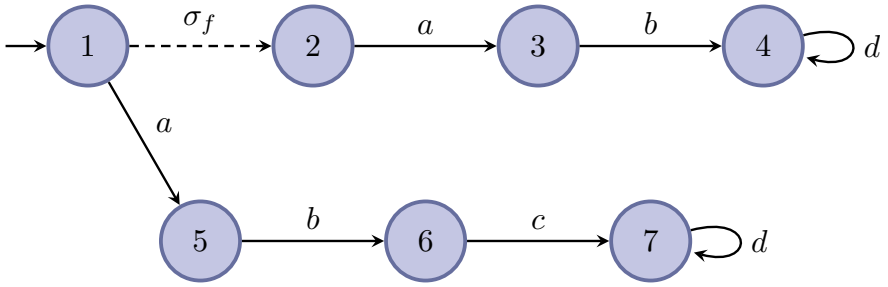


Figura 2.4: Autômato G que modela o sistema com a falha σ_f, para ser diagnosticada.

i. para cada $w \in L_N(G)$, $D(P_{\Sigma_o}(w)) = N$,

ii. para cada $w \in L_{Y\geq k}(G)$, $D(P_{\Sigma_o}(w)) = Y$. □

Definição 2.2. *Um SED G é dito diagnosticável-(Σ_o,k), se existe um diagnosticador-(Σ_o,k) para G.* □

Definição 2.3. *Um SED G é dito diagnosticável se existe $k \in \mathbb{N}$ tal que $L(G)$ é diagnosticável-(Σ_o,k).* □

Verificar se um SED G é diagnosticável pode ser feito em $\mathcal{O}(|X|^2)$, com $|X|$ o número de estados de G (Jiang et al., 2001; Moreira et al., 2011a; Qiu and Kumar, 2006). Determinar o mínimo k tal que G seja diagnosticável-(Σ_o,k) pode ser calculado em $\mathcal{O}(|X|^3)$ (Cassez et al., 2007; Yoo and Garcia, 2009).

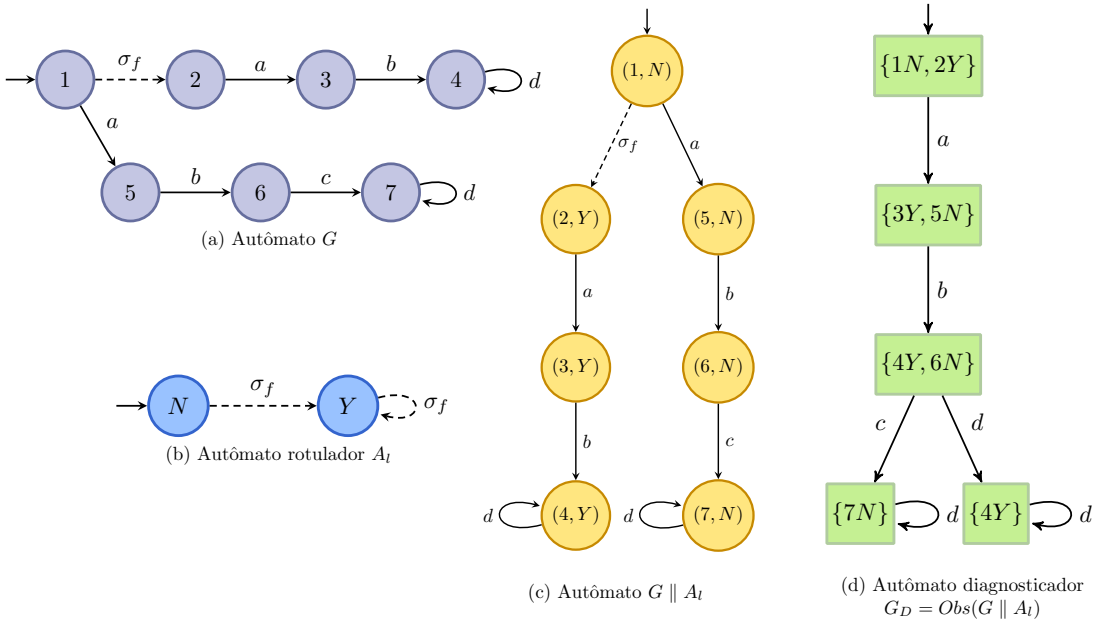
Exemplo 2.7. *Considere o sistema da figura 2.4. Se $\Sigma_o = \{a, b, c, d\}$, o sistema é $(\Sigma_o, 3) - diagnosticável$. O diagnosticador D nesse caso é $D(abdp) = Y$ para qualquer $p \in \Sigma_o^*$ e $D(abcp) = N$. Por outro lado, suponha que o conjunto de eventos observáveis seja $\Sigma_o = \{a, b, d\}$. Nesse caso os traços $\sigma abd^k \in L_{Y\geq k}(G)$ e $abcd^k \in L_N(G)$ têm ambiguidade para todo $k \in \mathbb{N}$ e, portanto, não existe um diagnosticador-(Σ_o,k) para nenhum valor de k e o sistema não é diagnosticável.* ❑

Caso G seja diagnosticável-Σ_o, é possível encontrar o autômato G_D que realiza o diagnosticador D (Sampath et al., 1995). Esse diagnosticador é calculado por meio da seguinte equação:

$$G_D = Obs(G \parallel A_l), \tag{2.10}$$

sendo A_l o autômato rotulador da figura 2.5(b).

Observação 2.2. *Se n representa a cardinalidade do espaço de estados do autômato G, a complexidade computacional no pior caso de G_D é 2^{2n} ($\mathcal{O}(2^{2n})$), devido ao cálculo do observador (na equação (2.10)) para o autômato de entrada $G \parallel A_l$, o qual pode ter (também no pior caso) $2n$ estados.*



(a) Autômato G

(b) Autômato rotulador A_l

(c) Autômato $G \parallel A_l$

(d) Autômato diagnosticador $G_D = Obs(G \parallel A_l)$

Figura 2.5: Passos para a construção de um diagnosticador para o autômato G (a)

Exemplo 2.8. *A figura 2.5 ilustra o cálculo do autômato diagnosticador G_D para o sistema G (figura 2.5(a)) com $\Sigma_{uo} = \{\sigma_f\}$. Note que quando acontecer a sequência $abdd^k$ ($k \in \mathbb{N}$) em G_D teremos certeza da ocorrência da falha σ_f.* ❑

Para verificar se um SED G é ou não diagnosticável é possível usar o algoritmo 2.2 proposto por Moreira et al. (2011a). O algoritmo está baseado no fato de que para um SED G diagnosticável, tem-se que:

$$P_{\Sigma_0}(L_{Y\geq k}(G)) \cap P_{\Sigma_0}(L_N(G)) = \emptyset, \tag{2.11}$$

isto é, não existe um par de traços (s_1, s_2) com $s_1 \in L_{Y\geq k}(G)$ e $s_2 \in L_N(G)$ tais que $P_{\Sigma_0}(s_1) = P_{\Sigma_0}(s_2)$. Assim, o algoritmo 2.2 fornece uma forma eficiente de buscar pares de traços (s_1, s_2) (que violam a condição (2.11)) por meio da construção de um *autômato verificador*, o qual será denotado por G_V. O autômato verificador G_V permite determinar, por meio de suas componentes fortemente conexas, se a condição (2.11) é violada por algum par de traços (s_1, s_2).

Observação 2.3. *Se n representa a cardinalidade do espaço de estados do autômato G, a complexidade computacional no pior caso de G_V é $2n^2$ ($\mathcal{O}(n^2)$) (Moreira et al., 2011a).*

Algoritmo 2.2: Verificação da diagnosticabilidade de um SED G usando o algoritmo proposto em Moreira et al. (2011a).

Entradas: O autômato G cuja diagnosticabilidade será verificada e o conjunto de eventos não-observáveis Σ_{uo}.
Saídas: O autômato verificador G_V e o resultado do teste de diagnosticabilidade.

Passo 1: A partir do autômato G construa o autômato de não falha G_N da seguinte forma:

- *Passo 1.1:* Obtenha o autômato G_{N1} apagando as transições de G rotuladas pelo evento σ_f.
- *Passo 1.2:* Obtenha o autômato de não falha, $G_N = Ac(G_{N1})$. Note que $G_N = (X_N, \Sigma_N, \delta_N, \Gamma_N, x_{o,N})$ com $\Sigma_N = \Sigma \setminus \{\sigma_f\}$.

Passo 2: Calcule o *autômato de falha* do sistema G_Y, da seguinte seguinte:

- *Passo 2.1:* Defina $A_l = (X_l, \Sigma_f, \delta_l, x_{o,l})$, em que $X_l = \{N, Y\}$, $\Sigma_f = \{\sigma_f\}$, $\delta_l(N, \sigma_f) = \{Y\}$, $\delta_l(Y, \sigma_f) = \{Y\}$ e $x_{o,l} = \{N\}$
- *Passo 2.2:* Calcule $G_l = G \parallel A_l$ e marque todos os estados cuja segunda componente seja Y.
- *Passo 2.3:* Obtenha o autômato de falha $G_Y = CoAc(G_l)$.

Passo 3: Renomeie os eventos não observáveis do autômato G_N de acordo com a função $R : \Sigma_N \to \Sigma_R$ definida por

$$R = \{(\sigma, \sigma_R) : \ \sigma \in \Sigma_{uo}\}.$$

Passo 4: Calcule o autômato verificador $G_V = G_N \parallel G_Y = (X_V, \Sigma_R \cup \Sigma, f_V, \Gamma_V, x_{0,V})$. Marque todos os estados da forma $(x_N, (x_Y, Y))$.

Passo 5: Verifique a existência de uma componente fortemente conexa (CFC) C não trivial em G_V satisfazendo a seguinte condição:

$$\exists x_V, \ (x_V \in X_{m,V} \cap C) \wedge (\Gamma(x_V) \cap \Sigma \neq \emptyset).$$

Se a resposta é sim, então a linguagem $L(G)$ não é diagnosticável, caso contrário é diagnosticável.

2.5 Controle supervisório

Ramadge and Wonham (1987) iniciaram a teoria de controle supervisório para sistemas a eventos discretos (chamada de teoria-RW), buscando propor uma metodologia abrangente para os problemas de controle que aparecem nos SEDs. Na teoria-RW,

a planta sob controle é modelada por um autômato e seu comportamento é descrito pelas linguagens formais associadas. O alvo do controle é limitar a linguagem da planta até atingir determinadas especificações de funcionamento. Para atingir esse resultado, usa-se um sistema de malha fechada em que o papel do controlador é feito por um dispositivo denominado *supervisor*, simbolizado por S, cuja finalidade é ativar ou desativar eventos.

Suponha que um SED seja modelado pelo autômato $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ que gera a linguagem $L(G)$ e marca a linguagem $L_m(G)$. No contexto de controle supervisório, os estados marcados representam tarefas especiais que devem ser alcançadas pelo sistema. Caso $L(G) = \overline{L_m(G)}$ o sistema é dito *não bloqueante*. O bloqueio acontece quando se alcança um conjunto de estados não coacessíveis ou, em outras palavras, quando o sistema atinge um conjunto de estados a partir do qual nunca completará uma tarefa.

E razoável supor que o supervisor pode inibir só alguns eventos, que chamaremos de *eventos controláveis*. Aqueles eventos que nunca puderem ser desabilitados serão denominados *não controláveis*. Um evento é modelado como não controlável por várias razões : é imprevisível, prioritário ou existem limitações físicas para desabilitá-lo (Cassandras and Lafortune, 2008). Formalmente, particionamos o conjunto dos eventos em dois conjuntos

$$\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$$

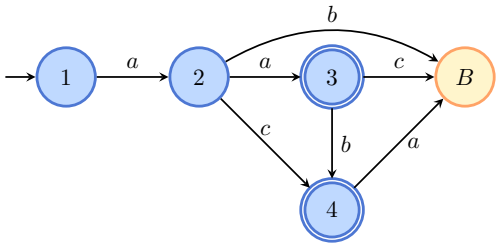
de eventos controláveis e não controláveis, respectivamente. Matematicamente um supervisor S é representado pela função

$$S : L(G) \to 2^{\Sigma}$$

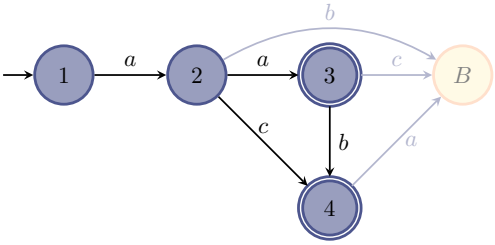
que especifica para cada traço $w \in L(G)$, o conjunto dos eventos que podem ser habilitados após da execução de w. Deve-se notar que os eventos não controláveis permanecerão sempre ativos uma vez que $S(w) \supseteq \Sigma_{uc}$. Uma vez que a sequência w tenha ocorrido, a próxima sequencia será $w\sigma$. Se o sistema G não tiver supervisão, o próximo evento σ está no conjunto $\Gamma(f^*(x_0, w))$; enquanto no mesmo sistema sob controle supervisório, σ fica restrito ao conjunto $\Gamma(f^*(x_0, w)) \cap S(w)$. Denotaremos o sistema em malha fechada como S/G (lê-se: "*S controlando G*").

A linguagem gerada pelo sistema supervisionado, denotada por $L(S/G)$, é definida recursivamente por:

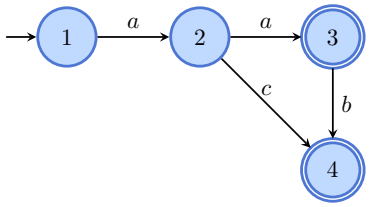
(LSG1) $\varepsilon \in L(S/G)$;
(LSG2) $w\sigma \in L(S/G) \Leftrightarrow (w \in L(S/G)) \wedge (w\sigma \in L(G)) \wedge (\sigma \in S(w))$;



(a) Sistema G sem supervisão



(b) Eventos desabilitados por S no sistema G



(c) Sistema S/G e linguagens $L(S/G)$ e $L_m(S/G)$

w	$\Gamma(f^*(x_0, w))$	$S(w)$
ε	$\Gamma(1) = \{a\}$	$\{a\}$
a	$\Gamma(2) = \{a, b, c\}$	$\{a, c\}$
aa	$\Gamma(3) = \{b, c\}$	$\{b\}$
ac, aab	-	$\{\}$

(d) Ação do Supervisor $S(w)$

Figura 2.6: Modelo G e seu comportamento sob controle supervisório

enquanto a linguagem marcada pelo sistema realimentado é definida como:

$$L_m(S/G) = L(S/G) \cap L_m(G).$$

Caso um sistema com controle supervisório não tenha bloqueio, então $\overline{L_m(S/G)} = L(S/G)$.

Exemplo 2.9. *Considere um sistema G cujo comportamento é descrito pelas linguagens $L = \overline{\{ab, aab, aac, aca\}}$ e $L_m = \{aa, ac, aab\}$ (veja a figura 2.6(a)) e suponha que todos os eventos sejam controláveis. O objetivo do controle supervisório é evitar o estado de bloqueio B. Quando o sistema G inicia, o supervisor S vai, progressivamente, restringindo os eventos viáveis (figura 2.6(b)) de acordo com o traço w que aconteceu. A figura 2.6(b) e a tabela 2.6(d) mostram como, por exemplo, quando a planta gera as sequências **a** e **ac**, são inibidos os eventos **b** e **a** respectivamente. A figura 2.6(c) representa o comportamento supervisado, dado pelas linguagens $L(S/G) = \overline{\{aab, ac\}}$ e $L_m(S/G) = \{aa, ac, aab\}$. O sistema controlado não tem mais bloqueio.* ❑

A finalidade do controle supervisório é sintetizar S usando métodos formais. Para isso, deve-se primeiro garantir a existência de uma solução. O papel fundamental na existência de soluções de problemas de controle supervisório é representado pela *controlabilidade*. As duas definições seguintes provêem a base do resultado fundamental da teoria-RW, dado pelo teorema da controlabilidade.

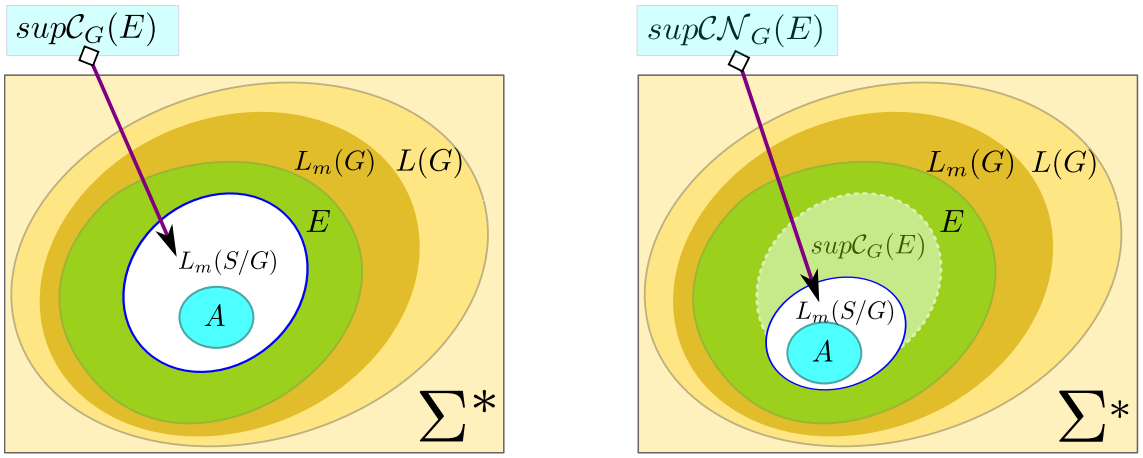


Figura 2.7: Diagrama de Venn das linguagens $sup\mathcal{C}_G(E)$ (a) e $sup\mathcal{CN}_G(E)$ (b).

Definição 2.4. *(Ramadge and Wonham, 1987)(Controlabilidade) Uma linguagem* $K \subseteq L(G)$ *é dita controlável em relação a* $L(G)$ *e* Σ_{uc} *se* $\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K}$. □

Definição 2.5. *(Ramadge and Wonham, 1987) A linguagem* $K \subseteq L_m(G)$ *é dita* $L_m(G)$*-fechada se* $K = \overline{K} \cap L_m(G)$. □

Teorema 2.1. *(Ramadge and Wonham, 1987) (Teorema da controlabilidade) Seja* $\emptyset \subset K \subseteq L_m(G)$ *uma linguagem não vazia. Então, existe um supervisor* S *não bloqueante se e somente se*

i. K *é controlável com relação a* $L(G)$.

ii. K *é* $L_m(G)$*-fechada.* □

É desejável obter um supervisor não bloqueante. Em alguns casos, obter um supervisor não bloqueante é impossível ou restritivo demais. Se a condição (ii) do teorema 2.1 for relaxada, obteremos um conjunto maior de linguagens factíveis, tolerando o possível bloqueio, como abordado por Chen and Lafortune (1991). O propósito central de um supervisor S é a coordenação de alto nível de sistemas compostos por muitos subsistemas com interações complexas. Esse objetivo é especificado formalmente por meio da seguinte inclusão (veja a figura 2.7(a)):

$$A \subseteq L_m(S/G) \subseteq E \tag{2.12}$$

A linguagem $E \subseteq L_m(G)$ é a *linguagem admissível*, na qual o projetista exclui todo comportamento considerado indesejável. A classe

$$\mathcal{C}_G(E) = \{K \subseteq E : K \text{ é controlável em relação a } \ L(G) \text{ e } \Sigma_{uc}\}$$

contém as possíveis linguagens controláveis que satisfazem a relação de inclusão direita na especificação (2.12). Essa classe é não vazia ($\emptyset \in \mathcal{C}_G(E)$) e fechada em relação a uniões arbitrárias. Em particular, $\mathcal{C}_G(E)$ possui um elemento supremo denotado $sup\mathcal{C}_G(E)$ e chamado *linguagem controlável suprema*. Encontrar a linguagem controlável suprema pode ser feito pelo método do ponto fixo (Wonham and Ramadge, 1987) ou pela fórmula fornecida por Kumar et al. (1991). Pelo teorema 2.1, um supervisor S que implementa o elemento supremo $sup\mathcal{C}_G(E)$ é *minimamente restritivo*. Caso E seja $L(G)$*-fechada*, obtém-se um sistema supervisionado não bloqueante. Caso a linguagem $sup\mathcal{C}_G(E)$ seja vazia, pode-se ultrapassar a especificação (2.12) usando a superlinguagem ínfima fechada e controlável (Lafortune and Chen, 1990) ou permitir algum tipo de bloqueio (Chen and Lafortune, 1991).

Considere os autômatos G e H, tais que $E = L_m(H)$, $\overline{E} = L(H)$ e $L(H) \subseteq L(G)$, em que $L_m(H)$, $L(H)$ e $L(G)$ representam as linguagens marcada e gerada por H, e a linguagem gerada por G, respectivamente. A linguagem controlável suprema $sup\mathcal{C}_G(E)$ pode ser calculada de acordo com o algoritmo 2.3 a seguir.

Algoritmo 2.3: Obtenção da linguagem controlável suprema $sup\mathcal{C}_G(E)$.

Entradas: Os autômatos G e H e o conjunto de eventos não-controláveis Σ_{uc}.

Saída: O autômato H_k tal que $L_m(H_k) = sup\mathcal{C}_G(E)$.

Passo 1: Construa o autômato $H_0 = H \times G = (Y_0, \Sigma, \delta_0, \Gamma_0, (y_0, x_0), Y_{0,m})$. Defina $k = 0$.

Passo 2: Apague os estados $(y, x) \in Y_k$ que violam a restrição no conjunto de eventos ativos $\Gamma_0[(y, x)] \supseteq \Gamma(x) \cap \Sigma_{uc}$, e suas transições.

Passo 3: Faça $k = k + 1$ e calcule

$$H_k = trim(H_{k-1}) = (Y_k, \Sigma, \delta_k, \Gamma_k, (y_0, x_0), Y_{k,m})$$

Passo 4: Repita os passos (2) e (3) até obter um ponto fixo ($H_k = H_{k-1}$) ou o autômato que representa a linguagem vazia.

Exemplo 2.10. *Consideremos novamente o SED G do exemplo 2.9. Suponhamos agora que o evento a seja não controlável. A nossa especificação está definida pela linguagem $E = \{aa, ac, aab\} \subseteq L_m(G)$ (Fig. 2.6(b)). Pode-se verificar que $E = \overline{E} \cap L(G)$ e, portanto, E é $L(G)$-fechada. A sequência $ac \in E$ continuada pelo evento $a \in \Sigma_{uc}$ produz a sequência $aca \in L(G) \setminus E$ que viola a condição de controlabilidade da definição 2.4. O mesmo acontece com a sequência $aab \in E$. Subtraindo essas sequências de E, obtemos a solução suprema $sup\mathcal{C}_G(E) = \{aa\}$. Dessa forma, o*

comportamento supervisor mais permissivo é $L_m(S/G) = \{aa\} \subset E$ *e* $L(S/G) = \overline{L_m(S/G)}$. ❑

2.5.1 Controle sob observação parcial

A extensão natural da teoria-RW considerando os eventos não-observáveis foi feita por Cieslak et al. (1988). Sob essa ótica, o conjunto dos eventos Σ é dividido em partições independentes

$$\Sigma \;=\; \Sigma_o \dot{\cup} \Sigma_{uo} \;=\; \Sigma_c \dot{\cup} \Sigma_{uc}$$

dependendo da observabilidade e controlabilidade dos eventos. A observabilidade é definida formalmente a seguir.

Definição 2.6. *(Cieslak et al., 1988)(Observabilidade) Uma linguagem* K *é dita observável em relação a* $L(G)$, Σ_o *e* Σ_c *se para todo* $w \in \overline{K}$ *e todo* $\sigma \in \Sigma_c$

$$(w\sigma \notin \overline{K}) \;\wedge\; (w\sigma \in L(G)) \Rightarrow P_{\Sigma_o}^{-1}[P_{\Sigma_o}(w)]\sigma \cap \overline{K} = \emptyset \qquad \square$$

Nesse ponto temos dois efeitos limitando o número de linguagens atingíveis sob controle supervisório: controlabilidade e observabilidade. Essas limitações refletem-se formalmente no próximo resultado.

Teorema 2.2. *(Cieslak et al., 1988) (Teorema da controlabilidade e observabilidade) Seja* $K \subseteq L_m(G)$ *uma linguagem não vazia. Existe um supervisor* S *não bloqueante se e somente se*

i. K *é controlável com relação a* $L(G)$ *e* Σ_{uc}.

ii. K *é observável com relação a* $L(G)$, Σ_o *e* Σ_c.

iii. K *é* $L_m(G)$*-fechada.* $\square$

A propriedade de observabilidade não é preservada sob uniões arbitrárias. Por essa razão não existe, em geral, um supervisor minimamente restritivo que satisfaça o teorema 2.2. Um problema mais bem comportado pode ser formulado substituindo-se a propriedade de observabilidade pela propriedade mais forte de *normalidade*, a qual se conserva sob operações de união (Cho and Marcus, 1989; Cieslak et al., 1988).

Definição 2.7. *(Normalidade) Considere a linguagem* $M = \overline{M} \in \Sigma^*$ *e a projeção natural* $P_{\Sigma_o} : \Sigma \to \Sigma_o$. *Então, a linguagem* $K \subseteq M$ *é dita normal em relação a* M *e* P_{Σ_o} *se*

$$\overline{K} = P^{-1}[P[K]] \cap M.$$

$\square$

Assim, pode-se obter uma solução ótima dada pela *linguagem suprema normal controlável*, denotada como $sup\mathcal{CN}_G(E)$. Os diagramas de Venn da figura 2.7(a,b) ilustram comparativamente como a solução $sup\mathcal{CN}_G(E)$ é ainda mais restritiva que solução $sup\mathcal{C}_G(E)$ o que é uma consequência inevitável da perda da informação de alguns sensores.

Além dos sistemas com observação parcial, várias extensões da teoria-RW foram e estão sendo desenvolvidas para lidar com problemas particulares no controle de SEDs. Dentre essas destacam-se: modelos temporizados, modelos estocásticos, controle robusto, controle tolerante a falhas e controle ótimo.

2.6 Comentários finais

O objetivo primordial deste capitulo foi apresentar os conceitos e problemas fundamentais em SEDs modelados por autômatos. As bases de linguagens formais e autômatos foram apresentados para introduzir a notação e alguns dos conceitos a serem usados nos capítulos posteriores. Além disso, foram apresentados os resultados fundamentais no estudo da diagnose de falhas e controle supervisório de SEDs que serão também utilizados no próximo capítulo. Um estudo contemporâneo do tema de autômatos é realizado em Lawson (2004). O tema de SEDs é amplamente desenvolvido em Cassandras and Lafortune (2008).

Capítulo 3

O programa DESLAB

O objetivo deste capítulo é apresentar o DESLAB, um programa de computação científica, escrito na linguagem PYTHON para o desenvolvimento de algoritmos para análise e síntese de sistemas a eventos discretos (SEDs) descritos por autômatos. O desenvolvimento do DESLAB surge a partir da revisão das ferramentas de computação existentes para SEDs (veja Clavijo et al. (2012) e referências), com o intuito de facilitar o estudo de algoritmos para SEDs descritos por autômatos usando métodos experimentais. Como será visto mais adiante, o DESLAB permite definir variáveis simbólicas de tipo autômato e incorpora instruções concisas para manipular, operar, analisar e visualizar essas variáveis, com uma sintaxe e nível de abstração próximos da notação matemática usual utilizada na teoria de SEDs.

Este capítulo está dividido da seguinte forma. Na seção 3.1 são revisados brevemente alguns programas importantes no projeto do DESLAB. A seção 3.2 trata de alguns conceitos importantes de computação científica que motivaram o desenvolvimento do DESLAB. A seção 3.3 apresenta as descrições da arquitetura funcional do DESLAB e das classes que compõem um objeto autômato. Na seção 3.4 é introduzido o programa na perspectiva do usuário: como são definidos os autômatos, as operações fundamentais incluídas no programa e a definição de novas funções. Na seção 3.5 são apresentados exemplos mais avançados do uso do programa no desenvolvimento de *toolboxes* para controle supervisório e diagnose de falhas. Na seção 3.6 são apresentados os comentários finais.

3.1 Ferramentas computacionais

A seguir revisamos algumas ferramentas computacionais que são importantes no projeto do DESLAB.

3.1.1 Graphviz

O Graphviz (AT&T Labs Research, 2012) é um programa de código fonte aberto para visualização de grafos . O programa usa uma linguagem especial de descrição de grafos chamada dot. O graphviz contém algoritmos muito eficientes para o desenho automático de grafos que contenham várias centenas de nós. Os algoritmos do graphviz têm sido "sintonizados" por vários anos para produzir automaticamente desenhos estéticos e legíveis. Em Wood (2005) foram testadas várias opções para o desenho automático de grafos e, na comparação com outros programas para visualização de grafos, o graphviz foi sempre superior. Por essa última razão, o graphviz é usado nesse trabalho para estabelecer, de forma preliminar, as localizações dos vértices no gráfico de um autômato.

3.1.2 NetworkX

O networkx (Hagberg et al., 2008, 2012) é um pacote escrito na linguagem python para a exploração e análise de grafos e algoritmos combinatórios. O programa fornece as estruturas de dados básicas para a representação de grafos simples, dígrafos e multígrafos com autolaços e pesos nas arestas. Consequentemente, sobre os grafos definidos pelo usuário, permite executar operações elementares de grafos como acrescentar ou remover vértices, compor grafos, entre outras. Também fornece funções de maior complexidade como: cálculo de métricas e estatísticas de grafos, conectividade, atratores, isomorfismo e análise algébrica de grafos. O networkx aproveita as bibliotecas especializadas de python, incorporando uma grande capacidade gráfica, matemática e de conversão de arquivos. A estrutura de grafo dos autômatos neste trabalho está construída sobre o networkx, fornecendo assim toda a capacidade de cômputo e análise de grafos, necessária para trabalhar com SEDs sobre um ambiente unificado.

3.1.3 Python

python (Python Software Foundation, 2012) é uma linguagem de código aberto criada em 1990 por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI). É uma linguagem de alto nível orientada a objetos, de tipagem dinâmica e forte, interpretada e interativa. Entre as estruturas numéricas, o python inclui números inteiros, de ponto flutuante, complexos e racionais de comprimento arbitrário. Dentre os tipos de dados inclui *strings*, listas, ênuplas, conjuntos, multiconjuntos e um tipo de arranjos associativos especiais denominados dicionários. Esses tipos de dados permitem escrever algoritmos complexos em python de uma forma clara, expressiva e eficiente que

parece pseudocódigo executável. Sua utilidade como linguagem de desenvolvimento no âmbito cientifico sofreu uma considerável aceleração na década passada (devido ao novo modelo código aberto adotado pela comunidade científica), sendo desenvolvidas bibliotecas em PYTHON para aplicações diversas que vão desde o simples cálculo matricial até pacotes sofisticados de modelagem e simulação de sistemas dinâmicos, processamento de sinais, neurociência (Langtangen, 2009; Oliphant, 2007; Perez et al., 2011) etc. A estrutura orientada a objetos da linguagem permite construir códigos complexos que refletem estreitamente a entidade física ou matemática modelada por eles.

3.2 Aspectos da computação científica

A *computação científica* pode ser entendida num sentido amplo como o uso de recursos computacionais para resolver problemas em áreas específicas de pesquisa, e tem crescentemente sido referida como uma "terceira abordagem", depois da teoria e a experimentação, para entender questões científicas fundamentais (Prabhu et al., 2011).

Tradicionalmente, as ferramentas de cálculo científico foram escritas em linguagens compiladas (primeiro exclusivamente em FORTRAN e depois em C e outras linguagens). Essas linguagens são eficientes no uso dos recursos computacionais, embora exijam um considerável esforço do desenvolvedor pelo inerente ciclo de compilação/depuração e de sua expressividade[1]. Os programas gerados são pouco interativos (precisa-se de trabalho adicional para fornecer interatividade) e torna-se difícil acrescentar novos módulos decorrente das necessidades da pesquisa desenvolvida. Para resolver esses problemas, apareceram nos anos 80 programas de alto nível como MATLAB, que forneceram ao cientista uma interface amigável, bibliotecas integradas extensivas, exploração interativa e visualização imediata de dados. Logo depois, nos anos 90, apareceram programas como MATHEMATICA e MAPLE que incorporaram capacidade simbólica. Tais programas permitiram aos pesquisadores efetuar cálculos e desenvolver ferramentas sofisticadas em seus domínios de pesquisa com um esforço de programação razoável, comparado com a carga imposta pela expressividade, tempos de compilação e curva de aprendizagem das linguagens compiladas. Com esses programas, problemas de cálculo científico que nos anos 70 eram considerados formidáveis, se tornaram cálculos elementares. Esses programas, ainda importantes, têm uma característica geral: estão baseados em linguagens de *scripting*, isto é, linguagens de muito alto nível que não precisam ser compiladas e cuja sintaxe é bem próxima às dos objetos estudados no domínio científico para o

[1] A palavra *expressividade*, nesse contexto, significa simplesmente quantas linhas de código é necessário escrever para formular um modelo matemático abstrato ou um algoritmo complexo.

qual foram desenvolvidas (alta expressividade). Além de realizar funções de cálculo numérico e lógico, a abrangência da *computação científica* moderna tem incorporado os seguintes requisitos:

1. Capacidade de integração de componentes escritos em diferentes linguagens e, também, de bibliotecas e ferramentas provenientes de várias disciplinas;
2. Visualização científica interativa, isto é, representação gráfica de dados usada como meio para ganhar entendimento do objeto estudado;
3. Processar, trocar e converter dados desde vários tipos de formatos de informação, dependendo do contexto, num entorno unificado. Além disso, ter a capacidade de acrescentar rapidamente novos módulos de tradução entre formatos;
4. Capacidade de armazenamento organizado de resultados de experimentos e simulações em bases de dados;
5. Controle e interação com diferentes dispositivos de *hardware* e sistemas de tempo real;
6. Exploração adequada das novas possibilidades da computação paralela;
7. Fornecer arquitetura de trabalho colaborativo entre pesquisadores, que podem estar distribuídos geograficamente.

As ferramentas baseadas em linguagens de *scripting* são altamente eficazes para construir *software* científico flexível que satisfaça os requisitos acima. Apesar de ser reconhecida a importância das ferramentas interativas baseadas em linguagens de *scripting* no âmbito científico, uma ferramenta com essas características ainda não foi desenvolvida para a análise, síntese e modelagem de SEDs modelados por autômatos.

3.3 O programa DESLAB

O DESLAB é um programa de computação científica, escrito na linguagem PYTHON, para o desenvolvimento de algoritmos para análise e síntese de SEDs descritos por autômatos. O objetivo central do DESLAB é fornecer uma ferramenta para facilitar o desenvolvimento de novos algoritmos propostos em SEDs, usando uma sintaxe simples e próxima da teoria matemática de SEDs. O programa segue a filosofia de ferramentas bem sucedidas que apareceram em outras áreas científicas. A proposta do DESLAB é fornecer uma ferramenta unificada que integra autômatos, algoritmos de grafos e cálculo numérico. O objeto computacional base sobre o qual se estrutura o DESLAB é o autômato finito. Sobre esse objeto agem um conjunto de instruções concisas para manipular, operar, analisar e visualizar SEDs. Com o conjunto de

instruções propostas e as estruturas básicas de controle da linguagem PYTHON, o DESLAB pode ser facilmente estendido, de maneira que novas bibliotecas e funções podem ser criadas à medida que apareçam novos resultados e necessidades dos usuários.

No projeto inicial do DESLAB, os quesitos (1) a (4) da seção 3.2 foram considerados: (*i*) necessidade de interação com programas de linguagens formais; (*ii*) integração de bibliotecas matemáticas e de análise de grafos como NUMPY e NETWORKX; (*iii*) visualização de autômatos usando LATEX, GRAPHVIZ e outros programas de visualização científica de redes complexas; (*iv*) capacidade de importação e exportação (em diversos formatos) para gráficos, grafos e autômatos.

A escolha do PYTHON para o desenvolvimento do DESLAB foi feita com base nas seguintes características que essa linguagem possui: (*i*) tipos abstratos de dados; ser livre e poder ser executada em diversos sistemas operacionais; (*ii*) potência para o desenvolvimento de aplicações científicas; (*iii*) contínua tendência de se tornar, de fato, a linguagem de *scripting* da computação científica (Perez et al., 2011).

Sem dúvida, as ferramentas de código aberto têm acrescentado possibilidades ilimitadas à comunidade científica: é quase inconcebível a composição de textos científicos complexos sem a ajuda de LATEX, a linguagem de edição científica que evoluiu graças à sua arquitetura e a seu código aberto. A tendência do PYTHON é igual e, com alta probabilidade de alcançar em poucos anos no campo da computação científica um impacto similar ao LATEX. O DESLAB é mais uma ferramenta de computação científica, dentro de uma gama extensa de aplicações, que conseguiram se desenvolver graças ao modelo de código aberto do PYTHON.

3.3.1 Estrutura do DESLAB

De uma maneira geral, as ferramentas especializadas de computação científica giram em torno de objetos matemáticos fundamentais dependentes do contexto de estudo. O pacote MAPLE, por exemplo, tem como objeto fundamental a expressão simbólica com as operações fundamentais definidas por regras algébricas. Para cada área, um programa de computação científica define os objetos, as relações, operações e funções que esses objetos determinam. Do ponto de vista da programação orientada a objetos, a implementação de um objeto matemático é uma *classe* e as operações que ele define são os *métodos* da classe. Os componentes da classe são as *propriedades* da classe.

O objeto computacional fundamental do DESLAB é o autômato finito (determinístico ou não determinístico) definido pela sêxtupla

$$G = (X, \Sigma, f, \Gamma, X_0, X_m), \tag{3.1}$$

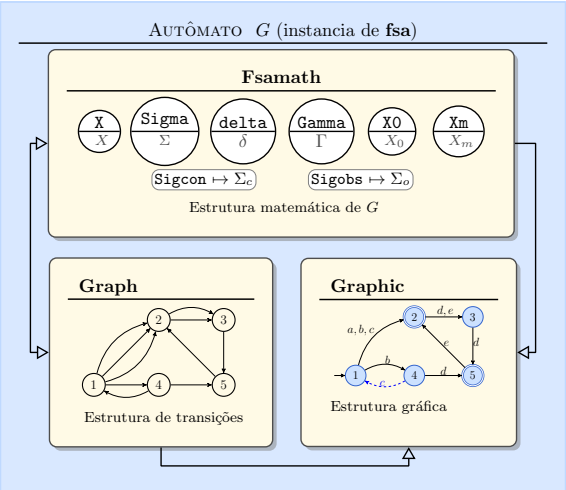


Figura 3.1: Autômato: o objeto fundamental no DESLAB, e suas classes.

O autômato no DESLAB (figura 3.1) é implementado pela classe **fsa**, a qual é uma entidade complexa formada por várias classes. As principais classes são explicadas a seguir.

Classe Fsamath. Essa classe implementa um autômato não determinístico (do qual o autômato determinístico é um caso particular) possuindo as propriedades `X`, `Sigma`, `delta`, `Gamma`, `X0` and `Xm` que correspondem a cada uma das componentes da sêxtupla da expressão (1), conforme mostrado na classe Fsamath da figura 3.1. As propriedades `X`, `Sigma`, `X0`, `Xm`. são definidas como conjuntos dentro do DESLAB e as propriedades `delta` e `Gamma` são funções implementadas utilizando estruturas de dados eficientes chamadas dicionários. Conforme será visto mais adiante, essas funções podem ser invocadas de forma similar à notação matemática. Essa classe também possui outras propriedades matemáticas tais como `Sigcon` e `Sigobs` que representam o conjunto de eventos controláveis e o conjunto de eventos observáveis, respectivamente.

Classe Graph. Alguns algoritmos importantes usados na análise de SEDs — por exemplo, verificar a diagnosticabilidade — são eficientemente implementados como algoritmos de grafos. No DESLAB é usada a perspectiva dual de autômato e grafo. Por essa razão, a estrutura de transições de um autômato no DESLAB está definida como uma classe de tipo *MultiDiGraph* do pacote NETWORKX. Sobre essa classe é possível realizar todo tipo de operações de grafos. Por exemplo, pode encontrar-se as componentes fortemente conexas, os autovalores da matriz de incidência, entre muitas operações.

Classe Graphic. Dependendo do contexto de pesquisa, a representação gráfica dos autômatos varia; por exemplo, em diagnose é usual desenhar autômatos com estados retangulares. Um outro ponto importante se refere à representação de transições nos autômatos. Para melhorar a visualização, quando houver várias transições

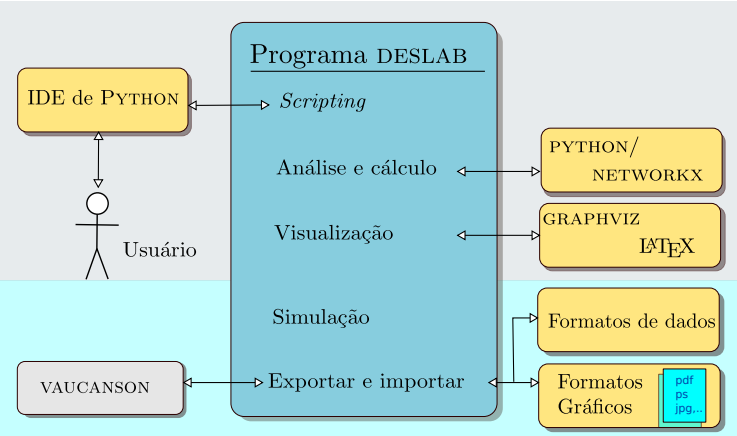


Figura 3.2: Arquitetura do DESLAB.

entre dois estados, essas serão substituídas por uma única transição rotulada por vários eventos; no exemplo apresentado na figura 3.1 a transição $(2 \xrightarrow{d,e} 3)$ do autômato representado na classe *Graphic*, substitui as transições $(2 \xrightarrow{d} 3)$ e $(2 \xrightarrow{e} 3)$. De forma similar, as transições rotuladas por eventos não observáveis, não controláveis ou associados a perdas intermitentes de observação, são representadas com setas especiais. Assim, por diversas razões é necessário que exista dentro da estrutura do autômato uma componente que especifique, de maneira flexível, a parte gráfica e que, inclusive, possa evoluir de acordo com os requisitos dos pesquisadores.

3.3.2 Arquitetura do DESLAB.

A arquitetura funcional do DESLAB (figura 3.2) fornece, para o caso especial de SEDs descritos por autômatos, capacidades similares a outros entornos de computação científica tais como: *programação, análise e cálculo, visualização, simulação e importação e exportação de arquivos.*

A nível de *programação*, o usuário interage usando comandos em um *shell* interativo (como em MATLAB ou MAPLE) ou por meio de *scripts* em um entorno de PYTHON. Uma vez carregado o módulo DESLAB em PYTHON (o que é feito escrevendo a linha: `from deslab import*`), o usuário pode, imediatamente, definir autômatos que agem como variáveis simples.

O *cálculo e a análise* dos autômatos são feitos utilizando algoritmos programados diretamente em PYTHON e os cálculos que envolvem grafos são feitos usando o pacote NETWORKX.

Um dos alvos principais na fase de projeto do DESLAB foi fornecer uma de alta qualidade que fosse capaz de interpretar rótulos em código LATEX fornecidos pelo usuário. Para consegui-lo, a *visualização* de autômatos no programa é realizada da seguinte forma: quando o método gráfico `draw` é invocado, um outro método da classe *Graphic* se encarrega de gerar a descrição gráfica do autômato na linguagem

DOT para, assim, calcular o "layout" do autômato mediante o programa GRAPHVIZ. Com o resultado desse cálculo, é gerado o código LATEX necessário para renderizar notação matemática real e também uma saída vetorizada de alta qualidade gráfica em formato PDF. Alguns dos tipos de saída que podem ser especificados são: apresentação em beamer, gráfica simples em preto e branco e tabela.

Os autômatos podem ser exportados em vários formatos gráficos (por exemplo, *pdf, eps, png*), e formatos de texto plano. As facilidades da linguagem PYTHON no processamento de arquivos, permitem acrescentar rapidamente novos formatos de conversão, caso seja necessário.

3.4 Trabalho com DESLAB

O DESLAB funciona por meio de comandos em um *shell* interativo ou por meio de *scripts* de programação. Os códigos apresentados nesta seção ilustram a funcionalidade do DESLAB e, para facilitar seu acompanhamento, serão usadas as seguintes convenções: (i) um *script* é representado como uma lista numerada e com fundo de cor ■ (ii) os comandos executados no *shell* interativo apresentam-se sem numeração e com fundo de cor ■. Nesse último caso, um comando de entrada do usuário segue ao símbolo `>>>`, enquanto o resultado desse comando é mostrado sem nenhum símbolo.

3.4.1 Definição do autômato

O exemplo a seguir ilustra a forma de definir um tipo de variável associada com um autômato no DESLAB.

Exemplo 3.1. *Considere um autômato* $G_1 = (X, \Sigma, \delta, \Gamma, X_0, X_m)$ *em que* $X = \{q_1, q_2, q_3\}$, $\Sigma = \{\alpha_1, \beta_1, e\}$, $\delta(q_1, \beta_1) = q_2$, $\delta(q_2, \beta_1) = \delta(q_2, e) = q_3$, $\delta(q_3, \alpha_1) = q_1$, $x_0 = \{q_0\}$ *e* $X_m = \{q_3\}$. *O tipo de variável autômato é definido usando as instruções mostradas na Listagem 3.1.*

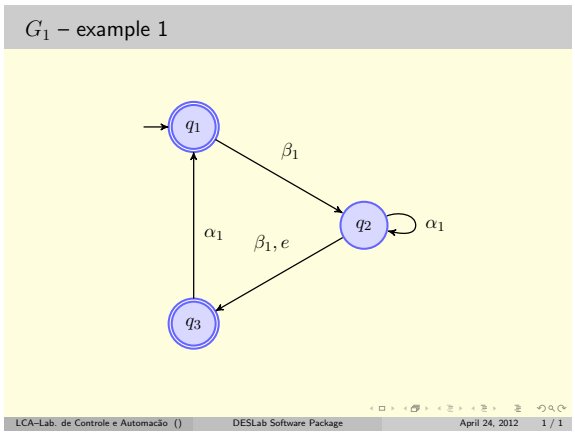


Figura 3.3: Diagrama de transição de estado de G_1, usando o método `draw` em modo apresentação.

Listagem 3.1: Definição de um autômaro em DESLAB.

```
1
2
3 from deslab import*
4 syms('q1 q2 q3 a1 b1 e') # defining symbolic variables to be used
5 table = [(a1,'\alpha_1'),(b1,'\beta_1'),(q1,'q_1'),(q2,'q_2'),(q3,'q_3')]
6 # automaton definition
7 X = [q1,q2,q3]
8 Sigma = [a1,b1,e]
9 X0 = [q1]
10 Xm = [q1,q3]
11 T =[(q1,b1,q2),(q2,b1,q3),(q2,e,q3),(q3,a1,q1),(q2,a1,q2)]
12 G1 = fsa(X,Sigma,T,X0,Xm,table,name='$G_1$ -- example 1')
13 draw(G1,'beamer')
```

Na linha 2 da Listagem 3.1 definem-se os *rótulos de programação* e na 3 linha define-se uma tabela que associa cada *rótulo de programação* com a correspondente componente gráfica. Note que o evento α_1 na variável `table` está associado com um par de rótulos da forma $\alpha_1 \mapsto$ (`a1,\alpha_1`). A primeira componente é o *rótulo de programação* e contém uma *string* com um nome de variável válido que será usado na interface textual interativa e na representação interna da estrutura de dados. A segunda componente especifica, por meio de código LATEX, o símbolo que será renderizado na saída gráfica. Caso não seja definida uma tabela de associação, os rótulos de programação serão usados na interface textual e gráfica.

Os conjuntos `X,S,X0 e Xm` são definidos por meio de *listas*. As transições entre estados são especificadas por uma lista das triplas que representam cada transição. Essa notação procura ser econômica para o usuário e não corresponde à representação interna do autômato no DESLAB. A instrução `fsa` constrói um objeto do tipo autômato, denotado pela variável `G1`, cuja representação interna é próxima da formulada na expressão (3.1). O método `draw`, usado na última linha, gera transparências de Beamer semelhantes àquela mostrada na figura 3.3.

As propriedades matemáticas do autômato G_1 são acessíveis usando a notação na Tabela 3.1.

Tabela 3.1: Sintaxe para o acesso das propriedades matemáticas do objeto *fsa*.

Símbolo	Sintaxe	Tipo de objeto
G_1	`G1`	fsa
X	`G1.X`	conjunto
Σ	`G1.Sigma`	conjunto
$\delta(q_1, \alpha_1)$	`G1.delta(q1,a1)`	função
$\Gamma(q_1)$	`G1.Gamma(q1)`	função
X_0	`G1.X0`	conjunto
X_m	`G1.Xm`	conjunto

Por exemplo, a obtenção dos conjuntos X e Σ para o autômato do exemplo 3.1 é feita digitando-se os seguintes comandos:

```
>>> G1.X
set(['q3', 'q2', 'q1'])
>>> G1.Sigma
set(['b1', 'e', 'a1'])
```

De forma similar, podemos calcular as funções $\Gamma(q_1)$, $\delta(q_2, \beta_1)$ com as seguintes instruções:

```
>>> G1.Gamma(q2)
set(['a1', 'e', 'b1'])
>>> G1.delta(q1,b1)
q2
```

É possível ainda realizar operações de conjuntos a partir das componentes matemáticas do autômato, usando a sintaxe de PYTHON. Por exemplo, $C = \Gamma(q_1) \cap \Gamma(q_2)$ é escrito:

```
>>> C = G1.Gamma(q1) & G1.Gamma(q2)
set(['b1'])
```

3.4.2 Operações no DESLAB

Definida a variável de tipo autômato, o passo seguinte é manipular essas variáveis por meio de operações definidas para esses elementos. A tabela 3.2 mostra as operações com autômatos disponíveis e a sintaxe usada para executá-las. Essas operações disponíveis são de tipo matemático (tabela 2(a)), de manipulação da estrutura do autômato (tabela 2(b)), de comparação de linguagens e autômatos (tabela 2(c)), e de entrada-saída (tabela 2(d)). O exemplo a seguir mostra que é possível definir novos autômatos como resultado de operações entre autômatos previamente definidos.

Exemplo 3.2. *Considere um autômato* $G_2 = (X, \Sigma, \delta_2, \Gamma_2, X_0, X_{m,2})$ *em que* X, Σ *e* X_0, *são definidas como no exemplo 1 e* $X_{m,2} = \{q_2, q_3\}$, $\delta_2(q_1, \beta_1) = q_2$, $\delta_2(q_2, e) =$

Tabela 3.2: Instruções fundamentais do programa DESLAB

(a) Operações matemáticas sobre objetos de tipo autômato

Definição	Notação em SEDs	Sintaxe
Composição paralela	$G_1 \parallel G_2$	`G1//G2`
Produto	$G_1 \times G_2$	`G1&G2`
União	$L_m(G_1) \cup L_m(G_2)$	`G1+G2`
Concatenação	$L_m(G_1)L_m(G_2)$	`G1*G2`
Trim	$Trim(G_1)$	`trim(G1)`
Parte Acessível	$Ac(G_1)$	`ac(G1)`
Parte CoAcessível	$CoAc(G_1)$	`coac(G1)`
Fecho em prefixo	$\overline{L_m(G_1)}$	`pclosure(G1)`
Projeção inversa	$P^{-}1\{L(G_1)\}$	`invproj(G1)`
Dif. de linguagens	$L_m(G_1) \setminus L_m(G_2)$	`G1-G2`
Complementar	$L_m(G_1)^C$	`G1'`
Observador	$Obs(G_1)$	`obs(G1)`
Projeção	$P(L(G))$	`proj(G1)`
Fecho de Kleene	$[L_m(G_1)]^*, \Sigma^*$	`kleeneclos(arg)`
Completude a Σ^*	$L(G_1) = \Sigma^*$	`complete(G1)`
Autômato com o número de estados mínimo	$L_m(G_{min}) = L_m(G_1)$	`statemin(G1)`

(b) Funções de manipulação da estrutura de autômato.

Ação	Sintaxe
Acrescentar $x \xrightarrow{a} y$ em G_1	`G1.addtransition(x,a,y)`
Apagar $x \xrightarrow{a} y$ em G_1	`G1.deltransition(x,a,y)`
Acrescentar estado x em G_1	`G1.addstate(x)`
Apagar estado x em G_1	`G1.delstate(x)`
Acrescentar evento e em G_1	`G1.addevent(e)`
Apagar evento a em G_1	`G1.delevent(a)`
Renomear eventos de $E' \subseteq E$ em G_1	`G1.renevents(E,mapping)`
Renomear estados de $X' \subseteq X$ em G_1	`G1.renstates(E,mapping)`
Redefinir $X_m, \Sigma_{con}, \Sigma_{obs}$ e propriedades gráficas	`G1.set(property=value)`

(c) Funções de comparação de linguagens e autômatos.

Definição	Notação em SEDs	Sintaxe
Inclusão	$L_m(G_1) \subseteq L_m(G_2)$?	`G1<=G2,G2>=G1`
Ling. vazia	$L_m(G_1) = \emptyset$?	`islangempty(G1)`
Equivalência de aut.	$G_1 = G_2$?	`G1==G2`
Equivalência de lings.	$L_m(G_1) = L_m(G_2)$?	`arelangequiv(G1,G2)`
Teste de completude	$L(G_1) = \Sigma^*$?	`islangcomplete(G1)`
Isomorfismo entre aut.	–	`areisomorph(G1,G2)`

(d) Funções de entrada saída

Ação	Sintaxe
Diagrama de transição do autômato G_1	`draw(G1,mode)`
Salvar autômato G_1	`save(G1,format)`
Carregar autômato G_1	`load(G1,format)`

q_3, $\delta_2(q_3, a_1) = q_3$. *Usando o* DESLAB *o autômato* $H = (G_1 \parallel G_2) \parallel G_3$ *é obtido digitando-se as seguintes instruções:*

```
>>> Xm2 = [q2,q3]
>>> T2 = [(q1,b1,q2),(q2,e,q3),(q3,a1,q3)]
>>> G2 = fsa(X,Sigma,T2,X0,Xm2,table)
>>> H = G1//G2
>>> draw (H, 'figure')
```

A última linha chama o método **draw** para gerar a figura 3.4 que representa o diagrama de transição de estados do autômato $H = G_1 \parallel G_2$.

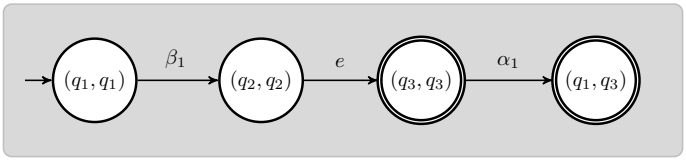


Figura 3.4: Diagrama de transição de estados de $H = G_1 \parallel G_2$ usando o método **draw** com saída de tipo figura.

3.4.3 Programação de funções usando o Python

A estrutura do DESLAB permite desenvolver rapidamente novas funções usando os seguintes recursos da linguagem PYTHON

- Tipos de dados como listas, ênuplas, conjuntos, multiconjuntos e rotinas para manipular *strings*.
- Operadores lógico-matemáticos e estruturas de controle de fluxo básicas de todas as linguagens de programação, tais como: `while, if-elif-else, for`.
- A sintaxe simples, parecida com pseudocódigo, em que os blocos de códigos são definidos pelo recuo de linha. Um aspecto bem importante da sintaxe do PYTHON, que facilita escrever algoritmos da teoria de SEDs, é a capacidade para definir dados e ciclos de controle por meio de propriedades descritivas.

No exemplo a seguir, combinaremos os recursos do PYTHON com as instruções fornecidas pelo DESLAB, para escrever uma nova função.

Exemplo 3.3. *Considere o autômato determinístico* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$. *A linguagem gerada por* G, *denotada por* $L(G)$, *é viva se* $\Gamma(x) \neq \emptyset$, *para todo* $x \in X$. *O código de* PYTHON *que verifica se uma linguagem gerada por um autômato é ou não viva é apresentado na Listagem 3.2*

Listagem 3.2: Função em DESLAB para verificar se a linguagem $L(G)$ é viva.

```
1 def islive(G):
2     live = all(G.Gamma(x) <> set([]) for x in G.X)   # ∀x ∈ X, Γ(x_i) ≠ ∅,
3     return live
```

Note que função `islive(G)` na Listagem 3.2 é definida por meio da instrução `def` seguida pelo nome da função (`islive`) e o argumento de entrada `(G)`. Note que o recuo de linha define os blocos de código (veja as linhas 2 e 3). A função finaliza com a instrução `return` que especifica que a variável booleana `live` (que contém o resultado do teste) é a variável de saída. Observe que a expressão matemática "$\forall x \in X, \Gamma(x) \neq \emptyset$" é codificada em PYTHON (linha 3) em uma forma muito semelhante à notação matemática.

Para determinar se as linguagens geradas pelos autômatos G_1 e H dos exemplos 3.1 e 3.2 são vivas, podemos invocar a nova função `islive` da forma seguinte:

```
>>> islive(G1)
True
>>> islive(H)
False
```

3.4.4 Algoritmos de grafos usando o networkx

Uma outra capacidade do DESLAB é a exploração e a análise de grafos e a disponibilidade de algoritmos combinatórios. Essa capacidade é fornecida pelo pacote NETWORKX (Hagberg et al., 2012). Uma vez que a estrutura de transições de um

Operação em g	Sintaxe
Acessar a estrutura de grafo g do autômato G	`g = G.Graph`
Componentes fortemente conexos	`strconncomps(G)`
Estados com auto laços	`selfloopnodes(G)`
Grafo condensado	`condensation(G)`
Ciclos simples	`simplecycles(G)`
Busca em profundidade	`dfs(G,source)`
Matriz de adjacência	`adjmatrix(G)`

Tabela 3.3: Alguns dos algoritmos de grafos fornecidos através de NETWORKX.

objeto autômato no DESLAB está construída sobre o NETWORKX, toda a potência de cálculo sobre grafos desse pacote está disponível para o usuário. A tabela 3.3 apresenta somente algumas das funções do NETWORKX que foram acrescentadas ao DESLAB. Com os novos recursos fornecidos pelo NETWORKX , é possivel obter os ciclos do autômato denotado pela variável `G1`, do exemplo 3.1. Para isso, usamos a estrutura de grafo de `G1`, por meio do seguinte comando:

```
>>> cycles = simplecycles(G1)
[['q3', 'q1', 'q2', 'q3'], ['q2', 'q2']]
```

A linha de saída apresenta os dois ciclos do autômato G_1, conforme esperado.

3.5 Criação de "toolboxes"

Nessa seção apresentamos a possibilidade de desenvolver, com relativa facilidade e curva de aprendizagem rápida, novas *toolboxes* para SEDs utilizando o DESLAB. Os exemplos a seguir ilustram o potencial desse pacote no desenvolvimento de novos algoritmos para SEDs e, por conseguinte, no desenvolvimento de novos *toolboxes*. A seguir apresentamos dois exemplos que ilustram a implementação de funções com base em algoritmos existentes.

Exemplo 3.4. *Considere os autômatos G e H, tais que $K = L_m(H)$, $\overline{K} = L(H)$ e $L(H) \subseteq L(G)$, em que $L_m(H)$, $L(H)$ e $L(G)$ representam as linguagens marcada e gerada por H, e a linguagem gerada por G, respectivamente. O objetivo desse exemplo é mostrar como o algoritmo 2.3 para o cálculo da linguagem controlável suprema $sup\mathcal{C}_G(E)$ pode ser implementado no* DESLAB.

A implementação do algoritmo 2.3 é a função `supcontrollable(H,G)` *mostrada na listagem 3.3. O passo 2 do algoritmo 2.3 é executado no ciclo* `for` *(linha 9). Os passos 3 e 4 são executados no ciclo* `while` *(linha 6). Finalmente, ao sair do ciclo* `while`*, a linguagem controlável suprema $sup\mathcal{C}_G(E)$ foi encontrada.*

Listagem 3.3: Implementação em DESLAB do algoritmo 2.3.

```
1  def supcontrollable(H,G,Sigma_uc):
2      # Inputs are automata G and H and Σ_uc
3      Hk = H & G                                  # STEP 1.  H_0 = H × G
4      Hk_changed = True                                  # flag for detecting fixed point
5      while (H0_changed & (H0 <> fsa())):         # STEP 4.
6          Hk_changed = False
7          for (y,x) in H0.X:
8              G_events_uc = G.Gamma(x) & Sigma_uc
9              if not(G_events_uc <= Hk.Gamma((y,x))):
10                 Hk = Hk.deletestate((y,x))      # STEP 2.
11                 Hk_changed = True                      # this is not a fixed point
12         Hk = trim(Hk)                           # STEP 4.  Calculating H_k = trim(H_{k-1})
13     return Hk
```

Exemplo 3.5. *O problema de diagnosticar uma falha num SED consiste em inferir a ocorrência de um evento não observável de falha, baseado nas ocorrências dos eventos observáveis desse sistema. Nesse contexto, a primeira pergunta é se para um determinado SED é sempre possível inferir a falha, isto é, se esse SED possui a propriedade de diagnosticabilidade. Nesse exemplo será implementado o algoritmo 2.2 de verificação em tempo polinomial da propriedade de diagnosticabilidade de um SED. Considere um autômato determinístico G que modela um SED e suponha que o conjunto Σ seja particionado da forma $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ em que Σ_o e Σ_{uo} são os eventos observáveis e não observáveis, respectivamente. Seja $\sigma_f \in \Sigma_{uo}$ o evento de falha.*

A implementação do algoritmo 2.2 é mostrada na Listagem 3.4. Os passos 2-4 são fáceis de acompanhar através dos comentários da Listagem 3.4 (linhas 3,5,7,9,18). No passo 2.1, usa-se a função `labelfailaut` *para construir o autômato A_l. No passo 5 (linhas 24,25), usam-se as funções* `selfloopnodes` *e* `strconncomps` *do pacote* NETWORKX *para encontrar as componentes fortemente conexas não triviais. O contraexemplo da diagnosticabilidade é procurado no ciclo* `for` *da linha 29 e, caso seja encontrado, o algoritmo retorna* `False`*.*

A listagem 3.5 define o sistema G mostrado na figura 3.5a e usa a função da listagem 3.4 (`verifydiagnosability`*) para determinar a diagnosticabilidade do sistema G considerando o conjunto de eventos observáveis $\Sigma_o = \{a, c, d\}$ e uma falha representada pelo evento f. Utilizando-se essa mesma função, obtém-se o autômato verificador G_V, conforme mostrado na figura 3.5b.*

Listagem 3.4: Implementação em DESLAB do algoritmo 2.2.

```
def verify(G, sf, Sigma_uo)
    # STEP 1.1
    GN_prime = G.deletevent(sf)
    # STEP 1.2
    GN = ac(GN_prime)
    # STEP 2.1
    Al = labelfailaut(sf)
    # STEP 2.2
    Gl = G//Al
    Xml = [(x,l) for (x,l) in Gl.X if l=='Y']
    Gl = Gl.setpar(Xm=Xml)
    # STEP 2.3
    GY = coac(Gl)
    # STEP 3
    R = [(event,event+'_R') for event in Sigma_uo]
    GN = GN.renameevents(R)
    # STEP 4
    GV = GN//GY
    XmV = [(xN,(xY,l)) for (xN,(xY,l)) in GV.X if l=='Y']
    GV = GV.setpar(Xm=XmV)
    # STEP 5
    SCC = strconncomps(GV)
    SCC_selfloop = [[l] for l in selfloopnodes(GV)]
    SCC_multiple = [C for C in SCC if (len(C)>1)]
    SCC_nontrivial = SCC_selfloop + SCC_multiple
    diagnosable = True
    # finding a counter example
    for C in SCC_nontrivial:
        for x_V in (set(C) & GV.Xm):
            if (GV.Gamma(x_V) & G.Sigma) <> set([]):
                diagnosable = False    # counter example founded
                break
    return GV, diagnosable
```

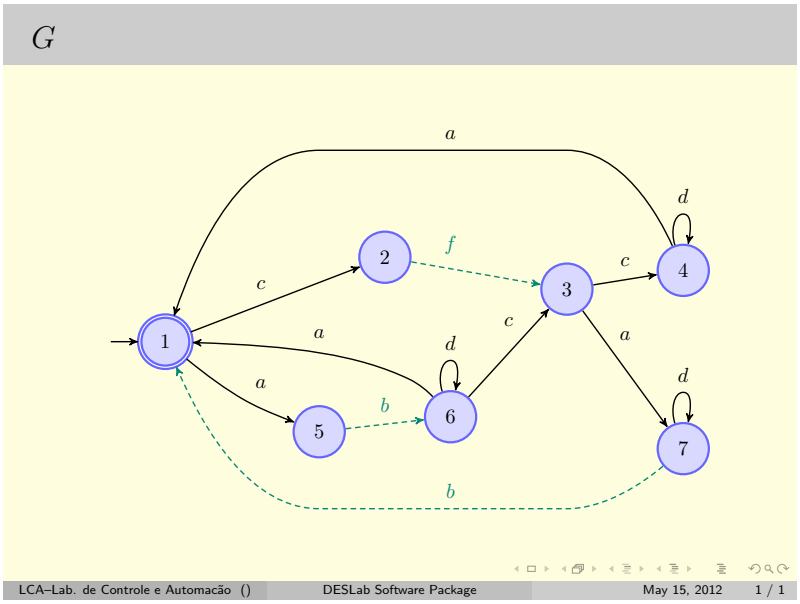
Listagem 3.5: Exemplo de uso da função `verifydiagnosability` da listagem 3.4.

```
syms('a b c d f g 0 1 2 3 4 5 6 7')
sigma= [a,b,c,d,f]
X=[x1,x2,x3,x4,x5,x6,x7]
X0, Xm = [x1], []
transicoes = [[x1,c,x2], [x1,a,x5], [x2,sf,x3], [x3,c,x4], [x4,a,x1], [x6,a,x1],
              [x6,c,x3], [x4,d,x4], [x1,a,x5], [x5,b,x6], [x6,d,x6], [x7,d,x7],
              [x3,a,x7], [x7,b,x1]]
sigmaobs = [a,c,d]
G=fsa(X, sigma, transicoes, X0, Xm, Sigobs=sigmaobs, name='G')
diagnosticavel, GV = verifydiagnosability(G,f)
draw(G,GV)
```

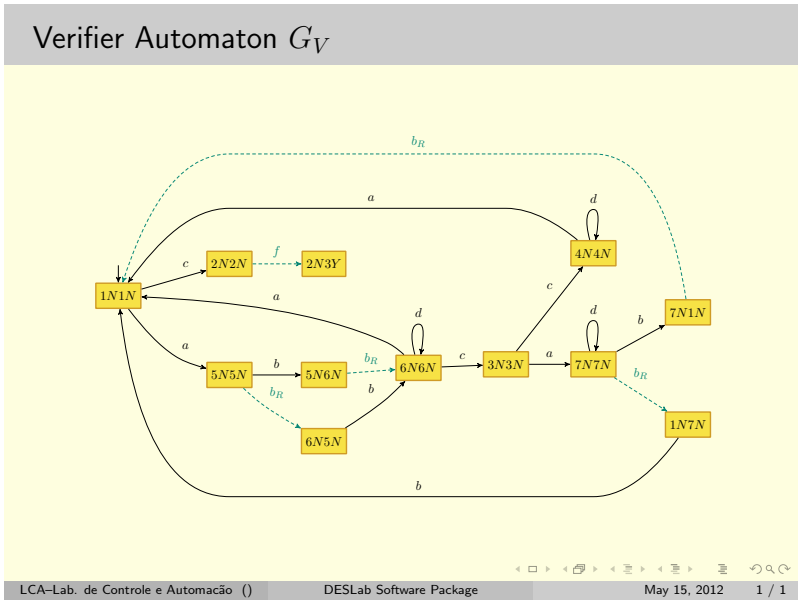
```
>>> print diagnosticavel
  True
```

Conforme mostrado pelo comando axima, o resultado da verificação da propriedade de diagnosticabilidade do sistema G representado na figura 3.5a confirma que

o sistema G é diagnosticável para o conjunto de eventos observáveis $\Sigma_o = \{a, b, d\}$.



(a) Planta G.



(b) Verificador G_V.

Figura 3.5: (a) Sistema G cuja diagnosticabilidade deseja-se verificar. (b) Verificador G_V obtido na verificação da diagnosticabilidade de G.

3.6 Comentários finais

Neste capítulo foi apresentado o programa DESLAB. As maiores contribuições que esse programa oferece são: (*i*) uma estrutura de autômato que define variáveis interativas "vivas"; (*ii*) uma sintaxe simples e expressiva que permite escrever algoritmos complexos de SEDs com poucas linhas de código; (*iii*) integração unificada de autô-

matos, grafos e cálculo numérico; (*iv*) visualização de alta qualidade integrada com o LaTeX.

Capítulo 4

Análise experimental exaustiva da complexidade média em diagnose de falhas de SEDs

Conforme mostrado no capítulo 2, para um autômato $G = (X, \Sigma, f, \Gamma, x_0)$ em tal que $|X| = n$, as complexidades no pior caso do diagnosticador e do verificador são, respectivamente, iguais a 2^{2n} (isto é, de ordem $\mathcal{O}(2^{2n})$), e $2n^2$ (isto é, de ordem $\mathcal{O}(n^2)$). Não obstante existirem esses resultados, a complexidade média também deve ser analisada uma vez que é possível que as instâncias que produzem a máxima complexidade podem raramente aparecer nos conjuntos que satisfazem as condições de diagnosticabilidade.

Neste capítulo são feitas as análises experimentais das complexidade média do diagnosticador proposto por Sampath et al. (1995) e do verificador de diagnosticabilidade proposto por Moreira et al. (2011a) explorando exaustivamente alguns conjuntos de autômatos que satisfazem as condições de diagnosticabilidade. A análise experimental exaustiva realizada neste capítulo tem dois alvos fundamentais: o primeiro é formular hipóteses preliminares sobre o comportamento estatístico das complexidades do diagnosticador e do verificador, e o segundo é determinar as condições para realizar a análise experimental com amostragem que será objeto do capítulo que segue. O diagrama de blocos da figura 4.1 descreve, de forma geral, a análise experimental que será realizada nesse capítulo. Conforme representado, o primeiro passo é usar um gerador exaustivo de autômatos (baseado no método proposto por Reis et al. (2005)) e um algoritmo de rejeição para obter *conjuntos de autômatos válidos* (um autômato válido possui uma estrutura básica que possibilita seu uso como objeto experimental do estudo realizado, conforme será visto mais adiante). O segundo passo é realizar o cálculo do diagnosticador e do verificador para cada *autômato válido* gerado. No terceiro passo os dados compilados são usa-

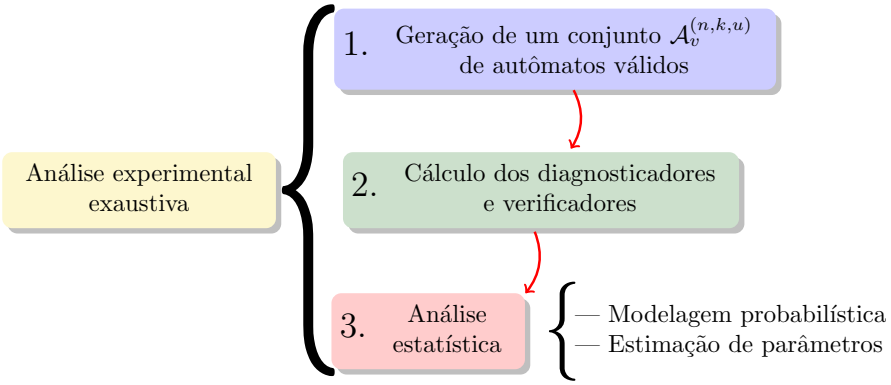


Figura 4.1: Diagrama de blocos do experimento exaustivo

dos para modelar a distribuição da complexidade do diagnosticador e estimar os parâmetros da distribuição selecionada.

Este capítulo está dividido da seguinte forma. Na seção 4.1 apresentamos o gerador exaustivo de autômatos proposto por Reis et al. (2005), o qual será a ferramenta computacional principal do experimento exaustivo proposto para analisar as complexidades médias do diagnosticador e do verificador. A seção 5.2 tem como intuito descrever o experimento exaustivo realizado para determinar as complexidades médias do diagnosticador e do verificador. Na mesma seção são apresentados os resultados experimentais obtidos e também é realizada a análise exploratória dos resultados experimentais. O objetivo da seção 4.3 é avaliar vários modelos probabilísticos candidatos para a descrição da complexidade do diagnosticador, dando lugar, assim, a uma hipótese estatística consistente sobre o modelo probabilístico que descreve de melhor forma a complexidade do diagnosticador. Na seção 4.4 apresentamos um exemplo para ilustrar as implicações de usar o modelo adotado na seção 4.3 como modelo probabilístico da complexidade do diagnosticador. Finalmente, na seção 4.5 são apresentados os comentários finais.

.

4.1 Gerador exaustivo de autômatos

O método de geração exaustiva de autômatos que será usado neste capítulo foi proposto por Reis et al. (2005) e está baseado na representação de autômatos como sequências. Utilizando essa representação é possível expressar autômatos determinísticos acessíveis completos de forma compacta usando um mínimo de memória. A teoria fundamental, conforme desenvolvida por Almeida (2010), é apresentada a seguir. Comecemos pelas seguintes definições.

Definição 4.1. *(Notação Θ) Diz-se que $f(n) \in \Theta(g(n))$ se existirem $c_1, c_2, n_0 > 0$,*

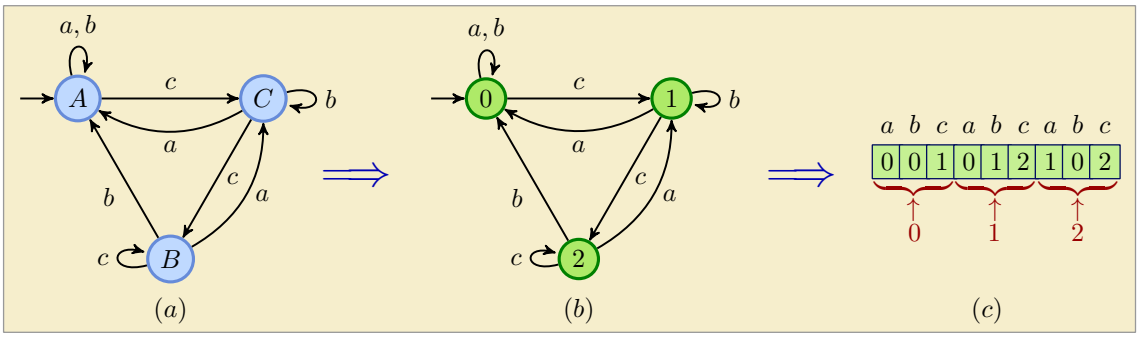


Figura 4.2: Automato G (a); autômato G' tal que $\varphi : G \to G'$ (b); sequência representativa do autômato G' (c).

tais que para todo $n > n_0$,

$$c_1 g(n) \leq f(n) c_2 \leq g(n).$$

□

Definição 4.2. *(Isomorfismo entre autômatos) Sejam* $G = (X, \Sigma, f, \Gamma, x_0)$ *e* $G' = (X', \Sigma, f', \Gamma', x'_0)$ *dois autômatos definidos sobre o mesmo alfabeto* Σ*. Um isomorfismo* φ *entre* G *e* G'*, denotado por* $\varphi : G \to G'$*, é uma função* $\varphi : X \to X'$ *satisfazendo as seguintes condições:*

i. φ *é bijetora;*

ii. $\varphi(x_0) = x'_0$*;*

iii. $\varphi(f(x, \sigma)) = f'(\varphi(x), \sigma)$. □

Seja $G = (X, \Sigma, f, \Gamma, x_0)$ um autômato determinístico acessível completo (ADAC) tal que $|X| = n$ e $|\Sigma| = k$, sendo que o alfabeto Σ possui uma ordem completa[1]. Dessa forma, é possível induzir uma ordem sobre o conjunto de estados X ao fazer uma busca em largura no autômato G, escolhendo as transições em cada estado segundo a ordem do alfabeto Σ. Para tanto, define-se um isomorfismo $\varphi : G \to G'$ ($G' = (X', \Sigma, f', \Gamma', 0)$) em que $X' = \{0, 1, \ldots, n-1\}$. Por exemplo, considere o autômato da figura 4.2(a) com $\Sigma = \{a, b, c\}$ ordenado alfabeticamente. A busca em largura induz a bijeção $\varphi = \{(A, 0), (C, 1), (B, 2)\}$ entre os conjuntos de estados X e o conjunto $X' = \{0, 1, 2\}$, em que os elementos de X' representam a ordem em que cada estado foi visitado. Note que o estado inicial de G' é 0. A figura 4.2(b) mostra o autômato G' isomorfo a G por meio de φ. Na representação

[1] Um conjunto A possui uma relação de ordem completa, denotada pelo simbolo $\leq$, se para todo $a, b, c \in A$ as seguintes condições são satisfeitas: *(i)* $a \leq a$ para todo a (reflexividade); *(ii)* se $a \leq b$ e $b \leq a$, então $a = b$ (antisimetria); *(iii)* se $a \leq b$ e $b \leq c$, então $a \leq c$ (transitividade); *(iv)* $a \leq b$, ou, então, $b \leq a$ (totalidade).

da figura 4.2(c) tem-se a palavra 001012102 de nk símbolos definida na base n representando o automato G'. Cada grupo de k símbolos define os estados para onde o autômato G' irá pelo disparo da transição rotulada pelo evento correspondente à ordenação de Σ. Por exemplo $0 \to 1$ na terceira posição indica que o estado 0 é ligado ao estado 1 pelos evento c.

Formalmente, seja $G = (X, \Sigma, f, \Gamma, x_0)$ um ADAC tal que $|X| = n$ e seja $\Sigma = \{\sigma_1 < \cdots < \sigma_k\}$ um alfabeto completamente ordenado. Denote por $I_{m:n}$ o conjunto de inteiros $\{m, m+1, \ldots n\}$. Seja $\varphi : X \to \{0, 1, \ldots, n-1\}$ um isomorfismo de X em X' induzido pela busca em largura realizada na ordem de Σ. Seja $G' = (X', \Sigma, f', \Gamma', 0)$ o autômato isomorfo a G obtido utilizando a função φ. A *sequência representativa* de G' é da forma[2]. $(s_i)_{i \in I_{0:kn-1}}$ com $s_i \in I_{0:n-1}$, e $s_i = f'(\lfloor i/k \rfloor, \sigma_{(i \mod k)+1})$, em que $k = |\Sigma|$ e $i \in I_{0:kn-1}$.

Seja $\mathcal{C}^{(n,k)}$ o conjunto de todos os autômatos completos com n estados e um alfabeto Σ com k eventos. O resultado a seguir estabelece uma bijeção entre as sequências representativas e o conjunto de ADACs com n estados sobre um alfabeto Σ de cardinalidade k.

Teorema 4.1. *(Reis et al., 2005) Seja $(s_i)_{i \in I_{0:kn-1}}$ em que $s_i \in I_{0:n-1}$, uma sequência satisfazendo as seguintes condições:*

R1. $(\forall m \in I_{2:n-1})(\forall i \in I_{0:kn-1})(s_i = m \Rightarrow (\exists j \in I_{0:i-1})[s_j = m-1])$,

R2. $(\forall m \in I_{1:n-1})(\exists j \in I_{0:km-1})[s_j = m]$.

Então, existe uma bijeção entre o conjunto de sequências satisfazendo **R1** *e* **R2** *e o conjunto de autômatos acessíveis completos $\mathcal{C}^{(n,k)}$.* □

A regra **R1** determina que um estado $x' \in X'$ rotulado por um número maior que zero somente pode aparecer na sequência $(s_i)_{i \in I_{0:kn-1}}$ após alguma ocorrência de seus predecessores. Isso é consequência direita da ordenação dos estados induzida pelo isomorfismo φ. Por exemplo, na sequência 001012102 da figura 4.2(c), $s_5 = 2$ ($m = 2$ e $i = 5$ na regra **R1**) o que implica que pelo menos um dos símbolos da subsequência $s_0 \cdots s_4 = 00101$ deve ser 1. A regra **R2** define a condição de acessibilidade do autômato cuja sequência representativa é $(s_i)_{i \in I_{0:kn-1}}$. Em palavras, **R2** implica que um estado rotulado por m deve aparecer pelo menos uma vez nas primeiras $km-1$ posições da sequência representativa. Por exemplo, a regra **R2**, quando aplicada à sequência da figura 4.2(c) com $m = 1$ e $k = 3$, determina que pelo menos um dos símbolos da subsequência $s_0 s_1 s_2 = 001$ deve ser 1.

Com o intuito de gerar, exaustivamente, autômatos, as regras **R1** e **R2** devem ser reformuladas convenientemente usando *índices*. Seja $(f_j), j \in I_{1:n-1}$, $f_j \in I_{0:kn-1}$ uma sequência de índices em que o símbolo f_j representa o índice da primeira ocor-

[2]A notação $\lfloor x \rfloor$ denota a *parte inteira de* x, definida como $\lfloor x \rfloor = \max\{m \in \mathbb{N} : m \leq x\}$.

rência do estado rotulado por j na sequência representativa $(s_i)_{i\in I_{0:kn-1}}$. Por exemplo na sequência 001012102 da figura 4.2(c), $f_j \in I_{0:8}$, $j \in I_{1:2}$, $f_1 = 2$ e $f_2 = 5$.

Em termos da sequência de índices $(f_j)_{j\in I_{1:n-1}}$, **R1** e **R2** correspondem, respectivamente, às seguintes regras:

G1. $(\forall j \in I_{2:n-1})[f_{j-1} < f_j]$,

G2. $(\forall m \in I_{1:n-1})[f_m \leq km - 1]$.

As regras **G1** e **G2** implicam que $f_1 \in I_{0:k-1}$ e $f_{j-1} \leq f_j \leq kj$ para $j \in I_{2:n}$. Note que, considerando novamente a sequência 001012102, a regra **G1** com $j = 2$ determina a existência de $f_1 < f_2 = 5$, o qual é uma forma equivalente a afirmar que se $s_5 = 2$, então pelo menos um dos símbolos da subsequencia $s_0 \cdots s_4$ deve ser 1 (regra **R1**). Para o mesmo exemplo, a regra **G2** com $m = 1$ determina, que $f_1 \in I_{0:2}$ que é equivalente a assegurar que pelo menos um dos símbolos da subsequência $s_0 s_1 s_2$ deve ser 1 (regra **R2**).

Para gerar todos os ADAC correspondentes a uma sequência de índices fixa $(f_j)_{j\in I_{0:n}}$, os símbolos s_i, em que $i \notin \{f_j : j \in I_{1:n-1}\} \cup I_{0:f_1}$ (isto é, os símbolos não definidos pela sequência de índices $(f_j)_{j\in I_{1:n}}$), devem ser inseridos conforme as regras a seguir:

G3. $(i < f_1)[s_i = 0]$;

G4. $(\forall j \in I_{1:n-2})[f_j < i < f_{j+1} \Rightarrow s_i \in I_{0:j}]$;

G5. $i > f_{n-1} \Rightarrow s_i \in I_{0:n-1}$.

As regras **G3**–**G5** são regras construtivas para obter uma sequência bem formada (representando um ADAC) a partir de uma sequência de índices fixa. Utilizando novamente a sequência 001012102, tem-se que a regra **G3** determina que a subsequência $s_0 s_1 = 00$ à esquerda de $s_{f_1} = s_2$ seja formada por zeros; a regra **G4** determina que a subsequência $s_3 s_4 = 01$ que aparece entre os símbolos $s_{f_1} = s_2$ e $s_{f_2} = s_5$ seja formada por dígitos em base 2; por último, a regra **G5** determina que a subsequência 102 à direita de s_5 seja formada por dígitos em base 3.

Usando a teoria apresentada acima é possível obter um algoritmo para geração exaustiva de autômatos que funciona em dois grandes procedimentos: geração recursiva de sequências e geração recursiva de índices. Esses procedimentos e suas correspondentes implementações serão apresentados a seguir.

1. Geração recursiva de sequências. Seja $(s_i)_{i\in I_{0:kn-1}}$ uma sequência representativa dada, e seja $(f_j)_{j\in I_{1:n-1}}$ uma sequência de índices dada. É possível gerar uma próxima sequência representativa, alterando consistentemente os símbolos de

$(s_i)_{i\in I_{0:kn-1}}$ (conforme as regras **G3**–**G5**) não prefixados pelos índices. O procedimento será descrito a seguir.

Defina o conjunto $NI = I_{f_1+1:kn-1} \setminus \{f_j : j \in I_{1:n-1}\}$ de subíndices dos símbolos de $(s_i)_{i\in I_{0:kn-1}}$ que não foram fixados pela sequência de índices. Conforme as regras **G4** e **G5** a base de cada dígito em $(s_i)_{i\in NI}$ depende da sua posição com relação aos índices. Para determinar a base do dígito s_i, calcule o conjunto

$$B_i = \{i - f_j : (f_j \in \{f_1, \ldots, f_{n-1}\}) \wedge (i - f_j > 0)\},$$

e represente por b_i o único índice f_l tal que $i - f_l = \min(B_i)$, com $f_l \in \{f_1, \ldots, f_{n-1}\}$. Note que $b_i + 1$ representa a base do digito s_i. A regra a seguir determina como alterar um dígito de $(s_i)_{i\in NI}$.

$$s_i' = \begin{cases} s_i + 1, & \text{se } s_i + 1 \leq b_i \\ 0, & \text{caso contrário.} \end{cases} \tag{4.1}$$

O procedimento começa usando a regra (4.1) no dígito s_{kn-1}. Se a condição $s_{kn-1} \leq b_{kn-1} = f_{n-1}$ da regra for verificada, então a próxima sequência representativa será $s_0 \cdots s_{nk-2}(s_{kn-1} + 1)$. Caso contrário, a regra (4.1) deve ser propagada aos outros dígitos de $(s_i)_{i\in NI}$ na ordem descendente de NI, até que a primeira condição seja verificada. O algoritmo 4.1 apresenta a implementação em pseudolinguagem do procedimento descrito acima. Note que, caso exista um dígito s_i satisfazendo $s_i + 1 \leq b_i$, o algoritmo retorna uma nova sequência $(s_i')_{i\in I_{0:kn-1}}$ (veja a linha $l1$). Caso contrário, é retornada a sequência 0^{nk} (veja a linha $l2$) que não corresponde a nenhum ADAC, sinalizando que já foi gerada toda a série de sequências representativas correspondentes à sequência atual de índices.

2. Geração recursiva de índices. Seja $(f_j)_{j\in I_{1:n-1}}$ uma sequência de índices dada. É possível gerar uma nova sequência de índices alterando-se os dígitos de $(f_j)_{j\in I_{1:n-1}}$ conforme as regras **G1**–**G2**. A regra a seguir mostra como alterar consistentemente um digito f_j da sequência $(f_j)_{j\in I_{1:n-1}}$.

$$f_j' = \begin{cases} f_j - 1, & \text{se } j > 1 \text{ e } f_j - 1 > f_{j-1} \\ f_j - 1, & \text{se } j = 1 \\ kj - 1, & \text{em outro caso.} \end{cases} \tag{4.2}$$

O procedimento começa usando a regra (4.2) no dígito f_{n-1}. Caso seja verificada a condição $f_{n-1} - 1 > f_{n-2}$, a próxima sequência de índices será $f_1 f_2 \cdots f_{n-2}(f_{n-1} - 1)$. Caso contrário, a regra (4.2) deve ser propagada na ordem decrescente de $I_{1:n-1}$, para cada f_i de $(f_i)_{i\in I_{1:n-2}}$ até que seja verificada a condição $f_j - 1 > f_{j-1}$ ou até

Algoritmo 4.1: função para geração recursiva de sequências

function GERARSEQUENCIA($(s_i)_{i\in I_{1:nk-1}}$, $(f_j)_{j\in I_{1:n-1}}$)
Entrada: A sequência $(s_i)_{i\in I_{1:nk-1}}$ e a sequência de índices $(f_j)_{j\in I_{1:n-1}}$
Saída: A próxima sequência gerada $(s'_i)_{i\in I_{1:nk-1}}$
$NI \longleftarrow I_{f_1+1:kn-1} - \{f_j : j \in I_{1:n-1}\}$
$(s'_i)_{i\in I_{1:nk-1}} \longleftarrow (s_i)_{i\in I_{1:nk-1}}$
for $i \in NI$, *percorra na ordem descendente de* NI **do**
  $B_i \longleftarrow \{i - f_j : (f_j \in \{f_1, \ldots, f_{n-1}\})[i - f_j > 0]\}$
  $b_i \longleftarrow$ o valor de f_l tal que $i - f_l = \min B_i$, com $f_l \in \{f_1, \ldots, f_l\}$
  if $s_i + 1 \leq b_i$ **then**
    $s'_i \longleftarrow s_i + 1$
$l1$ **return** $(s'_i)_{i\in I_{1:nk-1}}$
  else
    $s'_i \longleftarrow 0$
$l2$ **return** 0^{nk}

Algoritmo 4.2: função para geração recursiva de índices

function GERARINDICES($(f_j)_{j\in I_{1:n-1}}$)
Entrada: *A sequência a sequência de índices* $(f_j)_{j\in I_{1:n-1}}$
Saída: *A próxima sequência de índices gerada* $(f_j)_{j\in I_{1:n-1}}$
$(f'_j)_{j\in I_{1:n-1}} \longleftarrow (f_j)_{j\in I_{1:n-1}}$
for $j \in I_{1:n-1}$, *percorra na ordem descendente de* $I_{1:n-1}$ **do**
  if $j = 1$ **then**
    $f'_1 \longleftarrow f_1 - 1$
$c1$ **return** $(f'_j)_{j\in I_{1:n-1}}$
  if $(f_j - 1 > f_{j-1})$ **then**
    $f'_j \longleftarrow f_j - 1$
$c2$ **return** $(f'_j)_{j\in I_{1:n-1}}$
  else
    $f'_j \longleftarrow kj - 1$

chegar se chegarmos ao índice f_1. O algoritmo 4.2 representa a implementação em pseudolinguagem do procedimento para geração de índices. Note que o algoritmo 4.2 retorna uma nova sequência de índices $(f'_i)_{i\in I_{1:n-2}}$ caso seja verificada a condição $f_j - 1 > f_{j-1}$ (linha $c1$) ou quando f_1 for atingido (linha $c2$). O algoritmo 4.3 permite gerar todos os autômatos completos de um conjunto $C^{(n,k)}$ integrando, para isso, as funções de geração de sequências do algoritmo 4.1 e de geração de índices do algoritmo 4.2.

O seguinte exemplo ilustra o processo de geração de um conjunto $\mathcal{C}^{(n,k)}$ usando o algoritmo 4.3.

Exemplo 4.1. *Vamos ilustrar o funcionamento do algoritmo 4.3 usando como exemplo a geração exaustiva do conjunto* $\mathcal{C}^{(3,2)}$ *de autômatos completos com* $n = 3$ *estados e* $k = 2$ *eventos. Note que, para esse caso, cada sequência de índices é da forma*

Algoritmo 4.3: Procedimento de geração exaustiva de autômatos em $C^{(n,k)}$.

Entrada: *As cardinalidades $n = |X|$ do conjunto de estados e $k = |\Sigma|$ do conjunto de eventos dos autômatos $G \in C^{(n,k)}$.*

Saída: *O conjunto de autômatos completos $C^{(n,k)}$.*

1 $(f_j)_{j\in I_{1:n-1}} \longleftarrow (kj-1)_{j\in I_{1:n-1}}$ *(sequência inicial de índices)*
2 $(s_i)_{i\in I_{1:kn-1}} \longleftarrow 0^{k-1}10^{k-1}\dots(n-2)0^{k-1}(n-1)0^k$ *(sequência representativa inicial)*
3 $C^{(n,k)} \longleftarrow \{$autômato associado à sequência inicial $(s_i)_{i\in I_{1:kn-1}}\}$
4 $S_f \longleftarrow 12\dots(n-1)(n-1)^{(k-1)n+1}$ *(sequência representativa final)*
5 **while** $(s_i)_{i\in I_{1:kn-1}} \neq S_f$ **do**
6 $\quad (s_i)_{i\in I_{1:kn-1}} \longleftarrow$ GerarSequencia$((s_i)_{i\in I_{1:kn-1}})$
7 $\quad$ **if** $(s_i)_{i\in I_{1:kn-1}} = 0^{nk}$ **then**
8 $\quad\quad (f_j)_{j\in I_{1:n-1}} \longleftarrow$ GerarIndices$((f_j)_{j\in I_{1:n-1}})$
9 $\quad\quad$ **for** $j \in I_{1:n-1}$ **do**
10 $\quad\quad\quad s_{f_j} = j$
11 $\quad C^{(n,k)} \longleftarrow C^{(n,k)} \cup \{$autômato associado à sequência $(s_i)_{i\in I_{1:kn-1}}\}$

$(f_j)_{j\in I_{1:n-1}} = f_1f_2$ *e cada sequência representativa é da forma* $(s_i)_{i\in I_{0:kn-1}} = s_0\cdots s_5$. *A evolução do algoritmo 4.3 é apresentada, de forma sucinta, na tabela 4.1, na qual cada coluna representa a série de sequências representativas correspondente a uma sequência de índices particular e, por sua vez, cada elemento da coluna é uma sequência representativa com sua respetiva enumeração na ordem em que foi gerada. O algoritmo começa pela sequência de índices* $f_1f_2 = (kj-1)_{j\in I_{1:n-1}} = 13$ *e pela sequência representativa semente* $0^{k-1}10^{k-1}\dots(n-2)0^{k-1}(n-1)0^k = 010200$ *(linhas 1 e 2) de forma que a utilização recursiva da função* GerarSequencia$(s_0\cdots s_5)$ *(linha 6) apresentada no algoritmo 4.1 produz a série de sequências representativas que aparecem na coluna 1 da tabela 4.1 até chegar à ultima sequência representativa possível, dada por* 011222 *(esse fato é detectado pela aparição da sequência inválida* 000000 *na linha 7 do algoritmo). Neste ponto a função* GerarIndices(f_1f_2) *(linha 8) do é usada para gerar a próxima sequência de índices dada por* $f_1f_2 = 12$ *que, por sua vez, determina uma nova sequência representativa semente, dada por* 012000 *(linhas 9 e 10). A partir dessa semente, a função* GerarSequencia$(s_0\cdots s_5)$ *é usada novamente para gerar a série de sequências correspondente aos índices* $f_1f_2 = 12$ *(veja a coluna 2 da tabela 4.1) até chegar à sequência* 012222 *(seguida da sequência sinalizadora* 00000*). O processo continua de forma semelhante para as sequencias de índices* $f_1f_2 = 03$, $f_1f_2 = 02$ *cujas séries de sequências representativas aparecem, respectivamente, nas colunas 3 e 4 da tabela 4.1. Finalmente, quando gerada a sequência de índices* $f_1f_0 = 01$ *e a função* GerarSequencia$(s_0\cdots s_5)$ *progride (coluna 5 da tabela 4.1), é atingida a última sequência representativa possível* $S_f = 12\dots(n-1)(n-1)^{(k-1)n+1} = 122222$ *(essa condição é detectada na linha 5).*

❑

Tabela 4.1: Série de sequências representativas dos autômatos em $C^{(3,2)}$ geradas pelo algoritmo 4.3.

$f_1f_2 = 13$	$f_1f_2 = 12$	$f_1f_2 = 03$	$f_1f_2 = 02$	$f_1f_2 = 01$
(010200,1)	(012000,19)	(100200,46)	(102000,82)	(120000,136)
(010201,2)	(012001,20)	(100201,47)	(102001,83)	(120001,137)
⋮	⋮	⋮	⋮	⋮
(010222,9)	(012022,27)	(100222,54)	(102222,108)	(120222,162)
(011200,10)	(012100,28)	(101200,55)	(112000,109)	(121000,163)
⋮	⋮	⋮	⋮	⋮
(011221,17)	(012221,44)	(111221,80)	(112221,134)	(122221,215)
(011222,18)	(012222,45)	(111222,81)	(112222,135)	(122222,216)

A seguir apresentamos dois resultados que fornecem, respectivamente, uma cota superior e um valor exato para a cardinalidade do conjunto $\mathcal{C}^{(n,k)}$, precedidos pela seguinte definição:

Definição 4.3. *(Número de Stirling de segunda clase) O número de Stirling de segunda clase é definido como:*

$$\left\{ {m \atop n} \right\} = \frac{1}{n!} \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} i^m. \tag{4.3}$$

□

Teorema 4.2. *(Almeida et al., 2007) O número de autômatos da família $\mathcal{C}^{(n,k)}$, com $k \geq 1$ e $n > 1$ satisfaz*

$$\left|\mathcal{C}^{(n,k)}\right| \leq n^k \left\{ {k(n-1)+1 \atop n} \right\}. \tag{4.4}$$

□

Teorema 4.3. *(Almeida et al., 2007) O número de autômatos do conjunto $\mathcal{C}^{(n,k)}$, com $k \geq 1$ e $n > 1$ é*

$$\left|\mathcal{C}^{(n,k)}\right| = \sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \sum_{f_3=f_2+1}^{3k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} \prod_{i=1}^{n} i^{f_i - f_{i-1} - 1}, \tag{4.5}$$

em que $f_0 = -1$ e $f_n = kn$. □

Exemplo 4.2. *Consideremos, novamente, o conjunto $\mathcal{C}^{(3,2)}$ de ADACs com $n = 3$ estados e $k = 2$ eventos. Vamos estimar a cardinalidade aproximada desse conjunto*

usando a inequação (4.4).

$$\begin{aligned}\left|\mathcal{C}^{(3,2)}\right| &\leq n^k \left\{ {k(n-1)+1 \atop n} \right\} = 3^2 \left\{ {5 \atop 3} \right\} \\ &\leq \frac{9}{3!}\left[(-1)^3\binom{3}{0}\cdot 0^5 + (-1)^2\binom{3}{1}\cdot 1^5 + (-1)\binom{3}{2}\cdot 2^5 + (-1)^0\binom{3}{3}\cdot 3^5\right] \\ &\leq \frac{3}{2}\left[0+3-96+243\right] = 225.\end{aligned}$$

Agora vamos usar a equação (4.5) *para determinar a cardinalidade de* $C^{(3,2)}$*, obtida antes por enumeração exaustiva no exemplo 4.1.*

$$\begin{aligned}\left|\mathcal{C}^{(3,2)}\right| &= \sum_{f_1=0}^{k-1}\sum_{f_2=f_1+1}^{2k-1}\prod_{j=1}^{n} j^{f_j-f_{j-1}-1}\Big|_{f_0=-1,f_n=kn} \\ &= \sum_{f_1=0}^{1}\sum_{f_2=f_1+1}^{3}\prod_{j=1}^{3} j^{f_j-f_{j-1}-1}\Big|_{f_0=-1,f_3=6} \\ &= \sum_{f_2=1}^{3}\prod_{j=1}^{3} j^{f_j-f_{j-1}-1}\Big|_{f_0=-1,f_1=0,f_3=6} \quad + \sum_{f_2=2}^{3}\prod_{j=1}^{3} j^{f_j-f_{j-1}-1}\Big|_{f_0=-1,f_1=1,f_3=6} \\ &= (1\cdot 2^0\cdot 3^4 + 1\cdot 2^1\cdot 3^3 + 1\cdot 2^2\cdot 3^2) \quad + (1\cdot 2^0\cdot 3^3 + 1\cdot 2^1\cdot 3^2) = 216.\end{aligned}$$

O que confere com o número de sequências representativas mostradas na tabela 4.1, obtidas de acordo com o algoritmo 4.3 ❑

Seja $\mathcal{A}^{(n,k)}$ o conjunto de autômatos acessíveis em que $|X| = n$ e $|\Sigma| = k$. O método descrito acima foi concebido para gerar autômatos completos. Porém, o nosso intuito é gerar famílias de autômatos acessíveis que não, necessariamente, sejam completos. A bijeção estabelecida no lema 1 de Bassino et al. (2009, p. 8) (esse resultado será revisado no seguinte capítulo) permite formular o algoritmo 4.4, o qual gera todos os autômatos acessíveis pertencentes ao conjunto $\mathcal{A}^{(n,k)}$.

Algoritmo 4.4: Geração de exaustiva de autômatos acessíveis em $\mathcal{A}^{(n,k)}$.

Passo 1. Use o algoritmo 4.3 para gerar as sequências representativas do conjunto de autômatos completos $\mathcal{C}^{(n+1,k)}$ substituindo a sequência final S_f desse algoritmo (na linha 4) pela sequência $S'_f = 0^{k-1}12\ldots(n-1)(n-1)^{(k-1)(n-2)+1}$.

Passo 2. Transforme cada sequência gerada em um autômato completo $G_c \in \mathcal{C}^{(n+1,k)}$ e, para esse autômato, apague o estado inicial (rotulado por 0) e suas transições, definindo como novo estado inicial o estado 1. O autômato G resultante pertence ao conjunto $\mathcal{A}^{(n,k)}$.

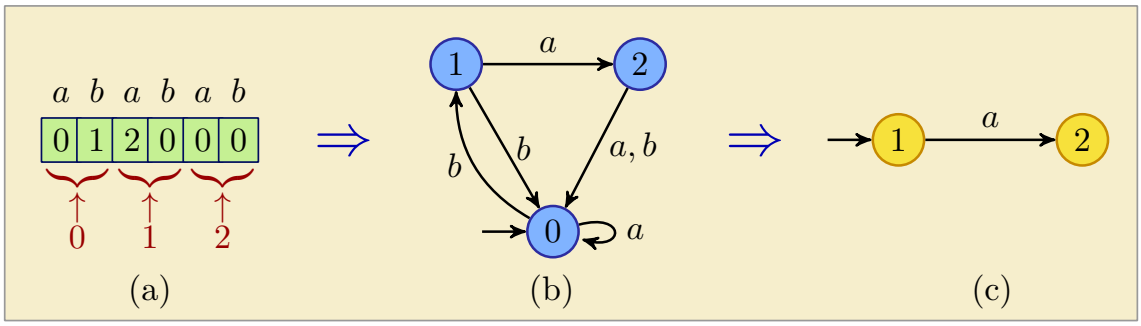


Figura 4.3: Sequencia representativa de um autômato em $\mathcal{C}^{(3,2)}$ (a); autômato completo $Gc \in \mathcal{C}^{(3,2)}$ (b); autômato acessível $G \in A^{(2,2)}$ (c).

O seguinte resultado fornece um limite assintótico para a cardinalidade do conjunto $A^{(n,k)}$ de autômatos acessíveis.

Teorema 4.4. *(Bassino et al., 2009) O número de autômatos acessíveis do conjunto $\mathcal{A}^{(n,k)}$, isto é $\left|\mathcal{A}^{(n,k)}\right|$, é de ordem $\Theta\left(n\left\{{kn \atop n}\right\}\right)$.* □

Como consequência do teorema 4.3 e da sequência S'_f no passo 1 do algoritmo 4.4, deriva-se a seguinte fórmula para determinar a cardinalidade de $\mathcal{A}^{(n,k)}$:

$$\left|\mathcal{A}^{(n,k)}\right| = \sum_{f_1=k}^{2k-1} \sum_{f_2=f_1+1}^{3k-1} \sum_{f_3=f_2+1}^{4k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{kn-1} \prod_{i=1}^{n} (i+1)^{f_i - f_{i-1} - 1}, \tag{4.6}$$

em que $f_0 = k-1$ e $f_n = k(n+1)$.

Exemplo 4.3. *Vamos ilustrar o uso do algoritmo 4.4 para gerar o conjunto $\mathcal{A}^{(2,2)}$ de autômatos acessíveis com $n = 2$ estados e $k = 2$ eventos. O primeiro passo é gerar as sequências do conjunto $\mathcal{C}^{(n+1,k)} = \mathcal{C}^{(3,2)}$, até chegar à sequência final*

$$S'_f = 0^{k-1}12\ldots(n-1)(n-1)^{(k-1)(n-1)+1} = 012222,$$

o que corresponde a gerar as 45 sequências das colunas 1 e 2 da tabela 4.1. Para ilustrar o passo 2 do algoritmo 4.4 considere a sequência 012000 *da figura 4.3(a), a qual representa o autômato completo $G_c \in \mathcal{C}^{(3,2)}$ mostrado na figura 4.3(b). Apagando o estado rotulado por 0 (junto com suas transições) e definindo o estado rotulado por 1 como estado inicial do autômato resultante, obtém-se o autômato acessível $G \in \mathcal{A}^{(2,2)}$ mostrado na figura 4.3(c).*

Podemos verificar a cardinalidade de $\mathcal{A}^{(2,2)}$ usando a equação (4.6)*. Assim:*

$$\begin{aligned}
\left|\mathcal{A}^{(2,2)}\right| &= \sum_{f_1=k}^{2k-1} \prod_{i=1}^{n} (i+1)^{f_i - f_{i-1} - 1}\Big|_{f_0=k-1, f_n=k(n+1)} \\
&= \sum_{f_1=2}^{3} \prod_{i=1}^{2} (i+1)^{f_i - f_{i-1} - 1}\Big|_{f_0=1, f_2=6} \\
&= 2^0 \times 3^3 + 2 \times 3^2 = 45.
\end{aligned}$$

❑

4.2 Experimento exaustivo para a análise da complexidade média em diagnose de falhas

4.2.1 Variáveis do experimento exaustivo

Seja Σ_k um alfabeto ordenado e suponha que Σ_k seja particionado como $\Sigma_k = \Sigma_{k,o} \dot{\cup} \Sigma_{k,u}$ em que $\Sigma_{k,o} = \{\sigma_1, \sigma_2, \ldots, \sigma_{k-u}\}$ é o conjunto de eventos observáveis e $\Sigma_{k,u} = \{\sigma_{k-u+1}, \sigma_{k-u+2}, \ldots, \sigma_k\}$ é o conjunto de eventos não-observáveis[3]. Note que $|\Sigma_{k,u}| = u$, isto é, os k últimos eventos de Σ_k serão, por hipótese, considerados não-observáveis. Seja o autômato $G_i^{(n,k,u)} = (X_n, \Sigma_k, f_i^{(n,k)}, \Gamma_i^{(n,k)}, 1) \in \mathcal{A}^{(n,k)}$ com $X_n = \{1, \ldots, n\}$ e que possui u eventos não-observáveis $\sigma_{k-u+1}, \sigma_{k-u+2}, \ldots, \sigma_k$. Suponha, ainda, que o evento σ_k representa o único evento de falha existente em $G_i^{(n,k,u)}$.

O diagnosticador associado ao autômato $G_i^{(n,k,u)}$ (obtido de acordo com a equação (2.10), será denotado denotado por:

$$G_{D,i}^{(n,k,u)} = (X_{D,i}^{(n,k,u)}, \Sigma_{k,o}, f_{D,i}^{(n,k,u)}, \Gamma_{D,i}^{(n,k,u)}, x_{0D,i}^{(n,k,u)})$$

e o verificador (construído usando o algoritmo 2.2) será denotado por:

$$G_{V,i}^{(n,k,u)} = (X_{V,i}^{(n,k,u)}, \Sigma_{V,k}, f_{V,i}^{(n,k,u)}, \Gamma_{V,i}^{(n,k,u)}, (0, (0, N))).$$

Definição 4.4. *(Conjunto de autômatos válidos) Um autômato $G_i^{(n,k,u)} \in \mathcal{A}^{(n,k)}$ é um autômato válido para análise experimental da complexidade média da construção de diagnosticadores e verificadores em diagnose de falhas se satisfizer as seguintes propriedades:*

- ***P1.*** *A linguagem gerada por $G_i^{(n,k,u)} \in \mathcal{A}^{(n,k)}$ é viva, isto é, $\Gamma_i^{(n,k)}(x) \neq \emptyset$ para todo $x \in X_n$.*
- ***P2.*** *Para todo $\sigma \in \Sigma_k$, existe $x \in X_n$ tal que $\sigma \in \Gamma_i^{(n,k)}(x)$, isto é, para cada evento de Σ_k existe pelo menos uma transição de $G_i^{(n,k,u)}$ rotulada por esse evento.*
- ***P3.*** *$G_i^{(n,k,u)}$ não possui ciclos cujas transições são rotuladas somente por eventos não-observáveis sendo uma dessa transições rotulada pelo evento de falha σ_k.*

*Um conjunto $\mathcal{A}_v^{(n,k,u)} \subset \mathcal{A}^{(n,k)}$ formado por todos os autômatos com n estados, k eventos e u eventos não observáveis e que satisfazem as propriedades **P1**–**P3** será denominado um conjunto de autômatos válidos.* □

[3]Note que o fato de incluir a definição de eventos observáveis e não-observáveis não altera a estrutura de transições de um autômato e, portanto, sua pertinência ao conjunto $\mathcal{A}^{(n,k)}$.

A propriedade **P1** é usualmente feita nos estudos de diagnosticabilidade de falhas de SEDs. Em nosso caso, dado que os autômatos são gerados automaticamente, a propriedade **P2** garante que os autômatos analisados possuem exatamente k eventos, dos quais exatamente u são não-observáveis (incluindo o evento de falha σ_k). A propriedade **P3** desconsidera autômatos estruturalmente não diagnosticáveis, os quais possuem falhas dentro de ciclos não observáveis[4]. Essa é uma propriedade menos restritiva do que hipótese imposta por Sampath et al. (1995), na qual desconsideram-se autômatos que possuem qualquer tipo de ciclos formados por eventos não observáveis. Finalmente, note que $1 < u < k$ porque pelo menos o evento de falha deve ser não-observável.

Instâncias consideradas no experimento exaustivo

No experimento exaustivo foi analisada a seguinte classe de conjuntos de autômatos válidos:

$$\mathcal{A}_E = \{\mathcal{A}_v^{(n,k,u)} : (n,k,u) \in I_E\},$$

em que

$$\begin{aligned} I_E = \{&(3,3,1),(3,3,2),(3,4,1),(3,4,2),(3,4,3),\\ &(3,5,1),(4,2,1),(4,3,1),(4,3,2),(5,2,1)\} \end{aligned}$$

Os conjuntos de $\mathcal{A}_E$ foram selecionados por possuírem uma cardinalidade "moderada" para serem gerados e analisados exaustivamente e, por sua vez, um número suficiente de autômatos que permita obter conclusões estatisticamente significantes.

Variáveis de saída do experimento

A complexidade dos algoritmos para o cálculo do diagnosticador e do verificador será medida pelo número de estados dos autômatos $G_{D,i}^{(n,k,u)}$ e $G_{V,i}^{(n,k,u)}$ para os conjuntos de autômatos válidos $\mathcal{A}_v^{(n,k,u)}$ em que $(n,k,u) \in I_E$.

As variáveis que serão consideradas no experimento são:

1. $\mathbf{D}^{(n,k,u)} = (D_1^{(n,k,u)}, D_2^{(n,k,u)}, \dots)$: variável contendo o número de estados do diagnosticador $D_i^{(n,k,u)} = |X_{D,i}^{(n,k,u)}|$ calculado para cada autômato $G_i^{(n,k,u)}$ no conjunto de autômatos válidos $\mathcal{A}_v^{(n,k,u)}$.

2. $\mathbf{V}^{(n,k,u)} = (V_1^{(n,k,u)}, V_2^{(n,k,u)}, \dots)$: variável contendo o número de estados do verificador $V_i^{(n,k,u)} = |X_{V,i}^{(n,k,u)}|$ calculado para cada autômato $G_i^{(n,k,u)}$ no conjunto de autômatos válidos $\mathcal{A}_v^{(n,k,u)}$.

[4]Note que essa propriedade pode ser verificada em tempo lineal por meio de uma exploração de componentes fortemente conexas no subautômato formado unicamente pelas transições não observáveis de $G_i^{(n,k,u)}$.

4.2.2 Procedimento experimental

Considere um conjunto $\mathcal{A}_v^{(n,k,u)}$ de autômatos válidos. Um teste exaustivo completo para esse conjunto compreende os seguintes passos:

Passo 1: *Geração do conjunto $\mathcal{A}_v^{(n,k,u)}$ de autômatos válidos.*

Esse passo é realizado combinando o gerador exaustivo de autômatos descrito na seção 4.1 com um algoritmo de *rejeição*. Para cada autômato produzido pelo gerador exaustivo define-se o conjunto de eventos observáveis $\Sigma_{k,o}$ e o conjunto de falha $\Sigma_f = \{\sigma_k\}$ obtendo, assim, o automato candidato $G_i^{(n,k,u)}$. A seguir o algoritmo de rejeição verifica se as propriedades **P1**–**P3** são satisfeitas pelo autômato candidato $G_i^{(n,k,u)}$. Caso $G_i^{(n,k,u)}$ satisfaça **P1**–**P3**, ele será acrescentado ao conjunto $\mathcal{A}_v^{(n,k,u)}$. O processo continua até que todos os autômatos de $A^{(n,k)}$ tenham sido gerados. Cada conjunto $\mathcal{A}_v^{(n,k,u)}$ gerado é guardado em uma base de dados geral de autômatos válidos.

Passo 2: *Cálculos da complexidade computacional.*

Nesse passo é processado um conjunto $\mathcal{A}_v^{(n,k,u)}$ de autômatos válidos tomado da base de dados produzida no passo 1. Para cada autômato $G_i^{(n,k,u)}$ do conjunto $\mathcal{A}_v^{(n,k,u)}$ são construídos o diagnosticador e o verificador associados. As variáveis $\mathbf{D}^{(n,k,u)}$ e $\mathbf{V}^{(n,k,u)}$ contendo os resultados dos cálculos da complexidade (número de estados do diagnosticador e do verificador, respectivamente) são guardadas em uma base de dados de resultados experimentais para pós-processamento estatístico.

4.2.3 Resultados experimentais

A tabela 4.2 apresenta os resultados do experimento exaustivo para os conjuntos indexados pelas triplas $(n, k, u) \in I_E$ que aparecem na primeira coluna.

As colunas 2 e 3 mostram, respectivamente, a cardinalidade cada um dos conjuntos de autômatos acessíveis $\mathcal{A}^{(n,k)}$ e de autômatos válidos $\mathcal{A}_v^{(n,k,u)}$. As colunas 4-6 mostram, respectivamente, o valor máximo, a média e o desvio padrão para cada variável $\mathbf{D}^{(n,k,u)}$ que representa a cardinalidade do espaço de estados do diagnosticador construído para cada autômato válido $G_i^{(n,k,u)} \in \mathcal{A}_v^{(n,k,u)}$. A coluna 7 apresenta, para propósito de comparação, a complexidade no pior caso do diagnosticador. Por exemplo, foram gerados 23,846,125 autômatos accessíveis em $A^{(4,3)}$ para obter o conjunto $\mathcal{A}_v^{(4,3,1)}$ que possui 4,577,395 autômatos válidos e, para esse último conjunto, a variável $\mathbf{D}^{(4,3,1)}$ possui um valor máximo de 53, um valor médio de 8,7 e um desvio padrão de 4,7, enquanto o valor de 256 da complexidade esperada no pior caso nunca é atingido.

Tabela 4.2: Resultados de complexidade computacional (estimada em número de estados) na construção do Diagnosticador e do Verificador para cada $\mathcal{A}_v^{(n,k,u)} \in \mathcal{A}_E$.

	Número de autômatos		Diagnosticador				Verificador			
(n,k,u)	$\lvert\mathcal{A}^{(n,k)}\rvert$	$\lvert\mathcal{A}_v^{(n,k,u)}\rvert$	max	m_D	s_D	2^{2n}	max	m_V	s_V	$2 \times n^2$
$(3,3,1)$	81856	18387	20	5,9	2,9	64	12	7,6	3,1	18
$(3,3,2)$	81856	7231	9	2,7	1,0	64	18	10,3	5,3	18
$(3,4,1)$	6516480	1472867	21	8,4	3,8	64	12	8,9	2,8	18
$(3,4,2)$	6516480	674727	20	4,8	2,4	64	18	12,4	5,0	18
$(3,4,3)$	6516480	280043	9	2,4	0,8	64	18	12,5	5,2	18
$(3,5,1)$	467590144	10830975	21	7,7	3,8	64	12	7,4	3.3	18
$(4,2,1)$	20225	3384	16	3,6	1,5	256	20	8,8	4,5	32
$(4,3,1)$	23846125	4577395	53	8,7	4,7	256	20	12,8	5,0	32
$(4,3,2)$	23846125	1201023	16	3,0	1,1	256	32	19,0	9,2	32
$(5,2,1)$	632700	90533	25	4,0	1,6	1024	30	12,2	6,4	50

As colunas 8-10 mostram os valores estatísticos (o valor máximo, a média e o desvio padrão) de cada variável $\mathbf{V}^{(n,k,u)}$ que representa a cardinalidade do espaço de estados do verificador construído para cada autômato válido $G_i^{(n,k,u)} \in \mathcal{A}_v^{(n,k,u)}$. Por exemplo, podemos verificar que a variável $\mathbf{V}^{(4,3,1)}$ possui um valor máximo de 20, um valor médio de 12,8, um desvio padrão de 5,0 e uma complexidade no pior caso de 32.

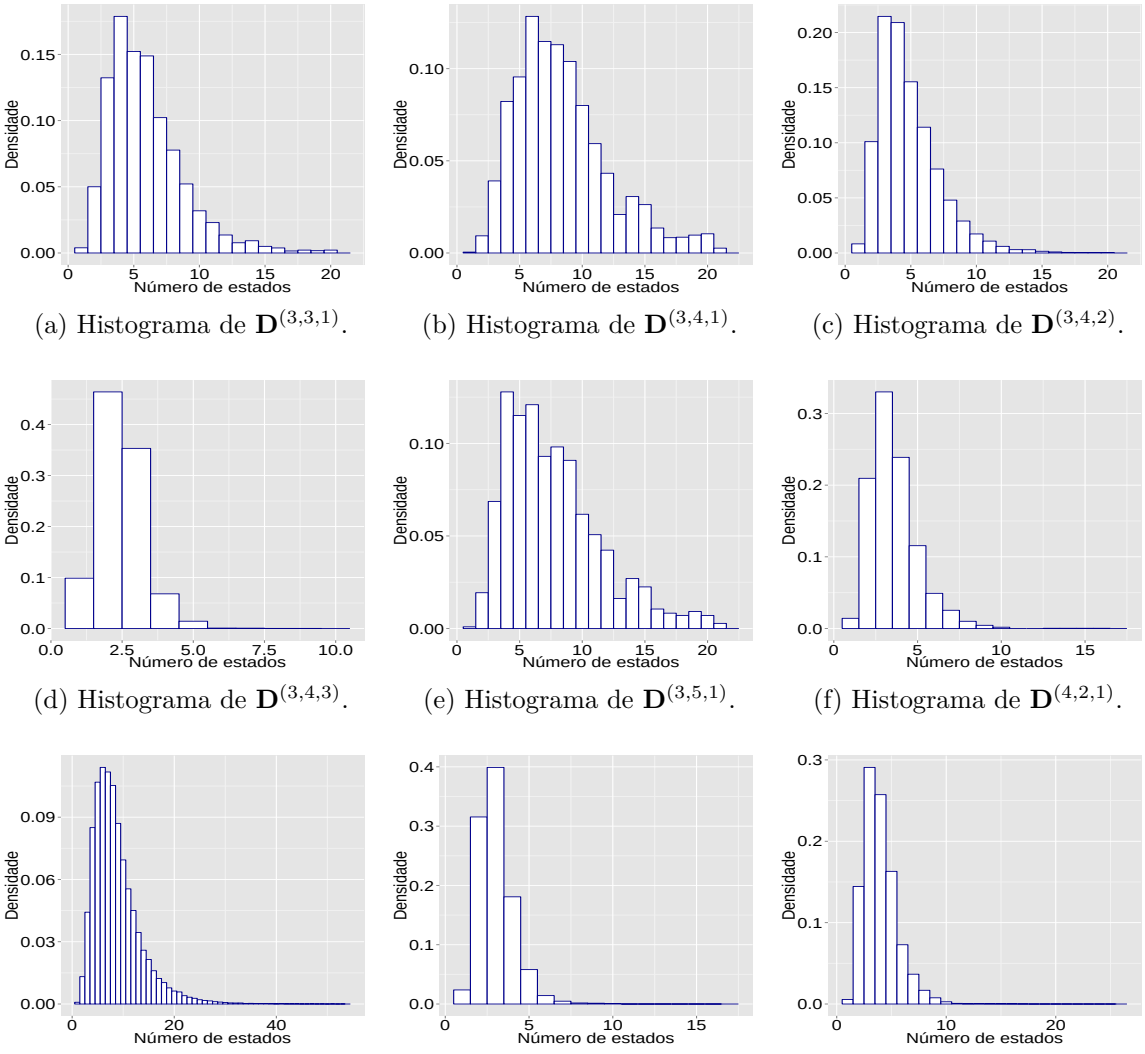
Note que para todos os conjuntos de $\mathcal{A}_v^{(n,k,u)}, (n,k,u) \in I_E$ analisados, o valor da complexidade média do diagnosticador é inferior (inclusive em várias ordens de grandeza) em relação à complexidade esperada no pior caso. Por outro lado, no caso do verificador, a complexidade média é da mesma ordem de grandeza da complexidade no pior caso.

É interessante observar o comportamento monotonicamente decrescente da complexidade média do diagnosticador com relação ao aumento no número de eventos não-observáveis; por exemplo, nos conjuntos de autômatos válidos $\mathcal{A}_v^{(3,4,1)}$, $\mathcal{A}_v^{(3,4,2)}$ e $\mathcal{A}_v^{(3,4,3)}$ (com 1, 2 e 3 eventos não observáveis, respectivamente) o valor médio da complexidade do diagnosticador diminui de 8,4 para 2,4 na medida que aumenta o número de eventos não observáveis.

Se considerarmos o caso do verificador, podemos observar que a complexidade possui um comportamento monotonicamente crescente em relação ao aumento do número de eventos não-observáveis; por exemplo para os mesmos conjuntos $\mathcal{A}_v^{(3,4,1)}$, $\mathcal{A}_v^{(3,4,2)}$ e $\mathcal{A}_v^{(3,4,3)}$, a complexidade média aumenta de 8.9 para 12.5 ao aumentar o número de eventos não observáveis. Observe, ainda, que as variáveis $V^{(3,3,2)}$, $V^{(3,4,2)}$, $V^{(3,4,3)}$ atingem a complexidade no pior caso de $2 \times 3^2 = 18$, assim como a variável $\mathcal{A}_v^{(4,3,2)}$ que atinge a complexidade no pior caso de $2 \times 4^2 = 32$.

4.2.4 Análise exploratória

A figura 4.4 apresenta os histogramas das variáveis $\mathbf{D}^{(n,k,u)}$ correspondentes aos conjuntos de autômatos válidos $\mathcal{A}_v^{(n,k,u)}$, $(n,k,u) \in I_E$, listados na tabela 4.2. De acordo com várias aplicações de modelagem estatística (Dey and Kundu, 2009; Gokhale and Mullen, 2005; Heyman et al., 1992; Mullen and Gokhale, 2005; Ni and Li, 2011; Parthiban et al., 2005) e com as formas características apresentadas nos histogramas mostrados na figura 4.4 é razoável considerar como modelos candidatos as distribuições de Weibull, Gamma, Lognormal e Negativa Binomial, cujas funções de densidade (ou massa) de probabilidade estão representadas na tabela 4.3.



(a) Histograma de $\mathbf{D}^{(3,3,1)}$. (b) Histograma de $\mathbf{D}^{(3,4,1)}$. (c) Histograma de $\mathbf{D}^{(3,4,2)}$.

(d) Histograma de $\mathbf{D}^{(3,4,3)}$. (e) Histograma de $\mathbf{D}^{(3,5,1)}$. (f) Histograma de $\mathbf{D}^{(4,2,1)}$.

(g) Histograma de $\mathbf{D}^{(4,3,1)}$. (h) Histograma de $\mathbf{D}^{(4,3,2)}$. (i) Histograma de $\mathbf{D}^{(5,2,1)}$.

Figura 4.4: Histogramas das variáveis $\mathbf{D}^{(n,k,u)}$.

Dentre os métodos de análise exploratória de dados, um método gráfico muito utilizado é o gráfico Q-Q, no qual são plotados os quantis[5] da distribuição teórica

[5]Se Y é uma variável aleatória com função de distribuição acumulada F, então o quantil q de

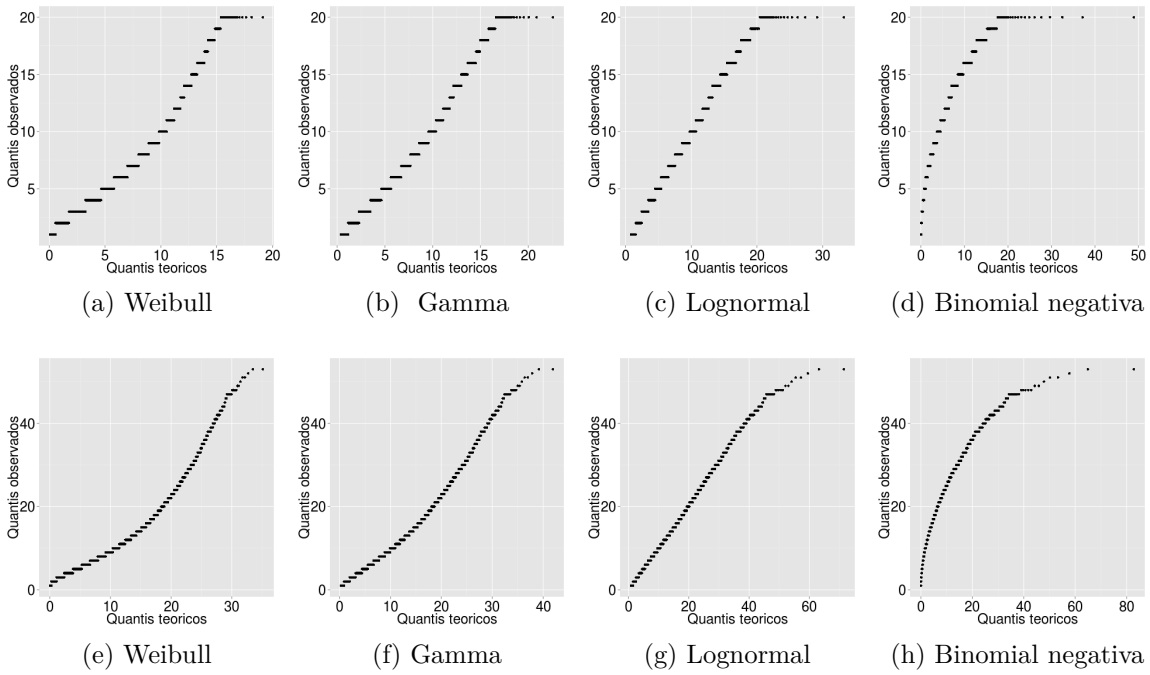
Tabela 4.3: Modelos probabilísticos candidatos.

Lognormal	Notação:	$Y \sim LN(\eta, \sigma)$
	fdp:	$f_{LN}(y;\eta,\sigma) = \frac{1}{\sqrt{2\pi}y\sigma} \exp\left(-\frac{\left(\log\left(\frac{y}{\eta}\right)\right)^2}{2\sigma^2}\right);\ y,\eta,\sigma > 0;$
	$E[Y]$:	$m_{LN} = \exp(\eta + \sigma^2/2)$
	$\mathrm{Var}[Y]$:	$s^2_{LN} = \exp[2(\eta+\sigma^2)] - \exp(2\eta+\sigma^2)$
	Mediana:	$\exp(\eta)$
Gamma	Notação:	$Y \sim GA(\alpha, \lambda)$
	fdp:	$f_{GA}(y;\alpha,\lambda) = \frac{1}{\lambda\Gamma(\alpha)} \left(\frac{y}{\lambda}\right)^{\alpha-1} e^{-(y/\lambda)};\ \ y,\alpha,\lambda > 0;$
	$E[Y]$:	$\alpha\lambda$
	$Var[Y]$:	$\alpha\lambda^2$
Weibull	Notação:	$Y \sim WE(a, b)$
	fdp:	$f_{WE}(y;a,b) = \frac{a}{b} \left(\frac{y}{b}\right)^{a-1} e^{-(y/b)^a};\quad y,a,b > 0;$
	$E[Y]$:	$b\Gamma(1+1/a)$
	$Var[Y]$:	$b^2\left[\Gamma(1+2/a) - \Gamma^2(1+1/a)\right]$
Binomial Negativa	Notação:	$Y \sim NB(n, p)$
	fdp:	$f_{NB}(y;n,p) = \frac{\Gamma(y+n)p^n(1-p)^y}{\Gamma(n)\Gamma(y+1)};\quad y,n,p > 0.$
	$E[Y]$:	$n(1-p)/p$
	$Var[Y]$:	$n(1-p)/p^2$

em relação aos quantis dos dados observados, constituindo-se em um método gráfico bastante apropriado para avaliar o ajuste de um modelo probabilístico a dados experimentais (Heyman et al., 1992; Law, 2007). No caso perfeito de correspondência entre o modelo probabilístico e os dados experimentais, o gráfico Q-Q formado é uma linha reta. As figuras 4.5(a)-(d) apresentam os gráficos Q-Q comparando-se as distribuições de Weibull, gamma, lognormal e binomial negativa para a variável $\mathbf{D}^{(3,3,1)}$. As figuras 4.5(e)-(h) apresentam a mesma comparação para a variável $\mathbf{D}^{(4,3,1)}$. Note que em ambos casos os gráficos Q-Q revelam que o melhor modelo descrevendo os dados é o modelo lognormal . Embora não mostrado aqui, resultados similares da análise Q-Q foi obtido para as outras variáveis $\mathbf{D}^{(n,k,u)}$ consideradas nesse experimento.

A figura 4.6 mostra os histogramas das variáveis $\mathbf{V}^{(n,k,u)}$ correspondentes a vários conjuntos $\mathcal{A}_v^{(n,k,u)}$ de autômatos válidos. Em particular, as figuras 4.6(a)-(d) apresentam, respectivamente, os histogramas das variáveis $\mathbf{V}^{(3,3,1)}$, $\mathbf{V}^{(3,4,1)}$, $\mathbf{V}^{(3,5,1)}$ e $\mathbf{V}^{(4,3,1)}$, correspondes a conjuntos de autômatos válidos com um único evento não observável (o evento de falha). As figuras 4.6(e)-(h) apresentam, respectivamente, as variáveis $\mathbf{V}^{(3,3,2)}$, $\mathbf{V}^{(3,4,2)}$, $\mathbf{V}^{(3,4,3)}$ e $\mathbf{V}^{(4,3,2)}$ correspondentes a conjuntos de autôma-

Y é o valor γ tal que $F(\gamma) = P(x \leq \gamma) = q$, para $0 < q < 1$. Caso F possua inversa, é possível escrever $\gamma = F^{-1}(q)$.



(a) Weibull (b) Gamma (c) Lognormal (d) Binomial negativa

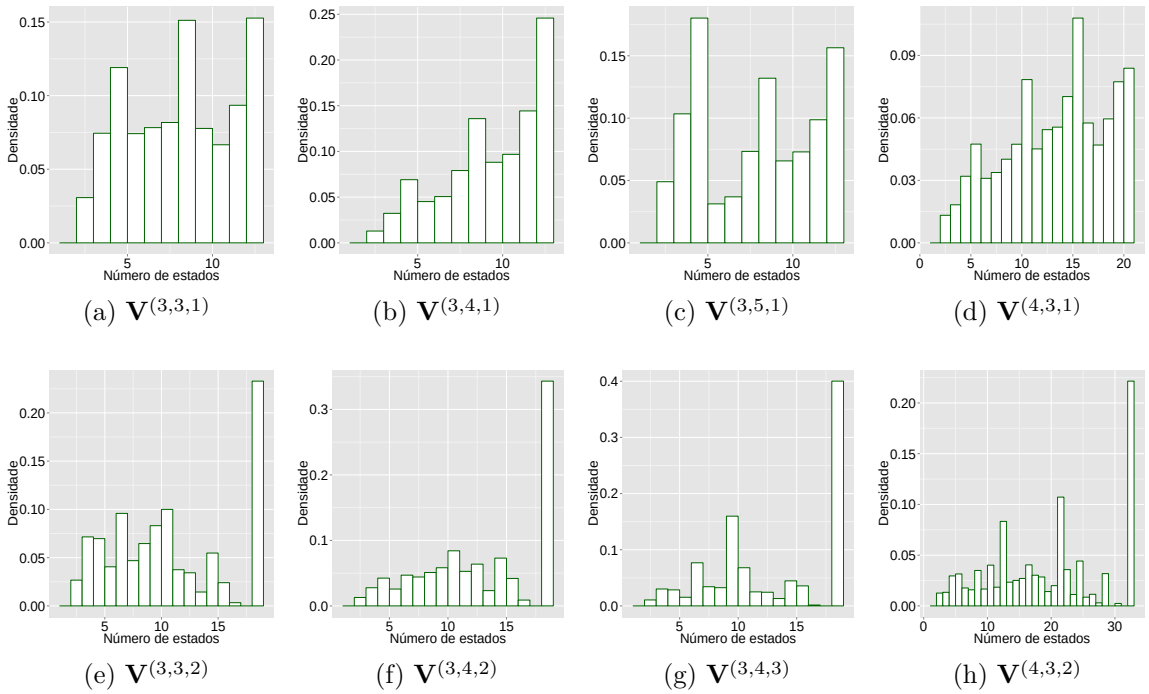
(e) Weibull (f) Gamma (g) Lognormal (h) Binomial negativa

Figura 4.5: Gráfico Q-Q das distribuições de Weibull, gamma, lognormal e binomial negativa para a variável $\mathbf{D}^{(3,3,1)}$ (a)-(d); Gráfico Q-Q das distribuições de Weibull, gamma, lognormal e binomial negativa para a variável $\mathbf{D}^{(4,3,1)}$ (e)-(h).

tos que possuem mais de um evento não observável. Note que o fato de acrescentar um evento não observável altera substancialmente a distribuição da complexidade do verificador que passa a se concentrar no valor de complexidade esperada no pior caso ($2n^2$) conforme pode ser visto nas figuras 4.6(d)-(f). Por exemplo, o histograma da variável $\mathbf{V}^{(3,4,3)}$ (figura 4.6(g)) mostra que uma proporção de aproximadamente 40% dos verificadores calculados para os autômatos do conjunto $\mathcal{A}_v^{(3,4,3)}$ chegam à complexidade limite no pior caso de $2 \times 4^2 = 32$.

A análise exploratória do verificador não evidência uma regularidade estatística que permita propor um modelo paramétrico simples. No entanto, sem necessidade de formular um modelo paramétrico, os resultados da tabela 4.2 e os histogramas da figura 4.6 permitem afirmar que a complexidade media na construção de verificadores para as instâncias consideradas é ainda quadrática ($\mathcal{O}(n^2)$) e, por conseguinte, o limite teórico do pior caso é estreito.

Os resultados obtidos nesta seção, embora exploratórios, sugerem que o modelo lognormal é o modelo probabilístico mais apropriado dentre os modelos considerados para o caso do diagnosticador. Na seção seguinte, vamos a usar critérios quantitativos para mostrar que, dentre os modelos candidatos escolhidos, o modelo lognormal é o melhor modelo para representar as variáveis $\mathbf{D}^{(n,k,u)}, (n,k,u) \in I_E$ representando a complexidade do diagnosticador.



(a) $\mathbf{V}^{(3,3,1)}$ (b) $\mathbf{V}^{(3,4,1)}$ (c) $\mathbf{V}^{(3,5,1)}$ (d) $\mathbf{V}^{(4,3,1)}$

(e) $\mathbf{V}^{(3,3,2)}$ (f) $\mathbf{V}^{(3,4,2)}$ (g) $\mathbf{V}^{(3,4,3)}$ (h) $\mathbf{V}^{(4,3,2)}$

Figura 4.6: Histogramas das variáveis $\mathbf{V}^{(n,k,u)}$.

4.3 Seleção do modelo probabilístico para $\mathbf{D}^{(n,k,u)}$ e estimação de parâmetros do modelo

Conforme ilustrado nos histogramas da figura 4.4 e nos gráficos Q-Q da figura 4.5 as variáveis $\mathbf{D}^{(n,k,u)}$, representando a complexidade do diagnosticador para cada conjunto $\mathcal{A}_v^{(n,k,u)}, (n,k,u) \in I_E$, possuem uma regularidade estatística que permitem propor, como modelos candidatos, as distribuições lognormal, Weibull, gamma e binomial negativa. Nesta seção vamos a apresentar uma forma de se estimar os parâmetros de cada modelo probabilístico definido na tabela 4.3 com base no método de máxima verosimilhança. Começemos pelas seguintes definições:

Definição 4.5. *(Amostra e observação)*

O vetor $\mathbf{Y} = (Y_1, Y_2, \ldots Y_N)$ *em que* $Y_1, Y_2, \ldots, Y_N$ *são variáveis aleatórias independentes e identicamente distribuídas é denominado amostra de tamanho* N.

Suponha que, para um determinado experimento, a amostra $\mathbf{Y} = (Y_1, Y_2, \ldots Y_N)$ *toma o valor específico* $\mathbf{y} = (y_1, y_2 \ldots, y_N)$, *isto é,* $Y_1 = y_1, Y_2 = y_2, \ldots, Y_N = y_N$. *Nesse caso, diz-se que* $\mathbf{y}$ *é uma observação da amostra* $\mathbf{Y}$. □

Definição 4.6. *(Função de verosimilhança) Seja* $\mathbf{Y} = (Y_1, Y_2, \ldots Y_N)$ *uma amostra de tamanho* N *tal que a distribuição das variáveis* $Y_i, i = 1, \ldots, N$ *seja* $f(y;\theta)$, *sendo* $\theta \in R^p$ *um vetor desconhecido de* p *parâmetros. Dada a observação* $\mathbf{y} = (y_1, \ldots, y_N)$,

a seguinte função:

$$L(\theta; (y_1, \ldots, y_N)) = \prod_{i=1}^{N} f(y_i; \theta) \tag{4.7}$$

é chamada função de verosimilhança. □

Observe que na definição da função de verosimilhança da equação (4.7), o vetor $\mathbf{y}$ de observações é considerado constante, enquanto o vetor de parâmetros θ é considerado variável.

Definição 4.7. *A estimativa de máxima verosimilhança (EMV) é o valor* $\hat{\theta} = \hat{\theta}((y_1, y_2, \ldots, y_N))$ *que maximiza* $L(\theta; (y_1, y_2, \ldots, y_N))$*, isto é,*

$$\hat{\theta} = \arg\max_{\theta \in R^p} L(\theta; \mathbf{y}).$$

□

Para encontrar a EMV $\hat{\theta}$, é usual definir a função de log-verosimilhança dada por:

$$l(\theta; \mathbf{y}) = \ln L(\theta; \mathbf{y}). \tag{4.8}$$

Para as distribuições lognormal, gamma, Weibull e binomial negativa (as quais formam o nosso conjunto de modelos candidatos), as funções de log-verossimilhança são definidas, respectivamente, por:

$$l_{LN}(\eta, \sigma; \mathbf{y}) = \sum_{i=1}^{N} \ln f_{LN}(y_i; \eta, \sigma), \tag{4.9a}$$

$$l_{GA}(\alpha, \lambda; \mathbf{y}) = \sum_{i=1}^{N} \ln f_{GA}(y_i; \alpha, \lambda), \tag{4.9b}$$

$$l_{WE}(a, b; \mathbf{y}) = \sum_{i=1}^{N} \ln f_{WE}(y_i; a, b), \tag{4.9c}$$

$$l_{NB}(n, p; \mathbf{y}) = \sum_{i=1}^{N} \ln f_{NB}(y_i; n, p). \tag{4.9d}$$

Substituindo as funções de densidade de probabilidade (e de massa de probabilidade no caso da distribuição binomial negativa) definidas na tabela 4.3 nas respetivas

funções de log-verossimilhança definidas pelas equações (4.9a-d), obtém-se:

$$l_{LN}(\eta, \sigma; \mathbf{y}) = -\left(\frac{N}{2}\ln(2\pi) + \sum_{i=1}^{N}\ln y_i + N\ln\sigma + \frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(\ln\left(y_i/\eta\right)\right)^2\right), \quad (4.10a)$$

$$l_{GA}(\alpha, \lambda; \mathbf{y}) = -\left(N\ln\Gamma(\alpha) + (1-\alpha)\sum_{i=1}^{N}\ln y_i + N\alpha\ln\lambda + \frac{1}{\alpha}\sum_{i=1}^{N} y_i\right), \quad (4.10b)$$

$$l_{WE}(a, b; \mathbf{y}) = N\ln a + (a-1)\sum_{i=1}^{N}\ln y_i - aN\ln b - \sum_{i=1}^{N}(y_i/b)^a, \quad (4.10c)$$

$$l_{NB}(n, p; \mathbf{y}) = Nn\ln(p) - N\ln(\Gamma(n)) + \sum_{i=1}^{N}\ln(\Gamma(y_i+n)) + y_i\ln(1-p) - \ln(\Gamma(y_i+1)). \quad (4.10d)$$

Assim, os parâmetros a serem estimados para cada distribuição candidata são os seguintes[6]:

- *Distribuição lognormal*: as EMV $\hat{\eta}$ e $\hat{\sigma}$ são obtidas diretamente das seguintes equações:

$$\hat{\eta} = \left(\frac{1}{N}\sum_{i=1}^{N}\ln y_i\right), \quad (4.11a)$$

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}[\ln y_i - \ln\hat{\eta}]^2. \quad (4.11b)$$

- *Distribuição gamma*: as EMV $\hat{\lambda}$ e $\hat{\alpha}$ são obtidas solucionando-se numericamente as seguintes equações:

$$\frac{\Gamma'(\hat{\lambda})}{\Gamma(\hat{\lambda})} - \ln\hat{\lambda} = \frac{1}{N}\sum_{i=1}^{N}\ln y_i - \ln\left(\frac{1}{N}\sum_{i=1}^{N} y_i\right), \quad (4.12a)$$

$$\hat{\alpha} = \frac{\hat{\lambda}}{\frac{1}{N}\sum_{i=1}^{N} y_i}. \quad (4.12b)$$

- *Distribuição Weibull:* as EMV $\hat{a}$ e $\hat{b}$ são calculadas solucionando-se numéricamente as seguintes equações:

$$\frac{\sum_{i=1}^{N} y_i^{\hat{a}}\ln y_i}{\sum_{i=1}^{N} y_i^{\hat{a}}} - \frac{1}{\hat{a}} = \frac{1}{N}\sum_{i=1}^{N}\ln y_i, \quad (4.13a)$$

$$\hat{b} = \frac{1}{N}\sum_{i=1}^{N} y_i^k. \quad (4.13b)$$

[6]A dedução desses resultados pode ser obtida diretamente usando os métodos do cálculo. O leitor pode consultar o procedimento algébrico em Aristizabal (2012) para a distribuição lognormal, em Miura (2011) para a distribuição gamma, em Nielsen (2011) para a distribuição Weibull e em Hilbe (2007) para a distribuição binomial negativa.

- *Distribuição binomial negativa:* a EMV $\hat{n}$ deve ser obtida solucionando-se numericamente a equação:

$$N \ln\left(\frac{Nn}{Nn + \sum_{i=1}^{N} y_i}\right) = N\Psi(n) - \sum_{i=1}^{N} \Psi(n + y_i), \tag{4.14a}$$

em que $\Psi(\cdot)$ e a função *digama*, definida como $\Psi(x) = \frac{d}{dx} \ln \Gamma(x)$. A EMV $\hat{p}$ é obtida a partir da seguinte equação:

$$p = \frac{Nn}{Nn + \sum_{i=1}^{N} y_i}. \tag{4.14b}$$

Partindo da função de verosimilhança, é possível definir diferentes critérios para selecionar um modelo probabilístico que melhor represente a amostra $\mathbf{Y}$ dentro de um conjunto de modelos candidatos. O critério que será usado aqui é o critério de informação de Akaike que é definido por:

$$AIC = -2l(\theta; \mathbf{y}) + 2p, \tag{4.15}$$

em que p é o número de parâmetros estimados do modelo. O melhor modelo, segundo o criterio de Akaike, é aquele que possui o menor AIC (Burnham and Anderson, 2002).

Testes de qualidade de ajuste

Uma forma de se verificar se uma observação de uma amostra de tamanho N não é consistente com uma distribuição probabilística adotada como modelo é por meio de um *teste de qualidade de ajuste* (Perkins et al., 2014). Nesse contexto, considere uma amostra $\mathbf{Y} = (Y_1, \ldots, Y_N)$ de tamanho N formada por variáveis aleatórias discretas. Seja $\mathbf{y} = (y_1, \ldots, y_N)$ uma observação de $\mathbf{Y}$. Escolha l intervalos chamados *categorias*, denotados por $[e_{j-1}, e_j)$, em que

$$e_0 < e_1 < \cdots < e_l,$$

e os valores de e_0 e e_l satisfazem

$$e_0 \leq \min_i y_i \text{ e } \max_i y_i \leq e_l.$$

A sequência $e_0, \ldots, e_l$ é chamada sequência de arestas e o intervalo $[e_{j-1}, e_j)$ é chamado j-ésima categoria.

Represente por $\mathbf{p}(\theta) = (p_1(\theta), p_2(\theta), \ldots, p_l(\theta))$ o vetor do modelo adotado em que cada $p_j(\theta) = P(e_{j-1} \leq Y_i < e_j)$, isto é, p_j é a frequência relativa esperada (segundo

o modelo adotado) de observações que ficam na j-ésima categoria. Represente por $\mathbf{q} = (q_1, q_2, \ldots, q_l)$ o vetor de frequências relativas da observação $\mathbf{y} = (y_1, \ldots, y_N)$ em que cada q_j é $1/N$ vezes o número de observações $y_i, i \in \{1, \ldots, N\}$, que ficam na j-ésima categoria. Note que $\sum_{j=1}^{l} p_j(\theta) = \sum_{j=1}^{l} q_j = 1$.

Um teste de qualidade de ajuste produz um valor denominado *nível de significância* que serve para avaliar a consistência estatística da observação $\mathbf{y}$ sob a hipótese de que o modelo $\mathbf{p}(\hat{\theta})$ representa o modelo probabilístico da amostra $\mathbf{Y}$, sendo $\hat{\theta}$ uma estimativa dos parâmetros do modelo (que para o nosso caso é a EMV). O propósito de um teste de qualidade de ajuste é determinar se a discrepância entre o modelo e a observação é maior que as flutuações aleatórias que aparecem devido aos efeitos experimentais.

Existem diferentes estatísticas para se medir a discrepância entre modelos e dados observados, dentre as quais destacam-se: a estatística chi-quadrado (χ^2), Freeman-Tukey, razão de log-verosimilhança, raiz da média quadrática, Kolmogorov-Smirnov e Anderson-Darling[7]. Apesar da estatística χ^2 (que dá lugar ao famoso teste do mesmo nome) ser usualmente adotada como medida de discrepância entre modelos e dados, no trabalho de Perkins et al. (2014) é mostrado que, no caso de variáveis discretas, o teste de qualidade de ajuste χ^2 produz resultados incorretos devido à sua dependência em relação à seleção das categorias. Em contrapartida, a estatística da raiz média quadrática é mais robusta à seleção de categorias. No trabalho de Carruth et al. (2012), recomenda-se usar, adicionalmente, a estatística de Kolmogorov-Smirnov. Levando em consideração essas observações, usaremos como medidas de discrepância, a estatística de *Kolmogorov-Smirnov*, denotada por d_1, a qual é definida pela seguinte equação:

$$d_1(\mathbf{q}, \mathbf{p}(\hat{\theta})) = \max_{k \in \{1,2\ldots,l\}} \left| \sum_{j=1}^{k} q_j - \sum_{j=1}^{k} p_j(\hat{\theta}) \right|; \tag{4.16a}$$

e a estatística da *raiz da média quadrática*, denotada por d_2, a qual é definida pela seguinte equação:

$$d_2(\mathbf{q}, \mathbf{p}(\hat{\theta})) = \sqrt{\frac{1}{l} \sum_{j=1}^{l} \left[q_j - p_j(\hat{\theta}) \right]^2}; \tag{4.16b}$$

O conceito do nível de significância de um teste de qualidade de ajuste surge ao considerar um experimento hipotético no qual é obtida uma observação $\tilde{\mathbf{y}} = (\tilde{y}_1, \ldots, \tilde{y}_N)$ de uma amostra $\tilde{\mathbf{Y}}$ de variáveis aleatórias cuja distribuição probabilística é, efetiva-

[7] Veja uma revisão dessas estatísticas em Huber-Carol (2002) e Steele and Chaseling (2006).

mente, $\mathbf{p}(\hat{\theta})$. Com base nesse experimento, o *nível de significância* α de um teste de qualidade de ajuste é definido pela seguinte probabilidade:

$$\alpha = P\left[d(\tilde{\mathbf{q}}, \mathbf{p}(\tilde{\theta})) \geq d(\mathbf{q}, \mathbf{p}(\hat{\theta}))\right], \tag{4.17}$$

em que

- d representa a estatística considerada para medir a discrepância entre o modelo e a observação, podendo ser a estatística de Kolmogorov-Smirnov (d_1) ou a estatística da raiz da média quadrática (d_2).
- $\hat{\theta}$ é a EMV de θ obtida a partir da observação $\mathbf{y}$.
- $\tilde{\mathbf{q}}$ é o vector de frequências relativas da observação $\tilde{\mathbf{y}} = (\tilde{y}_1, \ldots, \tilde{y}_N)$ cuja distribuição é $\mathbf{p}(\hat{\theta})$.
- $\tilde{\theta}$ é a EMV obtida a partir da observação $\tilde{\mathbf{y}}$ (note que, em geral, $\tilde{\theta} \neq \hat{\theta}$).
- $\mathbf{p}(\tilde{\theta})$ é o vetor de probabilidades calculado a partir do experimento simulado.

É importante ressaltar que ao calcular a probabilidade (4.17), $\tilde{\mathbf{q}}$ e $\tilde{\Theta}$ são consideradas variáveis aleatórias, enquanto $\mathbf{q}$ e $\mathbf{p}(\hat{\theta})$ são considerados valores fixos determinados pela observação $\mathbf{y} = (y_1, \ldots, y_N)$.

Observação 4.1. *É importante ressaltar que o nível de significância deve ser interpretado da seguinte forma: se o valor de α for muito pequeno, podemos ter um nível de confiança $1 - \alpha$ para rejeitar a hipótese de que $\mathbf{p}(\hat{\theta})$ é o modelo que representa a amostra $\mathbf{Y}$.*

Para o cálculo do nível de significância α da equação (4.17), pode ser usada, no caso geral, a simulação de Monte Carlo. Para tal efeito, o algoritmo 4.5 apresenta o método proposto em Clauset et al. (2009) e Carruth et al. (2012) para a determinação do nível de significância α. O erro no cálculo de α por meio do método proposto é $\sqrt{\alpha(1-\alpha)/i_{max}}$ (Perkins et al., 2014), sendo i_{max} o número máximo de iterações de Monte Carlo prefixadas.

É importante realçar que as estatísticas de qualidade de ajuste podem ser usadas para realizar uma comparação do ajuste de diferentes modelos de forma complementar ao critério de Akaike definido na equação (4.15) (Kundu and Manglick, 2005).

Na seção a seguir apresentaremos os resultados de avaliação dos modelos candidatos para as variáveis $\mathbf{D}^{(n,k,u)}$, $(n, k, u) \in I_E$.

Algoritmo 4.5: Método de Monte Carlo para obter o nível de confiança α.

Entradas:

- Observação $\mathbf{y} = (y_1, y_2, \ldots, y_N)$
- Distribuição adotada como modelo.
- Estatística d.
- Sequência de arestas $e_0, \ldots, e_l$.
- Número máximo de simulações i_{max}.

Saída: Nível de significância α.

1. Inicialize $n_\alpha = 0$ e $i = 1$.
2. Calcule a EMV $\hat{\theta}$ para o modelo adotado em relação à observação $\mathbf{y}$. Em seguida obtenha $\mathbf{q}$.
3. Calcule $d(\mathbf{q}, \mathbf{p}(\hat{\theta}))$ com a estatística d escolhida.
4. Gere aleatoriamente a observação $\tilde{\mathbf{y}} = (\tilde{y}_1, \ldots, \tilde{y}_N)$ de acordo com distribuição adotada, tomando como parâmetros da distribuição os valores de $\hat{\theta}$ estimados no passo 2.
5. Calcule a EMV $\tilde{\theta}$ da observação $\tilde{\mathbf{y}}$ obtida por meio da simulação realizada no passo 4.
6. Calcule $d(\tilde{\mathbf{q}}, \mathbf{p}(\tilde{\theta}))$ com a estatística d escolhida.
7. Se $d(\hat{\mathbf{q}}, \mathbf{p}(\tilde{\theta})) \geq d(\mathbf{q}, \mathbf{p}(\hat{\theta}))$, faça $n_\alpha \leftarrow n_\alpha + 1$
8. Faça $i \leftarrow i + 1$. Se o número de simulações de Monte Carlo prefixadas ainda não foi atingido (isto é, se $i < i_{max}$) vá ao passo 4. Caso contrario, estime o nível de significância como $\alpha \approx n_\alpha / i_{max}$.

4.3.1 Resultados de comparação de modelos e teste de bondade de ajuste.

A tabela 4.4 apresenta os resultados comparativos das distribuições lognormal, Gamma, Weibull e Binomial Negativa na modelagem de $\mathbf{D}^{(n,k,u)}$ para $(n, k, u) \in I_E$. Por razões de custo computacional e estabilidade numérica, os cálculos foram realizados tomando observações com amostragem uniforme de tamanho $N = 10000$ para as variáveis $\mathbf{D}^{(n,k,u)}$ cujos comprimentos sejam maiores que 10000. A comparação entre modelos é realizada com base no critério de Akaike, e nas estatísticas de Kolmogorov-Smirnov e da da raiz quadrática média. Por exemplo, o modelo de Weibull para a variável $\mathbf{D}^{(3,4,1)}$, quando avaliado segundo as três figuras de mérito adotadas resultou nos seguintes valores: $AIC = 52882$, $d_1 = 0.0183$ e $d_2 = 0.0285$. Considerando os valores do critério de Akaike e das estadísticas d_1 e d_2, o modelo lognormal possui, entre os 4 modelos candidatos, o melhor ajuste para todas as

Tabela 4.4: Estatísticas e parâmetros estimados para as distribuições lognormal, Weibull, gamma e binomial negativa considerando os conjuntos $\mathcal{A}_v^{n,k,u}$ da tabela 4.2.

		Distribuição			
	Critério	lognormal	Gamma	Weibull	Binomial Neg.
$\mathbf{D}^{(3,3,1)}$	AIC	46892	48100	47059	47778
	d_1	0,0154	0,061	0,0246	0,0522
	d_2	0,0050	0,0801	0,0385	0,0666
$\mathbf{D}^{(3,3,2)}$	AIC	19375	19961	19418	22991
	d_1	0,0245	0,0612	0,0249	0,1935
	d_2	0,0165	0,1252	0,0476	0,2731
$\mathbf{D}^{(3,4,1)}$	AIC	52826	53585	52882	53186
	d_1	0,0228	0,0543	0,0183	0,0346
	d_2	0,0069	0,0605	0,0285	0,0425
$\mathbf{D}^{(3,4,2)}$	AIC	42304	43646	42524	43505
	d_1	0,0100	0,0607	0,0340	0,0657
	d_2	0,0051	0,0975	0,0519	0,0905
$\mathbf{D}^{(3,4,3)}$	AIC	24554	25019	24655	30547
	d_1	0,0183	0,0459	0,0285	0,2336
	d_2	0,0237	0,1012	0,0521	0,3002
$\mathbf{D}^{(3,5,1)}$	AIC	52892	53580	52906	53342
	d_1	0,0295	0,0438	0,0248	0,0417
	d_2	0,0092	0,0684	0,0471	0,0601
$\mathbf{D}^{(4,2,1)}$	AIC	11386	11941	11452	12358
	d_1	0,0115	0,0650	0,0267	0,1140
	d_2	0,0047	0,1191	0,0458	0,1681
$\mathbf{D}^{(4,3,1)}$	AIC	55525	57313	56009	56584
	d_1	0,0099	0,0729	0,0419	0,0585
	d_2	0,0020	0,0770	0,0386	0,0558
$\mathbf{D}^{(4,3,2)}$	AIC	28416	29819	28440	33220
	d_1	0.0180	0.0717	0.0271	0.1574
	d_2	0.0106	0.1284	0.0462	0.2472
$\mathbf{D}^{(5,2,1)}$	AIC	35462	36948	35611	37880
	d_1	0.0095	0.0585	0.0187	0.0914
	d_2	0.0032	0.1024	0.0344	0.1413

variáveis $\mathbf{D}^{(n,k,u)}, (n,k,u) \in I_E$. A segunda melhor avaliação é para a distribuição de Weibull. As distribuições gamma e binomial negativa possuem índices de mérito menores para todas as instâncias. Se considerarmos a abrangência do modelo lognormal (é o melhor para todas as instancias consideradas), sua simplicidade (os parâmetros são estimados facilmente, sem ter que resolver equações não lineares como (4.12a), (4.13a) ou (4.14a)) está justificado adotar a distribuição lognormal como modelo probabilístico da complexidade do diagnosticador nas instâncias estudadas.

Sob a hipótese de que as variáveis que representam a complexidade do diag-

nosticador possuem distribuição lognormal, foi realizado um teste de qualidade de ajuste para cada $\mathbf{D}^{(n,k,u)}, (n,k,u) \in I_E$ usando, para tal efeito, o algoritmo 4.5 com as seguintes entradas:

- Observação de entrada: a própria variável $\mathbf{D}^{(n,k,u)}$ se seu comprimento for menor a 10000. Caso contrário é usada uma observação de $\mathbf{D}^{(n,k,u)}$ com amostragem uniforme de tamanho $N = 10000$.
- Estatísticas: Kolmogorov-Smirnov e da raiz da média quadrática.
- Número máximo de simulações de Monte Carlo: $i_{max} = 10^6$.
- Sequência de arestas: $e_j = j + 0.5, j = 0, \ldots, l-1$, sendo $l = \max_i \mathbf{D}^{(n,k,u)} + 1$, e $e_l = \infty$.

A tabela 4.5 mostra os resultados do teste de qualidade de ajuste do modelo lognormal e, como informação complementar, as estimativas dos parâmetros e dos momentos estatísticos de primeira e segunda ordem. As colunas 2 e 3 mostram o

Tabela 4.5: Nível de significância do teste de qualidade de ajuste para as variáveis as variáveis $\mathbf{D}^{(n,k,u)}$, $(n,k,u) \in I_E$, calculado usando o algoritmo 4.5 para um modelo lognormal.

	Nível de significância α		Parâmetros estimados		Momentos estimados		Momentos exatos	
$\mathbf{D}^{(n,k,u)}$	com d_1	com d_2	$\hat{\eta}$	$\hat{\sigma}$	$\hat{M}_{LN}$	$\hat{S}^2_{LN}$	m_{LN}	s^2_{LN}
$(3,3,1)$	0.699	0.786	1.67	0.47	5.9	8.7	5.9	8.4
$(3,3,2)$	0.997	0.607	0.92	0.37	2.7	1.0	2.7	0.9
$(3,4,1)$	0.901	0.890	2.03	0.45	8.4	15.8	8.4	14.1
$(3,4,2)$	0.694	0.920	1.47	0.47	4.9	5.5	4.8	5.5
$(3,4,3)$	0.688	0.952	0.83	0.37	2.5	0.8	2.4	0.7
$(3,5,1)$	0.991	0.880	1.92	0.50	7.7	16.5	7.7	14.6
$(4,2,1)$	0.833	0.785	1.20	0.39	3.6	2.1	3.6	2.1
$(4,3,1)$	0.857	0.440	2.04	0.50	8.7	22.7	8.7	22.5
$(4,3,2)$	0.709	0.483	1.04	0.35	3.0	1.2	3.0	1.2
$(5,2,1)$	0.593	0.543	1.30	0.38	4.0	2.6	4.0	2.5

níveis de significância calculados com as estatísticas de Kolmogorov-Smirnov (d_1) e da raiz média quadrática (d_2), respectivamente. Por exemplo, o nível de significância do teste de qualidade de ajuste para a variável $\mathbf{D}^{(3,4,2)}$ com a estatística d_1 é de 0.694 e assume o valor de 0.920 quando calculado com a estatística d_2. As colunas 4 e 5 mostram, respectivamente, as EMV dos parâmetros $\hat{\eta}$ e $\hat{\sigma}$ estimados usando as equações (4.11a,b). Por exemplo, as EMV dos parâmetros do modelo lognormal

associado à variável $\mathbf{D}^{(3,4,1)}$ são $\hat{\eta} = 2.03$ e $\hat{\sigma} = 0.45$. A coluna 6 mostra a estimativa do valor esperado, calculada usando a equação:

$$\hat{M}_{LN} = \exp(\hat{\eta} + \hat{\sigma}^2/2),$$

e a coluna 7 a estimativa da variância, calculada usando a equação:

$$\hat{S}^2_{LN} = \exp[2(\hat{\eta} + \hat{\sigma}^2)] - \exp(2\hat{\eta} + \hat{\sigma}^2).$$

Note que essas são as fórmulas que aparecem na tabela 4.3 substituindo-se os parâmetros η e σ pelas suas EMV. As colunas 8 e 9 mostram, novamente, os valores exatos do valor esperado e da variância para propósito de comparação com os valores estimados das colunas 7 e 8. Por exemplo, para a variável $\mathbf{D}^{(4,2,1)}$, $\hat{M}_{LN} = m_{LN} = 3.6$ e, por sua vez, $\hat{S}^2_{LN} = s^2_{LN} = 2.1$.

Podemos, então, sumarizar os resultados mais importantes reportados na tabela 4.5:

- Os valores de α exibidos nas colunas 2 e 3 indicam que a distribuição lognormal é um modelo consistente para a complexidade computacional do diagnosticador nas instâncias consideradas.
- O modelo lognormal permite estimar com bastante exatidão os momentos estatísticos de primeira e segunda ordem de cada variável $\mathbf{D}^{(n,k,u)} \in I_E$, conforme pode ser conferido nas colunas 6 e 8 para o valor esperado e nas colunas 7 e 9 para a variância.

4.4 Implicações da lognormalidade: exemplo

A seguir será apresentado um exemplo numérico para ilustrar algumas consequências que aparecem da adoção da distribuição lognormal como modelo da complexidade na construção do diagnosticador.

Considere um experimento no qual seja selecionado aleatoriamente um autômato do conjunto $\mathcal{A}_v^{(n,k,u)}$, $(n,k,u) \in I_E$ e, por sua vez, represente por $D^{(n,k,u)}$ a variável aleatória que resulta ao calcular a cardinalidade do conjunto de estados do diagnosticador construído para esse autômato. Conforme foi mostrado na seção anterior, a variável aleatória $D^{(n,k,u)}$ pode ser modelada, com bastante aproximação, pela distribuição lognormal com os parâmetros de ajuste da tabela 4.5.

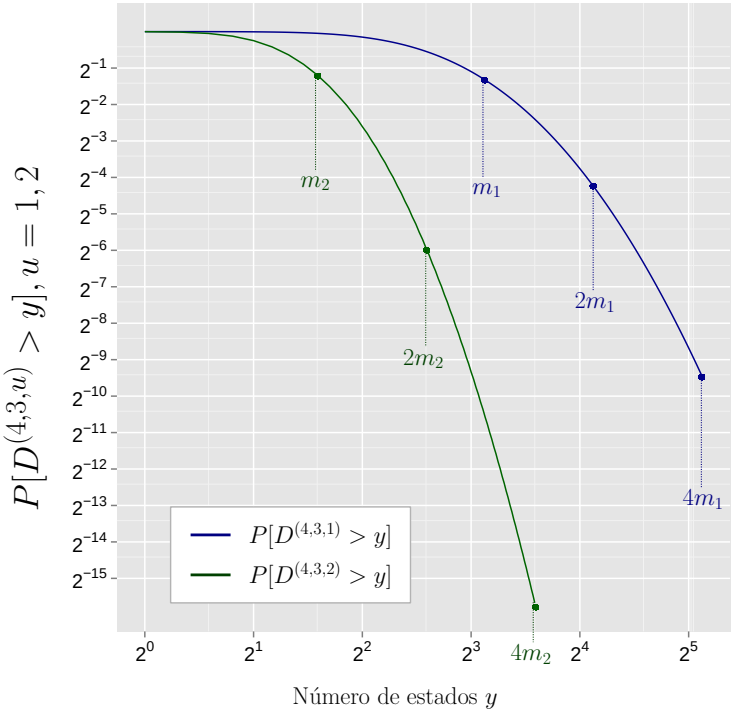


Figura 4.7: Funções de distribuição cumulativa complementar dos modelos de distribuição lognormal das variáveis aleatórias $D^{(4,3,1)}$ e $D^{(4,3,2)}$.

Nesse contexto, considere as variáveis aleatórias $D^{(4,3,1)} \sim LN(2.04, 0.50)$ e $D^{(4,3,2)} \sim LN(1.04, 0.35)$ com valores esperados $m_1 = 9$ e $m_2 = 3$, respectivamente. A figura 4.7 mostra as funções de distribuição cumulativa complementar de ambos modelos usando uma escala logarítmica de oitavas para ambos eixos. A curva em azul da figura 4.7 representa a probabilidade $P[D^{(4,3,1)} > y]$, enquanto a curva verde representa a probabilidade $P[D^{(4,3,2)} > y]$, sendo y o número de estados da abscissa.

Tabela 4.6: Cálculos de $P[D^{(4,3,u)} > d \cdot m_u]$ e $P[D^{(4,3,u)} \leq d \cdot m_u]$, $u = 1, 2$.

(a)

d	$P[D^{(4,3,1)} > d \cdot m_1]$	$P[D^{(4,3,1)} \leq d \cdot m_1]$
1	0.3769	62.30%
2^1	$0.3769 \times 2^{-3.02}$	95.35%
2^2	$0.3769 \times 2^{-8.34}$	99.88%
2^3	$0.3769 \times 2^{-16.17}$	99.999%

(b)

d	$P[D^{(4,3,2)} > d \cdot m_2]$	$P[D^{(4,3,2)} \leq d \cdot m_2]$
1	0.4382	56.17%
2^1	$0.4382 \times 2^{-4.74}$	98.36%
2^2	$0.4382 \times 2^{-14.48}$	99.99%
2^3	$0.3769 \times 2^{-16.17}$	99.999%

A coluna 2 da tabela 4.6(a) mostra os valores da probabilidade $P[D^{(4,3,1)} > d \cdot m_1]$, $d = 1, 2, 4, 8$ ressaltados na curva azul da figura 4.7. Note que ao duplicar o valor de $2m_1$, a probabilidade decresce num fator aproximado de 2^{-3}, ao quadruplicar m_1 desce num fator de 2^{-8}, e ao octuplicar m_1 a probabilidade desce num fator 2^{-16}. A coluna 3 da tabela 4.6(a) apresenta a probabilidade complementar $P[D^{(4,3,1)} \leq d \cdot m_1]$, $d = 1, 2, 4$ para evidenciar o fato de que 62.3% dos diagnosticadores associados

com autômatos de $\mathcal{A}_v^{(4,3,1)}$ possuem menos de $m_1 = 9$ estados, 95.35% menos de $2m_1 = 18$ estados, 99.88 menos de $4m_1 = 36$ estados e, praticamente, é certo que possuem menos de $8m_1 = 72$ estados (o valor real é 53). Note que, dado que o valor esperado é $m_1 \approx 2^3$ e que em $2^3 \times 2^2 = 2^5$ já estão incluídas praticamente todas as instâncias, então o limite teórico de complexidade do pior caso (2^8) é improvável.

A coluna 2 da tabela 4.6(b) mostra os valores da probabilidade $P[D^{(4,3,2)} > d{\cdot}m_2]$, $d = 1, 2, 4, 8$ ressaltados na curva verde da figura 4.7. Observe que ao duplicar o valor de m_2 a probabilidade decresce num fator aproximado de 2^{-5} , ao quadruplicar m_2 a probabilidade desce num fator de 2^{-14} e ao octuplicar a probabilidade desce num factor de 2^{-29} (do ponto de vista computacional é uma probabilidade 0). A coluna 3 da tabela 4.6(b) apresenta a probabilidade complementar $P[D^{(4,3,2)} \leq d \cdot m_2]$, $d = 1, 2, 4$ para ilustrar o fato de que 56.17% dos diagnosticadores associados com autômatos de $\mathcal{A}_v^{(4,3,2)}$ possuem menos de $m_2 = 3$ estados, 98.36% menos de $2m_2 = 6$ estados, 99.99 menos de $4m_2 = 12$ estados e é probabilisticamente certo que possuem menos de $8m_2 = 24$ estados (o valor real é 16). Note que, dado que o valor esperado é $m_2 \approx 2^2$ e que em $2^2 \times 2^2 = 2^4$ já estão incluídas todas as instâncias, então o limite teórico de complexidade do pior caso (2^8) é improvável.

Em síntese, é possível dizer que a adoção de um modelo lognormal implica que um aumento linear em relação ao logaritmo do número de estados esperados implica numa descida em uma proporção aproximadamente cúbica no logaritmo da probabilidade. Por essa razão, o limite teórico no cálculo do diagnosticador dado por $\mathcal{O}(2^{2n})$, é praticamente improvável.

4.5 Comentários finais

O objetivo primordial deste capítulo foi realizar um estudo preliminar da complexidade média do diagnosticador e do verificador por meio de um experimento exaustivo realizado sobre os conjuntos $\mathcal{A}^{(n,k,u)}, (n, k, u) \in I_E$. Os resultados mais significativos que foram obtidos nesse experimento são os seguintes:

- O modelo probabilístico lognormal é uma hipótese consistente para explicar a complexidade na construção do diagnosticador para os conjuntos de autômatos válidos estudados.
- O limite no pior caso $\mathcal{O}(2^{2n})$ para a complexidade na construção do diagnosticador é, sob adoção do modelo lognormal, pouco provável.
- Embora não seja fácil formular um modelo paramétrico simples para a complexidade do verificador, os resultados sugerem que o valor da complexidade média na construção do verificador é da mesma ordem de grandeza que o limite especificado para a complexidade do pior caso, isto é, $\mathcal{O}(n^2)$ também é

um limite apropriado para a complexidade média na construção do verificador.

As conclusões anteriores estão limitadas a um pequeno espaço de autômatos representados pelos conjuntos $\mathcal{A}_v^{(n,k,u)}, (n,k,u) \in I_E$. Por conseguinte, para estender esse estudo, e obter resultados que realmente sejam de importância prática do ponto de vista da diagnose de falhas em SEDs, é necessário considerar conjuntos maiores de autômatos accessíveis $\mathcal{A}^{(n,k)}$ e formular hipóteses quantitativas em relação ao comportamento dos valores esperados da complexidade do diagnosticador e do verificador. Levando em consideração os resultados dos teoremas 4.3 e 4.4, a explosão combinatória dificulta estender o teste exaustivo para conjuntos de autômatos acessíveis $\mathcal{A}^{(n,k)}$ de maior cardinalidade. O teste exaustivo poderia ser estendido um pouco mais usando um supercomputador e, simultaneamente, programas bem "sintonizados" em uma linguagem compilada; no entanto, a abrangência desse método está severamente limitada. Considere que, por exemplo, uma melhora computacional de ordem de 10^6 (por exemplo, um programa 1000 vezes mais rápido rodando em paralelo em 1000 computadores) é insignificante para testar o conjunto $\mathcal{A}^{(10,5)}$ cuja cardinalidade é aproximadamente 2.6×10^{47}. Nesse ponto requeremos uma nova abordagem para estender o método experimental usando amostragem. Essa abordagem será usada no capitulo seguinte.

Capítulo 5

Análise experimental da complexidade média em diagnose de falhas usando amostragem

Os resultados preliminares obtidos no capitulo anterior sugerem que os autômatos que produzem a complexidade no pior caso (de ordem $\mathcal{O}(2^{2n})$) do diagnosticador proposto por Sampath et al. (1995) aparecem como instâncias com pouca probabilidade e que, por outro lado, as complexidades no pior caso e média na construção do verificador proposto por Moreira et al. (2011a) são da mesma ordem ($\mathcal{O}(n^2)$). O propósito fundamental deste capítulo é estender esses resultados para conjuntos maiores de autômatos válidos formulando, assim, modelos experimentais do crescimento das complexidades médias na construção de verificadores e diagnosticadores de acordo com os métodos de análise experimental de algoritmos apresentados por McGeoch (2012), McGeoch et al. (2002) e Coffin and Saltzman (2000). O diagrama de blocos da figura 4.1 descreve, de forma geral, a análise experimental com amostragem que será realizada nesse capítulo. O primeiro passo é a obtenção de instâncias válidas por meio do gerador de autômatos determinísticos com distribuição uniforme (desenvolvido nos trabalhos de Bassino and Nicaud (2006, 2007), Bassino et al. (2008, 2009) Héam et al. (2010)) junto com um algoritmo de rejeição. O segundo passo é realizar o cálculo do diagnosticador e do verificador para cada *autômato válido* gerado. No terceiro passo, os dados experimentais são usados para obter uma estimativa do valor esperado das complexidades do diagnosticador e do verificador. Com base nessas estimativas, é ajustado um modelo de regressão para obter os modelos experimentais do crescimento assintótico dos valores esperados da complexidade do diagnosticador e do verificador.

Este capítulo está dividido da seguinte forma. Na seção 5.1 é apresentado o gerador de autômatos determinísticos com distribuição uniforme que será usado

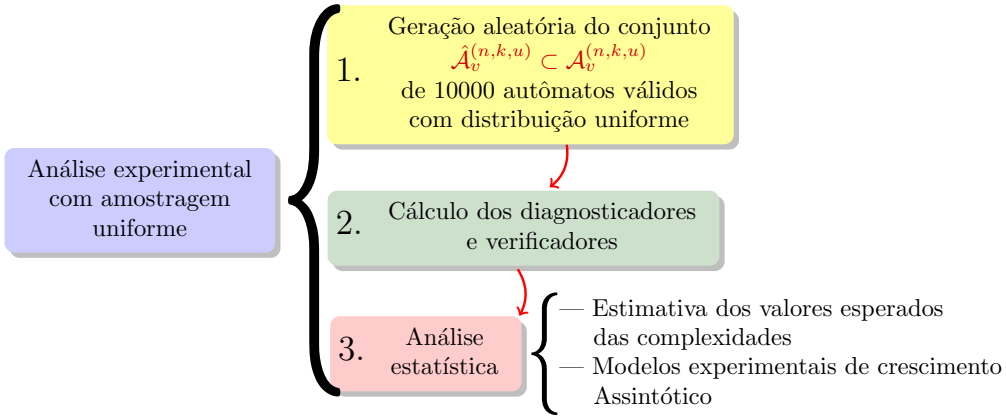


Figura 5.1: Diagrama de blocos do experimento exaustivo

para gerar as instâncias aleatórias do experimento com amostragem proposto neste capítulo. No começo da seção 5.2 são discutidas as razões da seleção do tamanho amostral. Em seguida, na mesma seção, é descrito o experimento realizado e, posteriormente, é realizada uma análise dos resultados a nível exploratório. Partindo da análise dos dados obtidos, na seção 5.3 são propostos modelos hipotéticos para caracterizar o crescimento assintótico da complexidade média na construção de diagnosticadores e verificadores. Finalmente, na seção 5.4 são apresentados os comentários finais.

5.1 Gerador uniforme de autômatos aleatórios

A seguir será apresentado o método de geração uniforme de autômatos conforme desenvolvido nos trabalhos de Bassino and Nicaud (2006, 2007), Bassino et al. (2008, 2009) e Héam et al. (2010).

5.1.1 Autômato representativo

Seja $G = (X, \Sigma, f, \Gamma, x_0)$ um autômato acessível tal que $|X| = n$ e $|\Sigma| = k$, sendo o alfabeto $\Sigma = \{\sigma_1 < \cdots < \sigma_k\}$ completamente ordenado. Seja $\psi : X \to \Sigma^*$ um mapeamento definido para cada $x \in X$ por

$$\psi(x) = \min_{lex} \{s \in \Sigma^* : f^*(x_0, s) = x \text{ e } s \text{ é um caminho simples em } G\}, \qquad (5.1)$$

em que o mínimo está definido conforme à *ordem lexicográfica*[1] $<_{lex}$. Note que $\psi(x)$ está definido para todo $x \in X$ pelo fato de G ser acessível. Note, também, que se

[1] A ordem lexicográfica é a ordem usada em um dicionário. Formalmente, seja $\Sigma = \{\sigma_1 < \cdots < \sigma_k\}$ um alfabeto ordenado. A ordem lexicográfica, denotada por $<_{lex}$, é a ordem definida como $s <_{lex} t$ se s é prefixo de t, ou, se $s = p\sigma_1 u'$ e $t = p\sigma_2 t'$ para algum $p \in \Sigma^*$ e $\sigma_1, \sigma_2 \in \Sigma$ com $\sigma_1 < \sigma_2$.

$x_1, x_2 \in X$, sendo $x_1 \neq x_2$, então $\psi(x_1) \neq \psi(x_2)$ pelo fato de G' ser determinístico. Assim, o mapeamento $\psi' : X \to X'$ em que $X' = \{\psi(x) : x \in X\}$ é uma bijeção que permite definir o isomorfismo $\psi' : G \to G'$ em que $G' = (X', \Sigma, f', \Gamma', \varepsilon)$. O autômato G' resultante é chamado *autômato representativo.*

Exemplo 5.1. *Considere o autômato G representado na figura 5.2(a), no qual o caminho mínimo na ordem lexicográfica para atingir o estado x_4 é aa, isto é, $\psi(x_4) = aa$ de acordo com a equação* (5.1)*. Observe que as transições em vermelho no diagrama de transição de estados de G da figura 5.2 servem para ilustrar graficamente os caminhos simples mínimos na ordem lexicográfica para atingir cada estado $x \in X$. A aplicação de ψ sobre cada $x \in X$ dá lugar à bijeção $\psi' : X \to X'$ ($X' = \{\varepsilon, a, aa, aab, aabc\}$) dada por:*

$$\psi' = \{(x_0, \varepsilon), (x_1, aabc), (x_2, aab), (x_3, a), (x_4, aa)\},$$

a qual permite definir o autômato representativo G', ilustrado na figura 5.2, tal que $\psi' : G \to G'$. ❑

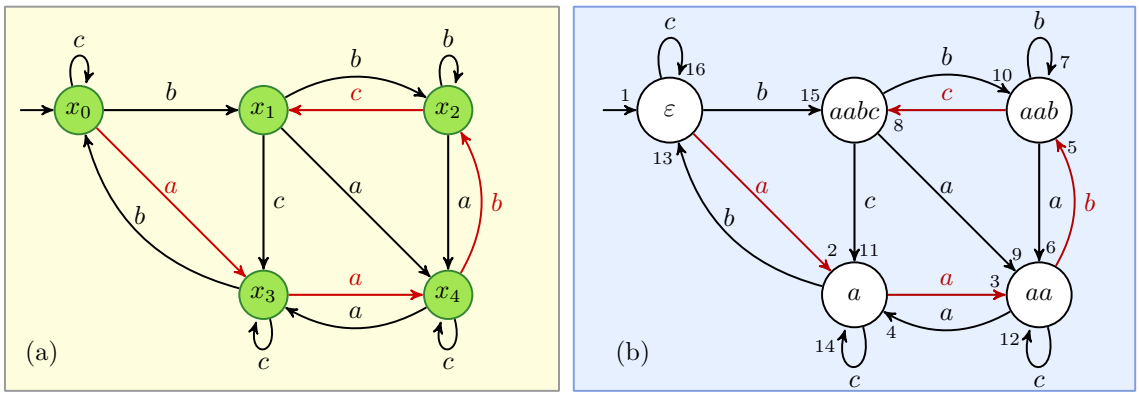


Figura 5.2: Autômato $G = (X, \Sigma, f, \Gamma, x_0)$ (a); autômato $G' = (X', \Sigma, f', \Gamma', \varepsilon)$ isomorfo com G. As transições em vermelho formam a árvore de extensão resultante da busca em profundidade em ordem lexicográfica.

Observação 5.1. *Note que, do ponto de vista algorítmico, o mapeamento ψ' é obtido realizando-se uma busca em profundidade (na ordem de Σ) no autômato G, associando cada estado $x \in X$ com a sequência s da árvore de extensão (representada pelas transições vermelhas na figura 5.2(a)) tal que $f^*(x_0, s) = x$.* □

5.1.2 Bijeção entre autômatos e partições de conjuntos

A seguir vamos a apresentar a bijeção existente entre um conjunto $\mathcal{C}^{(n,k)}$ de autômatos completos com n estados e k eventos e um subconjunto da classe de partições do conjunto $I_{1:kn+1}$. Comecemos pela seguinte definição:

Definição 5.1. *(**Partição Dick-k**) Represente por $\mathcal{Q}^{(p,n)}$ a classe das partições do*

conjunto $I_{1:p}$ em n blocos. Seja $Q = \{Q_0, \ldots, Q_{n-1}\}$ um elemento da classe $\mathcal{Q}^{(p,n)}$ em que os elementos de Q estão ordenados de acordo com seu elemento mínimo. Dado $k \geq 2$, uma partição $Q \in \mathcal{Q}^{(p,n)}$ é uma partição Dick-k *se*

$$\min Q_j \leq kj + 1, j = 0, 1, 2, \ldots, n-1. \tag{5.2}$$

□

Exemplo 5.2. *Considere a partição $Q = \{Q_0, Q_1, Q_2\}$ do conjunto $I_{1:10}$ em que $Q_0 = \{1, 4, 5, 7\}, Q_1 = \{2, 6, 8, 9\}$, e $Q_2 = \{3, 10\}$. Note que $Q \in \mathcal{Q}^{(10,3)}$, isto é, Q é uma das partições de $I_{1:10}$ em $n = 3$ subconjuntos não vazios. Suponha que $k = 2$ na condição* (5.2)*. Então, para cada Q_j tem-se:*

$$\begin{aligned} \min Q_0 = 1 &\leq 2 \times 0 + 1 = 1; \\ \min Q_1 = 2 &\leq 2 \times 1 + 1 = 3; \\ \min Q_2 = 3 &\leq 2 \times 2 + 1 = 5. \end{aligned}$$

e, em consequência, Q é uma partição Dick-2. ❑

Seja, agora, $G'_c = (X', \Sigma, f', \Gamma', \varepsilon) \in \mathcal{C}^{(n,k)}$ um autômato representativo completo e seja

$$T = \{(x', \sigma, f(x', \sigma)) : x' \in X', \sigma \in \Sigma\} \cup \{(\emptyset, \varepsilon, \varepsilon)\}$$

o conjunto de todas as transições de G'_c, incluindo a transição adicional $(\emptyset, \varepsilon, \varepsilon)$ (que pode ser interpretada como a seta que marca o estado inicial na representação gráfica de G'). Note que como G'_c é completo, então, $|T| = kn + 1$. É possível estabelecer uma bijeção de ordem $\nu : T \to I_{1:kn+1}$ da seguinte forma:

B1. $\nu((\emptyset, \varepsilon, \varepsilon)) = 1$,

B2. $\nu((\varepsilon, \sigma_1, \sigma_1)) = 2$,

B3. Para cada par de transições (r, σ_a, s) e (t, σ_b, u) em $T \backslash \{(\emptyset, \varepsilon, \varepsilon)\}$, $\nu((r, \sigma_a, s)) < \nu((t, \sigma_b, u))$ se e somente se $r\sigma_a <_{lex} t\sigma_b$.

Para exemplificar a ordem das transições definida acima, vamos usar novamente o autômato representativo mostrado na figura 5.2(b). Observe que a seta do estado inicial é enumerada por 1 (regra **B1**) e que $\nu((\varepsilon, a, a)) = 2$ (regra **B2**). Note que $\nu((aab, a, aa)) = 6 < \nu((aab, b, aab)) = 7$ sendo que $aaba <_{lex} aabb$, em conformidade com **B3**. Não é difícil verificar que a ordem das transições definida por ν é única e corresponde à ordem em que são percorridas as transições de G'_c em um algoritmo de busca em profundidade, seguindo uma ordem lexicográfica.

Vamos, agora, mostrar a forma como o autômato representativo completo $G'_c \in \mathcal{C}^{(n,k)}$ pode ser representado por uma partição do conjunto $I_{1:kn+1}$ em n conjuntos não vazios. Para tanto, denote por $Q_D = \{Q_0, \ldots, Q_{n-1}\}$ a partição construída a partir de G'_c usando as seguintes regras:

C1. $1 \in Q_0$

C2. Se i e j representam, respectivamente, as enumerações de um par de transições do automato G'_c, isto é, se $i = \nu((r, \sigma_a, s))$ e $j = \nu((t, \sigma_b, u))$ sendo que $(r, \sigma_a, s), (t, \sigma_b, u) \in T \setminus \{(\emptyset, \varepsilon, \varepsilon)\}$, então i, j pertencem ao mesmo elemento de Q_D se e somente se $s = u$.

O seguinte resultado estabelece a bijeção existente entre conjuntos de autômatos completos e partições Dick-k.

Teorema 5.1. *(Bassino et al., 2009) Seja Ω o mapeamento entre o conjunto $\mathcal{C}^{(n,k)}$ de autômatos completos com n estados e k eventos e o conjunto $\mathcal{Q}^{(kn+1,n)}$ em que $Q_D = \Omega(G'_c)$ é a partição construída de acordo com as regras* **C1** *e* **C2**. *Para cada $n \geq 1$ e $k \geq 2$, o mapeamento Ω que transforma G'_c em Q_D é uma bijeção entre o conjunto de autômatos completos $\mathcal{C}^{(n,k)}$ e o conjunto de partições* Dick-k *contido em $\mathcal{Q}^{(kn+1,n)}$.* □

Com propósito ilustrativo, vamos construir a partição associada com o autômato da figura 5.2. A regra **C1** implica que Q_D é da forma $\{\{1, \ldots\}, Q_2, \ldots, Q_{n-1}\}$. A regra **C2** implica que os números de ordem (obtidos usando o mapeamento ν) que correspondem às transições que chegam ao mesmo estado ficam no mesmo elemento de Q_D. Assim, usando a enumeração de cada transição conforme aparece na figura 5.2, obtém-se a seguinte partição Dick-3 associada ao autômato:

$$Q_D = \{\{1, 16, 13\}, \{2, 4, 11, 14\}, \{3, 6, 9, 12\}, \{5, 7, 10\}, \{8, 15\}\}.$$

5.1.3 Bijeção entre autômatos acessíveis e completos

Com vistas a gerar uniformemente autômatos acessíveis (possivelmente incompletos) é necessário definir uma transformação entre autômatos acessíveis e autômatos completos. Para tanto, seja $A = (X_A, \Sigma, f_A, \Gamma_A, \varepsilon) \in \mathcal{A}^{(n,k)}$ um autômato representativo acessível. Seja $\phi : \mathcal{A}^{(n,k)} \to \mathcal{C}^{(n+1,k)}$ a função que associa o autômato A com o autômato $\phi(A) = (X_\phi, \Sigma, f_\phi, \Gamma_\phi, \varepsilon) \in C^{(n+1,k)}$, sendo $X_\phi = \{\sigma_k x_a : x_a \in X_A\} \cup \{\varepsilon\}$,

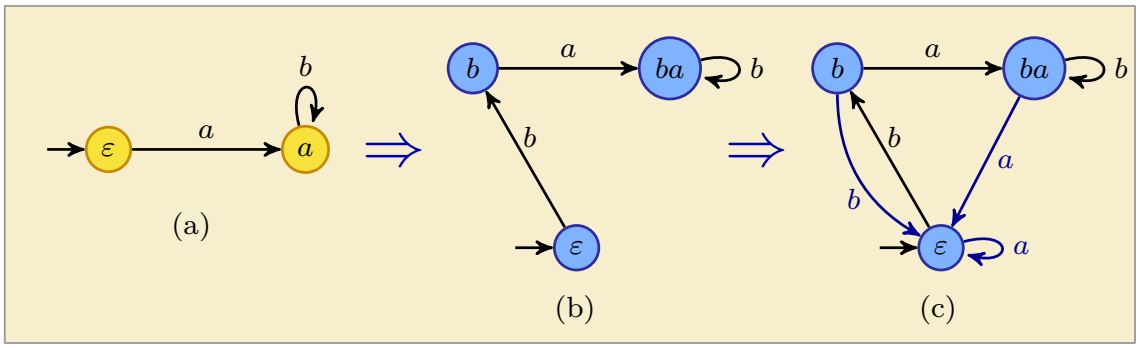


Figura 5.3: Autômato $A \in \mathcal{A}^{(n,k)}$ (a); passo inicial da transformação de A em um autômato completo (b); autômato completo $\phi(A)$

$\sigma_k = \max(\Sigma)$, e a função de transição de estados f_ϕ definida da seguinte forma:

- $f_\phi(\varepsilon, \sigma) = \begin{cases} \varepsilon, & \sigma \neq \sigma_k, \\ \sigma_k, & \sigma = \sigma_k. \end{cases}$
- $f_\phi(x_\phi, \sigma) = \begin{cases} \sigma_k f_A(x_A, \sigma), & \text{se } (\exists x_A \in \Sigma^*)[(x_\phi = \sigma_k x_A) \wedge (f_A(x_A, \sigma) \neq \emptyset)], \\ \varepsilon, & \text{se } (\exists x_A \in \Sigma^*)[(x_\phi = \sigma_k x_A) \wedge (f_A(x_A, \sigma) = \emptyset)]. \end{cases}$

Para ilustrar a construção do autômato $\phi(A)$ considere o autômato A da figura 5.3(a), supondo $\Sigma = \{a, b\}$ com $a < b$. A construção do autômato $\phi(A)$ pode ser descrita em dois passos conforme será ilustrado tomando como exemplo o autômato A da figura 5.3(a). O primeiro passo é renomear os estados do autômato A da forma $\{\sigma_k x_a : x_a \in X_A\} = \{b, ba\}$ e adicionar o estado rotulado por ε (que será o novo estado inicial) para obter o conjunto de estados $X_\phi = \{epsilon, b, ba\}$. Em seguida adicionada a transição $(\varepsilon, \sigma_k, \sigma_k) = (\varepsilon, b, b)$ conforme mostrado na figura 5.3(b). O segundo passo é completar o autômato acrescentando transições da forma $(x_\phi, \sigma, \varepsilon)$ até completar o autômato $\phi(A)$ representado na figura 5.3(c).

O seguinte resultado justifica a geração de autômatos acessíveis em $\mathcal{A}^{(n,k)}$ usando autômatos completos em $C^{(n+1,k)}$.

Lema 5.1. *(Bassino et al., 2009) Seja $\mathcal{E}^{(n+1,k)}$ o subconjunto de $\mathcal{C}^{(n+1,k)}$ formado por autômatos representativos completos $E = (X_E, \Sigma, f_E, \Gamma_E, \varepsilon)$ tais que $f_E(\varepsilon, \sigma) = \varepsilon$ para $\sigma \in \Sigma \setminus \{\sigma_k\}$. A função ϕ é uma bijeção de $\mathcal{A}^{(n,k)}$ em $\mathcal{E}^{(n+1,k)}$.* □

Observação 5.2. *Note que se $E \in \mathcal{E}^{(n+1,k)}$, o procedimento para obter $\phi^{-1}(E)$ é simplesmente apagar o estado inicial de E junto com suas transições definindo como novo estado inicial o segundo estado da ordem lexicográfica.* □

A discussão precedente pode ser resumida em dois fatos relevantes: *(i)* o lema 5.1 transforma o problema de se gerar autômatos acessíveis de n estados no problema de gerar autômatos completos com $n + 1$ estados; *(ii)* o teorema 5.1 converte o problema de se gerar autômatos completos de $n+1$ estados no problema

de se gerar partições de um conjunto em $n+1$ subconjuntos não vazios. A seguir vamos abordar esse último problema.

5.1.4 Geração uniforme de partições de um conjunto

Para resolver o problema de gerar uma partição de um conjunto em $n+1$ conjuntos não vazios é necessário introduzir alguns conceitos de combinatória analítica[2]. Começemos pela seguinte definição.

Definição 5.2. *(Classe combinatória rotulada) Uma classe combinatória rotulada $\mathcal{F}$ é um conjunto enumerável de objetos satisfazendo as seguintes propriedades:*

i. Possui uma função de tamanho $\|.\| : \mathcal{F} \to \mathbb{Z}^+$ tal que para cada elemento $\gamma \in \mathcal{F}$, seu tamanho $\|\gamma\|$ é um inteiro não negativo;

ii. Todo conjunto $\mathcal{F}_m = \{\gamma \in \mathcal{F} : \|\gamma\| = m\}$, formado por elementos de tamanho m, é finito;

iii. Todo objeto $\gamma \in \mathcal{F}$ é rotulado *no seguinte sentido: se um objeto é de tamanho m, então possui m rótulos diferentes no conjunto $I_{1:\|\gamma\|}$.*

□

Definição 5.3. *(Sequência de contagem) A sequência de contagem de uma classe combinatória $\mathcal{F}$ é uma sequência de inteiros $(f_m)_{m\geq 0}$ em que f_m é a cardinalidade de cada conjunto $\mathcal{F}_m$, isto é, $f_m = |\mathcal{F}_m|, m \geq 0$.* □

A classe $\mathcal{F}$ pode ser estudada, para propósitos de contagem e análise probabilística, associando-a com uma função geradora exponencial que será definida a seguir.

Definição 5.4. *(Função geradora exponencial (FGE)) Seja $\mathcal{F}$ uma classe combinatória de objetos rotulados. A função geradora exponencial (FGE) da classe combinatória rotulada $\mathcal{F}$ é a série de potências:*

$$F(z) = \sum_{m\geq 0} f_m \frac{z^m}{m!}, \tag{5.3}$$

□

Exemplo 5.3. *(Urnas, conjuntos não vazios) Considere a classe dos conjuntos não vazios $\mathcal{S}$ em que cada objeto de $\mathcal{S}$ é um conjunto que contém exatamente m elementos. Então, a classe $\mathcal{S}$ pode ser dada por:*

$$\mathcal{S} = \{\{1\}, \{1,2\}, \{1,2,3\}, \{1,2,3,4\} \dots\}$$

[2] O estudo das propriedades de estruturas combinatórias por meio de funções geradoras e cálculo de variável complexa é o objeto da *combinatória analítica* (Flajolet and Sedgewick, 2009).

Seja s_m a sequência de contagem de $\mathcal{S}$. Como a ordem entre os elementos não importa, então $s_m = 1$ para todo $m \geq 1$. Uma vez que não existe um conjunto vazio na classe $\mathcal{S}$, então $s_0 = 0$. Assim, a FGE da classe $\mathcal{S}$ será dada pela seguinte equação:

$$\begin{aligned} S(z) &= \sum_{m\geq 0} s_m \frac{z^m}{m!} = \sum_{m\geq 0} 1 \cdot \frac{z^m}{m!} - 1 \\ &= e^z - 1. \end{aligned} \tag{5.4}$$

❑

Sejam $\mathcal{F}$ e $\mathcal{G}$ duas classes combinatórias rotuladas cujos elementos são conjuntos. Para definir o produto cartesiano entre as classes $\mathcal{F}$ e $\mathcal{G}$ é necessário renomear consistentemente os elementos da classe resultante. Para tanto, defina $\pi_1 : I_{1:\|\gamma\|} \to \mathbb{N}$ e $\pi_2 : I_{1:\|\beta\|} \to \mathbb{N}$, duas funções de renomeação com imagens $\mathrm{Im}(\pi_1)$ e $\mathrm{Im}(\pi_2)$, respectivamente. Define-se o produto cartesiano entre as classes $\mathcal{F}$ e $\mathcal{G}$ da seguinte forma:

$$\begin{aligned} \mathcal{F} \times \mathcal{G} = \{(\gamma', \beta') :(\gamma', \beta') = (\pi_1(\gamma), \pi_2(\beta)), \gamma \in \mathcal{F}, \beta \in \mathcal{G}, \|\gamma'\| = \|\gamma\|, \|\beta'\| = \|\beta\|, \\ \mathrm{Im}(\pi_1) \cap \mathrm{Im}(\pi_2) = \emptyset, \mathrm{Im}(\pi_1) \cup \mathrm{Im}(\pi_2) = I_{1:\|\gamma\|+\|\beta\|}\} \end{aligned}$$

Note que o tamanho de cada objeto $(\gamma', \beta') \in \mathcal{F} \times \mathcal{G}$ é definido como $\|\gamma'\| + \|\beta'\|$.

Exemplo 5.4. *Considere os objetos $S_1 = \{1, 2\}$ e $S_2 = \{1, 2, 3\}$ da classe combinatória $\mathcal{S}$ definida no exemplo 5.4. É possível formar o par $(S_1', S_2') = (\pi_1(S_1), \pi_2(S_2)) = (\{4, 3\}, \{1, 2, 5\}) \in \mathcal{S} \times \mathcal{S}$ em que $\pi_1 = \{(1, 4), (2, 3)\}$ e $\pi_2 = \{(1, 1), (2, 2), (3, 5)\}$. Claramente, o tamanho de (S_1', S_2') é $\|S_1\| + \|S_2\| = \|S_1'\| + \|S_2'\| = 5$. Note que a classe combinatória $\mathcal{S} \times \mathcal{S}$ representa as partições ordenadas de um conjunto em dois blocos não vazios.* ❑

A seguir vamos obter a FGE do produto cartesiano de duas estruturas. Para tanto, sejam $\mathcal{F}$ e $\mathcal{G}$ duas classes combinatórias cujas FGE são

$$F(z) = \sum_{m\geq 0} f_m \frac{z^m}{m!}, \text{ e } G(z) = \sum_{m\geq 0} g_m \frac{z^m}{m!},$$

respectivamente. Um elemento $(\gamma', \beta') \in \mathcal{F} \times \mathcal{G}$ tal que $\|(\gamma', \beta')\| = m$ é obtido escolhendo-se um objeto de tamanho p em $\mathcal{F}$ e um objeto de tamanho $m - p$ em $\mathcal{G}$. Assim, a sequência de contagem de $\mathcal{F} \times \mathcal{G}$ será dada pela seguinte expressão:

$$h_m = \sum_{p\geq 0} \binom{m}{p} f_p g_{m-p}.$$

Se $H(z)$ representa FGE de $\mathcal{F} \times \mathcal{G}$, então, por definição, tem-se que:

$$\begin{aligned} H(z) &= \sum_{m\geq 0} h_m \frac{z^m}{m!} = \sum_{m\geq 0} \frac{1}{m!} \sum_{p\geq 0} \binom{m}{p} f_p g_{m-p} z^m \\ &= \sum_{m\geq 0} \sum_{p\geq 0} \frac{f_p}{p!} \frac{g_{m-p}}{(m-p)!} z^m. \end{aligned} \tag{5.5a}$$

A somatoria do lado direito na equação (5.5a) representa o produto das series de potências $\sum_{p\geq 0} f_m \frac{z^p}{p!}$ e $\sum_{p\geq 0} g_m \frac{z^p}{p!}$, portanto, tem-se que:

$$\begin{aligned} H(z) &= \left(\sum_{p\geq 0} f_m \frac{z^p}{p!} \right) \left(\sum_{p\geq 0} g_m \frac{z^p}{p!} \right) \\ &= F(z)G(z). \end{aligned} \tag{5.5b}$$

Então, a FGE do produto cartesiano de duas classes é o produto algébrico das FGE individuais.

Classes de partições de conjuntos

Defina a classe combinatória $\mathcal{R}_{ord}^{(n)} = \mathcal{S}^n$ das partições ordenadas de um conjunto em n blocos não vazios. Cada objeto de $\mathcal{R}_{ord}^{(n)}$ é uma n-upla da forma $(R_1, \ldots, R_n)$ em que $(R_1, \ldots, R_n) = (\pi_1(S_1), \ldots, \pi_n(S_n))$, $S_i \in \mathcal{S}$, $\bigcap_i \mathrm{Im}(\pi_i) = \emptyset$, $\bigcup_i \mathrm{Im}(\pi_i) = \{1, \ldots, \sum_i \|\alpha_i\|\}$, $i \in I_{1:n}$, sendo $\pi_i : I_{1:\|\alpha_i\|} \to \mathbb{N}, i \in I_{1:n}$ funções de renomeação. Note que o tamanho de cada objeto de $\mathcal{R}_{ord}^{(n)}$ é $\sum_{i=1}^n \|S_i\|$. Usando indução matemática sobre a FGE do produto cartesiano da equação (5.5b) e a FGE $S(z)$ obtida na equação (5.4), obtém-se que a FGE da classe combinatória $\mathcal{R}_{ord}^{(n)}$, denotada por $R_{ord}(z)$, será dada por:

$$R_{ord}(z) = (e^z - 1)^n. \tag{5.6}$$

Defina, agora, a classe combinatória $\mathcal{R}^{(n)}$ das partições de um conjunto em n blocos não vazios em que cada objeto $R \in \mathcal{R}^{(n)}$ é um conjunto da forma $\{R_1, \ldots, R_n\}$ renomeado conforme descrito acima. Note que o tamanho de cada elemento de $\mathcal{R}^{(n)}$ é $\|R_i\| = \sum_{i=1}^n \|R_i\|$. Assim sendo, a FGE da classe $\mathcal{R}^{(n)}$, denotada por $R_n(z)$, será dada pela seguinte equação:

$$R_n(z) = \frac{(e^z - 1)^n}{n!}, \tag{5.7}$$

em que o fatorial do denominador elimina a ordem das k-uplas do produto cartesiano na equação (5.6).

Exemplo 5.5. *Considere o elemento* $R = \{\{1, 2, 3, 4, 5\}, \{6, 7, 8\}\} \in \mathcal{R}^{(2)}$. *Observe*

que $\|R\| = 8$. *O objeto* R *é a estrutura que representa as* $\binom{8}{5} = 56$ *possíveis partições do conjunto* $I_{1:\|\lambda_1\|} = I_{1:8}$, *nas quais um bloco contém* 5 *elementos e o outro bloco contém* 3 *elementos.* ❑

Modelo de Boltzmann

De acordo com o teorema 5.1 existe uma bijeção entre o conjunto de autômatos completos $\mathcal{C}^{(n,k)}$ com n estados e k eventos e o conjunto das partições Dick-k contido no conjunto $\mathcal{Q}^{(kn+1,n)}$ das partições de $I_{1:kn+1}$ em n blocos não vazios. Com base nesse fato, vamos gerar inicialmente uma partição aleatória para, assim, gerar aleatoriamente um autômato. Uma ferramenta para a geração aleatória de partições (e, em geral, de diferentes tipos de classes combinatórias) é o *método do amostrador de Boltzmann* proposto por Duchon et al. (2004), o qual permite gerar um objeto γ de uma classe combinatória $\mathcal{F}$ cujo tamanho $\|\gamma\|$ é uma variável aleatória; porém, todos os objetos do mesmo tamanho ocorrem, invariavelmente, com a mesma probabilidade, isto é, se $\|\gamma\| = m$, então, $P[\gamma] = 1/|\mathcal{F}_m|$. O método do amostrador Boltzmann está basado no modelo probabilístico de Boltzmann que é definido da seguinte forma.

Definição 5.5. *(Modelo de Boltzmann (Duchon et al., 2004)) Dada uma classe combinatória rotulada* $\mathcal{F}$ *o modelo exponencial de Boltzmann associa a cada objeto* $\gamma \in \mathcal{F}$ *com probabilidade dada pela seguinte equação:*

$$P_z(\gamma) = \frac{1}{F(z)} \cdot \frac{z^{\|\gamma\|}}{\|\gamma\|!} \tag{5.8}$$

em que $z \in \mathbb{R}^+$ *é um parâmetro de controle que permite ajustar o tamanho do objeto* γ. □

O tamanho de um objeto resultante sob um modelo de Boltzmann é uma variável aleatória (dependente do parâmetro z) que será denotada por M. Como consequência da definição do modelo de Boltzmann na equação (5.8), a probabilidade de se obter um objeto de tamanho m será:

$$P_z(M = m) = \frac{f_m z^m}{m! F(z)}. \tag{5.9}$$

Pode-se mostrar (Duchon et al., 2004) que o valor esperado de M será dado pela seguinte equação :

$$E_z[M] = z\frac{F'(z)}{F(z)}. \tag{5.10}$$

Exemplo 5.6. *Vamos obter o modelo de Boltzmann para a classe* $\mathcal{S}$. *Para tanto, substituindo* $F(z) = e^z - 1$ *e* $f_m = 1, m > 0$ *na equação* (5.9), *tem-se que a probabi-*

lidade de se obter um objeto $S \in \mathcal{S}$ cujo tamanho seja m será dada por:

$$P_z[\|S\| = m | m > 0] = \frac{z^m}{m!(e^z - 1)} = \frac{z^m e^{-z}}{m!(1 - e^{-z})}. \tag{5.11}$$

A função massa de probabilidade na equação (5.11) *corresponde à distribuição de Poisson truncada em zero com parâmetro z, a qual será denotada por* $\text{Poisson}_{\geq 1}(z)$. ❑

Considere, agora, o problema de se gerar um objeto $R = \{R_1, \ldots, R_n\}$ da classe $\mathcal{R}^{(n)}$. Para esse fim, devem ser realizados n sorteios independentes usando o modelo $\text{Poisson}_{\geq}1(z)$ da equação (5.11) de forma que, em cada sorteio, obtenha-se o tamanho $\|R_i\|$ de cada elemento $R_i \in R$. O tamanho $\|R\| = \sum_{i=1}^{n} \|R_i\|$ do objeto resultante é uma variável aleatória. Porém, usando um *algoritmo de rejeição*, é possível obter um objeto R tal que $\|R\| = p$, sendo p o tamanho fixo desejado. Para maximizar a probabilidade de que $\|R\| = p$ (para que o algoritmo de rejeição seja eficiente), o parâmetro z do modelo $\text{Poisson}_{\geq}1(z)$ deve ser ajustado (Duchon et al., 2004) de tal sorte que $E_z[\|R\|] = p$. Assim, ao substituir $F(z)$ por $R_n(z) = (e^z - 1)^n/n!$ na equação (5.10), obtém-se:

$$E_z[\|R\|] = nz\frac{e^z}{e^z - 1}, \tag{5.12}$$

de forma que ao se fazer $E_z[\|R\|] = p$ no lado esquerdo da equação (5.12), obtém-se a seguinte equação:

$$z - \frac{p}{n}(1 - e^{-z}) = 0. \tag{5.13}$$

Denotando por z_0 a solução estritamente positiva da equação (5.13), tem-se, então, que o modelo de $\text{Poisson}_{\geq}1(z_0)$ é ajustado para obter, com elevada probabilidade, um objeto R de tamanho p.

Amostrador de Boltzmann para a geração de uma partição de $\mathcal{Q}^{(p,n)}$

O algoritmo 5.1 usa o *método do amostrador de Boltzmann* para gerar uniformemente um conjunto $Q \in \mathcal{Q}^{(p,n)}$, isto é, uma partição do conjunto $I_{1:p}$ em n conjuntos não vazios.
O método funciona da seguinte forma:

- Inicialmente é gerada a *estrutura da partição* (isto é, um objeto $R \in \mathcal{R}^{(n)}$ tal que $\|R\| = p$), usando o modelo de $\text{Poisson}_{\geq 1}(z_0)$ e um algoritmo de rejeição (passos 1–3 do algoritmo 5.1).

- Os elementos contidos em cada bloco da partição são gerados por meio de uma permutação do conjunto $I_{1:p}$ gerada uniformemente (passo 4 do algoritmo 5.1).

Algoritmo 5.1: Geração de uma partição $Q \in \mathcal{Q}^{(p,n)}$.

Entradas: os números p e n que definem a partição.
Saídas A estrutura de uma partição $R \in \mathcal{R}^{(n)}$ em que $\|R\| = p$ e a partição $Q \in \mathcal{Q}^{(p,n)}$.

1. Calcule o parâmetro z_0 de acordo com a equação (5.13).
2. Gere o vetor de números inteiros $\mathbf{r} = (r_1, \ldots, r_n)$ usando a distribuição de $\text{Poisson}_{\geq 1}(z_0)$ para gerar cada componente de $\mathbf{r}$.
3. Se $\sum_{i=1}^{n} r_i \neq p$ vá ao passo 2. Caso contrário forme o conjunto R (que define a estrutura da partição Q) da seguinte maneira:

$$\begin{aligned} R &= \{\{1, \ldots, r_1\}, \{r_1 + 1, \ldots, r_1 + r_2 + 1\}, \ldots, \{r_1 + \cdots + r_{n-1}, \ldots, p\}\}. \\ &= \{R_1, R_2, \ldots, R_n\} \end{aligned}$$

4. Gere uniformemente uma bijeção $\pi : I_{1:p} \to I_{1:p}$. Em seguida, construa a partição gerada $Q \in \mathcal{Q}^{(p,n)}$ da seguinte forma:

$$Q = \{\{\pi(i) : i \in R_j\} : R_j \in R, j \in I_{1:n}\}.$$

Exemplo 5.7. *Vamos gerar uma partição $Q \in \mathcal{Q}^{(5,3)}$ seguindo a sequência de passos do algoritmo 5.1. Nesse caso $p = 5$ e $n = 3$ e, portanto, $Q = \{Q_0, Q_1, Q_2\}$ em que $\sum_{i=1}^{3} |Q_i| = 5$.*

1. *Resolvendo a equação $z - \frac{5}{3}(1 - e^{-z}) = 0$, obtém-se um valor $z_0 = 1.1262$.*
2. *Utilizando a distribuição $\text{Poisson}_{>1}(1.1262)$ geramos $\mathbf{r} = (r_1, r_2, r_3)$. Uma primeira execução produz $\mathbf{r} = (1, 2, 1)$.*
3. *Dado que $\sum_{i=1}^{3} r_i \neq p = 5$, o vetor $\mathbf{r}$ deve ser gerado novamente. Depois de algumas iterações obtém-se o vetor $\mathbf{r} = (1, 2, 2)$ em que $\sum_{i=1}^{3} r_i = 5$. Dessa forma já temos o objeto $R = \{\{1\}, \{2, 3\}, \{4, 5\}\} \in \mathcal{R}^{(3)}$ que define a estrutura da partição.*
4. *Finalmente, gerando uniformemente uma bijeção aleatória $\pi : I_{1:5} \to I_{1:5}$ obtém-se, para um sorteio particular, o seguinte mapeamento:*

$$\pi = \{(1, 4), (2, 2), (3, 3), (4, 5), (5, 1)\}.$$

Em seguida substituindo cada número i de cada conjunto $R_j \in R, j = 1, 2, 3$ por $\pi(i)$ obtém-se a partição $Q = \{\{4\}, \{2, 3\}, \{5, 1\}\}$ gerada aleatoriamente com distribuição uniforme. ❑

5.1.5 Geração uniforme de um automato acessível

Para gerar uniformemente uma partição Dick-k $Q_D \in \mathcal{Q}^{(k(n+1)+1,n+1)}$ tal que $\Omega^{-1}(Q_D) \in \mathcal{E}^{(n+1,k)}$ utiliza-se o algoritmo 5.2, proposto por Bassino et al. (2008).

Algoritmo 5.2: Algoritmo de geração uniforme de um autômato $A \in \mathcal{A}^{(n,k)}$.

Entradas *O número de estados n e o número de eventos k do autômato que será gerado.*
Saída: *A partição* Dick-*2* $Q_D \in \mathcal{Q}^{(k(n+1)+1,n+1)}$ *tal que* $\Omega^{-1}(Q_D) \in \mathcal{E}^{(n+1,k)}$.

1. Use o algoritmo 5.1 para obter a estrutura de uma partição $R \in \mathcal{R}^{(n+1)}$ tal que $\|R\| = kn + 1$.
2. Gere uniformemente uma bijeção aleatória $\delta : I_{1:kn+1} \to I_{k+1:kn+1} \cup \{1\}$.
3. Forme o conjunto $Q_D = \{\{\delta(i) : i \in R_j\} : R_j \in R, j \in I_{1:n+1}\}$.
4. Defina $Q_D = \{Q_{D,0}, \ldots, Q_{D,n}\}$ de forma tal que os subconjuntos $Q_{D,0}, \ldots, Q_{D,n}$ estejam ordenados de acordo com seu elemento mínimo. Em seguida, redefina $Q_{D,0} = Q_{D,0} \cup I_{2:k}$.
5. Selecione uniformemente um inteiro $j \in I_{0:n}$ e redefina $Q_{D,j} = Q_{D,j} \cup \{k(n + 1) + 1\}$.
6. Se Q_D com as alterações dos passos 4 e 5 não for uma partição Dick-k então vá ao passo 1. Caso contrário foi gerada uma partição Dick-2 $Q_D \in \mathcal{Q}^{(k(n+1)+1,n+1)}$ tal que $\Omega^{-1}(Q_D) \in \mathcal{E}^{(n+1,k)}$.

O exemplo a seguir ilustra detalhadamente os passos do algoritmo 5.2.

Exemplo 5.8. *Seja $n = k = 2$. Vamos usar o algoritmo 5.2 para gerar uniformemente uma partição Dick-2 $Q_D \in \mathcal{Q}^{(7,3)}$ tal que $\Omega^{-1}(Q_D) \in \mathcal{E}^{(3,2)}$.*

1. *Suponha que nesse passo a estrutura $R = \{\{1\}, \{2, 3\}, \{4, 5\}\}$ tenha sido obtida realizando os passos 1–3 do algoritmo 5.1 conforme mostrado no exemplo 5.7.*
2. *Geramos uniformemente uma bijeção aleatória $\delta : I_{1:5} \to \{1, 3, 4, 5, 6\}$ na qual para um sorteio particular, obtém-se:*

$$\delta = \{(1, 5), (2, 3), (3, 4), (4, 6), (5, 1)\}.$$

3. *Substituímos cada número i de cada conjunto $R_j \in R, j = 1, 2, 3$ por $\delta(i)$ obtendo $Q_D = \{\{5\}, \{3, 4\}, \{6, 1\}\}$.*

4. *Ordenamos Q_D pelo elemento mínimo. Assim,* $Q_D = \{Q_{D,0}, Q_{D,2}, Q_{D,3}\} = \{\{1,6\},\{3,4\},\{5\}\}$. *Uma vez que* $I_{2:k} = \{2\}$, *redefinimos* $Q_{D,0} = Q_{D,0} \cup \{2\}$. *Dessa forma, obtém-se que* $Q_D = \{\{1,2,6\},\{3,4\},\{5\}\}$.

5. *Sorteamos um número aleatório* $j \in \{0,1,2\}$ *obtendo, para um sorteio particular,* $j = 1$. *Então,* $Q_{D,1} = Q_{D,1} \cup \{k(n+1)+1\} = Q_{D,1} \cup \{7\}$. *Assim, obtém-se que* $Q_D = \{\{1,2,6\},\{3,4,7\},\{5\}\}$.

6. *É fácil verificar que* Q_D *é uma partição* Dick-2 *e, em consequência, corresponde a um autômato.* ❑

Para construir o único (a menos de uma renomeação de estados) autômato acessível $G' \in \mathcal{A}^{(n,k)}$ que corresponde à partição Q_D, gerada pelo algoritmo 5.2, procede-se de acordo com o algoritmo 5.3.

Para justificar o procedimento realizado pelo algoritmo 5.3, vamos introduzir uma bijeção, denotada por ψ_{lex}. Para tanto, seja $G'_c = (X', \Sigma, f', \Gamma, \varepsilon)$ um autômato representativo completo com $n+1$ estados. O conjunto X' é completamente ordenado (pela ordem $<_{lex}$) e, por conseguinte, pode ser definida uma bijeção $\psi_{lex} : X' \to I_{0:n}$ na qual se $x'_1, x'_2 \in X'$, então, $\psi_{lex}(x'_1) < \psi_{lex}(x'_2)$ se e somente se $x'_1 <_{lex} x'_2$. Com base na bijeção ψ_{lex}, é possível definir um autômato $\phi_{lex}(G'_c) = (I_{0:n}, \Sigma, f, \Gamma, 0)$ isomorfo com G'_c em que cada estado está rotulado por um número em $I_{0:n}$. Assim, o algoritmo 5.3 constrói diretamente o seguinte autômato acessível:

$$A = \phi^{-1}(\psi_{lex}(\Omega^{-1}(Q_D))) = (I_{1:n}, \Sigma, f_A, \Gamma_A, 1). \tag{5.14}$$

A construção do autômato A, implementada no algoritmo 5.3, é realizada levando em consideração os seguintes fatos:

- Cada subconjunto $Q_{D,i} \in Q_D$ é associado diretamente com o rótulo de um estado $x \in I_{0:n}$ do autômato $\psi_{lex}(\Omega^{-1}(Q_D))$. Assim, tem-se que:

$$Q_{D,i} \to j,\ j = 0, \ldots, n.$$

- O ciclo principal do algoritmo (linhas 6–14) adiciona sequencialmente uma transição da forma (p, σ, q) ao conjunto de transições do autômato em construção. No entanto, a transição será adicionada unicamente se o estado q de chegada da transição for diferente de 0. A razão para isso reside no fato de que as transições que chegam ao estado inicial de $\Omega^{-1}(Q_D)$ devem ser apagadas ao se calcular ϕ^{-1} (veja a observação 5.2).

- A partição Q_D gerada pelo algoritmo 5.2 enumera as $k(n+1)+1$ transições do autômato $\Omega^{-1}(Q_D) \in \mathcal{E}^{(n+1,k)}$. Porém, o algoritmo 5.3 constrói diretamente o

autômato acessível $A \in \mathcal{A}^{(n,k)}$ dado pela equação (5.14), o qual possui $kn+1$ transições possíveis. De acordo com o lema 5.1, as primeiras $k+1$ transições do autômato $\Omega^{-1}(Q_D)$ saem do estado inicial e, por conseguinte, devem ser apagadas ao calcular ϕ^{-1} (veja a observação 5.2). Por essa razão, o ciclo principal do algoritmo 5.3 adiciona ao autômato em construção somente as transições (p,σ,q) do autômato $\Omega^{-1}(Q_D)$ tais que $\nu((p,\sigma,q)) \in I_{k+2:k(n+1)+1}$.

Algoritmo 5.3: Transformação de uma partição em um autômato acessível

PARTIÇÃO2ACESSÍVEL(Q_D)
Entrada: *Uma partição Dick-k denotada por* Q_D
Saída: *O autômato accessível* $A = (X_A, \Sigma, f_A, \Gamma_A, 1)$
1 STACK $S \longleftarrow \emptyset$
2 $T \longleftarrow \emptyset$
3 $q_{lex} \longleftarrow 2$ *(estado que deve seguir na ordem lexicográfica sendo o estado inicial* 1*)*
4 **for** $\sigma' \in \Sigma$ *percorra na ordem inversa* **do**
5 PUSH($S, (1, \sigma')$)
6 **for** $i \in I_{k+2:k(n+1)+1}$ **do**
7 $(p, \sigma) \longleftarrow$ POP(S)
8 $q = j$, em que j é o rótulo associado com o subconjunto $Q_{D,j}$ tal que $i \in Q_{D,j}$
9 **if** $q = q_{lex}$ **then**
10 $q_{lex} \longleftarrow q_{lex} + 1$
11 **for** $\sigma' \in \Sigma$ *percorra na ordem inversa* **do**
12 PUSH($S, (q, \sigma')$)
13 **if** $q \neq 0$ **then**
14 $T \longleftarrow T \cup \{(p, \sigma, q)\}$
fim Construa o autômato acessível $G = (X, \Sigma, f, \Gamma, 1)$, sendo que $X = \{1, \ldots, n\}$, $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$, e, f e Γ definidas pelas transições em T.

O exemplo a seguir ilustra detalhadamente os passos do algoritmo 5.3 na conversão de uma partição em um autômato acessível.

Exemplo 5.9. *Vamos transformar a partição* $Q_D = \{\{1,2,6\},\{3,4,7\},\{5\}\}$ *(obtida no exemplo 5.8) em um autômato* $A = (I_{1:2}, \{a,b\}, f_A, \Gamma_A, 1) \in \mathcal{A}^{(2,2)}$ *seguindo os passos do algoritmo 5.3. Note que as variáveis do algoritmo são:*

- *S: pilha inicialmente vazia.*
- *T: conjunto de transições do autômato A em construção, inicialmente vazio.*
- *q_{lex} : índice de ordem do próximo estado que pode ser visitado na ordem lexicográfica. Inicializado em 2, uma vez que o estado inicial é 1.*
- *(p, σ, q), transição candidata a ser adicionada no autômato em construção.*

O ciclo das linhas 6 e 7 inicializa a pilha com pares ordenados da forma $(1, \sigma')$ *em que a primeira componente é o estado inicial e a segunda componente é um*

evento do alfabeto. Note que os pares são colocados na pilha na ordem inversa de Σ*, assim,*

$$S = \begin{array}{|c|}\hline (1,a) \\ \hline (1,b) \\ \hline \end{array}\,.$$

O ciclo principal (linhas 6–14) percorre o conjunto $I_{k+2:k(n+1)+1} = \{4,5,6,7\}$ *que enumera todas as transições possíveis do autômato acessível em construção.*

Tabela 5.1: Execução do algoritmo 5.3 para transformar a partição $Q_D = \{\{1,2,6\},\{3,4,7\},\{5\}\}$ em um autômato acessível.

	q_{lex}	(p,σ,q)	S	T
i=4	2	$(1,a,1)$	$\boxed{(1,b)}$	$\{(1,a,1)\}$
i=5	3	$(1,b,2)$	$\begin{array}{\|c\|}\hline (2,b) \\ \hline (2,a) \\ \hline \end{array}$	$\{(1,a,1),(1,b,2)\}$
i=6	3	$(2,a,0)$	$\boxed{(2,b)}$	$\{(1,a,1),(1,b,2)\}$
i=7	3	$(2,b,1)$	$\boxed{}$	$\{(1,a,1),(1,b,2),(2,b,1)\}$

A tabela 5.1 mostra os valores das variáveis para cada valor de $i \in \{4,5,6,7\}$ *(ao final do ciclo). Por exemplo, a primeira fila da tabela 5.1 mostra os valores das variáveis para* $i = 4$*, que são obtidos da seguinte forma: o par ordenado* $(p,\sigma) = (1,a)$ *é obtido da pilha* S *(linha 7); uma vez que* $i = 4 \in Q_{D,1}$*, o estado de chegada da transição é* $q = 1$ *(linha 8); como* $q \neq 0$ *(verificado na linha 13), a transição* $(1,a,1)$ *é adicionada ao conjunto* T *de transições do autômato em construção (linha 14).*

Finalmente, quando $i = 7$ *obtém-se o seguinte conjunto de transições*

$$T = \{(1,a,1),(1,b,2),(2,b,1)\},$$

o qual permite construir o autômato acessível A *cujo diagrama de transição de estados está representado na figura 5.4.*

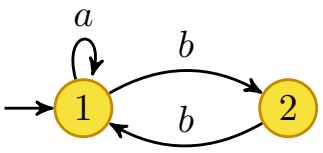


Figura 5.4: Autômato acessível $A \in \mathcal{A}^{(2,2)}$ correspondente à partição $Q_D = \{\{1,2,6\},\{3,4,7\},\{5\}\}$.

❑

5.2 Experimento com amostragem para a análise da complexidade média em diagnose de falhas

5.2.1 Tamanho amostral

Após discutir os aspectos relacionados com a implementação de um mecanismo para a geração aleatória uniformemente distribuída de autômatos, vamos agora considerar o problema da seleção do tamanho amostral necessário para obter conclusões estatisticamente significantes. Comecemos pela seguinte definição.

Definição 5.6. *(estimativa consistente) Seja* $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_N)$*, uma amostra aleatória de tamanho* N *em que* $E[Y_i] = m, i = 1, \ldots, N$*.* $\hat{M}_N$ *é uma estimativa consistente do valor de* m *obtida a partir da amostra* $\mathbf{Y}$ *se* $\lim_{N\to\infty} \hat{M}_N = m$*.*

Uma vez que o tamanho amostral N é finito, é necessário quantificar a proximidade entre o valor esperado m e a estimativa $\hat{M}_N$. Para tanto, é usada a seguinte probabilidade:

$$P\left\{m \in \left[\hat{M}_N - \delta, \hat{M}_N + \delta\right]\right\} = 1 - \alpha, \tag{5.15}$$

em que $[\hat{M}_N - \delta, \hat{M}_N + \delta]$ é o *intervalo de confiança* e $1 - \alpha$ é o *nível de confiança*[3]. Dado que $\hat{M}_N$ é uma variável aleatória, o intervalo de confiança é um conjunto aleatório. O nível de confiança é a probabilidade de que o parâmetro determinístico m esteja no intervalo de confiança aleatório. Note que, ao definir $\rho = \delta/\hat{M}_N$ como o erro percentual na estimação de m, o intervalo de confiança passa a ser escrito como:

$$[\hat{M}_N(1 - \rho), \hat{M}_N(1 + \rho)].$$

Tamanho amostral considerando um modelo lognormal

Uma primeira forma de se estimar o tamanho amostral para análise da complexidade computacional média do diagnosticador é supor, considerando o estudo realizado no capítulo 4, que as variáveis aleatórias que representam a complexidade na construção do diagnosticador estejam distribuídas lognormalmente. Assim, considere uma amostra aleatória $\mathbf{Y_D} = (Y_{D,1}, Y_{D,2}, \ldots, Y_{D,N})$ em que $Y_{D,i} \sim LN(\eta, \sigma)$, $E[Y_{D,i}] = m_{LN}$, $\text{var}[Y_{D,i}] = s^2_{LN}$, $i = 1, \ldots, N$. O valor esperado de uma variável aleatória com distribuição lognormal (tabela 4.3) é dado pela seguinte equação:

$$m_{LN} = \exp\left(\eta + \frac{\sigma^2}{2}\right). \tag{5.16a}$$

[3]Uma apresentação detalhada desses conceitos pode ser encontrada em Gubner (2006).

Assim, uma estimativa consistente para m_{LN}, denotada por $\hat{M}_{LN}$, será dada por:

$$\hat{M}_{LN} = \exp\left(\hat{\eta} + \frac{\hat{\sigma}^2}{2}\right) \tag{5.16b}$$

em que $\hat{\eta}$ e $\hat{\sigma}$ são os EMV da distribuição lognormal definidos nas equações (4.11a,b).

O tamanho amostral N que garante que m_{LN} pertença ao intervalo de confiança $[\hat{M}_{LN}(1-\rho), \hat{M}_{LN}(1+\rho)]$ com um nível de confiança especificado, pode ser calculado de forma aproximada usando a seguinte fórmula proposta em Hale (1972):

$$N = \frac{z_{1-\alpha/2}\hat{\sigma}^2}{\ln^2(\rho+1)}, \tag{5.17}$$

em que $z_{1-\alpha/2}$ é o quantil $1-\alpha/2$ de uma distribuição normal padrão. Uma vez selecionado o tamanho amostral, é possível calcular de forma exata o erro percentual na estimação de m_{LN} por meio da seguinte fórmula (Zou et al., 2009):

$$\rho_{LN} = 1 - \exp\left[-\left(z^2_{1-\alpha/2}\frac{\hat{\sigma}^2}{N} + \left(\frac{\hat{\sigma}^2}{2} - \frac{\hat{\sigma}^2(N-1)}{2\chi^2_{1-\alpha/2,N-1}}\right)^2\right)^{1/2}\right], \tag{5.18}$$

em que em que $\chi^2_{1-\alpha/2}$ é o percentil $1-\alpha/2$ da distribuição chi-quadrado com $N-1$ graus de liberdade.

Para determinar o tamanho amostral na análise do diagnosticador, vamos especificar um nível de confiança de 95% (isto é $1-\alpha = 0.95$ e, em consequência, $\alpha = 0.05$) e um erro porcentual $\rho = 1\%$. Usando a equação (5.17), obtém-se:

$$N = \frac{z_{1-\alpha/2}\hat{\sigma}^2}{\ln^2(\rho+1)} = \frac{1.96 \times 0.5^2}{\ln^2(0.01+1)} = 9700.$$

O valor de $\hat{\sigma} = 0.5$ usado no cálculo é o maior valor reportado na tabela 4.5. Agora, supondo que o tamanho amostral escolhido seja $N = 10000$, calculemos o valor exato de ρ usando a equação (5.18), obtendo:

$$\rho_{LN} = 1 - \exp\left[-\left(1.96^2\frac{0.5^2}{10000} + \left(\frac{0.5^2}{2} - \frac{0.5^2 \cdot 9999}{2 \cdot 10278.06}\right)^2\right)^{1/2}\right] = 0.010,$$

que mostra que, para um tamanho amostral $N = 10000$, o erro porcentual é, de fato de 1%.

Tamanho amostral sem considerar a distribuição da amostra

Mesmo sem adotar um modelo para uma amostra, ainda assim, é possível determinar um tamanho amostral usando dois resultados fundamentais da teoria probabilística: a lei forte dos grandes números e o teorema do limite central.

Considere, novamente, a amostra $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_N)$ de tamanho N tal que $E[Y_i] = m$ e $\text{var}[Y_i] = s^2$. Note que não estamos realizando nenhuma suposição sobre a distribuição das variáveis aleatórias Y_i, $i = 1, \ldots, N$. Então, a lei forte dos grandes números estabelece que a média amostral

$$\overline{M}_N = \frac{1}{N}\sum_{i=1}^{N} Y_i,$$

é uma estimativa consistente do valor de m. Se definirmos $E_N = \overline{M_N} - m$ como a variável aleatória que representa o erro na estimação de m, o teorema do limite central estabelece que $E_N \approx N(0, s^2/N)$, isto é, que a variável de erro na estimação de m possui distribuição aproximadamente normal. Com base nesses fatos[4], o intervalo de confiança para m está dado por:

$$\left[\overline{M}_N - \frac{sz_{1-\alpha/2}}{\sqrt{N}}, \overline{M}_N + \frac{sz_{1-\alpha/2}}{\sqrt{N}}\right]. \tag{5.19}$$

Note que, nesse caso, o erro porcentual na estimação de m será dado por:

$$\rho = sz_{1-\alpha/2}/(\overline{M}_N\sqrt{N}). \tag{5.20}$$

Observe que o intervalo de confiança (5.19) implica que a média amostral $\overline{M}_N$ está no intervalo

$$\left[m - \frac{sz_{1-\alpha/2}}{\sqrt{N}}, m + \frac{sz_{1-\alpha/2}}{\sqrt{N}}\right],$$

e também que o valor mínimo que pode tomar a média amostral (com probabilidade $1-\alpha$) é $\overline{M}_N = m - sz_{1-\alpha/2}/\sqrt{N}$. Substituindo esse último valor na equação (5.20), obtém-se a seguinte expressão para o valor do erro percentual máximo na estimação de m:

$$\rho_{max} = \frac{sz_{1-\alpha/2}}{m\sqrt{N} - sz_{1-\alpha/2}}. \tag{5.21}$$

O valor exato da média m e do desvio padrão s foi determinado exatamente para as instâncias do experimento exaustivo da seção 5.2 (veja a tabela 4.2). Com base nesses valores, vamos verificar se o tamanho amostral $N = 10000$ garante um intervalo de confiança tal que $\rho \leq 1\%$ com um nível de confiança de 95%. As colunas 2 e 3 da tabela 5.2 mostram, respectivamente, os valores do erro percentual máximo (cal-

[4]Veja os detalhes em Gubner (2006)

Tabela 5.2: Erro porcentual máximo na estimação da média ρ das variáveis $\mathbf{D}^{(n,k,u)}$ e $V^{(n,k,u)}, (n,k,u)$ com um tamanho amostral $N = 10000$.

(n,k,u)	$\rho_{max,D}$	$\rho_{max,V}$
$(3,3,1)$	0,0080	0,0097
$(3,4,1)$	0,0063	0,0088
$(3,4,2)$	0,0080	0,0096
$(3,4,3)$	0,0082	0,0068
$(3,5,1)$	0,0089	0,0098
$(4,3,1)$	0,0076	0,0108
$(4,3,2)$	0,0096	0,0071
$(5,2,1)$	0,0104	0,0078

culados usando a equação (5.21)) na estimação da média para as variáveis $\mathbf{D}^{(n,k,u)}$ e para as variáveis $\mathbf{V}^{(n,k,u)}$ (lembremos para o leitor $\mathbf{D}^{(n,k,u)}$ e $\mathbf{V}^{(n,k,u)}$ representam, respectivamente a complexidade na construção de diagnosticadores e verificadores para conjuntos de autômatos válidos com n eventos, k estados e u eventos não-observáveis conforme a notação usada no capítulo anterior). Por exemplo, para as variáveis $\mathbf{D}^{(3,4,3)}$ e $\mathbf{V}^{(3,4,3)}$, os erros porcentuais máximos são, respectivamente, $\rho_{max,D} = 0.0082$ e $\rho_{max,V} = 0.0068$. Note que para todas as variáveis, o erro máximo na estimação de m está próximo ao valor de 1% e, em consequência, $N = 10000$ representa um tamanho amostral confiável para um experimento com amostragem uniforme com base no conhecimento parcial obtido no experimento exaustivo.

5.2.2 Variáveis do experimento com amostragem uniforme

Considere um autômato $G_i^{(n,k,u)} = (X_n, \Sigma_k, f_i^{(n,k)}, \Gamma_i^{(n,k)}, 1)$ pertencente a um conjunto de autômatos válidos $\mathcal{A}_v^{(n,k,u)}$ de acordo com a definição 4.4. Em síntese, o autômato $G_i^{(n,k,u)}$ possui as três características (definidas pelas propriedades **P1**–**P3**), quais sejam: *(i)* representa uma linguagem viva; *(ii)* possui n estados, k eventos e u eventos não-observáveis; *(ii)* não possui ciclos formados por eventos não-observáveis dentre os quais está o evento de falha. Nesse contexto, sejam os autômatos

$$G_{D,i}^{(n,k,u)} = (X_{D,i}^{(n,k,u)}, \Sigma_{k,o}, f_{D,i}^{(n,k,u)}, \Gamma_{D,i}^{(n,k,u)}, x_{0D,i}^{(n,k,u)})$$

e

$$G_{V,i}^{(n,k,u)} = (X_{V,i}^{(n,k,u)}, \Sigma_{V,k}, f_{V,i}^{(n,k,u)}, \Gamma_{V,i}^{(n,k,u)}, (0,(0,N))),$$

respectivamente, o diagnosticador e o verificador associados ao autômato $G_i^{(n,k,u)}$.

Instâncias consideradas no experimento uniforme

Represente por $\hat{\mathcal{A}}_v^{(n,k,u)} \subset \mathcal{A}_v^{(n,k,u)}$ um conjunto de $N = 10000$ autômatos válidos gerados uniformemente usando o gerador descrito na seção 5.1. No experimento com amostragem foram considerados os conjuntos de autômatos válidos gerados uniformemente pertencentes à seguinte classe:

$$\mathcal{A}_U = \{\hat{\mathcal{A}}_v^{(n,k,u)} : (n,k,u) \in I_U\},$$

em que

$$\begin{aligned}
I_U &= I_1 \cup I_2, \cup I_3, \\
I_1 &= \{(n,k,u) : n \in I_{3:20}, k \in I_{2:10}, u \in I_{1:k-1}\}, \\
I_2 &= \{(n,k,u) : n \in I_{3:20}, k \in \{16,24\}, u \in I_{1:k-1}\}, \\
I_3 &= \{(n,k,u) : n \in \{32,64\}, k \in I_{2:10} \cup \{16,24\}, u \in I_{1:k-1}\}.
\end{aligned}$$

Em total foram selecionados 1,660 conjuntos de autômatos válidos gerados uniformemente, isto é, $|A_u| = 1{,}660$ de forma que o número total de autômatos considerado para esse experimento é de 16,000,000.

Variáveis de saída do experimento

Considere um conjunto de autômatos válidos gerado uniformemente $\hat{\mathcal{A}}_v^{(n,k,u)}$ em que $(n,k,u) \in I_U$. As variáveis que serão consideradas no experimento de geração uniforme são:

1. $\hat{\mathbf{D}}^{(n,k,u)} = (D_1^{(n,k,u)}, D_2^{(n,k,u)}, \ldots, D_{10000}^{(n,k,u)})$: variável contendo 10000 valores em que cada $D_i^{(n,k,u)} = |X_{D,i}^{(n,k,u)}|$, isto é, a cardinalidade do conjunto de estados do diagnosticador associado com o autômato $G_i^{(n,k,u)} \in \hat{\mathcal{A}}_v^{(n,k,u)}$.

2. $\hat{\mathbf{V}}^{(n,k,u)} = (V_1^{(n,k,u)}, V_2^{(n,k,u)}, \ldots, V_{10000}^{(n,k,u)})$: variável contendo 10000 valores em que cada $V_i^{(n,k,u)} = |X_{V,i}^{(n,k,u)}|$, isto é, a cardinalidade do conjunto de estados do verificador associado o autômato $G_i^{(n,k,u)} \in \hat{\mathcal{A}}_v^{(n,k,u)}$.

Note que, de acordo com a definição 4.5, $\hat{\mathbf{D}}^{(n,k,u)}$ e $\hat{\mathbf{V}}^{(n,k,u)}$ são observações de uma amostra de tamanho 10000 e, por simplicidade, serão chamadas simplesmente *variáveis* na discussão subsequente.

5.2.3 Procedimento experimental

Considere um conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$ de autômatos válidos gerados uniformemente. O teste com geração uniforme para esse conjunto compreende os seguintes passos:

Passo 1: *Geração do conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$ de autômatos válidos gerados uniformemente.*

Esse passo é realizado combinando o gerador uniforme de autômatos descrito na seção 5.1 com um algoritmo de *rejeição*. Para cada autômato produzido pelo gerador uniforme, define-se o conjunto de eventos observáveis $\Sigma_{k,o}$ e o conjunto de falha $\Sigma_f = \{\sigma_k\}$ obtendo, assim, o autômato candidato $G_i^{(n,k,u)}$. Em seguida, o algoritmo de rejeição verifica se as propriedades **P1**–**P3** são satisfeitas pelo autômato candidato $G_i^{(n,k,u)}$. Caso $G_i^{(n,k,u)}$ satisfaça **P1**–**P3**, ele será acrescentado ao conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$. O processo continua até gerar 10000 autômatos válidos que completam o conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$.

Passo 2: *Cálculos de complexidade.*

Nesse passo é processado um conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$ de autômatos válidos tomado da base de dados produzida no passo 1. Para cada autômato $G_i^{(n,k,u)}$ do conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$ é construído o diagnosticador e o verificador. O processo continua até processar todos os autômatos do conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$ obtendo, assim, as variáveis $\hat{\mathbf{D}}^{(n,k,u)}$ e $\hat{\mathbf{V}}^{(n,k,u)}$.

5.2.4 Análise exploratória

A seguir vamos apresentar os resultados estatísticos obtidos para as variáveis $\hat{\mathbf{D}}^{(n,k,u)}$ e $\hat{\mathbf{V}}^{(n,k,u)}$ em que $(n,k,u) \in I_1$. A informação experimental sobre a complexidade na construção de diagnosticadores e verificadores da coleção dos 8,100,000 (esse número aparece porque $|I_1| = 810$ e o tamanho amostral é 10000) autômatos está sendo representada, de forma sucinta, nas figuras 5.5(a,b) e 5.6(a,b). Para facilitar a discussão subsequente, observe que cada ponto dessas figuras representa o valor estatístico (média ou desvio padrão, conforme seja o caso) associado com o número de estados do eixo n, o número de eventos do eixo k e o número de eventos não-observáveis u do mapa de cores.

Complexidade do diagnosticador

O eixo vertical da figura 5.5a representa, em escala logarítmica, as médias das variáveis $\hat{\mathbf{D}}^{(n,k,u)}$. Para os conjuntos de autômatos testados, a informação mais relevante extraída da figura 5.5a pode ser sumarizada da seguinte forma:

- A média da complexidade no cálculo de diagnosticadores possui um comportamento monotônico subexponencial. Tal informação pode ser obtida diretamente na figura 5.5a, uma vez que as médias das variáveis $\hat{\mathbf{D}}^{(n,k,u)}$ estão representadas graficamente em escala logarítmica.

- Para uma classe formada por conjuntos de autômatos $\hat{\mathcal{A}}_v^{(n_0,k_0,u)}$ (em que n_0 e k_0 são números fixos) que diferem somente no número $u \in I_{0:k-1}$ de eventos não-observáveis, a maior média da complexidade na construção de diagnosticadores aparece no conjunto $\hat{\mathcal{A}}_v^{(n_0,k_0,1)}$, isto é, no conjunto de autômatos que possuem um único evento não-observável (tal fato é consistente com o resultado obtido em Jirásková and Masopust (2012) no qual se mostra que o pior caso de complexidade em projeções naturais de linguagens ocorre para autômatos com a menor quantidade de transições não observáveis). Em contrapartida, a menor média da complexidade aparece no conjunto $\hat{\mathcal{A}}_v^{(n_0,k_0,k-1)}$, isto é, no conjunto de autômatos que possui um único evento observável. A complexidade média decresce monotonicamente conforme aumenta o número de eventos não-observáveis.

O eixo vertical da figura 5.5b representa, em escala logarítmica, os desvios padrões das variáveis $\hat{\mathbf{D}}^{(n,k,u)}$, $(n,k,u) \in I_1$ e, conforme pode ser observado, segue um comportamento similar ao comportamento das médias discutido no parágrafo anterior.

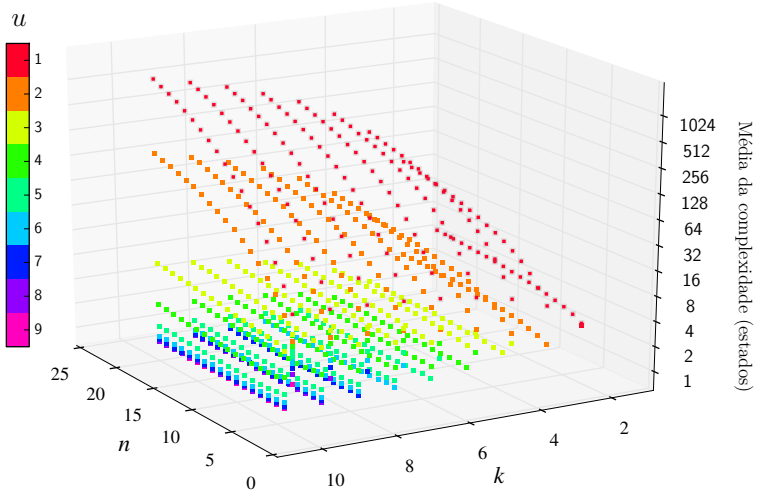
A figura 5.5c mostra os histogramas das variáveis $\hat{\mathbf{D}}^{(n,k,u)}$, $n \in I_{15:18}, k \in I_{6:9}, u = 1,2,3$. A abscissa representa o número de estados, a ordenada o número de eventos e o mapa de cores o número de eventos não-observáveis. É possível perceber que aparece um padrão de distribuição da complexidade na construção de diagnosticadores de tipo "lognormal", similar ao padrão que já foi observado nas instâncias que foram analisadas exaustivamente no capítulo anterior.

Complexidade do verificador

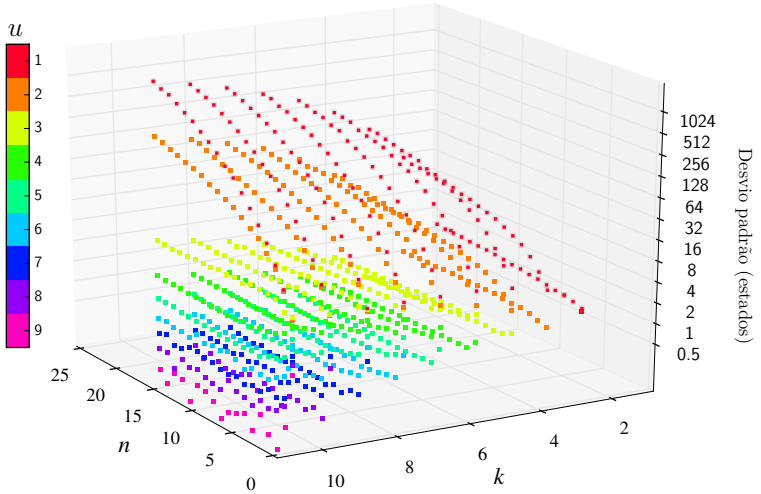
O eixo vertical da figura 5.6a representa, em escala de estados elevados ao quadrado, as médias das variáveis $\hat{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_1$. Limitando-nos às instâncias testadas nesse experimento, as informações mais relevantes que podem ser obtidas do gráfico são as seguintes:

O1. A média da complexidade na construção de verificadores é de ordem quadrática, o que pode ser notado facilmente uma vez que o eixo vertical está representado em escala quadrática. Para evidenciar ainda mais essa observação, o plano $2n^2$ (é um plano pela escala quadrática do eixo vertical) está sendo representado sobre os pontos das médias amostrais.

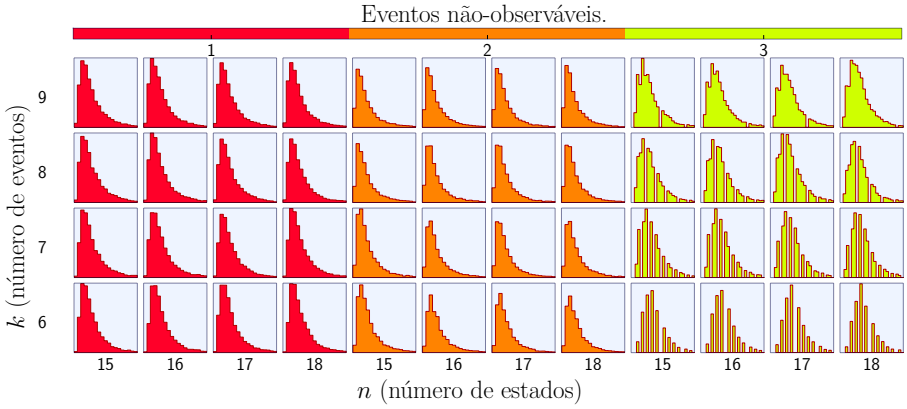
O2. Para uma classe formada por conjuntos de autômatos válidos $\hat{\mathcal{A}}^{(n_0,k_0,u)}$ (em que $n_0 \in I_{3:20}$ e $k_0 \in I_{2:10}$ são números fixos) que diferem somente no número de eventos não-observáveis, a menor média da complexidade na construção de verificadores aparece no conjunto $\hat{\mathcal{A}}^{(n_0,k_0,1)}$ de autômatos que possuem um único evento não-observável.



(a)

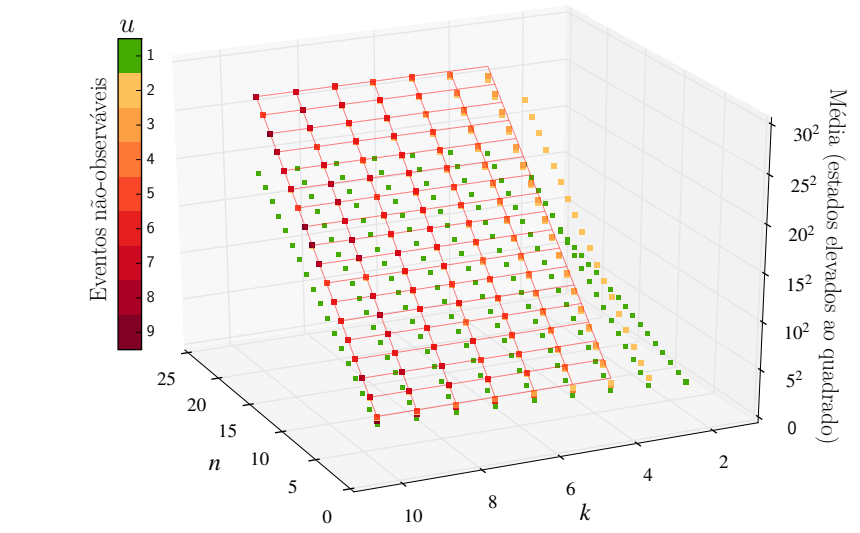


(b)

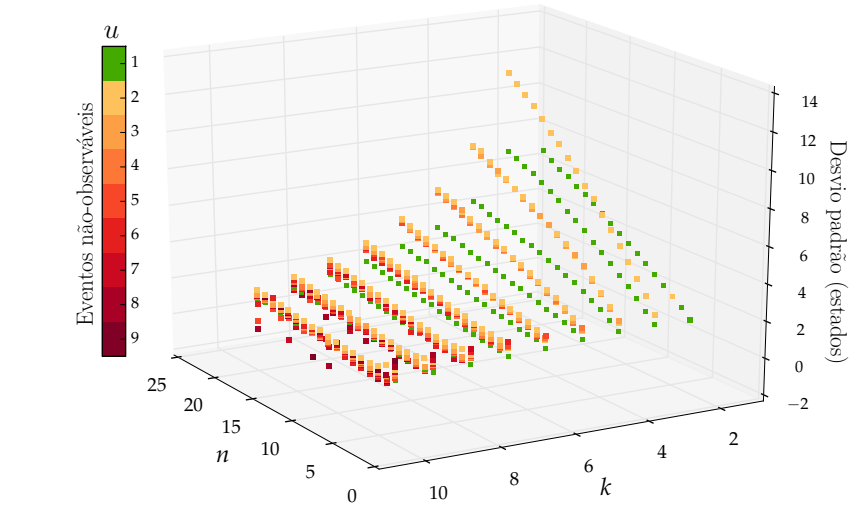


(c)

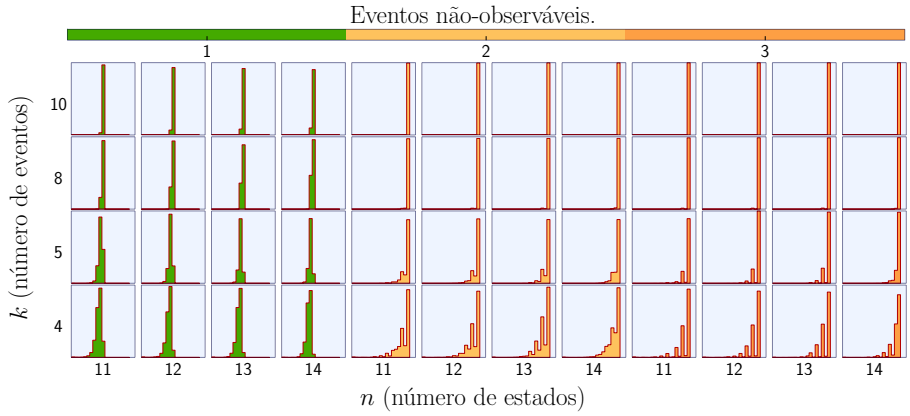
Figura 5.5: Médias das variáveis $\hat{\mathbf{D}}^{(n,k,u)}$, $(n,k,u) \in I_1$ (a); Desvios padrões das variáveis $\hat{\mathbf{D}}^{(n,k,u)}$, $(n,k,u) \in I_1$ (b); Histogramas das variáveis $\hat{\mathbf{D}}^{(n,k,u)}$, $n \in I_{15:18}$, $k \in I_{6:9}$, $u \in \{1,2,3\}$ (c).



(a)



(b)



(c)

Figura 5.6: Média das variáveis $\hat{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_1$ observadas (a); Desvios padrões das variáveis $\hat{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_1$ (b); Histogramas das variáveis $\hat{\mathbf{V}}^{(n,k,u)}$, $n \in I_{11:14}$, $k \in \{4,5,8,10\}$, $u \in \{1,2,3\}$ (c).

O3. Para um valor fixo $k_0 \geq 4$, a média da complexidade na construção de verificadores para os conjuntos $\hat{\mathcal{A}}^{(n,k_0,u)}$, $u \in I_{2:10}$ possui um comportamento bastante aproximado à curva $2n^2$.

O eixo vertical da figura 5.6b representa os desvios padrões das variáveis $\hat{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_1$. A informação mais importante revelada pela figura 5.6b é que o desvio padrão diminui à medida que aumenta o número de eventos. Note, por exemplo, que para $k = 10$, o desvio padrão é (para um valor de n qualquer) de no máximo 4 estados. Tal fato reforça a observação (**O3**) no sentido de que a complexidade média na construção de verificadores aproxima-se à curva $2n^2$ de forma cada vez mais determinística quando o número de eventos não-observáveis é maior que 1.

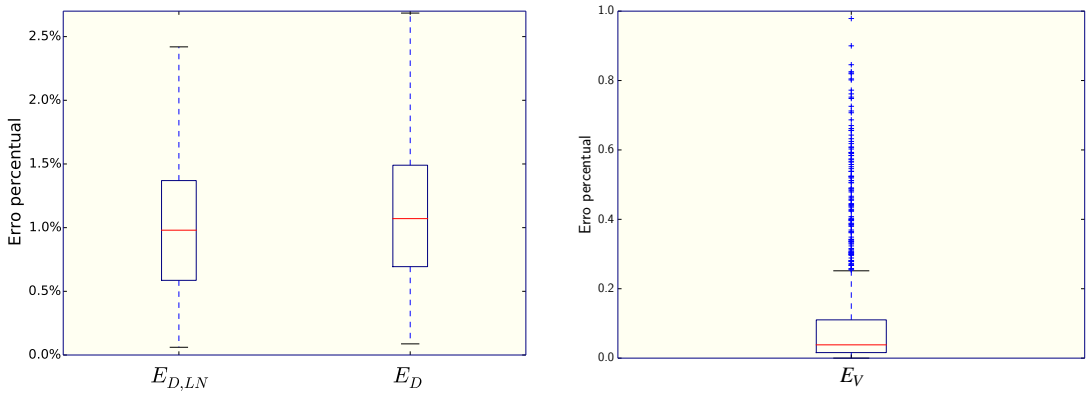
A figura 5.6c mostra os histogramas das variáveis $\hat{\mathbf{V}}^{(n,k,u)}$, $n \in I_{11:14}$, $k \in \{4,5,8,10\}$, $u = 1,2,3$. A abscissa representa o número de estados, a ordenada o número de eventos e o mapa de cores o número de eventos não-observáveis. A escala horizontal de cada histograma varia entre 0 e $2n^2$. Os histogramas mostram que, à medida que aumenta o número de eventos, a distribuição da complexidade na construção de verificadores vai se concentrando progressivamente em dois valores: para $u = 1$ a complexidade se concentra em um valor próximo a n^2 e, para $u = 2,3$ a complexidade se concentra em um valor próximo a $2n^2$.

Efeito do tamanho amostral

A seguir vamos verificar a validade das suposições feitas na seção 5.2.1 para a seleção do tamanho amostral. Para começar a discussão, vamos definir as seguintes variáveis:

- $E_D = \{\rho_D^{(n,k,u)} : (n,k,u) \in I_U\}$: conjunto de pontos contendo a distribuição do erro percentual na estimação dos valores esperados das variáveis $\mathbf{D}^{(n,k,u)}$ usando o teorema do limite central. Cada valor $\rho_D^{(n,k,u)} \in \mathbb{R}$ representa o erro percentual da média amostral $\overline{\hat{\mathbf{D}}^{(n,k,u)}}$ quando usada como estimativa de $\overline{\mathbf{D}^{(n,k,u)}}$, isto é, do valor esperado da complexidade na construção de diagnosticadores para um conjunto de autômatos válidos completo $\mathcal{A}_v^{(n,k,u)}$. O valor de cada $\rho_D^{(n,k,u)}$ foi calculado usando a equação (5.20) com um nível de confiança de 95%.

- $E_{D,LN} = \{\rho_{D,LN}^{(n,k,u)} : (n,k,u) \in I_U\}$: conjunto de pontos contendo a distribuição do erro percentual na estimação dos valores esperados das variáveis $\mathbf{D}^{(n,k,u)}$ sob a hipótese de que essas variáveis estão distribuídas lognormalmente. Cada valor $\rho_{D,LN}^{(n,k,u)} \in \mathbb{R}$ representa o erro percentual cometido ao usar $\hat{M}_{LN}$ (equação 5.16b) como estimativa de $\overline{\mathbf{D}^{(n,k,u)}}$. O valor de cada $\rho_D^{(n,k,u)}$ é calculado usando a equação (5.18) com um nível de confiança de 95%. .



(a) (b)

Figura 5.7: Diagramas de caixa do erro percentual na estimação da média das amostras $\tilde{\mathbf{D}}^{(n,k,u)}$, $(n,k,u) \in I_1$ com base nas observações tomadas (a); Diagrama de caixa do erro percentual na estimação da média $\tilde{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_1$ com base nas observações tomadas.

- $E_V = \{\rho_V^{(n,k,u)} : (n,k,u) \in I_U\}$: conjunto de pontos contendo a distribuição do erro percentual na estimação dos valores esperados das variáveis $\mathbf{V}^{(n,k,u)}$ usando o teorema do limite central. Cada valor $\rho_V^{(n,k,u)} \in \mathbb{R}$ representa o erro percentual da média amostral $\overline{\hat{\mathbf{V}}^{(n,k,u)}}$ quando usada como estimativa de $\overline{\mathbf{V}^{(n,k,u)}}$. O valor de cada $\rho_V^{(n,k,u)}$ é calculado usando a equação (5.20) com um nível de confiança de 95%.

Considere o erro percentual na estimação do valor esperado da complexidade na construção de diagnosticadores. A partir dos diagramas de caixa[5] de ρ_D e $\rho_{D,LN}$ mostrados na figura 5.7a é possível realizar as seguintes considerações sobre o comportamento do erro porcentual na estimação do valor esperado das variáveis $\mathbf{D}^{(n,k,u)}, (n,k,u) \in I_U$: *(i)* as medianas indicam que o erro sempre está em torno de 1%; *(ii)* os terceiros quartis indicam que o erro típico pode chegar até um valor de 1,5% e *(iii)* o valor máximo em ambos casos está em torno de 2,5% (sendo este um valor atípico). Note que os dois diagramas são muito próximos e, em consequência, a hipótese de um modelo lognormal tem um efeito apenas marginal sobre o erro porcentual na estimação do valor esperado.

Considere, agora, o erro porcentual na estimação da complexidade média na construção de verificadores. O diagrama de caixa de E_V mostrado na figura 5.7b permite realizar as seguintes observações sobre do erro percentual na estimação do

[5]Os diagramas de caixas são uma representação gráfica das características estatísticas de um conjunto de observações. A linha no meio da caixa indica o valor da *mediana* e os limites superior e inferior da caixa indicam, respectivamente, os quartis 25º e 75º. Assim, a caixa contém 50% dos valores centrais dos dados. As duas linhas que se estendem fora da caixa representam a distancia à maior e à menor observação que estão a menos de um quartil da caixa. As observações atípicas são as que variam entre 1,0 e 1,5 quartis de distância da caixa. Os valores extremos são as observações que estão a mais de 1,5 quartis do extremo da caixa (Hair et al., 2007).

valor esperado de cada uma das variáveis $\mathbf{V}^{(n,k,u)}$:*(i)* a mediana mostra que o valor típico do erro percentual é de 0,03%; *(ii)* o terceiro quartil indica que o erro tipico pode alcançar 0,1%;*(iii)* o valor máximo encontra-se próximo a 1%, se bem que é um valor extremo.

De acordo com a discussão acima é possível concluir que, independentemente da adoção de um modelo lognormal, a seleção de um tamanho amostral de 10000 garante condições estatísticas próximas às esperadas para o caso do diagnosticador. No caso de verificador, tendo em vista que as menores variâncias foram obtidas para un número de eventos $k \geq 4$ podemos concluir que tamanho amostral escolhido (10000 autômatos) é bastante conservador.

5.3 Análise de dados

O objetivo desta seção é propor modelos experimentais da complexidade média na construção de diagnosticadores e verificadores com base nos resultados do experimento com amostragem uniforme realizado neste capítulo. Para facilitar a discussão, vamos definir as seguintes funções:

- $\mathbf{M}_D(n,k,u) = \overline{\mathbf{D}^{(n,k,u)}}$ é a função que associa um ponto $(n,k,u) \in I_U$ com o valor esperado da complexidade dos diagnosticadores construídos para o conjunto $\mathcal{A}_v^{(n,k,u)}$ de todos os autômatos válidos.
- $\hat{\mathbf{M}}_D(n,k,u) = \overline{\hat{\mathbf{D}}^{(n,k,u)}}$ é a função que associa um ponto $(n,k,u) \in I_U$ com a média amostral da complexidade dos diagnosticadores construídos para o conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$ de autômatos válidos gerados aleatoriamente com distribuição uniforme.
- $\mathbf{M}_V(n,k,u) = \overline{\mathbf{V}^{(n,k,u)}}$ é a função que associa um ponto $(n,k,u) \in I_U$ com o valor esperado da complexidade dos verificadores construídos para o conjunto $\mathcal{A}_v^{(n,k,u)}$ de todos os autômatos válidos.
- $\hat{\mathbf{M}}_V(n,k,u) = \overline{\hat{\mathbf{V}}^{(n,k,u)}}$, é a função que associa um ponto $(n,k,u) \in I_U$ com a média amostral da complexidade dos verificadores construídos para o conjunto $\hat{\mathcal{A}}_v^{(n,k,u)}$ de autômatos válidos gerados aleatoriamente com distribuição uniforme.

5.3.1 Modelo experimental da complexidade média do verificador

De acordo com a discussão realizada na seção 5.2.1 e com os resultados experimentais mostrados na figura 5.7b, o tamanho amostral $N = 10000$ garante que $\hat{\mathbf{M}}_V(n,k,u)$ é

uma estimativa bastante próxima do valor esperado $\mathbf{M}_V(n,k,u)$ da complexidade na construção de verificadores (com um erro típico de 0.1%). Com base nesse fato, vamos quantificar o crescimento de $\hat{\mathbf{M}}_V(n,k,u)$ para propor um modelo experimental da complexidade média na construção de verificadores. A estratégia para quantificar o crescimento de $\hat{\mathbf{M}}_V(n,k,u)$ é realizar uma regressão não linear de mínimos quadrados sobre os contornos $\hat{\mathbf{M}}_V(n,k_0,u_0)$ (sendo k_0 e u_0 valores fixos), usando-se o seguinte modelo de regressão não linear (Fox and Weisberg, 2011):

$$\hat{\mathbf{M}}_V(n,k_0,u_0) = f(n;(a,b)) + e_n \tag{5.22}$$

em que $f(n;(a,b))$ é o modelo com parâmetros a e b, e_n é o n-ésimo resíduo e $n \in I_{3:20}$ é o regressor. Para encontrar o modelo correspondente a cada contorno é usado um algoritmo numérico de minimização da seguinte soma ponderada dos resíduos quadráticos :

$$S(a,b) = \sum_{n \in I_{3:20}} w_n(\hat{\mathbf{M}}_V(n,k_0,u_0) - f(n;(a,b)))^2 \tag{5.23}$$

em que w_n é um parâmetro que serve para ponderar a qualidade do ajuste em diferentes pontos. A função $f(n;(a,b))$ na equação (5.22) é definida (dependendo do número de eventos não observáveis) da seguinte forma:

- Para o caso $u_0 = 1$, usa-se o modelo $f(n;(a,b)) = (n+a)^b$ para os 10 contornos $\hat{\mathbf{M}}_V(n,k_0,1)$ que correspondem, respectivamente, a cada valor $k_0 \in I_{2:10} \cup \{16\}$.
- Para o caso $u_0 = 2$ usa-se o modelo $f(n;(a,b)) = 2(n+a)^b$ para os 9 contornos $\hat{\mathbf{M}}_V(n,k_0,2)$ que correspondem, respectivamente, a cada valor $k_0 \in I_{3:10} \cup \{16\}$.

Tabela 5.3: Modelos de regressão dos contornos $\hat{\mathbf{M}}_V(n,k_0,u_0)$ para aproximar o crescimento da complexidade média do verificador

	$u_0 = 1$		$u_0 = 2$	
k_0	Modelo	$\mathcal{O}(\cdot)$	Modelo	$\mathcal{O}(\cdot)$
2	$(n+0.25)^{1.51}$	$n^{1.51}$	—	—
3	$(n-0.62)^{1.93}$	$n^{1.93}$	$2(n-0.88)^{1.92}$	$n^{1.92}$
4	$(n-0.01)^{1.97}$	$n^{1.97}$	$2(n-0.42)^{1.98}$	$n^{1.98}$
5	$(n+0.23)^{1.99}$	$n^{1.99}$	$2(n+0.19)^{1.99}$	$n^{1.99}$
6	$(n+0.33)^{1.99}$	$n^{1.99}$	$2(n+0.11)^{2.00}$	$n^{2.00}$
7	$(n+0.39)^{2.00}$	$n^{2.00}$	$2(n+0.06)^{2.00}$	$n^{2.00}$
8	$(n+0.42)^{2.00}$	$n^{2.00}$	$2(n+0.04)^{2.00}$	$n^{2.00}$
9	$(n+0.44)^{2.00}$	$n^{2.00}$	$2(n+0.03)^{2.00}$	$n^{2.00}$
10	$(n+0.45)^{2.00}$	$n^{2.00}$	$2(n+0.02)^{2.00}$	$n^{2.00}$
16	$(n+0.48)^{2.00}$	$n^{2.00}$	$2(n+0.00)^{2.00}$	$n^{2.00}$

A tabela 5.3 apresenta as curvas de regressão para os contornos $\hat{\mathbf{M}}_V(n, k_0, u_0)$ que foram calculados[6] usando um vetor $w_n = 1, n \in I_{3:20}$, isto é, com a mesma qualidade de ajuste em cada ponto da regressão. Por exemplo para $k_0 = 3$, na terceira fila da tabela 5.3, obtém-se os seguintes modelos: se $u = 1$ o modelo da regressão $(n - 0.62)^{1.93}$ que produz a estimativa de crescimento assintótico da complexidade média $n^{1.93}$ e se $u = 2$ o modelo encontrado é $2(n - 0.88)^{1.92}$ que implica uma estimativa assintótica da complexidade média dada por $n^{1.92}$. As figuras 5.8(a-f) mostram, usando uma escala logarítmica para ambos os eixos, o ajuste entre vários contornos $\hat{\mathbf{M}}_V(n, k_0, u_0)$ e seus modelos correspondentes (observe que cada modelo de regressão é representado graficamente com linha tracejada). Os valores de $n = 32$ e $n = 64$ correspondem a resultados experimentais utilizados para validar a predição de cada modelo sobre valores extrapolados diferentes dos usados na regressão. Conforme pode ser observado, a concordância entre cada contorno $\hat{\mathbf{M}}_V(n, k_0, u_0)$ e seu modelo correspondente é elevada.

Os resultados obtidos nos modelos de regressão para os contornos $\hat{\mathbf{M}}_V(n, k_0, u_0)$ indicam que para valores de $k_0 \geq 5$, a média da complexidade na construção de verificadores pode ser considerada quadrática. Tal fato está ilustrado nas figuras 5.9(a,b). No caso de um evento não-observável, a figura 5.9(a) é uma representação gráfica da superfície $\hat{\mathbf{M}}_V(n, k, 1), n \in I_{3:20} \cup \{32, 64\}, k \in I_{2:10} \cup \{16\}$ e, conforme esperado, essa superfície converge ao plano $(n + 0.48)^2$. No caso de dois ou mais eventos não-observáveis, a figura 5.9(b) é uma representação gráfica simultânea das superfícies $\hat{\mathbf{M}}_V(n, k, u), n \in I_{3:20} \cup \{32\}, k \in I_{2:10} \cup \{16\}, u \in \{2, \ldots, k-1)\}$ que são limitadas superiormente pelo plano $2n^2$.

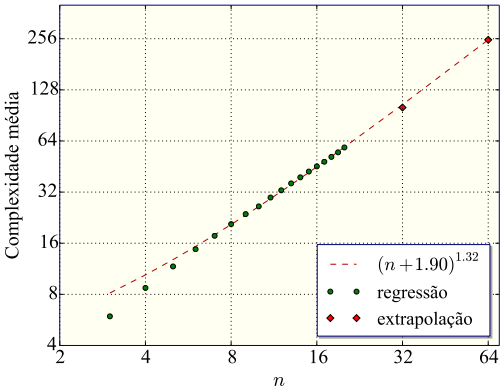
Em síntese, o crescimento assintótico da superfície $\hat{\mathbf{M}}_V(n, k, u), (n, k, u) \in I_U$ pode ser descrito pelo modelo n^2 (o fator linear de diferença deve ser desconsiderado como consequência do teorema da aceleração linear (Greenlaw and Hoover, 1998)). Com base nesse fato e levando em consideração que $\mathbf{M}_V(n, k, u)$ pertence ao intervalo de confiança (com nível de confiança de 95%)

$$[0.99\hat{\mathbf{M}}_V(n, k, u), 1.01\hat{\mathbf{M}}_V(n, k, u)],$$

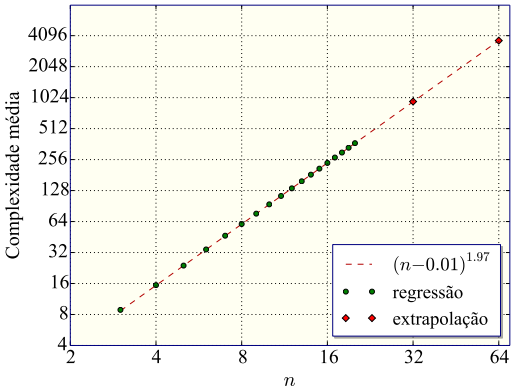
é possível formular o seguinte modelo experimental que descreve o comportamento assintótico da complexidade média na construção de verificadores para os conjuntos testados nesse experimento.

Modelo Experimental 5.1. *Para cada conjunto $\mathcal{A}_v^{(n,k,u)}, (n, k, u) \in I_U$ de autômatos válidos, a complexidade média na construção do verificador $G_{V,i}^{(n,k,u)}$ associado com o autômato $G_i^{(n,k,u)} \in \mathcal{A}_v^{(n,k,u)}$ é de ordem $\mathcal{O}(n^2)$ sendo n a cardinalidade do conjunto de estados do autômato $G_i^{(n,k,u)}$.* □

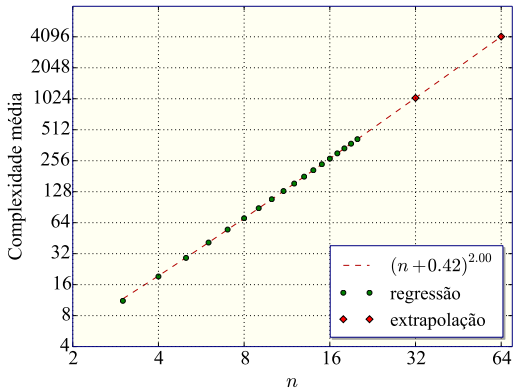
[6]Os cálculos foram realizados usando o pacote de cálculo estatístico *open source* R.



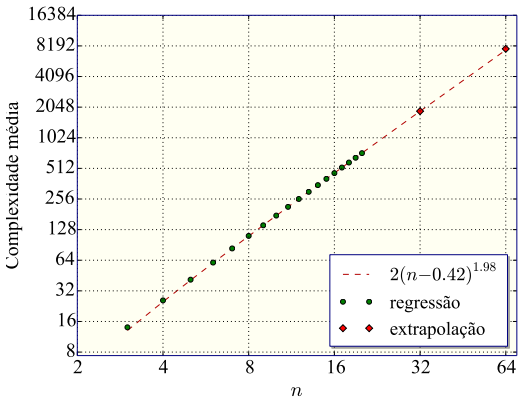
(a) $\hat{\mathbf{M}}_V(n, 3, 1)$ e modelo.



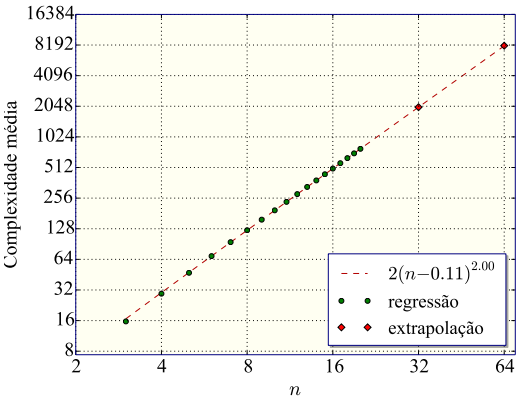
(b) $\hat{\mathbf{M}}_V(n, 4, 1)$ e modelo.



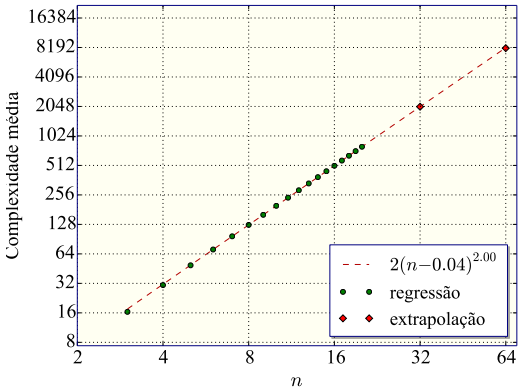
(c) $\hat{\mathbf{M}}_V(n, 8, 1)$ e modelo.



(d) $\hat{\mathbf{M}}_V(n, 4, 2)$ e modelo.



(e) $\hat{\mathbf{M}}_V(n, 6, 2)$ e modelo.



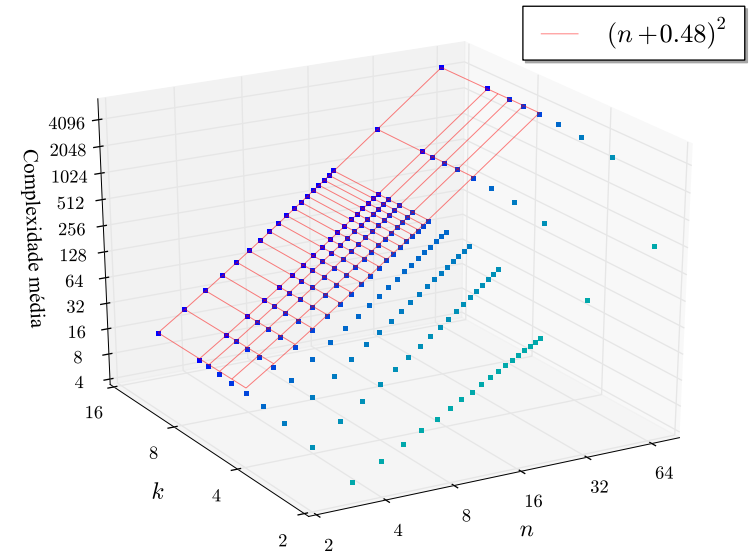
(f) $\hat{\mathbf{M}}_V(n, 8, 2)$ e modelo.

Figura 5.8: Regressão dos contornos $\mathbf{M}_V(n, k_0, u_0)$ para alguns valores de k_0 e u_0. Os pontos em verde são os valores usados para a regressão, a linha tracejada representa graficamente o modelo de regressão de cada contorno e os pontos em vermelho são valores experimentais usados para validar a extrapolação de cada modelo.

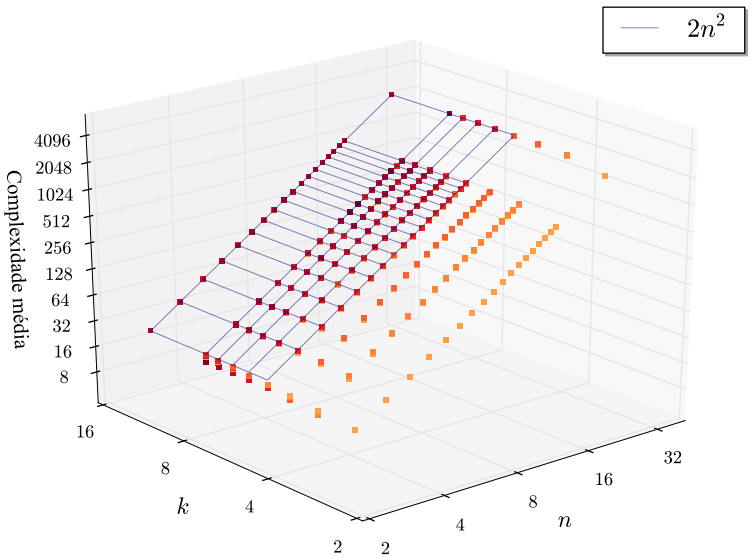
Conforme pode ser observado nas figuras 5.8 e 5.9(a,b), o modelo de crescimento n^2 ao ser extrapolado para valores de n e k fora dos intervalos de regressão foi também verificado. Embora o método experimental usado não permite caracterizar a

complexidade média na construção de verificadores para todo conjunto de autômatos válidos, a evidência experimental justifica a formulação da seguinte conjetura.

Conjectura 5.1. *Para cada conjunto* $\mathcal{A}_v^{(n,k,u)}$ *de autômatos válidos com* $n \geq 3, k \geq 2$ *e* $0 \leq u \leq k-1$*, o verificador* $G_{V,i}^{(n,k,u)}$ *associado com o autômato* $G_i^{(n,k,u)} \in \mathcal{A}_v^{(n,k,u)}$ *possui complexidade média de ordem quadrática em relação à cardinalidade do espaço de estados do autômato* $G_i^{(n,k,u)}$*.*



(a)



(b)

Figura 5.9: Convergência da superfície $M_v(n,k,1), n \in I_{3:20} \cup \{32,64\}, k \in I_{3:10} \cup \{16\}$ ao plano $(n+0.48)^2$ (a); Convergência da superfície $M_v(n,k,u), n \in I_{3:20} \cup \{32\}, k \in I_{3:10} \cup \{16\}, u \in \{2,\ldots,\max(k-1,8)\}$ ao plano $2n^2$ (b).

5.3.2 Modelo experimental da complexidade média na construção do diagnosticador

De acordo com as considerações da seção 5.2.1 (verificadas pelos resultados experimentais mostrados na figura 5.7a), o tamanho amostral $N = 10000$ garante que o valor esperado $\mathbf{M}_D(n, k, u)$ da construção de diagnosticadores pode ser estimado de forma confiável usando a média amostral da complexidade $\hat{\mathbf{M}}_D(n, k, u)$ dentro de um erro típico de 1%. Por essa razão, é possível usar a função $\hat{\mathbf{M}}_D(n, k, u)$ na formulação de um modelo experimental da complexidade na construção de diagnosticadores. Além disso, de acordo com os resultados obtidos nesse experimento, as médias amostrais da complexidade das instâncias testadas no experimento estão limitadas superiormente pela superfície $\hat{\mathbf{M}}_D(n, k, 1)$, isto é:

$$\hat{\mathbf{M}}_D(n, k, k-1) \leq \cdots \leq \hat{\mathbf{M}}_D(n, k, 2) \leq \hat{\mathbf{M}}_D(n, k, 1). \tag{5.24}$$

Dessa forma, a maior média amostral da complexidade aparece nos conjuntos $\mathcal{A}_v^{(n,k,1)}$ formados por todos os autômatos válidos que possuem um único evento não-observável (a figura 5.5(a) ilustra esse fato para valores de $\hat{\mathbf{M}}_D(n, k, u), (n, k, u) \in I_1$). Com base nesse fato, é possível usar a superfície $\hat{\mathbf{M}}_D(n, k, 1)$ para caracterizar a complexidade média na construção de diagnosticadores, uma vez que essa superfície limita superiormente as médias amostrais da complexidade de conjuntos de autômatos com um número maior de eventos não observáveis. De forma similar ao caso do verificador, a estratégia usada para estimar o crescimento da complexidade é usar o modelo de regressão não-linear

$$\hat{\mathbf{M}}_D(n, k_0, 1) = (n + a)^b + e_n, \tag{5.25}$$

em que a e b, e_i são os parâmetros do modelo, e_n é o n-ésimo resíduo e $n \in I_{3:20}$ é o regressor. Os parâmetros do modelo são estimados minimizando-se a seguinte soma ponderada dos resíduos quadráticos:

$$S(a, b) = \sum_{n \in I_{3:20}} w_n (\hat{\mathbf{M}}_D(n, k_0, 1) - (n + a)^b)^2,$$

em que a escolha dos pesos é $w_n = n, n \in I_{3:20}$, uma vez que o intuito do modelo é identificar o crescimento assintótico das curvas $\hat{\mathbf{M}}_D(n, k_0, 1)$, sendo menos importante o ajuste do modelo em valores pequenos de n.

A tabela 5.4 apresenta os modelos de regressão encontrados para os contornos $\hat{\mathbf{M}}_D(n, k_0, 1), k_0 \in I_{2,10} \cup \{16, 20, 24\}$. Por exemplo, na quinta fila da tabela 5.4, para um valor de $k_0 = 6$ o modelo encontrado é $(n + 1.61)^{2.00}$ que implica numa estimativa de crescimento assintótico $n^{2.00}$. As figuras 5.8(a-f) mostram, usando uma

escala logarítmica para ambos eixos, o ajuste de alguns dos contornos $\hat{\mathbf{M}}_D(n, k_0, u_0)$ mostrados na tabela 5.4 com seus respectivos modelos (note que cada modelo de regressão é representado graficamente em linha tracejada). Conforme pode ser observado nas figuras 5.8(a-f) o ajuste de cada modelo é melhor para valores altos de n, sendo isso uma consequência da seleção de pesos $w_n = n$, que permite estimar com melhor aproximação, o comportamento assintótico de cada contorno. Observe, ainda, que os valores $n = 32$ e $n = 64$ usados para validar a extrapolação do modelo permanecem limitados superiormente pelos modelos de regressão obtidos.

Tabela 5.4: Modelos para a estimação da complexidade média do diagnosticador

k_0	Modelo	$\mathcal{O}(\cdot)$
2	$(n + 4.72)^{0.62}$	$n^{0.62}$
3	$(n + 1.90)^{1.32}$	$n^{1.32}$
4	$(n + 2.29)^{1.63}$	$n^{1.63}$
5	$(n + 2.06)^{1.84}$	$n^{1.84}$
6	$(n + 1.61)^{2.00}$	$n^{2.00}$
7	$(n + 1.36)^{2.13}$	$n^{2.13}$
8	$(n + 0.95)^{2.24}$	$n^{2.24}$
9	$(n + 0.62)^{2.34}$	$n^{2.34}$
10	$(n + 0.30)^{2.42}$	$n^{2.42}$
16	$(n - 1.20)^{2.76}$	$n^{2.76}$
24	$(n - 2.49)^{3.05}$	$n^{3.05}$

A figura 5.11(a) mostra o exponente b do primeiro termo da equação (5.25) em função do número de eventos k_0 para cada modelo obtido na tabela 5.4. Para aproximar o crescimento do exponente em função do número de eventos é usado o seguinte modelo linear:

$$b = c_1 \log k_0 + c_2.$$

em que c_1 e c_2 são os parâmetros do modelo e k_0 (o número de eventos) é o regressor. Realizando a regressão linear com base nos valores de b da tabela 5.4, obtém-se o seguinte modelo:

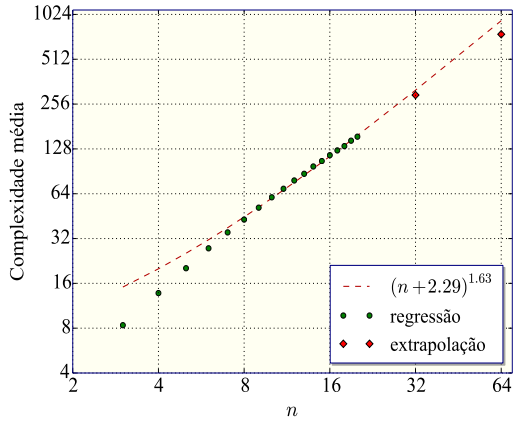
$$b = 0.77 \log k_0 + 0.63,$$

a partir do qual obtém-se a função

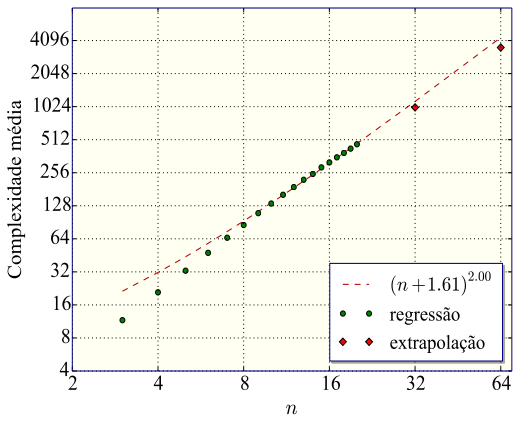
$$g(n, k) = n^{0.77 \log k + 0.63}, \tag{5.26}$$

que representa uma estimativa do crescimento da superfície $\hat{\mathbf{M}}_D(n, k, 1), n \in I_{3:20} \cup \{32, 64\}, k \in I_{2:10} \cup \{16, 24, 32\}$ conforme mostrado na figura 5.11(b).

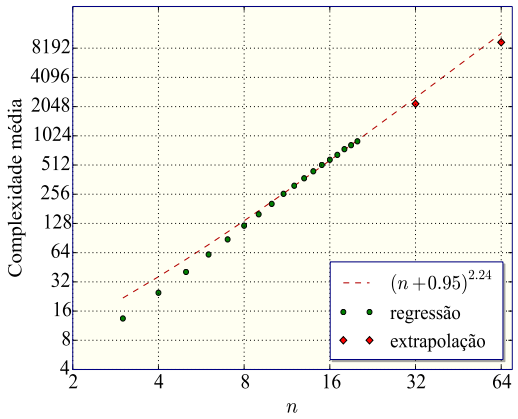
O resultado experimental descrito pela desigualdade (5.24) e o fato estatístico de



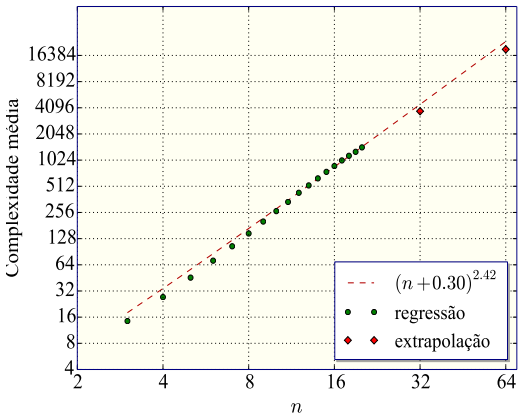
(a) $\hat{\mathbf{M}}_D(n, 4, 1)$ e modelo.



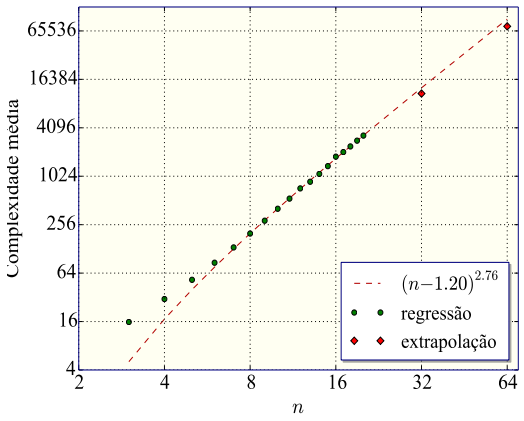
(b) $\hat{\mathbf{M}}_D(n, 6, 1)$ e modelo.



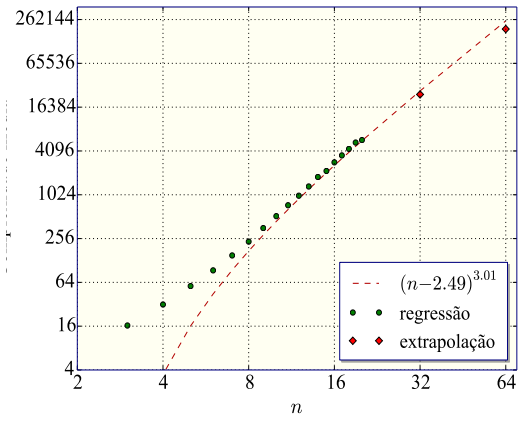
(c) $\hat{\mathbf{M}}_D(n, 8, 1)$ e modelo.



(d) $\hat{\mathbf{M}}_D(n, 10, 1)$ e modelo.

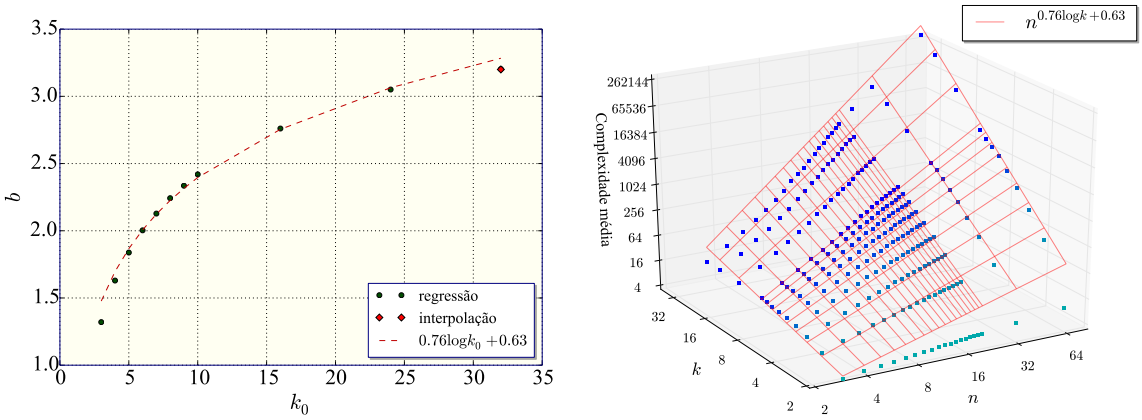


(e) $\hat{\mathbf{M}}_D(n, 16, 1)$ e modelo.



(f) $\hat{\mathbf{M}}_D(n, 24, 1)$ e modelo.

Figura 5.10: Regressão dos contornos $\mathbf{M}_D(n, k_0, u_0)$ para alguns valores de k_0 e u_0. Os pontos em verde são os valores usados para a regressão, a linha tracejada é a representação gráfica do modelo de regressão de cada contorno e os pontos em vermelho são os valores experimentais usados para validar a extrapolação de cada modelo.



(a) (b)

Figura 5.11: Curva do exponente b e o modelo de regressão em relação ao número de eventos (a); $\mathbf{M}_D(n,k,1)$ e superficie $n^{0.77\log k+0.63}$ (b).

que $M_D(n,k,u)$ pertence ao intervalo de confiança $[0.99\hat{M}_D(n,k,u), 1.01\hat{M}_D(n,k,u)]$ (com 95% de probabilidade), permitem concluir que o modelo de crescimento assintótico da equação (5.26) pode ser considerado confiavelmente como um limite superior da complexidade média para todas todos os conjuntos de autômatos válidos $\mathcal{A}_v^{(n,k,u)}, (n,k,u) \in I_U$ considerados nesse experimento e, por conseguinte, é possível formular o seguinte modelo experimental para descrever o comportamento assintótico da complexidade média na construção de diagnosticadores.

Modelo Experimental 5.2. *Para cada conjunto $\mathcal{A}_v^{(n,k,u)}, (n,k,u) \in I_U$ de autômatos válidos, a complexidade média do diagnosticador $G_{D,i}^{(n,k,u)}$ associado com o autômato $G_i^{(n,k,u)} \in \mathcal{A}_v^{(n,k,u)}$ é de ordem $\mathcal{O}(n^{0.77\log k+0.63})$ sendo n e k, respectivamente, as cardinalidades do conjunto de estados e de eventos do autômato $G_i^{(n,k,u)}$.* □

Conforme ilustrado na figura 5.11(b), o modelo de crescimento da equação (5.26) pode ser extrapolado para valores de n e k fora dos intervalos de regressão. No entanto, ao usar uma abordagem experimental, não é possível afirmar categoricamente que o modelo proposto caracteriza a complexidade média para todo conjunto $\mathcal{A}_v^{(n,k,u)}$ de autômatos válidos e, em consequência, o modelo experimental 5.2 é valido unicamente para os conjuntos de autômatos válidos testados. Porém, existe evidencia experimental que permite formular a seguinte conjetura sobre a complexidade média de na construção de diagnosticadores para diagnose de falhas:

Conjectura 5.2. *Para cada conjunto $\mathcal{A}_v^{(n,k,u)}$ de autômatos válidos com $n \geq 2, k \geq 2, 0 \leq u \leq k-1$, o diagnosticador $G_{D,i}^{(n,k,u)}$ associado com o autômato $G_i^{(n,k,u)} \in \mathcal{A}_v^{(n,k,u)}$ possui complexidade média de ordem $\mathcal{O}(n^{0.77\log k+0.63})$ sendo n e k, respectivamente, as cardinalidades dos conjuntos de estados e de eventos.* □

5.4 Comentários finais

O objetivo fundamental deste capítulo foi realizar um estudo da complexidade média do diagnosticador e do verificador por meio de um experimento com amostragem realizado sobre os conjuntos $\mathcal{A}^{(n,k,u)}, (n,k,u) \in I_U$ de autômatos válidos. Os resultados fundamentais que foram obtidos a partir do experimento levaram aos modelos experimentais 5.1 e 5.2.

É importante ressaltar que, conforme foi sinalado por McGeoch (2012), o objetivo central do estudo dos algoritmos usando algoritmia experimental é obter compreensão sobre a complexidade dos algoritmos estudados e formular hipóteses quantitativas da complexidade que possam ser estendidas (para o nosso caso, continuando a exploração computacional para conjuntos de autômatos válidos cada vez maiores) ou validadas posteriormente (criando uma teoria explicativa do comportamento observado) de acordo com o método científico.

Capítulo 6

Diagnosticabilidade e modelagem de falhas intermitentes na operação de sensores em sistemas a eventos discretos

Os sensores desempenham um papel vital em qualquer sistema realimentado. Dessa forma, a confiabilidade e segurança de um sistema dependem, em última instância, da operação adequada dos seus sensores. Falhas em sensores têm sido reportadas como causa de vários acidentes, produzindo perdas materiais e, até mesmo, de vidas (da Silva et al., 2012). Por essa razão, é importante encontrar formas de se distinguir apropriadamente entre o mau funcionamento de um sensor e seu funcionamento adequado ou normal.

Existem três métodos fundamentais para tratar o problema de se lidar com leituras incorretas nos sensores (Frank, 1990): *(i)* uso de redundância de *hardware* com votação majoritária; *(ii)* métodos baseados em modelos e, *(iii)* métodos baseados em conhecimento. A redundância é a forma mais simples de melhorar a confiabilidade de um sensor e consiste na adição de sensores redundantes que possuem a mesma função, sendo a leitura adotada decidida por votação majoritária. O *projeto baseado em modelo* depende de um modelo desenvolvido para o sistema considerado em que a decisão é tomada a partir de comparações entre as saídas do modelo e do sistema real. O *projeto baseado em conhecimento* usa técnicas de inteligência computacional tais como redes neurais, lógica nebulosa e sistemas especialistas.

Entre os métodos baseados em modelos, os trabalhos mais relevantes reportados na literatura são: o trabalho preliminar de Clark (1978) que propôs o assim denominado esquema de observador dedicado[1] (DOS) no qual são projetados observadores

[1] O termo original é *dedicated observer scheme*.

dedicados separados para cada um dos sensores (as entradas dos observadores são os sinais de saída dos sensores correspondentes e o sinal de entrada da planta); o artigo de Frank (1990) que além de apresentar uma revisão da literatura também melhora o esquema desenvolvido em Clark (1978), conduzindo ao assim denominado *esquema de observador generalizado* (GOS), no qual mais de uma saída é disposta como entrada dos observadores; Lunze and Schröder (2004) que propuseram um método para detecção e identificação de falhas em sensores e atuadores usando a teoria de sistemas a eventos discretos e modelando a planta como um autômato estocástico e, Ding et al. (2004) que apresentaram um esquema baseado em modelo para o monitoramento dos sensores no sistema de controle de estabilidade dinâmica veicular (ESP) formado por um sistema de freio antibloqueio, um controle de tração e um controle de torque de direção.

Uma abordagem de sistemas especialistas foi proposta por Athanasopoulou and Chatziathanasiou (2009) que desenvolveram um sistema inteligente para identificação e substituição de medições derivadas de sensores com falhas em plantas de geração termoelétrica baseado em um procedimento para identificar as falhas e reconstruir as medições erradas. Como abordagem alternativa baseada em conhecimento, da Silva et al. (2012) propuseram um sistema de diagnóstico de falhas de sensores por meio de redes neurais artificiais.

Neste capítulo iremos propor uma abordagem baseada na teoria de sistemas a eventos discretos para o problema de diagnosticar falhas intermitentes de sensores modelando a planta como um autômato determinístico. O sistema de diagnose de falhas na operação de sensores deve ser construído separadamente do sistema que diagnostica as falhas comuns do sistema e do sistemas de controle supervisório, conforme mostrado na figura 6.1.

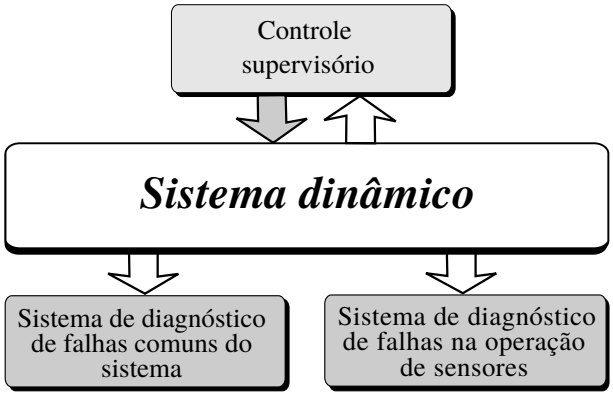


Figura 6.1: Diagrama esquemático dos sistemas de controle supervisório e de diagnose de falhas usando modelos de sistemas a eventos discretos.

Falhas em sensores foram abordadas no contexto de controle supervisório por Rohloff (2005), Sanchez and Montoya (2006) e Alves et al. (2014); e no contexto

de diagnose de falhas de SEDs por Carvalho et al. (2012, 2013b). Diferentemente das abordagens feitas nos trabalhos acima, neste trabalho não estamos propondo um sistema que possa remediar o problema de falhas em sensores mas um sistema que realmente detecta o mau funcionamento de um sensor. Para tanto, o modelo de perda de observações introduzido em Carvalho et al. (2012) será modificado limitando-o ao problema de detetar o mau funcionamento de um sensor e, dessa maneira, converter o problema de detetar falhas na operação de sensores em um problema equivalente de diagnose de falhas intermitentes conforme proposto no trabalho de Contant et al. (2004). Serão apresentadas condições necessárias e suficientes para a diagnosticabilidade de falhas intermitentes sensores, além disso, será proposto um teste baseado em um autômato diagnosticador para verificar a diagnosticabilidade de falhas intermitentes na operação de sensores.

Este capítulo está dividido da seguinte forma. Na seção 6.1 apresentamos uma modificação do modelo de perdas intermitentes de observação usando autômatos que foi desenvolvido em Carvalho et al. (2012). O intuito da modificação realizada é lidar, expressamente, com falhas na operação de sensores. Utilizando o modelo introduzido na seção 6.1, o objetivo da seção 6.2 é apresentar a forma em que o problema de diagnose de falhas na operação de sensores pode ser transformado num problema equivalente de diagnosticar falhas intermitente, no qual o evento de falha a ser diagnosticado é o evento registrado por um sensor cuja possível falha deseja-se diagnosticar. Na seção 6.3 apresentamos as condições necessária e suficiente para a diagnose de falhas intermitentes na operação de sensores e, em seguida, um teste (baseado no autômato diagnosticador) para verificar a diagnosticabilidade de uma linguagem em presença de falhas intermitentes na operação de sensores. A seção 6.4 apresenta dois exemplos que ilustram os resultados mais importantes do capítulo. Finalmente, na seção 6.5 são apresentados os comentários finais.

6.1 Modelagem de SEDs sujeitos a falhas intermitentes na operação de sensores

Nesta seção apresentaremos dois modelos de sensores usando autômatos: o primeiro modelo considera o funcionamento ideal de um sensor e o segundo modela o comportamento do sensor estando sujeito a falhas intermitentes. A hipótese subjacente para a formulação desses modelos é que o evento a ser registrado não é observável do ponto de vista do sistema.

Para começar a discussão, considere um autômato determinístico

$$G = (X, \Sigma, f, \Gamma, x_0, X_m),$$

sendo que o conjunto Σ de eventos é particionado como $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ em que Σ_o e Σ_{uo} representam, respetivamente, o conjunto de eventos observáveis e não-observáveis. Seja $\Sigma_{isf} = \{\sigma\} \subset \Sigma_o$, defina o conjunto $\Sigma_{dil} = \Sigma \cup \Sigma'_{isf}$ em que $\Sigma'_{isf} = \{\sigma' : \sigma \in \Sigma_{isf}\}$ e forme o seguinte conjunto: $\Sigma_{dil} = \Sigma \cup \Sigma'_{isf}$. No contexto do estudo da diagnosticabilidade de falhas intermitentes na operação de sensores é importante a operação de dilatação D (Carvalho et al., 2012) definida a seguir.

Definição 6.1. *(Dilatação) A dilatação D é definida pelo mapeamento*

$$\begin{array}{rcccl} D & : & \Sigma^* & \rightarrow & 2^{\Sigma^*_{dil}}, \\ & & s & \mapsto & D(s), \end{array} \tag{6.1}$$

em que

$$\begin{array}{rcl} D(\varepsilon) & = & \{\varepsilon\}, \\ D(\sigma) & = & \left\{ \begin{array}{l} \{\sigma\}, \ se\ \sigma \in \Sigma \backslash \Sigma_{isf}, \\ \{\sigma, \sigma'\}, \ se\ \sigma \in \Sigma_{isf}, \ com\ \sigma' \in \Sigma'_{isf}, \end{array} \right. \\ D(s\sigma) & = & D(s)D(\sigma), s \in \Sigma^*, \sigma \in \Sigma. \end{array} \tag{6.2}$$

Note que a operação de dilatação D pode ser estendida naturalmente de sequências para linguagens aplicando-a a todas as sequência da linguagem, isto é,

$$L_{dil} = D(L) = \bigcup_{s \in L} D(s).$$

6.1.1 Modelagem de falhas intermitentes de sensores usando autômatos

A figura 6.2(a) mostra um autômato S_{ideal} modelando o comportamento de um sensor ideal. Para esse autômato, a_u denota o evento não-observável cuja ocorrência deve ser registrada pelo sensor e a_r denota o evento que representa a leitura do sensor quando esse registra a ocorrência do evento a_u. A figura 6.2(b) mostra o autômato S_{isf} modelando o comportamento de um sensor sujeito a falhas intermitentes ao registrar a ocorrência de a_u. A diferença essencial entre o autômato S_{ideal} e o autômato S_{isf} é a inclusão da transição rotulada pelo evento a_f, representando a ocorrência de uma falha na operação do sensor, isto é, o sensor deixa de registrar a ocorrência do evento a_u. Os auto-laços existentes nos estados iniciais das figuras 6.2(a) and 6.2(b) foram adicionados para garantir que outros eventos da planta, diferentes de a_u, cujas ocorrências também devem ser registradas por outros sensores (ou eventos de comando), possam ocorrer. Note que o conjunto de eventos da planta ainda é representado por Σ e, portanto, contém o evento a_u. Note também que os únicos eventos que podem ocorrer após da ocorrência de a_u são a_r ou a_f, uma vez que são eventos privados dos autômatos S_{ideal} e/ou S_{isf}. O comportamento do sistema, considerando que o sensor esteja sujeito a falhas intermitentes na sua

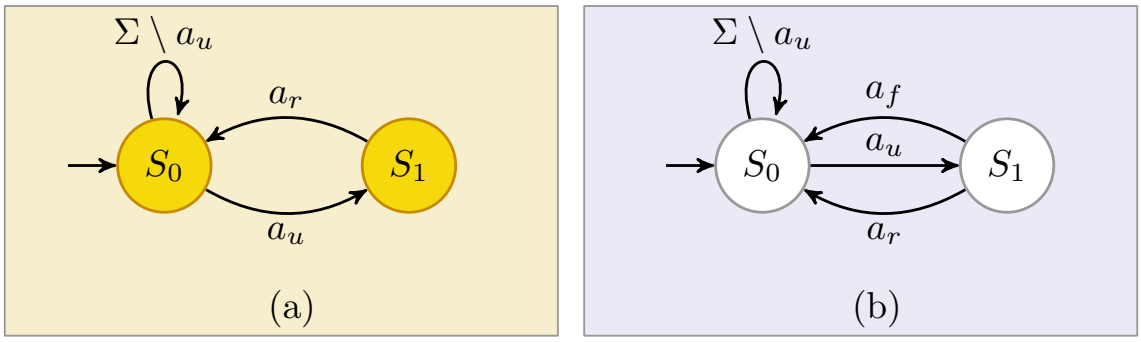


Figura 6.2: Autômatos modelando o comportamento de sensores: autômato modelando o comportamento de um sensor ideal (S_{ideal})(a); autômato que modela um sensor sujeito a falhas intermitentes (S_{isf})(b).

operação, pode ser obtido realizando a composição paralela entre o autômato G que modela a planta e o autômato S_{isf}.

Com o propósito de ilustrar a influência de falhas na operação de sensores no modelo da planta, considere o diagrama de transição de estados de parte do modelo de uma planta mostrado na figura 6.3(a). Realizando a composição paralela entre G e S_{isf}, obtemos o autômato $G_{dil} = G \parallel S_{isf}$ mostrado na figura 6.3(b). Observe que a ocorrência de a_u determina as seguintes sequências: $a = a_u a_r$ e $a' = a_u a_f$. Note que a sequência a termina com um evento observável (o evento correspondente ao registro do sensor) enquanto os dois eventos de a' são não-observáveis. Dessa forma, a e a' podem ser associadas, respetivamente, a dois eventos: um evento observável a associado à ocorrência de a_u e o evento a' representando que o sensor falhou em registrar a ocorrência de a_u. Como consequência, o modelo de G da figura 6.3(a) pode ser substituído pelo modelo da figura 6.3(c) ao se levar em conta possíveis falhas intermitentes na operação do sensor associado ao evento a_u.

É importante ressaltar que o modelo descrito na figura 6.3(c) é idêntico ao modelo proposto por Carvalho et al. (2012), no contexto de diagnose de falhas, para abordar o problema de perdas intermitentes de observação. No entanto, diferentemente do modelo proposto em Carvalho et al. (2012), não estamos considerando os eventos de falha em G. Outra diferença importante entre o problema abordado aqui e o problema resolvido em Carvalho et al. (2012) é que em Carvalho et al. (2012), as falhas são permanentes e os sensores estão sujeitos a perdas intermitentes. No modelo proposto aqui, por outro lado, o problema de diagnose de falhas é formulado de tal forma que no modelo final todos os sensores associados a eventos diferentes do sensor com falha são considerados ideais e a falha a ser considerada é *intermitente*, entendendo o termo *intermitente* da seguinte forma: o evento a ser registrado pode ocorrer e ser observado; pode ocorrer e não ser observado; e, finalmente, pode ocorrer e não ser observado e, caso ocorra novamente, ser ou não observado.

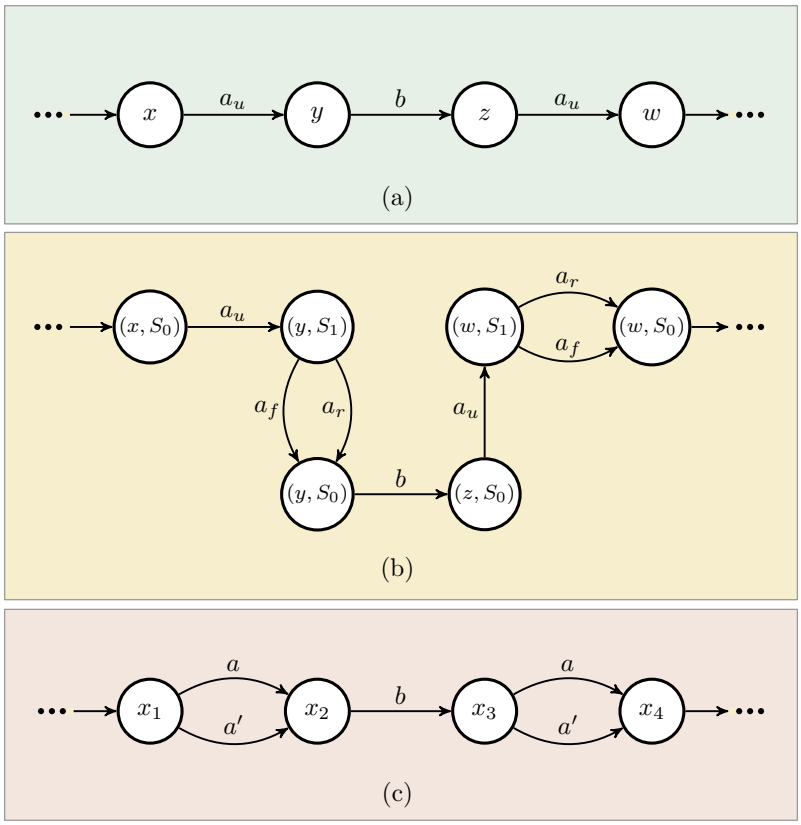


Figura 6.3: Parte do modelo de uma planta modelada pelo autômato G (a); autômaton $G_{dil} = G \parallel S_{isf}$ (b), e modelo de G considerando falhas intermitentes na operação de sensores (c).

6.1.2 Modelo por autômatos de SEDs sujeitos a falhas intermitentes na operação de sensores

Não é difícil verificar (Carvalho et al., 2012) que o autômato G_{dil}, que leva em conta as falhas intermitentes na operação de sensores, pode ser obtido a partir de G, adicionando-se transições rotuladas por $\sigma' \in \Sigma'_{isf}$ em paralelo com cada transição rotulada por σ. Assim, o autômato G_{dil} pode ser definido como:

$$G_{dil} = (X_{dil}, \Sigma_{dil}, f_{dil}, \Gamma_{dil}, x_{0,dil}), \tag{6.3}$$

em que $X_{dil} = X$, $x_{0,dil} = x_0$, $\Gamma_{dil}(x) = \Gamma(x) \cup \{\sigma'\}$, se $\sigma \in \Gamma(x)$ ou $\Gamma_{dil}(x) = \Gamma(x)$, caso contrário, $f_{dil}(x, \sigma) = f(x, \sigma), \forall \sigma \in \Gamma(x)$ e $f_{dil}(x, \sigma') = f(x, \sigma), \forall \sigma' \in \Gamma_{dil}(x)$. Conforme provado em Carvalho et al. (2012), a linguagem gerada por G_{dil} (L_{dil}) é dada por:

$$L_{dil} = D(L),$$

em que $D(\cdot)$ denota operação de dilatação introduzida na definição 6.1.

6.2 Diagnosticabilidade de falhas intermitentes na operação de sensores

O autômato A_ℓ, representado na figura 6.4, modela o comportamento dinâmico de um sensor sujeito a falhas intermitentes, em que $\sigma \in \Sigma_{isf}$ é o evento registrado pelo sensor cujo mau funcionamento desejamos detectar e $\sigma' \in \Sigma'_{isf}$ é o correspondente evento de falha associado a σ. Note que, enquanto o sensor está registrando corretamente a ocorrência do evento σ, podemos dizer que o sensor tem um comportamento normal (representado pelo estado inicial N de A_ℓ). No entanto, quando ocorre uma falha do sensor, o autômato A_ℓ passa para o estado F e permanece nele enquanto o sensor continuar falhando. Se o sensor voltar a funcionar novamente ou, dito de outra forma, se o evento σ ocorrer novamente, o autômato A_ℓ passará para o estado R e permanecerá nele enquanto o sensor continuar registrando a ocorrência de σ. A última consideração do modelo é que se em algum momento o sensor falhar novamente no registro da ocorrência de σ, o autômato A_ℓ retornará ao estado F.

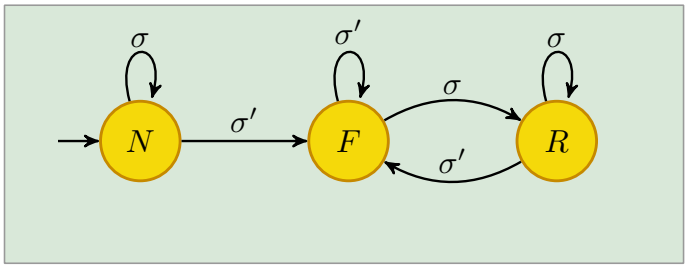


Figura 6.4: Autômato A_ℓ modela o comportamento dinâmico das falhas intermitentes nos sensores.

A detecção da falha na operação de um sensor pode ser efetuada detectando-se a ocorrência do evento σ'. Como o evento σ' não é um evento de G, a definição da diagnosticabilidade de falhas na operação de sensores não pode ser formulada em termos de L e sim em termos de L_{dil}. Por conseguinte, embora o modelo usado aqui para levar em conta falhas na operação de sensores seja uma modificação do modelo apresentado em Carvalho et al. (2012), a definição de diagnosticabilidade não pode ser formulada de forma semelhante àquela introduzida em Carvalho et al. (2012). Isso decorre do fato de que o evento de falha considerado em Carvalho et al. (2012) era, de fato, um evento de G e a operação de dilatação foi usada para produzir diferentes possibilidades de sequências que envolviam ou não falhas na operação de sensores. Aqui a dilatação de L produz a linguagem gerada pelo autômato que é, segundo descrito acima, o modelo proposto para levar em conta falhas intermitentes na operação de sensores; e, embora algum evento de falha possa aparecer no modelo do sistema, não estamos interessados em detectar sua ocorrência. A dilatação também possibilita que σ possa ocorrer, mesmo após a ocorrência de

σ', caso o sensor recobre seu funcionamento normal.

Agora formularemos as seguintes hipóteses:

A1. A linguagem gerada por G é viva, isto é , $\Gamma(x_i) \neq \emptyset$ para todo $x_i \in X$;

A2. Existe somente um tipo de sensores com falha intermitente na operação, isto é, $\Sigma_{isf} = \{\sigma\}$ and $\Sigma'_{isf} = \{\sigma'\}$.

Suponha que s_f denote o último evento de uma sequência s. Então, $\Psi(\Sigma'_{isf}) = \{s \in L : s_f \in \Sigma'_{isf}\}$ é conjunto de todas as sequências de L que terminam com o evento σ_f e $L/s = \{t \in \Sigma^* : st \in L\}$ a continuação da linguagem de L após uma sequência s. Suponha ainda que $\overline{s}$ denote o fecho de prefixo de s. Com um ligeiro abuso de notação, a relação de pertinência $\Sigma'_{isf} \in s$ é usada para denotar que $\overline{s} \cap \Psi(\Sigma'_{isf}) \neq \emptyset$. Assim, podemos dizer que $s \in L$ é uma sequência que contém uma falha, se $\Sigma'_{isf} \in s$. A seguir apresentamos as definições de *diagnosticabilidade-F* e *diagnosticabilidade-R*.

Definição 6.2. *(diagnosticabilidade-F e diagnosticabilidade-R) Seja L_{dil} a linguagem gerada pelo autômato G_{dil} (o autômato que modela o comportamento da planta em presença de falhas intermitentes na operação de sensores). Então:*

- *L_{dil} é diagnosticável-F em relação à projeção $P_{dil,o} : \Sigma^*_{dil} \to \Sigma^*_o$ e Σ'_{isf}, se a seguinte condição for verificada:*

$$(\exists n \in \mathbb{N})(\forall s \in \Psi(\Sigma'_{isf}))(\forall t \in L_{dil}/s, \Sigma_{isf} \notin t)(\|t\| \geq n) \Rightarrow D_F,$$

sendo a condição de diagnosticabilidade D_F expressa por:

$$(\forall \omega = \omega'\omega'' \in P^{-1}_{dil,o}(P_{dil,o}(st)) \cap L_{dil})((\omega' \in \Psi(\Sigma'_{isf})) \wedge (\Sigma'_{isf} \cup \Sigma_{isf} \notin \omega'')).$$

- *L_{dil} é diagnosticável-R em relação à projeção $P_{dil,o} : \Sigma^*_{dil} \to \Sigma^*_o$, Σ'_{isf} e Σ_{isf}, se a seguinte condição for verificada:*

$$\begin{aligned}&(\exists n \in \mathbb{N})(\forall r \in \Psi(\Sigma'_{isf}))(\forall s \in L_{dil}/r, s \in \Psi(\Sigma_{isf}))\\ &(\forall t \in L_{dil}/rs, \Sigma'_{isf} \notin t)(\|t\| \geq n) \Rightarrow D_R,\end{aligned}$$

sendo a condição de diagnosticabilidade D_R expressa por:

$$(\forall \omega = \omega'\omega'' \in P^{-1}_{dil,o}(P_{dil,o}(rst)) \cap L_{dil})((\omega' \in \Psi(\Sigma'_{isf})) \wedge (\Sigma'_{isf} \notin \omega'') \wedge (\Sigma_{isf} \in \omega'')).$$

Observação 6.1. *A ideia subjacente nas definições de diagnosticabilidade-F e diagnosticabilidade-R, apresentadas acima, é a mesma das definições de diagnosticabilidade de falhas intermitentes formulada por Contant et al. (2004); a diferença*

principal reside no fato de que a definição apresentada em Contant et al. (2004), além de requerer um evento de falha, precisa de outro evento não-observável denominado evento de "reset". Porém, acima de tudo, como foi indicado por Contant et al. (2004), ambas as noções de diagnosticabilidade (a que foi apresentada em Contant et al. (2004) e a formulação dada na definição 6.2) são "extensões naturais da noção de diagnosticabilidade de falhas permanentes introduzida em Sampath et al. (1995)"

Observação 6.2. *No trabalho de Contant et al. (2004) supõe-se que não existem caminhos cíclicos de eventos não-observáveis em G e, além disso, todos os resultados obtidos em Contant et al. (2004) fazem essa hipótese. Essa hipótese é removida aqui pela seguinte razão: embora fosse considerado que G não possui caminhos cíclicos de eventos não observáveis, esses tipos de caminhos ainda poderiam aparecer em G_{dil}, tendo em vista que a dilatação introduz uma transição rotulada por σ' (sendo este um evento não-observável) em paralelo com σ (sendo este um evento observável). Assim, poderia ocorrer que σ fosse o único evento observável em um caminho cíclico s e que, em consequência, aparecesse um caminho adicional de eventos não observáveis s' em G_{dil} com os mesmos eventos de s, exceto com σ substituído por σ'. Dessa forma, todos os testes propostos em Contant et al. (2004) não podem ser aplicados ao problema considerado aqui e, consequentemente, é necessário um desenvolvimento adicional levando em conta os caminhos cíclicos com eventos não-observáveis.*

6.3 Verificação da diagnosticabilidade de falhas intermitentes em sensores usando diagnosticadores

Nesta seção vamos propor um teste para verificar a diagnosticabilidade de falhas intermitentes em sensores usando o autômato diagnosticador. Deve ser observado que diagnosticadores servem também para realizar a diagnose online. Com esse intuito, vamos inicialmente construir o autômato

$$G_{dil,\ell} = G_{dil} \| A_\ell, \tag{6.4}$$

sendo A_ℓ o autômato rotulador da figura 6.4. Note que os estados de $G_{dil,\ell}$ são obtidos adicionando-se rótulos N, F ou R aos estados da planta para indicar, respectivamente, se o sensor não falhou (ou equivalentemente, se o sensor está em operação normal), se falhou, ou, se falhou e recuperou da falha. Denote por $L_{dil,\ell}$ e $\Sigma_{dil,\ell}$ a linguagem gerada e o conjunto de evento de G, respectivamente. Não é difícil verificar que $L_{dil,\ell} = L_{dil}$ uma vez que $\Sigma_{dil,\ell} = \Sigma_{dil}$.

O fato de que a diagnosticabilidade se baseia unicamente em eventos observáveis,

sugere o cálculo do seguinte autômato:

$$G_{dil,d} = (X_{dil,d}, \Sigma_{dil,d}, f_{dil,d}, \Gamma_{dil,d}, x_{0_{dil,d}}) = Obs\,(G_{dil,\ell}), \tag{6.5}$$

sendo $\Sigma_{dil,d} = \Sigma_o$. Note que novas combinações de rótulos podem aparecer nos estados do diagnosticador, tornando necessário introduzir a seguinte classificação[2].

Definição 6.3. *Um estado $x_{dil,d} \in X_{dil,d}$ é denominado:*

- *Normal (ou de não-falha) se $\ell = N$ para todo $(x, \ell) \in x_{dil,d}$;*
- *Certo da ocorrência da falha nos sensores (ou certo-F), se $\ell = F$ para todo $(x, \ell) \in x_{dil,d}$;*
- *Certo da recuperação da falha nos sensores (ou certo-R), se $\ell = R$ para todo $(x, \ell) \in x_{dil,d}$.*

Além disso, se existir:

- *$(x, \ell), (y, \tilde{\ell}) \in x_{dil,d}$, x não necessariamente distinto de y, tal que $\ell = N$ e $\tilde{\ell} = F$ então $x_{dil,d}$ é denominado incerto-NF, isto é, incerto se o sensor falhou ou não;*
- *$(x, \ell), (y, \tilde{\ell}) \in x_{dil,d}$, x não necessariamente distinto de y, tal que $\ell = N$ e $\tilde{\ell} = R$ então $x_{dil,d}$ é denominado incerto-NR, isto é, incerto se o sensor não falhou ou se falhou e se recuperou da falha;*
- *$(x, \ell), (y, \tilde{\ell}) \in x_{dil,d}$, x não necessariamente distinto de y, tal que $\ell = F$ e $\tilde{\ell} = R$ então $x_{dil,d}$ é denominado incerto-FR, isto é, incerto se o sensor falhou e não se recuperou ou se falhou e se recuperou da falha;*
- *$(x, \ell), (y, \tilde{\ell}), (z, \hat{\ell}) \in x_{dil,d}$, x, y, z não necessariamente distintos, tal $\ell = N$, $\tilde{\ell} = F$ e $\hat{\ell} = R$ então $x_{dil,d}$ é denominado incerto-NFR, isto é, incerto se o sensor não falhou, se falhou e não se recuperou da falha, ou ainda, se falhou e se recuperou da falha.*

Com vistas a apresentar condições necessárias e suficientes para as diagnosticabilidades-F e -R, devemos definir ciclos indeterminados apropriados que expressem incertezas por parte do diagnosticador em relação às sequências da linguagem gerada pela planta que podem ser afetadas pelas falhas intermitentes dos sensores. Note que a possibilidade de aparecerem caminhos cíclicos de eventos não observáveis requer que sejam considerados não somente os ciclos observáveis indeterminados como, também, os ciclos escondidos indeterminados. (Basilio and Lafortune, 2009; Basilio et al., 2012; Carvalho et al., 2012).

[2]É importante ressaltar que a classificação de estados introduzida aqui é uma extensão natural das definições originalmente apresentadas em Sampath et al. (1995) e Basilio et al. (2010).

Definição 6.4. *(Ciclo de estados) Um conjunto de estados* $\{x_1, x_{,2}, \ldots, x_n\} \subseteq X$ *forma um ciclo em um autômato* $H = (X, \Sigma, f, \Gamma, x_0)$ *se existe uma sequência* $s = \sigma_1\sigma_2 \ldots \sigma_n \in L(H, x_1)$ *tal que* $f(x_k, \sigma_k) = x_{k+1}$, $k = 1, \ldots, n-1$, *e* $f(x_n, \sigma_n) = x_1$, *em que* $L(H, x) = \{s \in \Sigma^* : f(x, s) \in X\}$.

Definição 6.5. *(ciclo observável indeterminado) Um conjunto de estados incertos* $\{x^1_{dil,d}, x^2_{dil,d}, \ldots, x^p_{dil,d}\} \subseteq X_{dil,d}$ *forma um ciclo observável indeterminado em* $G_{dil,d}$ *se as seguintes condições forem satisfeitas:*

B1. $x^1_{dil,d}, x^2_{dil,d}, \ldots, x^p_{dil,d}$ *formam um ciclo em* $G_{dil,d}$*;*

B2. $\exists (x_i^{k_i}, \ell), (\tilde{x}_i^{r_i}, \tilde{\ell}), (\hat{x}_i^{q_i}, \hat{\ell}) \in x_{dil,d}$, *se* $(\hat{x}_i^{q_i}, \hat{\ell})$ *existir,* $(x_{dil,d} \in X_{dil,d})$, $x_i^{k_i}$ *não necessariamente distinto de* $\tilde{x}_i^{r_i}$ *e* $\hat{x}_i^{q_i}$, $i = 1, 2, \ldots, p$, $k_i = 1, 2, \ldots, m_i$, *e* $r_i = 1, 2, \ldots, \tilde{m}_i$ *e* $q_i = 1, 2, \ldots, \hat{m}_i$ *tais que as sequência de estados* $\{x_i^{k_i}\}$, $\{\tilde{x}_i^{r_i}\}$ *e* $\{\hat{x}_i^{q_i}\}$, $i = 1, 2, \ldots, p$, $k_i = 1, 2, \ldots, m_i$, $r_i = 1, 2, \ldots, \tilde{m}_i$ *e* $p_i = 1, 2, \ldots, \hat{m}_i$ *podem ser rearranjadas para formar ciclos em* $G_{dil,\ell}$*, cujas sequências correspondentes* s, $\tilde{s}$ *e* $\hat{s}$ *(formadas com os eventos definidos de acordo com a definição 6.4) têm como projeções* $P_{dil,o}(s) = P_{dil,o}(\tilde{s}) = P_{dil,o}(\hat{s}) = \sigma_1\sigma_2 \ldots \sigma_p \neq \varepsilon$.

Além disso, se:

- $(\ell = N, \tilde{\ell} = F)$*, então o ciclo é um ciclo observável indeterminado-F (coi-F);*
- $(\ell = F, \tilde{\ell} = R)$*, então o ciclo é um ciclo observável indeterminado-R (coi-R);*
- $(\ell = N, \tilde{\ell} = R)$ *ou* $(\ell = N, \tilde{\ell} = F, \hat{\ell} = R)$*, então o ciclo é um ciclo observável indeterminado-FR (coi-FR);*

Definição 6.6. *(ciclo escondido, ciclo escondido indeterminado-F, -R e FR) Seja* $x_{dil,d} = \{x_1\ell_1, x_2\ell_2, \ldots, x_n\ell_n\}$ *um estado de* $G_{dil,d}$*. Existe um ciclo escondido em* $x_{dil,d}$ *se para algum* $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, n\}$*, as seguintes condições forem satisfeitas:*

HC1. $x_{i_1}\ell_{i_1}, x_{i_2}\ell_{i_2}, \ldots, x_{i_k}\ell_{i_k}$ *formam um ciclo em* $G_{dil,\ell}$*;*

HC2. $\{\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_k}\} \subseteq \Sigma_{uo}$, *em que* $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_k}$ *são tais que* $f_{dil,\ell}(x_{i_j}\ell_{i_j}, \sigma_{i_j}) = x_{i_{j+1}}\ell_{i_{j+1}}$, $j = 1, 2, \ldots, k-1$, *e* $f_{dil,\ell}(x_{i_k}\ell_{i_k}, \sigma_{i_k}) = x_{i_1}\ell_{i_1}$.

Se além das condições **HC1** *e* **HC2**,

- $x_{dil,d}$ *é um estado incerto-NF de* $G_{dil,d}$ *e* $\ell_{i_j} = F$, $j = 1, 2, \ldots, k$, *então* $x_{dil,d}$ *possui um ciclo escondido indeterminado-F (cei-F);*
- $x_{dil,d}$ *é um estado incerto-FR de* $G_{dil,d}$ *e* $\ell_{i_j} = R$, $j = 1, 2, \ldots, k$, *então* $x_{dil,d}$ *possui um ciclo escondido indeterminado-R (cei-R).*

- *$x_{dil,d}$ é um estado incerto-NR de $G_{dil,d}$ e $\ell_{i_j} = R$, $j = 1, 2, \ldots, k$, então $x_{dil,d}$ possui um ciclo escondido indeterminado-FR (cei-FR);*

- *$x_{dil,d}$ é um estado incerto-NFR de $G_{dil,d}$ e*

  - *$\ell_{i_j} = F$, $j = 1, 2, \ldots, k$, e não existe $\tilde{\ell}_{i_{\tilde{j}}} = R$, $\tilde{j} = 1, 2, \ldots, \tilde{k}$, para qualquer $\{i_1, i_2, \ldots, i_{\tilde{k}}\} \subseteq \{1, 2, \ldots, n\}$ então $x_{dil,d}$ possui um ciclo escondido indeterminado-F (cei-F);*
  - *$\ell_{i_j} = R$, $j = 1, 2, \ldots, k$, então $x_{dil,d}$ possui um ciclo escondido indeterminado-FR (cei-FR).*

Com base na definição 6.5 e na definição 6.6 podemos enunciar o seguinte teorema.

Teorema 6.1.

A linguagem L_{dil} gerada pelo automato G_{dil} é diagnosticável-F em relação à projeção $P_{dil,o} : \Sigma_{dil}^* \to \Sigma_o^*$ e Σ'_{isf} se, e somente se, o diagnosticador $G_{dil,d}$ não possui ciclos (observáveis ou escondidos) indeterminados-F nem -FR.

A linguagem L_{dil} gerada pelo autômato G_{dil} é diagnosticável-R em relação à projeção $P_{dil,o} : \Sigma_{dil}^* \to \Sigma_o^*$, Σ'_{isf} e Σ_{isf} se, e somente se, o diagnosticador $G_{dil,d}$ não possui ciclos (observáveis ou escondidos) indeterminados-R nem -FR.

Demonstração. Vamos apresentar somente a prova para a diagnosticabilidade-F, uma vez que a prova para a diagnosticabilidade-R pode ser obtida seguindo passos e argumentos semelhantes ao caso da diagnosticabilidade-F.

($\Leftarrow$) Suponha que L_{dil} não seja diagnosticável-F em relação a $P_{dil,o}$ e Σ'_{isf}. Assim, como consequência da definição de diagnosticabilidade-F, existem pelo menos dois sequências da seguinte forma: *(i)* uma sequência $s_F = st \in L_{dil}$, $s \in \Psi(\Sigma'_{isf})$, $t \in L_{dil}/s$, $\Sigma_{isf} \notin t$, $\|t\| \geq n$, n arbitrariamente longo, e; *(ii)* uma sequência s_N ou s_R (podem ser ambas) satisfazendo as seguintes condições:

D1. s_N é tal que $\Sigma'_{isf} \notin s_N$ e $P_{dil,o}(s_N) = P_{dil,o}(s_F)$;

D2. $s_R = s'_R s''_R$ é tal que $(s'_R \in \Psi(\Sigma'_{isf})) \wedge (\Sigma_{isf} \in s''_R) \wedge (\Sigma'_{isf} \notin s''_R)$ e $P_{dil,o}(s_R) = P_{dil,o}(s_F)$;

Dado que $G_{dil,d}$ é um autômato determinístico, então existirão estados incertos $x_{dil} \in X_{dil}$ tais que:

D3. $x_{dil,d} = f_{dil,d}(x_{0_{dil,d}}, P_{dil,o}(st)) = f_{dil,d}(x_{0_{dil,d}}, P_{dil,o}(s_N))$: estado incerto-NF;

D4. $x_{dil,d} = f_{dil,d}(x_{0_{dil,d}}, P_{dil,o}(st)) = f_{dil,d}(x_{0_{dil,d}}, P_{dil,o}(s_R))$: estado incerto-FR;

D5. $x_{dil,d}$ é um estado incerto-NFR quando as condições **D1** e **D2** são verificadas simultaneamente.

Não é difícil verificar que, embora s_F deva ser uma sequência arbitrariamente longa[3], $P_{dil,o}(s_F)$ pode ter comprimento finito ou ser arbitrariamente longa, uma vez que G_{dil} pode ter caminhos cíclicos de eventos não observáveis. Um argumento similar aplica-se a s_N e s_R.

Considere inicialmente o caso em que $P_{dil,o}(s_F)$ seja uma sequência arbitrariamente longa. Nesse caso, $P_{dil,o}(s_N)$ e $P_{dil,o}(s_R)$ também serão arbitrariamente longas. Suponha que $|X_{dil,d}| = N_{dil,d}$. Fazendo $n > N_{dil,d}$, então existirá um ciclo de estados para o qual pelo menos um evento associado com as transições entre os estados do ciclo seja observável, definindo, assim, um ciclo observável de estados incertos em $G_{dil,d}$. Para mostrar que esse ciclo é indeterminado é suficiente provar que esse ciclo de estados incertos define dois ciclos em $G_{dil,\ell}$: um com estados rotulados por F, e outro rotulado por N e/ou R. Dado que $L_{dil,d} = P_{dil,o}(L_{dil,\ell})$ e $P_{dil,o}(s_i)$ é arbitrariamente longa para $i \in \{F, N, R\}$, então $s_{i,\ell} = P^{-1}_{dil,o}(s_i) \cap L_{dil,\ell}$ é arbitrariamente longa para $i \in \{F, N, R\}$. Supondo que $|X_{dil,\ell}| = N_{dil,\ell}$, então existe $m > N_{dil,\ell}$ tal que $\|s_i\| \geq m$, $i \in \{F, N, R\}$ definindo, assim, um ciclo de estados F, N e/ou R em $G_{dil,\ell}$.

Considere agora o caso em que $P_{dil,o}(s_F)$ seja uma sequência ilimitada. Dado que s_F é uma sequência arbitrariamente longa, então

$$x_{dil,\ell} = f_{dil,\ell}(x_{0,\ell}, st)$$

será um estado certo para todo $st \in \overline{s_F}$, $s \in \Psi(\Sigma'_{isf})$. Assim, conforme a definição 6.6, existirá um ciclo escondido indeterminado-F nos estados incertos definidos pelas condições **D3**, **D4** e **D5**.

($\Rightarrow$) Considere inicialmente o caso em que $G_{dil,d}$ possui ciclos observáveis indeterminados-F. Assim, de acordo com a definição 6.5, existem pelo menos duas sequências $s_{F,\ell}$ e $s_{N,\ell}$ e/ou $s_{R,\ell}$ em $L_{dil,\ell}$, em que que $s_{F,\ell}$ é arbitrariamente longa e $s_{N,\ell}$ e/ou $s_{R,\ell}$ são também arbitrariamente longas dependendo do ciclo indeterminado formado por estados incertos-NFR, -FR, ou -NF, tais que $P_{dil,o}(s_{N,\ell}) = P_{dil,o}(s_{F,\ell})$ e/ou $P_{dil,o}(s_{R,\ell}) = P_{dil,o}(s_F)$. Dado que $L_{dil} = L_{dil,\ell}$, então $s_{i,\ell} \in L_{dil}$, $i \in \{F, N, R\}$, o que de acordo com a definição 6.2, mostra L_{dil} não é diagnosticável-F em relação a projeção $P_{dil,o}$ e Σ'_{isf}.

Suponha, finalmente, que existe um ciclo escondido inderterminado-F num estado $x_{dil,d} = \{x_1\ell_1, x_2\ell_2, \ldots, x_n\ell_n\} \in X_{dil,d}$ de $G_{dil,d}$, em que $x_i\ell_i$ é um estado de $G_{dil,\ell}$. Assim, como consequência da definição 6.6, existe $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, n\}$ tal que $x_{i_1}\ell_{i_1}, x_{i_2}\ell_{i_2}, \ldots, x_{i_k}\ell_{i_k}$ formam um ciclo em $G_{dil,\ell}$, e $\ell_{i_j} = F$, $j = 1, 2, \ldots, k$. Então, não é difícil verificar que existe uma sequência $s_p = stu_p \in$

[3]Uma sequência s será arbitrariamente longa se seu comprimento $p = ||s||$ for arbitrariamente longo.

L_{dil} satisfazendo as seguintes condições:

E1. $\Sigma_f \in s$ e $f_{dil,d}(x_{0_{dil,d}}, P_{dil,o}(s)) = x_{dil,d}$;

E2. $t \in (\Sigma'_{isf} \cup \Sigma_{uo})^*$ tal que $f_{dil,d}(x_{0_{dil,d}}, P_{dil,o}(st)) = x_{dil,d}$;

E3. $u_p \in (\Sigma'_{isf} \cup \Sigma_{uo})^*$, $\|u_p\| = p$, com p arbitrariamente longo, é tal que

$$f_{dil,d}(x_{0_{dil,d}}, P_{dil,o}(stu_p)) = x_{dil,d}$$

e $f_{dil,d}(x_{i_1}\ell_{i_1}, u_j) = x_{i_j}\ell_{i_j}$, sendo que $j = (p \bmod k) + 1$.

Além disso, dado que $x_{dil,d}$ é um estado incerto, existe $s_i \in L_{dil}$, $i \in \{N, R\}$, tal que as condições **E1**, **E2**, ou ambas, são verificadas (dependendo de $x_{dil,d}$ ser, respectivamente, um estado incerto-NFR, -NF, ou -FR) e, por conseguinte, a condição de diagnosticabilidade D_F é violada. □

6.4 Exemplos

A seguir apresentamos dois exemplos para ilustrar os resultados apresentados na seção anterior.

Exemplo 6.1. *Considere o autômato G_1 cujo diagrama de transição de estados está representado na figura 6.5(a), em que todos os eventos são considerados observáveis e suponha que $\Sigma_{isf} = \{b\}$. O modelo correspondente a G_1 (denotado por $G_{dil,1}$) que leva em consideração possíveis falhas intermitentes na operação do sensor que registra a ocorrência do evento b está representado na figura 6.5(b).*

Para a construção do diagnosticador, o próximo passo é obter, de acordo com a equação (6.4), o autômato $G^1_{dil,\ell} = G_{dil,1} \| A_\ell$ representado na figura 6.5(c). De acordo com a equação (6.5), o diagnosticador será dado por $G^1_{dil,d} = Obs(G^1_{dil,l})$ que resulta no diagrama de transição de estados mostrado na figura 6.5(d). É fácil verificar que $G^1_{dil,1}$ não possui ciclos (observáveis ou escondidos) indeterminados-F, nem -R e, portanto, a linguagem gerada por $G_{dil,1}$ é diagnosticável-F em relação à projeção $P_{dil,o}$ e $\Sigma'_{isf} = \{b'\}$ e diagnosticável-R em relação à projeção $P_{dil,o}$, $\Sigma'_{isf} = \{b'\}$ e $\Sigma_{isf} = \{b\}$.

Exemplo 6.2. *Considere, agora, o autômato G_2 cujo diagrama de transição de estados está mostrado na figura 6.6(a). Suponha que os conjunto de eventos observáveis e não-observáveis sejam, respectivamente, $\Sigma_o = \{b, c\}$ e $\Sigma_{uo} = \{a\}$. O modelo (denotado por $G_{dil,2}$) que leva em conta as falhas intermitentes na operação do sensor responsável pelo registro do evento b e seu correspondente diagnosticador (denotado por $G^2_{dil,d}$) são mostrados, respectivamente, nas figuras 6.6(b) e 6.6(c) . Na figura 6.6(c) podemos observar que $G^2_{dil,d}$ possui ciclos (observados e escondidos)*

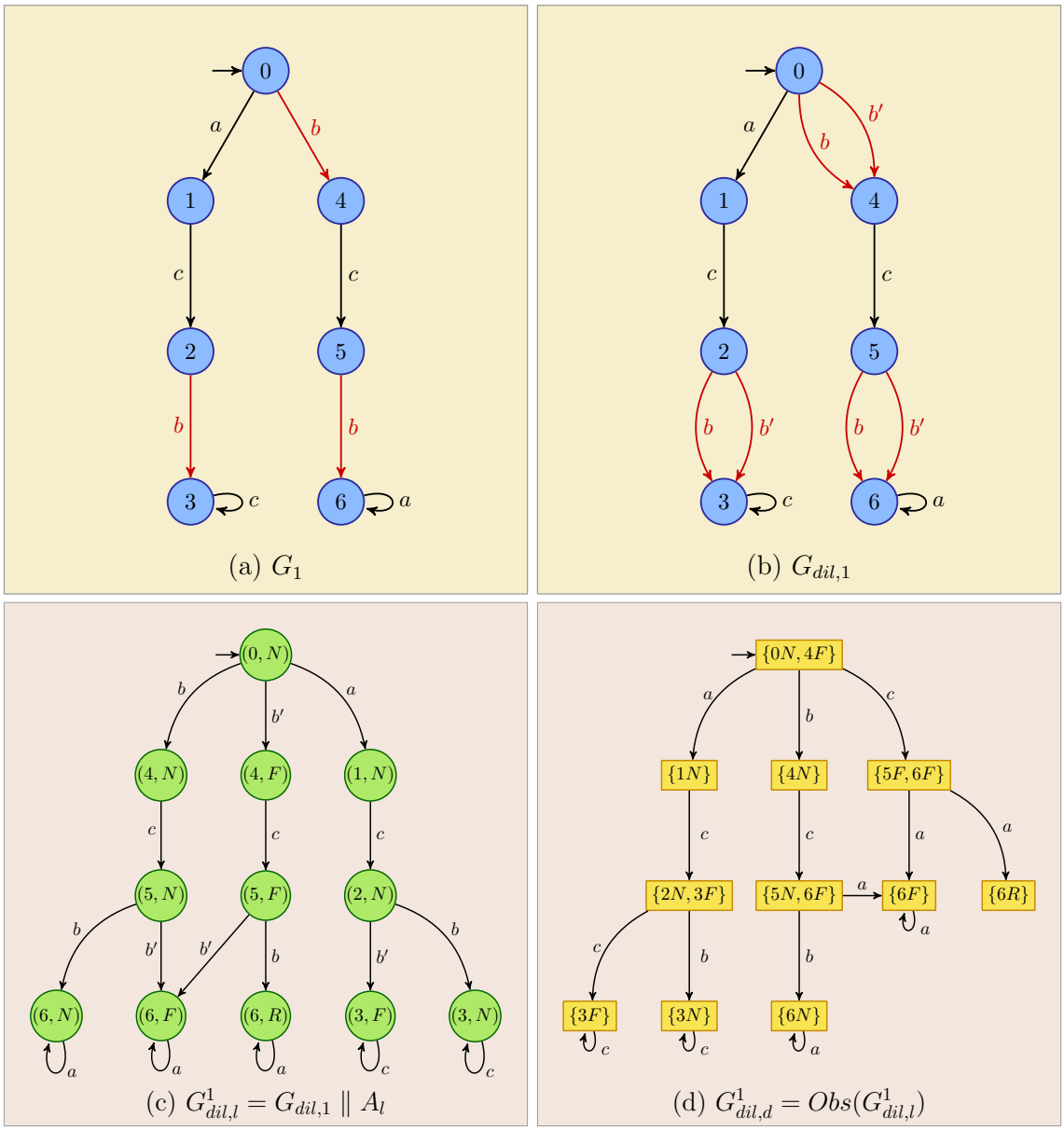


Figura 6.5: Autômato G_1 (a); autômato $G_{dil,1}$ (b); autômato $G^1_{dil,\ell}$ (c); diagnosticador $G^1_{dil,d}$ (d).

indeterminados-F e, também, -R. Por essa razão, de acordo com o teorema 6.1, a linguagem gerada por $G_{dil,2}$, não é diagnosticável-F (nem diagnosticável-R) em relação à projeção $P_{dil,o}$, $\Sigma_{isf} = \{b\}$ e $\Sigma'_{isf} = \{b'\}$. A não diagnosticabilidade pode ser justificada da seguinte forma:

- *Em relação à existência de ciclos escondidos indeterminados-F, considere a existência do cei-F no estado incerto-NF $\{2N, 3F, 4F, 7F, 8F\}$. Não é difícil verificar a existência de duas sequências: uma sequência de falha $s_F = b'acb'a^n$, $n \in \mathbb{N}$ e uma sequência normal $s_N = ac$ para as quais $P_{dil,o}(s_N) = P_{dil,o}(s_F) = cb$, e, por conseguinte, a condição de diagnosticabilidade-F é violada.*

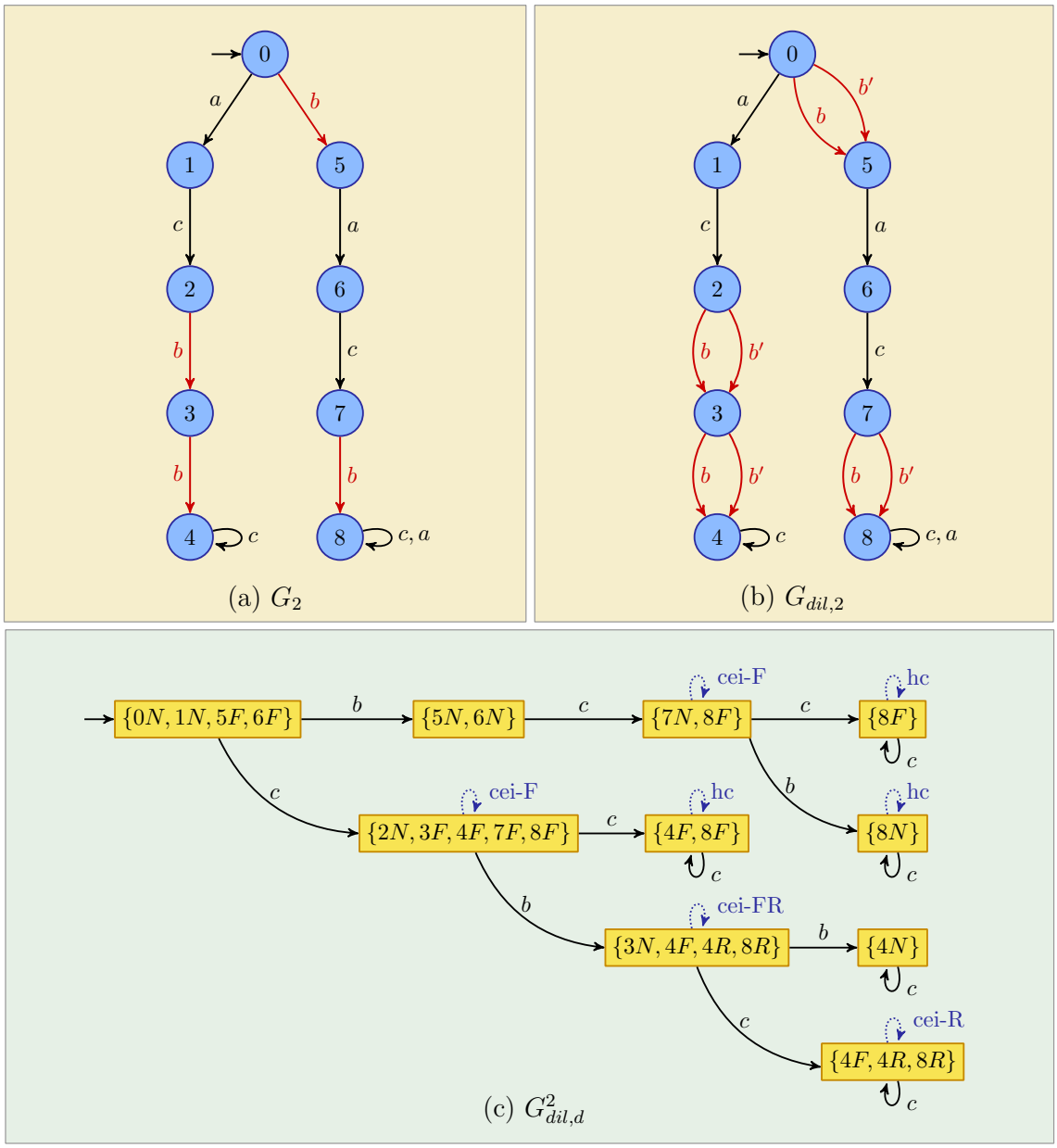


Figura 6.6: Autômato G_2 (a); autômato $G_{dil,2}$ (b); autômato diagnosticador $G^2_{dil,d}$ (c).

- *Em relação à existência de ciclos escondidos indeterminados-R, considere o cei-R no estado incerto* $\{3N, 4F, 4R, 8R\}$. *Para esse caso, é possível encontrar três sequências: uma sequência normal* $s_N = acb$, *uma sequência de falha* $s_F = acbb'$ *e uma sequência* $s_R = b'acba^n$ *na qual o evento b ocorre após uma falha no sensor. Para essas sequências,* $P_{dil,o}(s_R) = P_{dil,o}(s_N) = P_{dil,o}(s_F) = cb$, *de tal forma que a condição de diagnosticabilidade-R é violada.*

- *Em relação à existência dos ciclos observáveis indeterminados-F e -R que se formam no estado* $\{4F, 4R, 8R\}$ *pode ser identificada a existência de uma sequência de falha* $s_F = acbb'c^n$ *e duas sequências com ocorrências do evento*

b após da ocorrência de b', $s_{R,1} = acb'bc^n$ e $s_{R,2} = b'acbc^n$, $n \in \mathbb{N}$. Todas essas sequências têm a mesma projeção sobre Σ_o^*, isto é, $P_{dil,o}(s_F) = P_{dil,o}(s_{R,1}) = P_{dil,o}(s_{R,2}) = cbc^n$ de forma que, novamente, as condições de diagnosticabilidade-R e -F são violadas. Note que a existência das sequências s_F, $s_{R,1}$ e $s_{R,2}$ implica que não é possível determinar de forma única se o sensor falhou permanentemente, ou, se falhou e se recuperou da falha.

É interessante notar que existe um cei-R adicional no estado $\{4F, 4R, 8R\}$, o qual também viola a condição de diagnosticabilidade-R.

6.5 Comentários finais

Neste capítulo abordamos o problema da diagnosticabilidade de falhas intermitentes na operação de sensores. É importante ressaltar que: *(i)* diferente do que se poderia esperar, o uso do modelo adaptado para mau funcionamento de sensores não transforma o problema de falhas intermitentes na operação de sensores em um problema semelhante ao apresentado em Carvalho et al. (2012); *(ii)* a formulação do problema de deteção de falhas intermitentes na operação de sensores proposta aqui difere da abordagem apresentada em Contant et al. (2004) no sentido de que os eventos de falha e *reset* agora ficam associados, respectivamente, com o mau funcionamento e o funcionamento normal do sensor e, além disso, o evento de *reset* considerado aqui é observável enquanto em Contant et al. (2004) é um evento não observável. Essa diferença é fundamental e conduz às definições de diagnosticabilidade introduzidas aqui, as quais diferem das apresentadas em Contant et al. (2004).

Capítulo 7

Conclusões e trabalhos futuros

O objetivo principal deste trabalho foi estudar a complexidade média de algoritmos para diagnose de falhas em SEDs usando uma abordagem de algoritmia experimental. Nesse sentido, a primeira contribuição desse trabalho foi introduzir uma metodologia de análise experimental no estudo da complexidade computacional de algoritmos para SEDs usando geradores de autômatos determinísticos acessíveis (reportados em trabalhos recentes da literatura) e, complementarmente, análise de dados. A seguir apresentamos as conclusões centrais derivadas da análise experimental dos algoritmos de diagnose de falhas para SEDs estudados neste trabalho.

O problema de se construir o diagnosticador proposto por Sampath et al. (1995), no caso médio, é um problema computacionalmente mais tratável do que esperado, se atendemos ao limite conhecido ($\mathcal{O}(2^{2n})$) da complexidade no pior caso. A complexidade no pior caso na construção de diagnosticadores é superior em vários ordens de grandeza à complexidade média, uma vez que que as instâncias que produzem a maior complexidade aparecem com pouca probabilidade nas distribuições probabilísticas que caracterizam a complexidade na construção de diagnosticadores para todos os conjuntos de autômatos válidos estudados. Tal fato foi analisado em detalhe no capitulo 4 em que, para alguns conjuntos de autômatos válidos, mostrou-se que a distribuição lognormal explica consistentemente a complexidade na construção de diagnosticadores. Desde uma perspectiva complementar, a mesma divergência entre a complexidade média e a complexidade no pior caso do diagnosticador foi evidenciada pelo modelo hipotético de ordem $\mathcal{O}(n^{0.77 \log k + 0.63})$ (em que n e k são, respectivamente o número de estados e o número de eventos do autômato de entrada) formulado no capítulo 5.

A complexidade média na construção do verificador proposto por (Moreira et al., 2011a) é de ordem quadrática em relação ao número de estados do autômato de entrada. O fato de que tanto a complexidade média como a complexidade no pior caso serem de ordem quadrática implica que não é possível esperar uma maior eficiência do algoritmo de construção de verificadores formulado por Moreira et al. (2011a)

no caso médio. Conforme foi mostrado no capítulo 5, a medida que se aumenta o número de eventos no autômato de entrada, as distribuições da complexidade do verificador vão se concentrando progressivamente nos valores n^2 e $2n^2$ (dependendo do número de eventos não observáveis), evidenciando, assim, que o limite no pior caso de complexidade na construção do verificador é, para todos os conjuntos de autômatos válidos analisados, estreito.

O processo de realizar a análise experimental de algoritmos de diagnose de falhas levou, como contribuição secundaria, ao desenvolvimento do programa DESLAB. Além das características do programa DESLAB que já foram sinaladas no capítulo 3, é interessante comentar que a ferramenta já foi usada em alguns trabalhos de mestrado (Bouzan, 2013) para a implementação de algoritmos para SEDs e, paralelamente, como ferramenta pedagógica de apoio a alguns cursos.

Além disso, no capítulo 4 foram estabelecidos dois resultados em geração e enumeração de autômatos, quais sejam: o algoritmo 4.4 que permite a geração exaustiva de conjuntos de autômatos com n estados e k eventos e, com base nesse algoritmo, a fórmula (4.6) que permite o cálculo da cardinalidade desses conjuntos.

Uma outra contribuição da tese foi realizar uma formulação diferente do modelo proposto por Carvalho et al. (2012) (concebido originalmente para a diagnose de falhas com perda intermitente de observações nos sensores) para considerar o problema da diagnose de falhas intermitentes na operação de sensores. Nesse contexto, foram obtidas condições necessárias e suficientes para a diagnosticabilidade de falhas intermitentes na operação de sensores e, com base nesses resultados, foi proposto um teste de diagnosticabilidade baseado no autômato diagnosticador.

Possíveis tópicos de pesquisa que podem dar continuidade a esse trabalho são listados a seguir:

(i) A verificação da diagnosticabilidade usando diagnosticadores requer busca por ciclos que, no pior caso, tem complexidade EXPTIME. Assim, a comparação mais justa para a verificação da propriedade de diagnosticabilidade de um SED deve ser feita a partir do autômato diagnosticador realizando uma busca de componentes fortemente conectadas (essa busca possui complexidade linear) no seguinte autômato:

$$G_{test} = G_D \parallel (G \parallel A_l) \tag{7.1}$$

em que G_D é o diagnosticador associado à planta G e A_l é o autômato rotulador da figura 2.5; embora o trabalho aqui apresentado para G_D tenha importância no que se refere à diagnose online. O trabalho comparativo envolvendo G_{test} será objeto de investigação futura.

(ii) Desenvolvimento de uma plataforma de computação de alto desempenho (possivelmente usando computação paralela) para analisar a complexidade média

de algoritmos em SEDs. Uma das aplicações possíveis para essa plataforma seria dar continuidade ao experimento com amostragem para conjuntos de autômatos válidos de maior cardinalidade, com o intuito de estender e refinar o modelo hipotético proposto para a complexidade do diagnosticador de acordo com os passos do método científico.

(iii) Uso da metodologia de análise experimental para avaliar o desempenho comparativo e a complexidade média de outros algoritmos usados em SEDs. Por exemplo, essa metodologia pode ser usada para avaliar a complexidade média do algoritmo de busca de bases mínimas para diagnose de falhas proposto em Basilio et al. (2012) cuja complexidade no pior caso é exponencial. A metodologia pode ser usada, no mesmo caso, para comparar diferentes algoritmos de obtenção de bases mínima para diagnose de falhas estabelecendo, assim, o desempenho comparativo dos mesmos.

(iv) Uma vez que a combinatória analítica está se desenvolvendo recentemente como uma abordagem poderosa para estudar a complexidade média de algoritmos que envolvem autômatos (Broda et al., 2012), é de interesse pesquisar a fatibilidade da aplicação dos métodos de combinatória analítica para estudar a complexidade média de algoritmos de SEDs, tanto em diagnose de falhas como em controle supervisório. A combinação dos métodos teóricos da combinatória analítica com os métodos da algoritmia experimental pode ser muito proveitosa na análise do desempenho real de algoritmos em SEDs.

(v) Levando em consideração que a evidência experimental mostra que a construção de diagnosticadores é, no caso médio, computacionalmente tratável, o seguinte passo é estudar estruturas eficientes de realização computacional de diagnosticadores para sua correspondente implementação nos diferentes tipos de dispositivos computacionais sobre os quais tem-se construído os SEDs existentes.

Referências Bibliográficas

Almeida, M. (2010). *Equivalence of Regular Languages: an Algorithmic Approach and Complexity Analysis*. Tese de doutorado, Universidade de Porto.

Almeida, M., Moreira, N., and Reis, R. (2007). Enumeration and generation with a string automata representation. *Theor. Comput. Sci.*, 387(2):93–102.

Alves, M. V. S., Basilio, J. C., Carrilho da Cunha, A. E., Carvalho, L. K., and Moreira, M. V. (2014). Robust supervisory control against intermittent loss of observations. In *12th IFAC International Workshop on Discrete Event Systems*. accepted paper.

Aristizabal, R. J. (2012). *Estimating Parameters of the Three-Parameter Lognormal Distribution*. Dissertação de mestrado, Florida International University.

Arora, S. and Barak, B. (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, USA, first edition.

Athanasopoulou, C. and Chatziathanasiou, V. (2009). Intelligent system for identification and replacement of faulty sensor measurements in thermal power plants (ippamas: Part 1). *Expert Systems with Applications*, 36(5):8750–8757.

AT&T Labs Research (visitado: 21-01-2012). Graphviz – graph visualization software. Website. `http://www.graphviz.org/`.

Balemi, S., Hoffmann, G., Wong-Toi, H., and Franklin, G. (1993). Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059.

Basilio, J. C., Carvalho, L. K., and and, M. V. M. (2010). Diagnose de falhas em sistema a eventos discretos modelados por autômatos finitos. *Revista Controle & Automação*, 21(5):510–533.

Basilio, J. C. and Lafortune, S. (2009). Robust codiagnosability of discrete event systems. In *American Control Conference, 2009. ACC '09.*, pages 2202–2209.

Basilio, J. C., Lima, S. T. S., Lafortune, S., and Moreira, M. V. (2012). Computation of minimal event bases that ensure diagnosability. *Discrete Event Dynamic Systems: Theory and Applications*, 22(3):249–292.

Bassino, F., David, J., and Nicaud, C. (2008). Random generation of possibly incomplete deterministic automata. In *Génération Aléatoire de Structures COMbinatoires (Gascom'08)*, pages 31–40.

Bassino, F., David, J., and Nicaud, C. (2009). Enumeration and random generation of possibly incomplete deterministic automata. *Pure Mathematics and Applications*, 19(2-3):1–16.

Bassino, F. and Nicaud, C. (2006). Accessible and deterministic automata: Enumeration and boltzmann samplers. *DMTCS Proceedings*, 0(1).

Bassino, F. and Nicaud, C. (2007). Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1-3):86–104.

Bouzan, B. (2013). *Algoritmos em tempo polinomial para verificação da observabilidade e normalidade de linguagens regulares*. Dissertação de mestrado, Universidade Federal do Rio de Janeiro.

Bovet, D. P. and Crescenzi, P. (1994). *Introduction to the theory of complexity*. Prentice Hall, Englewood Cliffs, USA.

Broda, S., Machiavelo, A., Almeida, M., Moreira, N., and Reis, R. (2012). An introduction to descriptional complexity of regular languages through analytic combinatorics. Technical report, Universidade do Porto.

Broda, S., Machiavelo, A., Moreira, N., and Reis, R. (2011). On the average complexity of partial derivative automata: an analytic combinatorics approach. *International Journal of Foundations of Computer Science*, 22(07):1593–1606.

Brunsch, T. and Rudie, K. (2008). Discrete-event systems model of an outbreak response. In *American Control Conference, 2008*, pages 1709–1714.

Burnham, K. and Anderson, D. (2002). *Model Selection and Multi-Model Inference: A Practical Information-Theoretic Approach*. Springer, New Yok, USA.

Carruth, J., Tygert, M., and Ward, R. (2012). A comparison of the discrete kolmogorov-smirnov statistic and the euclidean distance. eprint arXiv:1206.6367 [stat.ME].

Carvalho, L., Basilio, J., Moreira, M., and Clavijo, L. (2013a). Diagnosability of intermittent sensor faults in discrete event systems. In *American Control Conference (ACC), 2013*, pages 929–934.

Carvalho, L. K. (2011). *Diagnose robusta de sistemas a eventos discretos*. Tese de doutorado, Universidade Federal de Rio de Janeiro.

Carvalho, L. K., Basilio, J. C., and Moreira, M. V. (2012). Robust diagnosis of discrete-event systems against intermittent loss of observations. *Automatica*, 48(9):2068–2078.

Carvalho, L. K., Moreira, M. V., Basilio, J. C., and Lafortune, S. (2013b). Robust diagnosis of discrete-event systems against permanent loss of observations. *Automatica*, 49(1):223–231.

Cassandras, C. G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer, 233 Spring Street, New York, NY 10013, USA.

Cassez, F. (2012). The complexity of codiagnosability for discrete event and timed systems. *IEEE Trans. Automat. Contr.*, 57(7):1752–1764.

Cassez, F., Tripakis, S., and Altissen, K. (2007). Sensor minimization problems with static or dynamic observers for fault diagnosis. In *Seventh International Conference on Application of Concurrency to System Design (ACSD 2007)*, pages 90–99.

Chen, E. and Lafortune, S. (1991). Dealing with blocking in supervisory control of discrete-event systems. *IEEE Transactions on Automatic Control*, 36(6):724–735.

Cho, H. and Marcus, S. I. (1989). On supremal languages of classes of sublanguages that arise in supervisor synthesis. *Mathematics of Control, Signals, and Systems*, 2(1):47–69.

Cho, K.-H. and Lim, J.-T. (1998). Synthesis of fault-tolerant supervisor for automated manufacturing systems: A case study in photolithographic process. *IEEE Transactions on Robotics and Automation*, 14(2):348–351.

Cieslak, R., Desclaux, C., Fawaz, A. S., and Varaiya, P. (1988). Supervisory control of discrete-event processes with partial observations. *IEEE Transactions on Automatic Control*, 33(3):249–260.

Clark, R. N. (1978). Instrument fault detection. *IEEE Trans. on Aerospace and Electronic Systems*, 14(3):456–465.

Clauset, A., Shalizi, C. R., and Newman, M. E. J. (2009). Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703.

Clavijo, L. B., Basilio, J. C., and Carvalho, L. K. (2012). Deslab: A scientific computing program for analysis and synthesis of discrete-event systems. In *Preprints of the 11th International Workshop on Discrete Event Systems*, pages 349–355.

Coffin, M. and Saltzman, M. J. (2000). Statistical analysis of computational tests of algorithms and heuristics. *INFORMS Journal on Computing*, 12(1):24–44.

Contant, O., Lafortune, S., and Teneketzis, D. (2004). Diagnosis of intermittent faults. *Discrete Event Dynamic Systems-Theory and Applications*, 14(2):171–202.

da Silva, J. C., Saxena, A., Balaban, E., and Goebel, K. (2012). A knowledge-based system approach for sensor fault modeling, detection and mitigation. *Expert Systems with Applications*, 39(12):10977–10989.

Debouk, R., Lafortune, S., and Teneketzis, D. (2000). Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 10(1-2):33–86.

Dey, A. K. and Kundu, D. (2009). Discriminating among the log-normal, weibull, and generalized exponential distributions. *IEEE Transactions on Reliability*, 58(3):416–424.

Ding, E. L., Fennel, H., and Ding, S. X. (2004). Model-based diagnosis of sensor faults for esp systems. *Control Engineering Practice*, 12(7):847–856.

Duchon, P., Flajolet, P., Louchard, G., and Schaeffer, G. (2004). Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4-5):2004.

Flajolet, P. and Sedgewick, R. (2009). *Analytic Combinatorics*. Cambridge University Press, New York, USA, 1st edition.

Fox, J. and Weisberg, S. (2011). *An R Companion to Applied Regression*. SAGE Publications, Thousand Oaks, USA.

Frank, P. M. (1990). Fault-diagnosis in dynamic-systems using analytical and knowledge-based redundancy - a survey and some new results. *Automatica*, 26(3):459–474.

Gohari, P. and Wonham, W. M. (2000). On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man, and Cybernetics*, 30(5):643–652.

Gokhale, S. and Mullen, R. (2005). Dynamic code coverage metrics: a lognormal perspective. In *Software Metrics, 2005. 11th IEEE International Symposium*, pages 10 pp.–33.

Grastien, A. and Anbulagan (2013). Diagnosis of discrete event systems using satisfiability algorithms: A theoretical and empirical study. *IEEE Trans. Automat. Contr.*, 58(12):3070–3083.

Greenlaw, R. and Hoover, H. (1998). *Fundamentals of the Theory of Computation: Principles and Practice*. Elsevier Science, San Francisco, USA.

Gubner, J. A. (2006). *Probability and random processes for electrical and computer engineers*. Cambridge University Press., New York, USA.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (visitado: 21-01-2012). Networkx. Website. `http://networkx.lanl.gov/index.html#`.

Hair, J., Anderson, R., and Tatham, R. (2007). *Análise Multivariada de Dados*. Bookman, Porto Alegre: Brasil.

Hale, W. E. (1972). Sample size determination for the lognormal distribution. *Atmospheric Environment*, 6(6):419–422.

Héam, P.-C., Nicaud, C., and Schmitz, S. (2010). Parametric random generation of deterministic tree automata. *Theor. Comput. Sci.*, 411(38-39):3469–3480.

Heyman, D., Tabatabai, A., and Lakshman, T. V. (1992). Statistical analysis and simulation study of video teleconference traffic in atm networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 2(1):49–59.

Hilbe, J. (2007). *Negative Binomial Regression*. Cambridge University Press, New York, USA, second edition.

Holmes, R. (1998). *Elementary Set Theory with a Universal Set*. Academia-Bruylant, Louvain-La-Neuve, Belgium.

Hsu, A., Eskafi, F., Sachs, S., and Varaiya, P. (1993). Protocol design for an automated highway system. *Discrete Event Dynamic Systems: Theory and Applications*, 2(3):183–206.

Huber-Carol, C. (2002). *Goodness-of-fit Tests and Model Validity*. Springer-Verlag, New Yor, USA.

Jiang, S., Huang, Z., Chandra, V., and Kumar, R. (2001). A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Trans. on Automatic Control*, 46(8):1318–1321.

Jiang, S., Kumar, R., and Garcia, H. E. (2003). Optimal sensor selection for discrete-event systems with partial observation. *IEEE Trans. on Automatic Control*, 48(3):369–381.

Jirásková, G. and Masopust, T. (2012). On a structural property in the state complexity of projected regular languages. *Theoretical Computer Science*, 449:93–105.

Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. In Goldwasser, M. H., Johnson, D. S., and McGeoch, C. C., editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. American Mathematical Society.

Kumar, R. and Garg, V. (1995). *Modeling and Control of Logical Discrete Event Systems*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, New York, USA.

Kumar, R., Garg, V., and Marcus, S. I. (1991). On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17(3):157–168.

Kundu, D. and Manglick, A. (2005). Discriminating between the log-normal and gamma distributions. *Journal of Applied Statistical Sciences*, 14(1):175–187.

Lafortune, S. (1988). Modeling and analysis of transaction execution in database systems. *IEEE Transactions on Automatic Control*, 33(5):429–447.

Lafortune, S. and Chen, E. (1990). The infimal closed and controllable superlanguage and its applications in supervisory control. *IEEE Transactions on Automatic Control*, 35(4):398–405.

Langtangen, H. P. (2009). *Python Scripting for Computational Science (Texts in Computational Science and Engineering)*. Springer-Verlag, Berlin, Germany, 3rd edition.

Law, A. (2007). *Simulation Modeling and Analysis*. McGraw-Hill Education (India) Pvt Limited, Noida, India, fourth edition.

Lawson, M. V. (2004). *Finite Automata*. CRC Press LLC, Boca Raton, USA, first edition.

Liao, H., Wang, Y., Cho, H. K., Stanley, J., Kelly, T., Lafortune, S., Mahlke, S. A., and Reveliotis, S. A. (2013). Concurrency bugs in multithreaded software: modeling and analysis using petri nets. *Discrete Event Dynamic Systems: Theory and Applications*, 23(2):157–195.

Lin, F. (1994). Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems Systems: Theory and Applications*, 4(2):197–212.

Lunze, J. and Schröder, J. (2004). Sensor and actuator fault diagnosis of systems with discrete inputs and outputs. *IEEE Trans. on Systems, Man and Cybernetics. Part B: Cybernetics*, 34(2):1096–1107.

McGeoch, C. (2012). *A Guide to Experimental Algorithmics*. Cambridge University Press, 32 Avenue of the Americas, New York, NY 10013-2473, USA.

McGeoch, C., Sanders, P., Fleischer, R., Cohen, P. R., and Precup, D. (2002). Experimental algorithmics. chapter Using Finite Experiments to Study Asymptotic Performance, pages 93–126. Springer-Verlag New York, Inc., New York, NY, USA.

Medrano, J. and Enari, E. (2010). Utilização dos conceitos de modelagem de sistemas de eventos discretos em processos logísticos na indústria de máquinas e equipamentos. In *VI Congresso Nacional de Excelência em Gestão*, pages 301–318.

Miura, K. (2011). An introduction to maximum likelihood estimation and information geometry. *Interdisciplinary Information Sciences*, 17(3):155–174.

Moore, F. (1971). On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Transactions on Computers*, 20(10):1211–1214.

Moreira, M. V., Jesus, T. C., and Basilio, J. C. (2011a). Polynomial time verification of decentralized diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 56(7):1679–1684.

Moreira, M. V., Jesus, T. C., and Basilio, J. C. (2011b). Polynomial time verification of decentralized diagnosability of discrete event systems. *IEEE Trans. on Automatic Control*, 56(7):1679–1684.

Mullen, R. and Gokhale, S. (2005). Software defect rediscoveries: a discrete lognormal model. In *IEEE International Symposium on Software Reliability Engineering, 2005. ISSRE 2005.*, pages 203–212.

Muller-Hannemann, M. and Schirra, S., editors (2010). *Algorithm Engineering: Bridging the Gap Between Algorithm Theory and Practice*. Springer-Verlag, Berlin, Germany.

Ni, Z. and Li, T. (2011). Negative binomial model with an application to special treatment count data. In *2011 International Conference on Management and Service Science (MASS)*, pages 1–4.

Nicaud, C. (2009). On the average size of glushkov's automata. In Dediu, A. H., Ionescu, A.-M., and Martín-Vide, C., editors, *3rd International Conference on Language and Automata Theory and Applications – LATA 2009*, volume 5457 of *Lecture Notes in Computer Science*, pages 626–637. Springer.

Nielsen, M. (2011). *Parameter Estimation for the Two-parameter Weibull Distribution*. Brigham Young University. Department of Statistics.

Nunes, C. E. V. (2012). *Sistema Inteligente de Suporte Operacional em Processos de Tratamento Primário de Petróleo*. Dissertação de mestrado, Universidade Federal de Sergipe.

Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science and Engineering*, 9(3):10–20.

Opitz, B. (2006). *Methods of Supervisory Control: A Software Implementation*. Dissertação de mestrado, University of Erlangen-Nuremberg.

Parthiban, A., Madhavan, J., Radhakrishna, P., Savitha, D., and Kumar, L. (2005). On the probability distribution function for airborne radar clutter. In *2005 IEEE International Radar Conference*, pages 464–468.

Peisert, S. and Bishop, M. (2007). How to design computer security experiments. In *Proceedings of the Fifth World Conference on Information Security Education*, pages 141–148.

Pena, P., Bravo, H., da Cunha A.E.C., Malik, R., Lafortune, S., and Cury, J. (2014). Verification of the observer property in discrete event systems. *IEEE Transactions on Automatic Control*, DOI:10.1109/TAC.2014.2298985.

Pena, P., Cury, J. E. R., and Lafortune, S. (2008). Polynomial-time verification of the observer property in abstractions. In *American Control Conference, 2008*, pages 465–470.

Pencolé, Y. and Cordier, M.-O. (2005). A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164(1-2):121–170.

Perez, F., Granger, B. E., and Hunter, J. D. (2011). Python: An ecosystem for scientific computing. *Computing in Science Engineering*, 13(2):13–21.

Perkins, W., Tygert, M., and Ward, R. (2014). Some deficiencies of χ^2 and classical exact tests of significance. *Applied and Computational Harmonic Analysis*, 36(3):361–386.

Prabhu, P., Jablin, T. B., Raman, A., Zhang, Y., Huang, J., Kim, H., Johnson, N. P., Liu, F., Ghosh, S., Beard, S., Oh, T., Zoufaly, M., Walker, D., and August, D. I. (2011). A survey of the practice of computational science. In *State of the Practice Reports*, SC '11, New York, NY, USA. ACM.

Python Software Foundation (visitado: 21-01-2012). Python programming language – official website. Website. `http://www.python.org/`.

Qiu, W. and Kumar, R. (2006). Decentralized failure diagnosis of discrete event systems. *IEEE Trans. on Systems, Man and Cybernetics, Part A*, 36(2):384–395.

Ramadge, P. J. and Wonham, W. M. (1987). Supervisory control of a class of discrete-event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230.

Reis, R., Moreira, N., and Almeida, M. (2005). On the representation of finite automata. In *Proceedings of the 7th International Workshop on Descriptional Complexity of Formal Systems*, pages 269–276.

Rohloff, K., Yoo, T.-S., and Lafortune, S. (2003). Deciding co-observability is pspace-complete. *Automatic Control, IEEE Transactions on*, 48(11):1995–1999.

Rohloff, K. R. (2005). Sensor failure tolerant supervisory control. In *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005*, pages 3493–3498.

Rohloff, K. R., Khuller, S., and Kortsarz, G. (2006). Approximating the minimal sensor selection for supervisory control. *Discrete Event Dynamic Systems: Theory and Applications*, 16(1):143–170.

Rudie, K. and Wonham, W. M. (1992). Protocol verification using discrete-event systems. In *Proceedings of the 31st IEEE Conference on Decision and Control (CDC)*, pages 3770–3777.

Sakarovitch, J. (2009). *Elements of Automata Theory*. Cambridge University Press, New York, USA, 1st edition.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. (1996). Failure diagnosis using discrete event models. *IEEE Trans. on Control Systems Technology*, 4(2):105–124.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketziz, D. (1995). Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1574.

Sanchez, A. M. and Montoya, F. J. (2006). Safe supervisory control under observability failure. *Discrete Event Dynamic Systems: Theory and Applications*, 16(4):493–525.

Sedgewick, R. (visitado: 12-12-2013). The role of the scientific method in programming. Website. `http://www.cs.princeton.edu/~rs/talks/ScienceCS.pdf`.

Sipser, M. (2006). *Introduction to the Theory of Computation*. Thomson Course Technology, Belmont, USA, second edition.

Steele, M. and Chaseling, J. (2006). Powers of discrete goodness-of-fit test statistics for a uniform null against a selection of alternative distributions. *Commun. Stat., Simulation Comput.*, 35(4):1067–1075.

Tabakov, D. and Vardi, M. Y. (2005). Experimental evaluation of classical automata constructions. In *12th International Conference on Logic for Programming Artificial Intelligence and Resoning – LPAR 2005*, pages 396–411. Springer.

Vahidi, A. (2004). *Efficient Analysis of Discrete Event Systems - Supervisor Synthesis with Binary Decision Diagrams*. Tese de doutorado, Chalmers University of Technology.

van Glabbeek, R. and Ploeger, B. (2008). Five determinization algorithms. In *CIAA 2008, Lecture Notes in Computer Science*, pages 161–170.

Wang, X., Mallapragada, G., and Ray, A. (2005). Language-measure-based supervisory control of a mobile robot. In *American Control Conference*, pages 4897–4902.

Wang, Y., Kelly, T., and Lafortune, S. (2007). Discrete control for safe execution of it automation workflows. *SIGOPS Oper. Syst. Rev.*, 41(3):305–314.

Wang, Y., Reza, H., and Singhai, S. (2010). A language - based framework for analyzing service representation models and service composition approaches. In *2010 IEEE 7th International Conference on e-Business Engineering*, pages 201–208.

Wong, K. C. (1998). On the complexity of projections of discrete-event systems. In *In IEE Workshop on Discrete Event Systems*, pages 201–208.

Wonham, W. M. and Ramadge, P. J. G. (1987). On the supremal controllable sublanguage of a given sublanguage. *SIAM Journal on Control and Optimization*, 25(3):637–659.

Wood, M. M. (2005). *Application, Implementation and Integration of Discrete-Event Systems Control Theory*. Dissertação de mestrado, Queen's University.

Yoo, T.-S. and Garcia, H. E. (2009). Event counting of partially-observed discrete-event systems with uniformly and nonuniformly bounded diagnosis delays. *Discrete Event Dynamic Systems: Theory and Applications*, 19(2):167–187.

Yoo, T.-S. and Lafortune, S. (2002a). Np-completeness of sensor selection problems arising in partially observed discrete-event systems. *Automatic Control, IEEE Transactions on*, 47(9):1495–1499.

Yoo, T.-S. and Lafortune, S. (2002b). Polynomial-time verification of diagnosability of partially observed discrete-event systems. *Automatic Control, IEEE Transactions on*, 47(9):1491–1495.

Zaytoon, J. and Lafortune, S. (2013). Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308 – 320.

Zou, G. Y., Taleban, J., and Huo, C. Y. (2009). Confidence interval estimation for lognormal data with application to health economics. *Computational Statistics & Data Analysis*, 53(11):3755–3764.