COPPE/UFRJ

DIAGNOSTICABILIDADE E DIAGNOSE ON-LINE DE FALHAS DE SISTEMA A EVENTOS DISCRETOS MODELADOS POR AUTÔMATOS FINITOS

Thiago Cerqueira de Jesus

Dissertação de Mestrado 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 Mestre em Engenharia Elétrica.

Orientador: Marcos Vicente de Brito Moreira

Rio de Janeiro

Fevereiro de 2011

DIAGNOSTICABILIDADE E DIAGNOSE ON-LINE DE FALHAS DE SISTEMA A EVENTOS DISCRETOS MODELADOS POR AUTÔMATOS FINITOS

Thiago Cerqueira de Jesus

DISSERTAÇÃO 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 MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

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

Prof. Amit Bhaya, Ph.D.

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

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2011

Jesus, Thiago Cerqueira de

Diagnosticabilidade e diagnose on-line de falhas de sistema a eventos discretos modelados por autômatos finitos/Thiago Cerqueira de Jesus. – Rio de Janeiro: UFRJ/COPPE, 2011.

XIII, 92 p. 29, 7cm.

Orientador: Marcos Vicente de Brito Moreira

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2011.

Referências Bibliográficas: p. 89 – 92.

1. Sistema a Eventos Discretos. 2. Autômatos. 3. Verificação de Falhas. 4. Diagnose on-line de falhas. 5. Complexidade Computacional. I. Moreira, Marcos Vicente de Brito. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À Deus, por me conceder meios de chegar onde estou, uma vez que "o homem não pode receber coisa alguma se do céu não lhe for dada"(Jo 3:27), e à toda minha família, principalmente ao meu irmão Diego, por sempre me incentivar e acreditar que eu poderia fazer qualquer coisa, e à minha avó Anita, por quem tenho um amor muito especial.

Agradecimentos

Dedico meus sinceros agradecimentos para:

– Deus, primeiramente e acima de tudo, pois sem ele não sei o que eu seria capaz de fazer;

– a minha família pelo constante e incondicional apoio. Em particular, agradeço ao meu pai, Luiz Batista, por ter feito de tudo para me dar uma boa educação, à minha mãe, Noemia Cerqueira, pelas infinitas orações que demonstram o seu imensurável amor, e à minha tia Lindinalva e à minha avó Anita, que sempre cuidaram de mim como um filho;

– meu irmão, Diego de Jesus, e a sua esposa Tina de Jesus, por depositar em mim uma fé muito maior do que a que eu mesmo tenho;

– o meu primo José de Jesus e sua família, Rose, Rubens e Juliana, por todo apoio, carinho, segurança, conforto e abrigo que me deram na cidade do Rio de Janeiro;

– Mima Barbosa, por ser minha companheira em todos os momentos, me aconselhar, incentivar, compreender e entender sempre. Você é muito especial para mim;

– meus amigos e colegas de apartamento, Victor Ferraz e Paulo Souza, pelas noites e finais de semana de diversão, pelas sugestões e por ter compartilhado de perto comigo essa fase na minha vida;

– meu amigo, professor e orientador Marcos Moreira. Muito obrigado por cada conselho e lição de vida que me fizeram ser uma pessoa melhor. Muito obrigado por sempre exigir e esperar de mim mais do que eu acreditava poder dar. Muito obrigado pela paciência e persistência. Muito obrigado pelo almoço no CBA (=D);

– os amigos Prof. João Carlos Basílio, Lilian Kawakami e Lucas Molina, por toda a ajuda e pela amizade que cultivarei sempre. Eu não tenho palavras para dizer o quanto vocês foram e são importantes para mim;

– todos meus outros verdadeiros amigos, que prefiro não citar seus nomes para não cometer nenhuma injustiça;

– o pessoal do LABCON que me acolheu de braços abertos no primeiro ano de mestrado;

– o pessoal do LCA, que me acolheu de braços abertos no segundo ano de mestrado;

– aos professores Amit Bhaya e Antonio Eduardo Carrilho da Cunha, por avaliarem este trabalho dando dicas valiosas para seu melhoramento e continuidade;

– o Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro;

– a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho, cujos nomes não citei.

A todos vocês, muito obrigado.

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

DIAGNOSTICABILIDADE E DIAGNOSE ON-LINE DE FALHAS DE SISTEMA A EVENTOS DISCRETOS MODELADOS POR AUTÔMATOS FINITOS

Thiago Cerqueira de Jesus

Fevereiro/2011

Orientador: Marcos Vicente de Brito Moreira

Programa: Engenharia Elétrica

A diagnose de falhas é uma tarefa importante em sistemas grandes e complexos e, por isso, tem recebido considerável atenção na literatura. O primeiro passo para diagnosticar a ocorrência de uma falha em um sistema a eventos discretos é a verificação da diagnosticabilidade do sistema. Vários trabalhos na literatura abordam esse problema utilizando diagnosticadores ou verificadores para as arquiteturas centralizada e descentralizada. O segundo passo é a diagnose *on-line*. Ambos os passos requerem a utilização de sensores para a observação dos eventos do sistema. Visando reduzir custos com o uso de sensores, diversos trabalhos na literatura abordam o problema de seleção de sensores.

Neste trabalho um novo algoritmo de complexidade polinomial para a verificação da diagnosticabilidade de um sistema a eventos discretos é proposto. O algoritmo tem complexidade computacional menor do que outros métodos propostos na literatura e pode ser aplicado em ambas as arquiteturas centralizada e descentralizada (codiagnosticabilidade). Baseado nesse algoritmo de verificação, um novo algoritmo para a diagnose de falhas *on-line* é também proposto neste trabalho. Esse algoritmo de diagnose baseia-se na observação dos eventos do sistema apenas quando for estritamente necessário, permitindo que o processo seja interrompido caso a falha seja detectada ou nenhuma falha tenha ocorrido e não possa mais ocorrer.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DIAGNOSABILITY AND ON-LINE FAILURE DIAGNOSIS OF DISCRETE EVENT SYSTEMS MODELED BY FINITE-STATE AUTOMATA

Thiago Cerqueira de Jesus

February/2011

Advisor: Marcos Vicente de Brito Moreira

Department: Electrical Engineering

Failure diagnosis is an important task in large complex systems and, as such, this problem has received considerable attention in the literature. The first step to diagnose failure occurrences in discrete event systems is the verification of the system diagnosability. Several works in the literature address this problem using either diagnosers or verifiers for the centralized and decentralized architectures. The second step is on-line diagnosis. Both steps require the use of sensors to observe the events of the system. In order to reduce the costs related to the use of sensors, several works in the literature address the sensor selection problem.

In this work a new polynomial time algorithm to verify the diagnosability of a discrete event system is proposed. The algorithm has lower computational complexity than other methods proposed in the literature and can be applied to both centralized and decentralized (codiagnosability) architectures. Based on this algorithm, a new algorithm for on-line failure diagnosis of discrete event systems described by finite-state automata is also proposed. The main characteristic of this algorithm is that the system events are observed only when they are strictly necessary for the diagnosis process, allowing that the process be interrupted if a failure is detected or if the failure has not occurred and can not occur anymore.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Sistemas a eventos discretos (SED) são sistemas cujo espaço de estados é um conjunto discreto, e cuja evolução se dá pela ocorrência de eventos. Alguns desses eventos podem levar o sistema a desempenhar um comportamento indesejado, sendo chamados de eventos de falha. Um comportamento indesejado representado por uma falha pode significar, por exemplo, uma falha de segurança como um tanque transbordando, ou uma deterioração do desempenho do sistema, como um motor consumindo mais corrente do que o necessário. Desta forma, motivada pela necessidade prática de assegurar o funcionamento correto e seguro de sistemas grandes e complexos, a diagnose de falha de SED tem recebido considerável atenção na literatura nos últimos anos [1–11]. Diversos trabalhos na literatura abordam o problema de diagnose *on-line* de SED modelados por redes de Petri [1–5] ou por autômatos [6–11]. Este trabalho limita-se a abordagens do problema de diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos.

O problema de diagnosticar uma falha em um SED consiste em inferir a ocorrência de algum evento de falha não observável[1] de falha, baseado nas ocorrências dos eventos observáveis deste sistema. A diagnose de um evento de falha de uma determinada classe pode ser feita usando duas arquiteturas básicas: centralizada ou descentralizada. No caso centralizado, o problema de diagnosticar a ocorrência de uma falha de uma determinada classe pode ser expresso como o problema de identificar a ocorrência desses eventos baseado na observação de uma sequência de eventos observáveis, com atraso limitado, executados pelo sistema.

[1]Um evento é dito ser observável se ele é detectado por algum sensor.

Por outro lado, na arquitetura descentralizada, as observações de eventos são distribuídas entre m diagnosticadores locais, sendo que cada diagnosticador local tem seu próprio conjunto de eventos observáveis. Diversos protocolos para diagnose descentralizada são apresentados em [12]. Neste trabalho, assim como em [13], [14], [15], [16] e [17], o protocolo utilizado considera que não existe comunicação entre os diagnosticadores locais ou entre um diagnosticador local e um coordenador (protocolo 3 de [12]). Neste caso, a ocorrência de eventos de falha de uma determinada classe é diagnosticável se pelo menos um diagnosticador local identificar com um atraso limitado a ocorrência de um evento de falha pertencente a essa classe. Por essa razão, o problema de diagnosticar todas as ocorrências de eventos de falha em uma arquitetura descentralizada sem comunicação entre os diagnosticadores locais ou entre um diagnosticador local e um coordenador é usualmente chamado de codiagnose.

Em [10] e [11], uma abordagem para a diagnose de falhas em SED é apresentada e um diagnosticador é proposto com dois propósitos: (i) detecção *on-line* e isolação de falhas de um sistema e; (ii) verificação *off-line* das propriedades de diagnosticabilidade de um sistema. Em [10] é mostrado que o problema da detecção *on-line* de falhas de um sistema pode ser resolvido com complexidade polinomial em cada passo do procedimento de diagnose. Entretanto, o problema de verificar se uma falha pode ser diagnosticada com um atraso limitado tem, no pior caso, complexidade exponencial no espaço de estados do sistema. Isso se deve ao fato de que a condição necessária e suficiente para a diagnosticabilidade é obtida a partir do diagnosticador *off-line*, cujo espaço de estados cresce, no pior caso, exponencialmente com a cardinalidade do espaço de estados do modelo do sistema.

Para suprir a necessidade de obter o diagnosticador *off-line* para a verificar a diagnosticabilidade de uma sistema a eventos discretos, algoritmos em tempo polinomial, baseados na construção de autômatos não determinísticos e na busca por ciclos com determinadas propriedades, são propostos em [13] e [14], baseados na mesma noção de diagnosticabilidade apresentada em [10]. Em [13] é mostrado que a complexidade computacional do algoritmo é de quarta ordem no número de estados e de primeira ordem no número de eventos do sistema. A complexidade computacional do algoritmo proposto em [14] é de segunda ordem no número de estados e de

primeira ordem no número de eventos do sistema e, portanto, possui menor complexidade computacional do que o método proposto em [13]. É importante observar que em [13] e [14] são feitas as hipóteses de que a linguagem gerada pelo sistema é viva e que não existem ciclos de eventos não observáveis no autômato do modelo do sistema.

Com o objetivo de obter um método para verificar a diagnosticabilidade para o caso descentralizado, ou codiagnosticabilidade, algoritmos em tempo polinomial são propostos em [15], [16] e [17]. Esses algoritmos são baseados na construção de autômatos de teste e na busca por ciclos com determinadas características nesses autômatos. O algoritmo proposto em [15] pode também ser usado para a verificação da diagnosticabilidade e tem complexidade computacional de ordem $(m+1)$ no número de estados e eventos do sistema, sendo m o número de diagnosticadores locais. No caso centralizado, $m = 1$ e a complexidade do algoritmo é de segunda ordem no número de estados e eventos do modelo do sistema, o que é maior do que a ordem da complexidade do algoritmo proposto em [14]. Por outro lado, em [15] as hipóteses de vivacidade da linguagem gerada por um sistema e de não existência de ciclos de eventos não observáveis são removidas. O algoritmo proposto em [16] também remove essas hipóteses e o autômato verificador tem, no máximo, $2^{m+1} \times |X|^{m+1}$ estados e $2^{m+1} \times |X|^{m+1} \times |\Sigma| \times (m+1)$ transições, sendo X e Σ o espaço de estados e o conjunto de eventos do sistema, respectivamente, e $|.|$ denota a cardinalidade de um conjunto.

É importante ressaltar que, embora os métodos de verificação propostos por JIANG *et al.* [13], YOO e LAFORTUNE [14], QIU e KUMAR [15] e WANG *et al.* [16] sejam polinomiais, o número de estados e transições dos seus respectivos verificadores pode ser muito elevado, mesmo para sistemas pequenos e simples. Em muitos casos, esse verificadores chegam a ser maiores do que o próprio diagnosticador *off-line* do sistema proposto por SAMPATH *et al.* [10].

Um outro problema abordado no contexto de diagnose de falhas em SED é a escolha dos sensores necessários para realizar o processo da diagnose. Diversos trabalhos na literatura tratam do problema de seleção de sensores para a diagnose de falhas utilizando diferentes abordagens [18–22]. YOO e LAFORTUNE [19] apresentam um algoritmo em tempo polinomial para verificação da diagnosticabilidade

de SED descritos por autômatos e é mostrado que o problema de encontrar um conjunto de eventos observáveis de menor cardinalidade que mantenha a propriedade de diagnosticabilidade de um SED é NP-completo. Visando estender os resultados de YOO e LAFORTUNE [19], JIANG *et al.* [20] apresentam um algoritmo em tempo polinomial para obtenção de um conjunto ótimo de classes de equivalência de eventos utilizando máscaras de não projeção. O objetivo do método proposto em [20] é obter a informação mínima necessária sobre a observação de eventos para que o sistema permaneça diagnosticável.

Mais recentemente, CASSEZ *et al.* [21] e CASSEZ e TRIPAKIS [22], propõem a construção de um diagnosticador dinâmico que seleciona a cada passo do processo de diagnose um conjunto de eventos observáveis mínimo mantendo a propriedade de diagnosticabilidade do sistema. A motivação para esse problema é a redução no tempo de computação gasto com informações fornecidas pelos sensores consideradas não relevantes e de energia necessária para operar esses sensores, como acontece na utilização de redes de sensores sem fio [23]. Note que o problema formulado por CASSEZ *et al.* [21] e CASSEZ e TRIPAKIS [22] é diferente dos problemas formulados por YOO e LAFORTUNE [19] e JIANG *et al.* [20] uma vez que nestes trabalhos os diagnosticadores são estáticos, ou seja, o conjunto de eventos observáveis permanece o mesmo em todas as etapas da diagnose.

Embora o diagnosticador dinâmico proposto em [21] e [22] leve a uma economia de tempo de computação e energia com relação ao uso de sensores, o método leva, em geral, a um aumento no atraso para a identificação de um evento de falha. Esse atraso é indesejado na maioria dos casos uma vez que pode representar um gasto maior de energia no sistema em comparação com a economia com o uso de sensores, ou até mesmo o alcance de um estado em que não seja mais possível o sistema se recuperar da falha.

Outra forma de reduzir o tempo de computação gasto com informações fornecidas pelos sensores e energia necessária para operar esses sensores, seria desligar os sensores e interromper o processo de diagnose quando ele não fosse mais necessário, ou seja, interromper o processo caso a falha seja diagnosticada ou se a sequência observada indicar que a falha não ocorreu e não pode mais ocorrer no sistema.

Neste trabalho, um novo algoritmo para a verificação da diagnosticabilidade para

os casos centralizado e descentralizado de um SED é proposto [24, 25]. O algoritmo tem menor complexidade computacional do que outros métodos propostos na literatura, e gera um autômato verificador cujo número de estados e transições, em geral, menor do que o número de estados e transições dos autômatos verificadores da diagnosticabilidade de um SED apresentados em [13], [14], [15] e [16], uma vez que somente sequências que podem levar o sistema a ser não diagnosticável são representadas no autômato verificador proposto. As hipóteses de vivacidade da linguagem gerada pelo sistema e de não existência de ciclos de eventos não observáveis são removidas como ocorre em [15] e [16]. Outra contribuição deste trabalho é um novo algoritmo para a diagnose *on-line* centralizada de SED, em que somente as sequências de eventos observáveis estritamente necessárias para a diagnose de falhas são percorridas pelo diagnosticador, ou seja, o processo de diagnose é interrompido caso a falha seja diagnosticada ou então se a sequência observada indicar que a falha não ocorreu e não pode mais ocorrer no sistema [26]. O diagnosticador proposto é baseado no autômato verificador também proposto neste trabalho [24].

Este trabalho está organizado da seguinte forma. No capítulo 2 são apresentados os fundamentos teóricos de diagnose de falhas em sistemas a eventos discretos necessários para o entendimento deste trabalho. No capítulo 3 são apresentados os métodos de verificação da diagnosticabilidade de SED propostos em [13–16], e um novo método para realizar essa verificação é proposto [24]. No capítulo 4 um novo método para a diagnose de falhas em SED, baseado no verificador proposto no capítulo 3 [24], é apresentado [26]. Por fim, no capítulo 5, é feito um resumo dos principais resultados apresentados neste trabalho e são propostos trabalhos futuros.

Capítulo 2

Fundamentos teóricos de diagnose de falhas em sistemas a eventos discretos

Neste capítulo são apresentadas as notações e alguns conceitos preliminares necessários para o entendimento deste trabalho. Dentre eles está o conceito de linguagem gerada por um sistema a eventos discretos, e o conceito de autômato, que é utilizado para representar tais sistemas. São ainda apresentadas as operações realizadas sobre linguagens e autômatos. Além disso, é apresentado o problema da diagnose de falhas em sistemas a eventos discretos, revisando a definição de diagnosticabilidade para os casos centralizado e descentralizado (codiagnosticabilidade). É ainda apresentado um método proposto na literatura para verificação dessas propriedades em sistemas a eventos discretos e, por fim, é apresentada uma maneira de avaliar a complexidade computacional de um algoritmo. Exemplos são utilizados para ilustrar os diversos conceitos.

2.1 Sistemas a eventos discretos

Sistemas a eventos discretos (SED) são sistemas dinâmicos de estados discretos cuja transição de estados se dá através da ocorrência, em geral assíncrona, de eventos. O fato do estado do sistema ser discreto implica que ele pode assumir valores simbólicos, como por exemplo $\{ligado, desligado\}$, $\{verde, amarelo, vermelho\}$, ou

valores discretos tais como valores numéricos pertencentes aos conjuntos $\mathbb{N}$ ou $\mathbb{R}$, ou ser formado por um subconjunto enumerável de elementos de $\mathbb{R}$. Eventos podem estar associados a ações específicas (por exemplo, alguém aperta um botão, um avião levanta vôo etc) ou ser o resultado de diversas condições que são satisfeitas (uma peça atinge um determinado ponto de uma linha de produção, o líquido dentro de um tanque atinge uma determinada altura etc). Embora seja possível modelar qualquer sistema físico como um SED de acordo com o grau de abstração considerado, determinados sistemas são naturalmente discretos e com evolução determinada pela ocorrência de eventos [27, 28].

Assim como na modelagem de sistemas dinâmicos de variáveis contínuas (SDVC), um modelo para um SED deve ser capaz de reproduzir, dentro de limites de tolerância pré-estabelecidos, o comportamento do sistema. Enquanto nos SDVC as trajetórias dos estados são descritas em função do tempo, nos SED elas são função de uma sequência de eventos. Todas as sequências de eventos possíveis de serem geradas por um SED caracterizam a linguagem desse SED, sendo esta definida sobre o conjunto de eventos (alfabeto) do sistema [27]. Assim, ao se considerar a evolução dos estados de um SED, a maior preocupação é com a sequência de estados visitados e com os eventos que causaram as correspondentes transições de estado, isto é, o modelo de um SED é composto basicamente de dois elementos, estados e eventos, como é mostrado no exemplo 2.1, que apresenta um sistema a eventos discretos comum no dia a dia, que é um sistema de uma fila de atendimento em uma clínica.

Exemplo 2.1 *Um sistema em que haja recursos limitados, logo necessite de uma espera para se utilizar estes recursos, é um exemplo de um sistema a eventos discretos. Um caso particular de um sistema de filas é o processo de atendimento em uma clínica. Esse sistema funciona da seguinte forma:*

1. *Os clientes que querem ser atendidos devem ir até a clínica, entrar em uma fila esperando o atendente estar livre para atender;*
2. *A clínica possui um espaço físico para comportar três clientes por vez;*
3. *Quando o atendente estiver livre, o primeiro cliente da fila será atendido;*
4. *Quando o atendimento ao cliente for finalizado, o atendente passa a ficar livre para atender o próximo cliente na fila de espera;*

5. Após ser atendido, o cliente pode deixar a clínica a qualquer momento.

Esse sistema pode ser modelado da seguinte forma: o conjunto de eventos é dado por $\Sigma = \{$ ***"chegada de cliente"***, ***"início de atendimento ao cliente"***, ***"fim de atendimento ao cliente"*** *e* ***"partida de cliente"***$\}$, *e o espaço de estados é* $X = \{$ ***"fila vazia e atendente livre"***, ***"fila com um cliente e atendente livre"***, ***"fila com um cliente e atendente ocupado"***, ***"fila com dois clientes e atendente livre"***, ***"fila com dois clientes e atendente ocupado"***, ***"fila cheia e atendente livre"***, ***"fila cheia e atendente ocupado"*** $\}$. *Note que todos os eventos de* Σ *representam ações que ocorrem em instantes de tempo não determinados e que podem levar o sistema de um estado de* X *para outro. Por exemplo, se o sistema estiver no estado* ***"fila vazia e atendente livre"*** *e houver a* ***"chegada de cliente"***, *então ocorrerá no sistema uma transição para o estado* ***"fila com um cliente e atendente livre"***. *Assim, podemos dizer que este sistema de atendimento em uma clínica é um SED.*

É importante perceber que, assim como em SDVC, a modelagem de um mesmo SED pode ser feita de formas diferentes. Isso depende do grau de abstração que se deseja obter com o sistema, podendo considerar ou desprezar elementos externos ou pouco relevantes. No caso deste exemplo, uma alternativa seria considerar que o cliente deixa a clínica imediatamente após ter o seu atendimento concluído, sendo desnecessário a existência do evento ***"partida de cliente"***. □

2.2 Linguagens

Um SED está necessariamente associado a um conjunto de eventos Σ. Esse conjunto pode ser interpretado como o *alfabeto* de uma linguagem, enquanto que as possíveis sequências formadas por elementos desse conjunto podem ser interpretadas como *palavras* de uma linguagem. Assim, é possível utilizar linguagens para modelar o comportamento de SED. Formalmente, a definição de linguagem é apresentada a seguir [29].

Definição 2.1 (Linguagem) *Uma linguagem definida sobre um conjunto de eventos* Σ *é um conjunto formado por sequências de comprimento finito construídas a partir de eventos pertencentes a* Σ . □

A seguir são apresentados alguns exemplos de linguagens.

Exemplo 2.2 *Seja o conjunto de eventos* $\Sigma = \{a, b, c\}$*, então uma possível linguagem seria*

$$L_1 = \{a, ab, abc\}$$

contendo apenas três sequências. Outra possibilidade de linguagem seria

$$L_2 = \{ \textit{ todas as possíveis palavras de comprimento três, sem o evento } \mathbf{a}\}$$

contendo oito sequências. Ou ainda

$$L_3 = \{ \textit{ todas as possíveis palavras de comprimento finito que terminam com } \mathbf{a}\}$$

contendo um número infinito de sequências. □

Algumas operações podem ser definidas sobre o conjunto de eventos ou sobre sequências, a fim de formar novas sequências. A *concatenação*, por exemplo, consiste em unir duas ou mais sequências para formar uma. A sequência ab é a concatenação dos eventos a e b. A sequência vazia ϵ é o elemento identidade da concatenação, sendo $u\epsilon = \epsilon u = u$, para qualquer sequência u. Por sua vez, o *Fecho de Kleene* de um conjunto Σ, denotado por $\Sigma^\star$, é o conjunto de todas as sequências finitas formadas por elementos de Σ, incluindo a sequência vazia. Assim, $\Sigma^\star$ é um conjunto infinito, uma vez que contém sequências arbitrariamente longas. Se $\Sigma = \{a, b\}$, então $\Sigma^\star = \{\epsilon, a, b, aa, ab, bb, ba, aaa, bbb, \ldots\}$.

Essas operação podem também ser definidas sobre linguagens como segue [28].

Definição 2.2 (Concatenação) *Sejam* $L_1, L_2 \subseteq \Sigma^\star$*, então a concatenação* L_1L_2 *é definida como:*

$$L_1L_2 = \{s \in \Sigma^\star : (s = s_1s_2)[s_1 \in L_1 \ e \ s_2 \in L_2]\}.$$

□

Definição 2.3 (Fecho de Kleene) *Seja* $L \subseteq \Sigma^\star$*, então o fecho de Kleene de* L*,* $L^\star$*, é definido como:*

$$L^\star = \{\epsilon\} \cup L \cup LL \cup LLL \ldots.$$

□

Além dessa operações ainda são definidas, também sobre linguagens, as operações prefixo fechamento e pós linguagem, como apresentado a seguir.

Definição 2.4 (Prefixo fechamento) *Seja $L \subseteq \Sigma^\star$, então o prefixo fechamento de L, $\bar{L}$, é definido como:*

$$\bar{L} = \{s \in \Sigma^\star : (\exists t \in \Sigma^\star)[st \in L]\}.$$

Se $L = \bar{L}$, diz-se que a linguagem L é prefixo-fechada. □

Definição 2.5 (Pós linguagem) *Seja $L \subseteq \Sigma^\star$ e $s \in L$. Então a pós linguagem de L após s, denotada por L/s, é a linguagem*

$$L/s = \{t \in \Sigma^\star : st \in L\}.$$

□

Outra operação bastante útil sobre linguagens é a projeção. Essa operação é definida como em [28, 30].

Definição 2.6 (Projeção) *A projeção $P_s : \Sigma_l^\star \to \Sigma_s^\star$, sendo $\Sigma_s \subset \Sigma_l$, é realizada como segue:*

$$\begin{aligned} P_s(\epsilon) &= \epsilon, \\ P_s(\sigma) &= \begin{cases} \sigma, \ \textit{se } \sigma \in \Sigma_s \\ \epsilon, \ \textit{se } \sigma \in \Sigma_l \setminus \Sigma_s \end{cases}, \\ P_s(s\sigma) &= P_s(s)P_s(\sigma), \ \textit{para todo } s \in \Sigma_l^\star, \sigma \in \Sigma_l, \end{aligned} \tag{2.1}$$

em que $\setminus$ denota a diferença entre conjuntos. □

Pode-se ver que a projeção substitui na sequência $s \in \Sigma_l^\star$ todos eventos $\sigma \in \Sigma_l \setminus \Sigma_s$ pela sequência vazia ϵ, que é equivalente a apagar esses eventos da sequência s.

Outra operação que pode ser realizada sobre uma linguagem é a projeção inversa, que é definida a seguir [28].

Definição 2.7 (Projeção inversa) *A projeção inversa $P_s^{-1} : \Sigma_s^\star \to 2^{\Sigma_l^\star}$ é definida como:*

$$P_s^{-1}(t) = \{s \in \Sigma_l^\star : P_s(s) = t\}. \tag{2.2}$$

□

O exemplo 2.3 mostra como realizar a projeção e a projeção inversa sobre uma linguagem.

Exemplo 2.3 *Considere* $L = \{abc, abb, aba\}$ *a linguagem gerada por um autômato* G*, cujo conjunto de eventos é* $\Sigma = \{a, b, c\}$*. Seja* $P_s : \Sigma^\star \to \Sigma_s^\star$*, com* $\Sigma_s = \{b, c\}$*. Assim,* $P_s(L) = \{bc, bb, b\}$*. A projeção inversa da linguagem* $P_s(L)$ *é o conjunto* $P_s^{-1}(P_s(L)) = \{a\}^\star\{b\}\{a\}^\star\{c\}\{a\}^\star \cup \{a\}^\star\{b\}\{a\}^\star\{b\}\{a\}^\star \cup \{a\}^\star\{b\}\{a\}^\star$. □

Existem diversas formas de representar linguagens que descrevem o comportamento de um SED. Na literatura encontra-se a predominância de representações por autômatos e por redes de Petri. Este trabalho se limita à abordagem por autômatos finitos.

2.3 Autômatos

Um autômato é um dispositivo capaz de representar uma linguagem de acordo com regras bem definidas. Esse dispositivo é comum no contexto de SED por ser fácil de manipular, realizar operações e analisar. Formalmente, podemos definir um autômato determinístico da seguinte forma [28, 29].

Definição 2.8 (Autômato Determinístico) *Um autômato determinístico, denotado por* G*, é uma sêxtupla*

$$G = (X, \Sigma, f, \Gamma, x_0, X_m)$$

em que:

- X *é o conjunto de estados;*
- Σ *é o conjunto finito de eventos;*
- $f : X \times \Sigma \to X$ *é a função de transição de estados;*
- $\Gamma : X \to 2^\Sigma$ *é a função de eventos ativos;*
- x_0 *é o estado inicial;*
- $X_m \subseteq X$ *é o conjunto de estados marcados.*

□

Observação 1 *Por conveniência, f é sempre estendida do domínio $X \times \Sigma$ para o domínio $X \times \Sigma^\star$ de forma recursiva, sendo*

$$\begin{aligned} f(x, \epsilon) &= x \\ f(x, se) &= f\left[f(x, s), e\right] \end{aligned}$$

para qualquer $x \in X$, $e \in \Sigma$ e $s \in \Sigma^\star$. Além disso, por simplicidade de notação, os componentes Γ e X_m são omitidos da definição de determinados autômatos, quando isso não interferir no entendimento do leitor. □

Um autônomo é um conjunto, como mostrado na definição 2.8, porém pode ser representado de forma gráfica por meio de um *diagrama de transição de estados*. Nesse diagrama os *estados* são representados por nós circulares, enquanto que as *transições* são representadas por arcos rotulados pelos *eventos*. O estado inicial é o estado indicado por uma seta, e os estados marcados são representados por círculos duplos concêntricos. O exemplo 2.4 apresenta esses conceitos.

Exemplo 2.4 *Considere o diagrama de transição de estados da figura 2.1. Esse grafo direcionado representa um autômato $G = (X, \Sigma, f, \Gamma, x_0, X_m)$, com $X = \{0, 1, 2\}$, $\Sigma = \{a, b\}$, $X_m = \{2\}$, $x_0 = 0$, $f(0, b) = 1$, $f(0, a) = 2$, $f(1, a) = 2$, $f(2, b) = 2$, $\Gamma(0) = \{a, b\}$, $\Gamma(1) = \{a\}$ e $\Gamma(2) = \{b\}$.* □

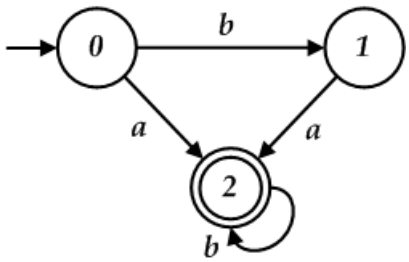


Figura 2.1: Diagrama de transição de estados do autômato do exemplo 2.4.

Quando um evento ocorre em um autômato, mas não gera a mudança de estado, diz-se que o estado tem um *autolaço*, como no caso do estado 2 do exemplo 2.4. Em contrapartida, quando um estado não possui nenhum evento ativo, diz-se ser um estado de *bloqueio* ou *deadlock*.

A definição 2.8 trata especificamente de um autômato determinístico, que difere de um não determinístico no conjunto de eventos, que passa a possuir a sequência vazia ϵ, ou seja, o conjunto de eventos é dado por $\Sigma \cup \{\epsilon\}$, e na função de transição de estados, que passa a ser definida como $f_{nd}^{est} : X \times \Sigma^\star \rightarrow 2^X$. Desta forma, a transição de estado a partir da ocorrência de um evento pode não ser única, e é possível evoluir de um estado para outro por meio da transição rotulada pela sequência vazia ϵ, ou seja, $f(x, \epsilon)$ não é necessariamente igual a x como no caso determinístico.

A ligação entre linguagens e autômatos pode ser feita inspecionando-se o diagrama de transição de estados de um autômato. Para tanto, é preciso apresentar a definição de caminho em um autômato, como segue.

Definição 2.9 (Caminho) *Em um autômato, um caminho* (x_k, σ_1, x_{k+1}, σ_2, ..., σ_l, x_{k+l}), *para* $l > 0$, *é a sequência de estados e eventos tais que* $x_{k+i} = f(x_{k+i-1}, \sigma_i)$ *para todo* $i \in \{1, 2, \ldots, l\}$. *O caminho forma um ciclo se* $x_{k+l} = x_k$. □

Assim, considere todos os caminhos que podem ser percorridos a partir do estado inicial; considere agora, entre esses caminhos, aqueles que terminam em um estado marcado. Isso leva às definições de linguagem gerada e linguagem marcada por um autômato [28].

Definição 2.10 (Linguagem gerada) *A linguagem gerada por um autômato* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ *é definida como*

$$\mathcal{L}(G) = \{s \in \Sigma^\star : f(x_0, s) \text{ é definida}\}.$$

□

Definição 2.11 (Linguagem marcada) *A linguagem marcada por um autômato* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ *é definida como*

$$\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) : f(x_0, s) \in X_m\}.$$

□

CASSANDRAS e LAFORTUNE [28] ressaltam que a linguagem $\mathcal{L}(G)$ representa todos os caminhos direcionados compatíveis com o diagrama de transição de estados, começando do estado inicial. A sequência correspondente a um caminho é a

concatenação dos rótulos dos eventos das transições pertencentes a este caminho. Assim, uma sequência s pertence a $\mathcal{L}(G)$ se, e somente se, ela corresponde a um caminho admissível no diagrama de transição de estados de G, ou seja, se $f(x_0, s)$ é definida. De forma equivalente, a linguagem marcada $\mathcal{L}_m(G)$, que é um subconjunto de $\mathcal{L}(G)$, consiste em todas as sequências s tais que$f(x_0, s) \in X_m$, isto é, o conjunto das sequências correspondentes ao caminhos que terminam em um estado marcado do diagrama de transição de estados. Em geral, a linguagem marcada representa a linguagem de interesse de um autômato, podendo representar, por exemplo, a finalização de uma tarefa, ou a disponibilidade de um recurso físico do sistema.

Exemplo 2.5 *Considere o autômato G da figura 2.2, e suponha que $\Sigma = \{a, b, c\}$. Assim, a linguagem gerada por G é $\mathcal{L}(G) = \overline{\{\{a, b\}\{c\}\}^\star}$, enquanto que a linguagem marcada por G é $\mathcal{L}_m(G) = \{a, b\}\{\{c\}\{a, b\}\}^\star$, consistindo de todas as sequências de $\mathcal{L}(G)$ que terminam com os eventos a ou b, com possíveis ocorrências do evento c intercaladas com as ocorrências de a ou b.* □

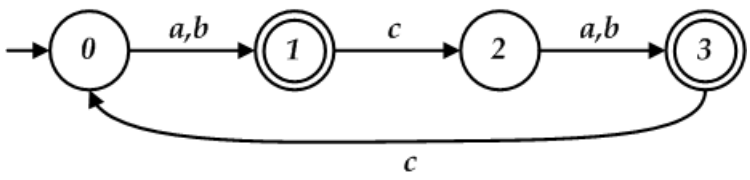


Figura 2.2: Autômato que gera $\{\{a, b\} \cup \{a, b\}\{c\}\}^\star$ e marca $\{a, b\}\{\{c\}\{a, b\}\}^\star$.

Em alguns casos, apenas uma parte do autômato se mostra pertinente para análise, como, por exemplo, o conjunto de estados a partir dos quais seja possível alcançar um estado marcado. Além disso, muitas vezes se deseja criar um novo autômato a partir da composição de outros autômatos. Para tanto, devem ser realizadas operações com autômatos que são mostradas a seguir.

2.4 Operações com Autômatos

Para a análise de sistemas a eventos discretos modelados por autômatos, em geral, um conjunto de operações se faz necessário, seja para modificar um autômato, ou para combinar ou compor dois ou mais autômatos a fim de representar um sistema completo a partir de modelos de componentes individuais do sistema.

A parte acessível de um autômato G é a operação unária que elimina todos os estados de G que não são alcançáveis a partir do estado inicial x_0, assim como as transições relacionadas com esses estados eliminados. Formalmente, a parte acessível de G é definida como a seguir [28].

Definição 2.12 (Parte Acessível) *Seja $G = (X, \Sigma, f, x_0, X_m)$. A parte acessível de G, denotada por $Ac(G)$, é o subautômato*

$$Ac(G) \quad = \quad (X_{ac}, \Sigma, f_{ac}, x_0, X_{ac,m})$$

sendo,

$$\begin{aligned} X_{ac} &= \{x \in X : (\exists s \in \Sigma^\star)[f(x_0, s) = x]\} \ ; \\ X_{ac,m} &= X_m \cap X_{ac} \ ; \\ f_{ac} &: X_{ac} \times \Sigma \to X_{ac} \ ; \end{aligned}$$

em que f_{ac} denota a nova função de transição obtida restringindo-se o domínio de f para o domínio dos estados acessíveis X_{ac}, ou seja, os estados que podem ser alcançados a partir do estado inicial. □

A parte coacessível de um autômato G é também uma operação unária e é obtida eliminando-se todos os estados de G a partir dos quais não é possível alcançar um estado marcado. Essa operação é formalmente definida a seguir [28].

Definição 2.13 (Parte Coacessível) *Seja $G = (X, \Sigma, f, x_0, X_m)$. A parte coacessível de G, denotada por $CoAc(G)$, é o subautômato*

$$CoAc(G) \quad = \quad (X_{coac}, \Sigma, f_{coac}, x_{0,coac}, X_m)$$

sendo,

$$\begin{aligned} X_{coac} &= \{x \in X : (\exists s \in \Sigma^\star)[f(x, s) \in X_m]\} \\ x_{0,coac} &= \begin{cases} x_0, & \text{se } x_0 \in X_{coac} \\ \text{indefinido}, & \text{se } x_0 \notin X_{coac} \end{cases} \\ f_{coac} &: X_{coac} \times \Sigma \to X_{coac} \end{aligned}$$

em que f_{coac} denota a nova função de transição obtida restringindo-se o domínio de f aos estados coacessíveis X_{coac}, ou seja, os estados a partir dos quais pode-se alcançar um estado marcado. □

Exemplo 2.6 *Considere o autômato determinístico G apresentado na figura 2.3(a). Assim, é possível obter o autômato da parte acessível de G, $Ac(G)$, apresentado na figura 2.3(b), assim como o autômato da coacessível de G, $CoAc(G)$, apresentado na figura 2.3(c). Neste trabalho, todos os autômatos são considerados acessíveis.* □

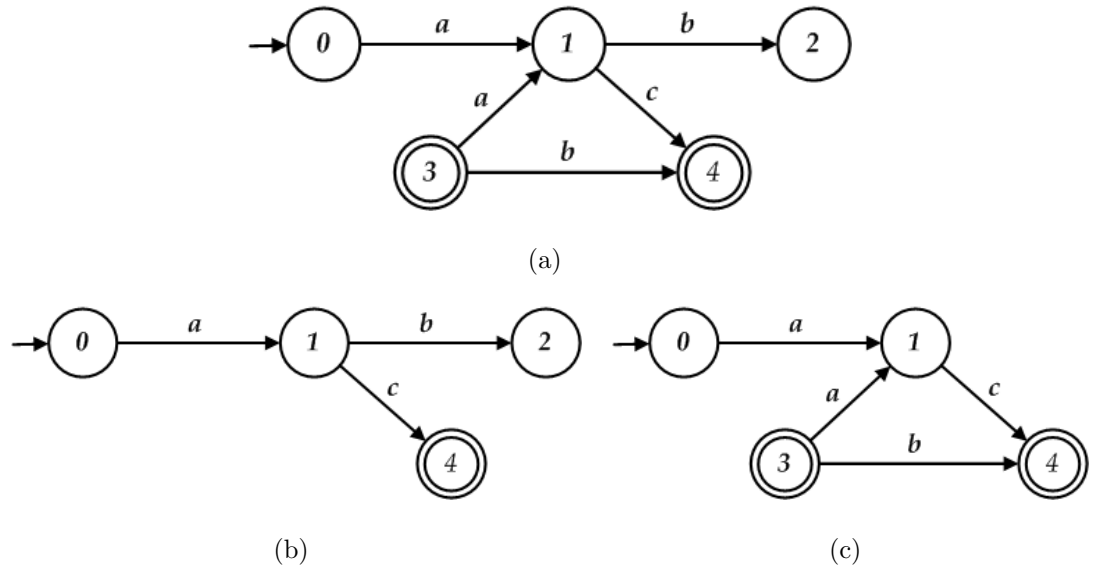


Figura 2.3: (a) Autômato G, (b) parte acessível e (c) parte coacessível de G.

As operações de projeção e projeção inversa, que foram definidas sobre linguagens, também podem ser aplicadas sobre autômatos. A projeção das linguagens $L = \mathcal{L}(G)$ e $L_m = \mathcal{L}_m(G)$ pode ser implementada em G substituindo-se todas as transições rotuladas por eventos de $\Sigma_l \setminus \Sigma_s$ pela sequência vazia ϵ, levando a um autômato não determinístico. Para obter um autômato determinístico que gere a linguagem $P_s(L)$, um observador deve ser calculado [28]. Por outro lado, é possível modificar o autômato G para que esse novo autômato gere a linguagem $P_s^{-1}(L)$ e marque a linguagem $P_s^{-1}(L_m)$. Para tanto, basta adicionar a cada estado de G um autolaço rotulado por todos os eventos em $\Sigma_l \setminus \Sigma_s$. Com um leve abuso de notação, esse autômato que gera $P_s^{-1}(L)$ e marca $P_s^{-1}(L_m)$ é denotado neste trabalho por $P_s^{-1}(G)$.

Observação 2 *As operações de parte acessível, coacessível, projeção e projeção inversa são aqui definidas para autômatos determinísticos, porém podem ser definidas e implementadas de forma similar em autômatos não determinísticos [28].* □

Além das operações unárias, em alguns casos, pode ser feita uma composição de

dois ou mais autômatos para formar um novo autômato. Para tanto, pode-se usar o *produto* e a *composição paralela* definidos a seguir [28].

Definição 2.14 (Produto) *Sejam os autômatos* $G_1 = (X_1, \Sigma_1, \Gamma_1, f_1, x_{01}, X_{m1})$ *e* $G_2 = (X_2, \Sigma_2, \Gamma_2, f_2, x_{02}, X_{m2})$. *Então, o produto de* G_1 *e* G_2, *denotado por* $G_1 \times G_2$, *é definido como:*

$$G_1 \times G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\times 2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}), \tag{2.3}$$

sendo

$$f_{1\times 2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), \text{ se } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2); \\ \text{indefinido, caso contrário.} \end{cases} \tag{2.4}$$

□

Note que a transição no produto $G_1 \times G_2$ ocorre se, e somente se, a transição é possível em ambos autômatos G_1 e G_2. Isso significa que a evolução de estados em $G_1 \times G_2$ é completamente sincronizada com a evolução de estados dos autômatos G_1 e G_2. Uma consequência importante desse fato é que a linguagem gerada por $G_1 \times G_2$ é igual a $\mathcal{L}(G_1) \cap \mathcal{L}(G_2)$, e a linguagem marcada é $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$ [28].

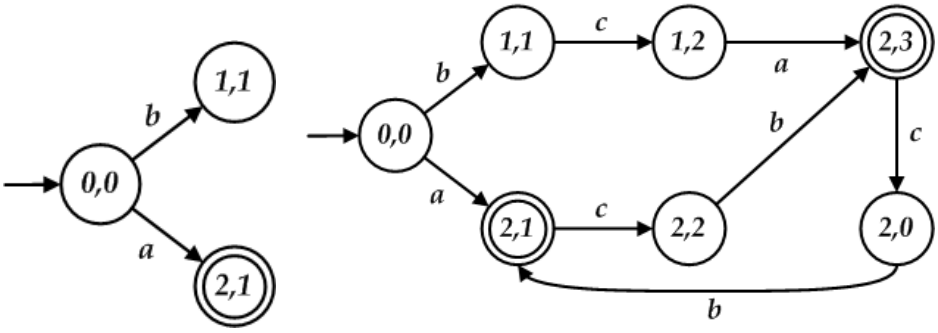
Definição 2.15 (Composição paralela) *Sejam* $G_1 = (X_1, \Sigma_1, \Gamma_1, f_1, x_{01}, X_{m1})$ *e* $G_2 = (X_2, \Sigma_2, \Gamma_2, f_2, x_{02}, X_{m2})$. *Então, a composição paralela, denotada por* $G_1 \| G_2$, *é definida como:*

$$G_1 \| G_2 = P_1^{-1}(G_1) \times P_2^{-1}(G_2) \tag{2.5}$$

sendo $P_i : (\Sigma_1 \cup \Sigma_2)^\star \to \Sigma_i^\star$, *para* $i = 1, 2$. □

Utilizando-se as projeções P_i, pode-se definir as linguagens gerada e marcada da composição paralela $G_1 \| G_2$, respectivamente, como $\mathcal{L}(G_1 \| G_2) = P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)]$ e $\mathcal{L}_m(G_1 \| G_2) = P_1^{-1}[\mathcal{L}_m(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)]$.

Exemplo 2.7 *A figura 2.4(a) mostra o produto entre os autômatos das figuras 2.1 e 2.2, e a figura 2.4(b) apresenta o autômato resultante da composição paralela desses autômatos.* □



(a) (b)

Figura 2.4: Autômatos resultantes do (a) produto e (b) composição paralela dos autômatos das figuras 2.1 e 2.2.

2.5 SED parcialmente observados

SED parcialmente observados são sistemas cujos conjuntos de eventos são particionados em dois subconjuntos disjuntos: Σ_o e Σ_{uo}, ou seja, $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. Σ_o consiste no conjunto de eventos que podem ser observados (observáveis), e Σ_{uo} consiste no conjunto de eventos que não podem ser observados (não observáveis). Um evento é dito ser observável quando sua ocorrência puder ser registrada através de sensores ou quando estiver associado a comandos. Os eventos não observáveis, por sua vez, designam aqueles eventos do sistema cuja ocorrência não pode ser observada por sensores, ou que ocorrem em um local remoto, mas não são comunicados a um coordenador, o que é uma situação típica em um sistema de informação distribuída. Eventos de falha, que não causam mudanças imediatas nas leituras de sensores, são também exemplos de eventos não observáveis [27, 28].

Seja G um autômato determinístico que modela o comportamento de um SED. A linguagem gerada observada de G é a linguagem formada por todas as sequências de $\mathcal{L}(G)$ desconsiderando-se os eventos não observáveis, ou seja, a linguagem gerada observada de G é igual a $P_o[\mathcal{L}(G)]$, sendo $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$. Do mesmo modo, a linguagem marcada observada de G é $P_o[\mathcal{L}_m(G)]$. Visando obter um autômato determinístico cujas linguagens gerada e marcada são iguais às linguagens gerada e marcada observadas de um autômato parcialmente observado, deve-se obter um observador. Contudo, antes de apresentar esse conceito, é necessário definir alcance não observável de um estado $x \in X$, denotado por $UR(x)$, como segue [28]:

$$UR(x) = \{y \in X : (\exists t \in \Sigma_{uo}^\star)[f(x,t) = y]\}. \tag{2.6}$$

O alcance não observável é também definido para um conjunto $B \in 2^X$ como:

$$UR(B) = \bigcup_{x \in B} UR(x). \tag{2.7}$$

Note que o alcance não observável de um estado x gera um conjunto de informações de todos os estados que podem ser alcançados a partir de x por transições rotuladas por eventos não observáveis. Assim, o alcance não observável fornece uma estimativa dos estados alcançados a partir de x por transições não observáveis. A seguir é apresentado o conceito de observador, que utiliza o alcance não observável para gerar uma estimativa de estados alcançados pela execução de uma sequência observável de eventos a partir do estado inicial. O observador de um autômato G em relação a um conjunto de eventos observáveis Σ_o, denotado por $Obs(G, \Sigma_o)$, é definido como [28]:

$$Obs(G, \Sigma_o) = (X_{Obs}, \Sigma_o, f_{Obs}, \Gamma_{Obs}, x_{0,Obs}, X_{m_{Obs}}), \tag{2.8}$$

sendo $X_{Obs} \subseteq 2^X$ e $X_{m_{Obs}} = \{B \in X_{Obs} : B \cap X_m \neq \emptyset\}$. f_{Obs}, Γ_{Obs} e $x_{0,Obs}$ são definidos de acordo com o algoritmo 2.1, que apresenta o procedimento para a criação de um observador.

Algoritmo 2.1 (Observador)

- *Passo 1: Defina* $x_{0,Obs} = UR(x_0)$ *e faça* $X_{Obs} = \{x_{0,Obs}\}$ *e* $\tilde{X}_{Obs} = X_{Obs}$.
- *Passo 2:* $\overline{X}_{Obs} = \tilde{X}_{Obs}$ *e* $\tilde{X}_{Obs} = \emptyset$.
- *Passo 3: Para cada* $B \in \overline{X}_{Obs}$,
  - *Passo 3.1:* $\Gamma_{Obs}(B) = \left(\bigcup_{x \in B} \Gamma(x)\right) \cap \Sigma_o$.
  - *Passo 3.2: Para cada* $e \in \Gamma_{Obs}(B)$

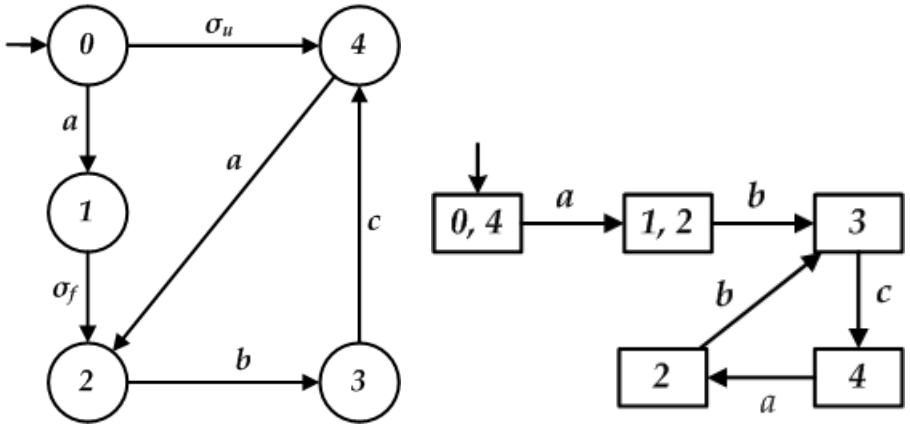
    $$f_{Obs}(B, e) = UR(\{x \in X : (\exists y \in B)[x = f(y, e)]\}).$$

  - *Passo 3.3:* $\tilde{X}_{Obs} \leftarrow \tilde{X}_{Obs} \cup \{f_{Obs}(B, e)\}$.
- *Passo 4:* $X_{Obs} \leftarrow X_{Obs} \cup \tilde{X}_{Obs}$.

- *Passo 5: Repita os passos 2 a 4 até que toda a parte acessível de* $Obs(G, \Sigma_o)$ *tenha sido construída.*

- *Passo 6:* $X_{m_{Obs}} = \{B \in X_{Obs} : B \cap X_m \neq \emptyset\}$. □

Exemplo 2.8 *Considere o autômato G mostrado na figura 2.5(a), com* $\Sigma_o = \{a, b, c\}$, $\Sigma_{uo} = \{\sigma_f, \sigma_u\}$. *A figura 2.5(b) mostra o observador de G, construído de acordo com o algoritmo 2.1.* □



(a) (b)

Figura 2.5: (a) Autômato G e (b) observador de G, $Obs(G, \Sigma_o)$.

Os conceitos apresentados até aqui formam a base teórica necessária para tratar do problema de diagnose de falhas em SED. Esse problema é formulado a seguir, tanto para uma arquitetura centralizada quanto para uma arquitetura descentralizada. São apresentadas ainda as definições de linguagens diagnosticável e codiagnosticável, bem como um método para verificar a diagnosticabilidade e realizar a diagnose de falhas nas arquiteturas centralizada e descentralizada.

2.6 Diagnose de falhas em SED

Seja $G = (X, \Sigma, f, x_0, X_m)$ um autômato determinístico que modela um SED parcialmente observado, ou seja, $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, e seja $\Sigma_f \subseteq \Sigma_{uo}$ o conjunto dos eventos de falha do sistema. Considere que o conjunto Σ_f pode ser particionado em diferentes subconjuntos Σ_{fi}, $i = 1, 2, \ldots, l$, não necessariamente unitários, em que cada conjunto Σ_{fi} é formado por eventos que modelam falhas que sejam, de alguma forma,

correlacionadas, ou seja, falhas de um mesmo tipo. Suponha que

$$\Pi_f = \{\Sigma_{f1}, \Sigma_{f2}, \ldots, \Sigma_{fl}\} \tag{2.9}$$

denote uma partição do conjunto de eventos de falha, ou seja

$$\Sigma_f = \bigcup_{i=1}^{l} \Sigma_{fi}. \tag{2.10}$$

Assim, cada vez que for dito que uma falha do tipo F_i ocorre, deve ser entendido que algum evento do conjunto Σ_{fi} ocorreu.

A diagnose de um evento de falha pertencente ao conjunto Σ_{fi} pode ser feita usando duas arquiteturas básicas: centralizada ou descentralizada. No caso centralizado, o problema de diagnosticar a ocorrência de uma falha da classe Σ_{fi} pode ser expresso como o problema de diagnosticar a ocorrência desses eventos baseado na observação de uma sequência finita de eventos gerada por G, sendo que apenas os eventos em Σ_o são observados. Nesse caso, as ocorrências de eventos são observadas por um único autômato diagnosticador $Diag(G)$, distinguindo os eventos observáveis do sistema dos não observáveis por meio da projeção P_o, como mostra a figura 2.6. Esse autômato diagnosticador se encarrega de, baseado nos eventos observáveis, gerar informações que possibilitam diagnosticar a ocorrência de algum evento de falha.

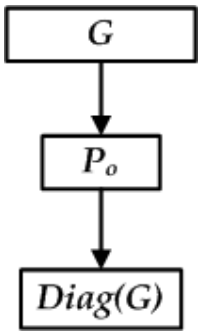


Figura 2.6: Arquitetura centralizada.

Por outro lado, na arquitetura descentralizada, as observações de eventos são distribuídas entre m diagnosticadores locais $Diag_i(G)$, sendo que cada diagnosticador tem seu próprio conjunto de eventos observáveis Σ_{o_i}, para $i = 1, \ldots, m$, sendo $\Sigma_o = \bigcup_{i=1}^{m} \Sigma_{o_i}$, além de ser considerado que não existe comunicação entre os diagnosticadores locais ou entre um diagnosticador local e um coordenador. Nesse caso, as

ocorrências de eventos são observadas pelos m diagnosticadores locais, considerando as projeções $P_{o_i} : \Sigma^\star \rightarrow \Sigma_{o_i}^\star$, $i = 1, \ldots, m$, como mostra a figura 2.7. Assim, na arquitetura descentralizada a ocorrência de eventos de falha da classe Σ_{fi} é diagnosticável se pelo menos um diagnosticador local identificar a ocorrência de um evento de falha pertencente ao conjunto Σ_{fi}. Por essa razão, o problema de diagnosticar todas as ocorrências de eventos de falha em uma arquitetura descentralizada, como a descrita anteriormente, é usualmente chamado de codiagnose.

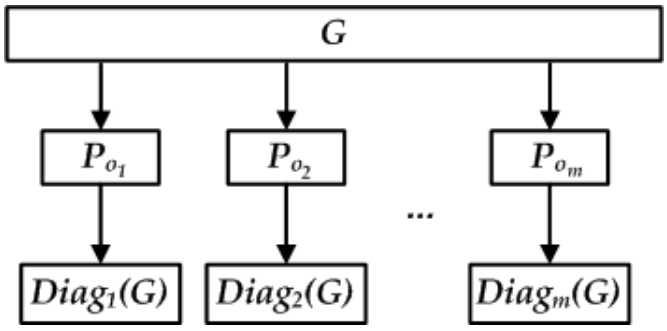


Figura 2.7: Arquitetura descentralizada.

Nas definições 2.16 e 2.17 são apresentados os conceitos de linguagem diagnosticável e de linguagem codiagnosticável de um SED, respectivamente [15]. Para tanto, considere a linguagem $L_N \subseteq L$ que representa o comportamento de não falha do sistema. Assim, L_N é uma linguagem prefixo-fechada formada por todos os traços de L que não contém qualquer evento de falha pertencente ao conjunto Σ_f.

Definição 2.16 (Linguagem diagnosticável) *Seja L a linguagem prefixo-fechada gerada por um autômato G e seja $L_N \subset L$ a linguagem prefixo-fechada do comportamento normal do sistema. Considere a partição do conjunto de eventos $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ e seja $\Sigma_f \subseteq \Sigma_{uo}$ o conjunto de eventos de falha. Então, L é diagnosticável em relação a $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$ e Σ_f se, e somente se,*

$$(\exists n \in \mathbb{N})(\forall s \in L \setminus L_N)$$
$$(\forall st \in L \setminus L_N, |t| \geq n \text{ ou } st \text{ leva a um estado de bloqueio }) \Rightarrow$$
$$(\forall w \in P_o^{-1}(P_o(st)) \cap L, w \in L \setminus L_N).$$

□

Definição 2.17 (Linguagem codiagnosticável) *Seja L a linguagem prefixo-fechada gerada por um autômato G e seja $L_N \subset L$ a linguagem prefixo-fechada do comportamento normal do sistema. Suponha que existam m módulos locais com projeções $P_{o_i} : \Sigma^\star \to \Sigma_{o_i}^\star$ ($i \in I_M = \{1, \ldots, m\}$) e seja $\Sigma_f \subseteq \Sigma_{uo}$ o conjunto de eventos de falha. Então, L é codiagnosticável em relação a P_{o_i} e Σ_f se, e somente se,*

$$
\begin{aligned}
&(\exists n \in \mathbb{N})(\forall s \in L \setminus L_N) \\
&(\forall st \in L \setminus L_N, |t| \geq n \text{ ou } st \text{ leva a um estado de bloqueio }) \Rightarrow \\
&(\exists i \in I_M)(\forall w \in P_{o_i}^{-1}\left[P_{o_i}(st)\right] \cap L, w \in L \setminus L_N).
\end{aligned}
$$

□

De acordo com a definição 2.16, L é diagnosticável em relação a P_o e Σ_f se, e somente se, para todas as sequências st de comprimento arbitrariamente longo após a ocorrência de um evento de falha ou que levam a um estado de bloqueio, não existe uma sequência $s' \in L_N$, tal que $P_o(s') = P_o(st)$. De forma similar, segundo a definição 2.17, L é codiagnosticável com relação a P_{o_i} e Σ_f se, e somente, se para todas as sequências st de comprimento arbitrariamente longo após a ocorrência de um evento de falha ou que levam a um estado de bloqueio, não existem sequências $s_i \in L_N$, não necessariamente distintas, tais que $P_{o_i}(s_i) = P_{o_i}(st)$ para todo $i = 1, \ldots, m$. Note que, se um sistema é não diagnosticável, então ele também é não codiagnosticável, uma vez que a arquitetura descentralizada concede menos informações sobre o sistema do que a arquitetura centralizada, no sentido de eventos observados.

Para verificar as propriedades de diagnosticabilidade da linguagem gerada por um autômato, pode-se utilizar um autômato *verificador* (apresentado no capítulo 3), que basicamente gera informações sobre a existência de sequências que violam as condições da definição 2.16, para o caso centralizado, ou que violam as condições da definição 2.17, para o caso descentralizado. Pode-se ainda utilizar um autômato *diagnosticador* para essa tarefa. O diagnosticador ainda é capaz de realizar a diagnose *on-line* no caso centralizado, e pode ser utilizado para o caso descentralizado, se feitas algumas modificações. A seguir, é apresentado o diagnosticador proposto por SAMPATH *et al.* [10].

2.7 Diagnosticador proposto por Sampath *et al.* [10]

O diagnosticador proposto por SAMPATH *et al.* [10] apresenta dois propósitos: (*i*) detecção *on-line* e isolação de falhas de um sistema e; (*ii*) verificação *off-line* das propriedades de diagnosticabilidade de um sistema. O método original para a construção do diagnosticador é apresentado em [10] e reformulado em [28], porém mantendo o resultado. Aqui é apresentada a abordagem de CASSANDRAS e LAFORTUNE [28]. Em [10, 28] são supostas as seguintes hipóteses:

H1.A linguagem gerada por G é viva, isto é, $\Gamma(x) \neq \emptyset$ para todo $x \in X$;
H2.O autômato G não possui nenhum ciclo formado somente por eventos não observáveis.

Note que, de acordo com a hipótese $H1$, não existe uma sequência $st \in L$ tal que $\sigma_f \in s$, e st leva a um estado de bloqueio. Assim, para a verificação da diagnosticabilidade centralizada ou codiagnosticabilidade de um SED, apenas sequências de comprimento arbitrariamente longo após a ocorrência de um evento de falha são consideradas quando a hipótese $H1$ é válida.

O diagnosticador de um autômato G, denotado por $Diag(G)$, é construído fazendo o observador da composição paralela de G com o autômato A_{label} apresentado na figura 2.8 (aqui, por simplicidade, é suposto que $\Sigma_f = \{\sigma_f\}$), sendo este último um autômato de rótulos: ele permanece no estado N enquanto não ocorrer falha, e muda para o estado Y quando a falha ocorre. Assim,

$$Diag(G) = Obs(G \| A_{label}, \Sigma_o) \tag{2.11}$$

é um estimador de estados, já que ele é um observador, e um estimador da ocorrência de falha, uma vez que rótulos Y e N são atribuídos ao estados indicando se um determinado estado foi alcançado por uma sequência contendo um evento de falha ou não. É importante ressaltar que $\mathcal{L}(G \| A_{label}) = L$ e que $\mathcal{L}(Diag(G)) = P_o\left[\mathcal{L}(G \| A_{label})\right] = P_o(L)$.

Se todos os estados de G no estado corrente de $Diag(G)$ possuem o rótulo N, então temos a certeza de que uma falha não ocorreu. Assim, esse estado é chamado

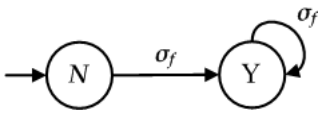


Figura 2.8: Autômato A_{label}.

de estado *negativo* ou *normal*. Por outro lado, se todos os estados de G no estado corrente de $Diag(G)$ possuem o rótulo Y, então temos certeza de que uma falha ocorreu, sendo esse, portanto, um estado *positivo* ou *certo*. Caso haja estados em $Diag(G)$ contendo ao menos um estado rotulado por N e ao menos um estado rotulado por Y, então tem-se um estado *incerto*, uma vez que não é possível precisar a ocorrência da falha. Se em $Diag(G)$ existe um ciclo tal que todos os estados que o compõem sejam estados incertos, então esse ciclo é chamado de *ciclo incerto*. Se um ciclo incerto em $Diag(G)$ pode ser associado a dois ciclos em $G \| A_{label}$, sendo que todos os estados que compõem um desses ciclos sejam apenas estados rotulados por N, e todos os estados que compõem o outro ciclo sejam apenas estados rotulados por Y, então esse ciclo em $Diag(G)$ é um ciclo *indeterminado*.

A existência de um ciclo indeterminado implica que a linguagem gerada por G é não diagnosticável. Para verificar esse fato, considere $s_D \in \mathcal{L}(Diag(G))$ a sequência associada a um caminho em $Diag(G)$ que contenha um ciclo indeterminado. Logo, existe uma sequência $st \in \mathcal{L}(G)$ associada a um caminho com um ciclo de estados positivos em $G \| A_{label}$, tal que $\sigma_f \in s$, t possui comprimento arbitrariamente longo, e $P_o(st) = s_D$, assim como existe uma sequência $s' \in \mathcal{L}(G)$ associada a um caminho com um ciclo de estados normais em $G \| A_{label}$, tal que $\sigma_f \notin s'$ e $P_o(s') = s_D$. Portanto, a existência desse ciclo implica que existe uma sequência $st \in L \setminus L_N$ de comprimento arbitrariamente longo após a falha, e uma sequência $s' \in L_N$, tais que $P_o(st) = P_o(s')$, o que, de acordo com a definição 2.16, caracteriza uma linguagem ser não diagnosticável.

É importante notar a necessidade das hipóteses $H1$ e $H2$. Se a hipótese $H1$ não fosse verdadeira, então poderia existir uma sequência $u \in L$ de comprimento limitado que leva a um estado de bloqueio em G, tal que $\sigma_f \in u$, e uma sequência $s' \in L_N$, tais que $P_o(u) = P_o(s')$, o que, de acordo com a definição 2.16, caracteriza L como uma linguagem não diagnosticável em relação a P_o e Σ_f. Porém, uma vez que $\mathcal{L}(Diag(G)) = P_o\left[\mathcal{L}(G \| A_{label})\right] = P_o(L)$, então a sequência $P_o(u) = P_o(s')$ está

associada a um caminho em $Diag(G)$, entretanto, esse caminho não forma um ciclo, uma vez que u é uma sequência de comprimento limitado e, consequentemente, a sequência $P_o(u)$ não possui comprimento arbitrariamente longo. Desta forma, se a hipótese $H1$ for falsa, a não existência de um ciclo indeterminado em $Diag(G)$ não é uma condição necessária e suficiente para a diagnosticabilidade em SED. A não existência de um ciclo indeterminado também deixa de ser condição necessária e suficiente caso a hipótese $H2$ não seja verdadeira. Nesse caso, pode existir uma sequência st, sendo s uma sequência de comprimento limitado, $\sigma_f \in s$ e $t \in \Sigma_{uo}^{\star}$ uma sequência arbitrariamente longa, e pode existir uma sequência $s' \in L_N$, tais que $P_o(st) = P_o(s')$, o que, de acordo com a definição 2.16, caracteriza L como uma linguagem não diagnosticável em relação a P_o e Σ_f. Entretanto, como t é uma sequência formada apenas por eventos não observáveis, então $P_o(st) = P_o(s)$ e $P_o(st)$ possui comprimento limitado. Assim, como a sequência $P_o(st) \in \mathcal{L}(Diag(G))$, ela está associada a um caminho em $Diag(G)$, porém esse caminho não forma um ciclo, fazendo com que a não existência de um ciclo indeterminado $Diag(G)$ não seja uma condição necessária e suficiente para a diagnosticabilidade em SED.

O teorema a seguir estabelece a condição necessária e suficiente para a verificação da diagnosticabilidade por meio do diagnosticador proposto por SAMPATH *et al.* [10].

Teorema 2.1 *Suponha que as hipóteses $H1$ e $H2$ sejam válidas. Então, uma linguagem L gerada por um autômato G é diagnosticável com relação à projeção $P_o : \Sigma^{\star} \rightarrow \Sigma_o^{\star}$ e ao conjunto de eventos de falha Σ_f, se, e somente se, o diagnosticador $Diag(G)$ não possui ciclos indeterminados.* □

Demonstração Ver [10]. □

A seguir é apresentado o algoritmo 2.2, que detalha o método de criação do diagnosticador e mostra como usá-lo para a verificação da diagnosticabilidade para o caso centralizado.

Algoritmo 2.2 *Seja $G = (X, \Sigma, f, x_0)$ o autômato que modela um SED e considere a partição $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, e a projeção $P_o : \Sigma^{\star} \rightarrow \Sigma_o^{\star}$.*

- *Passo 1: Defina* $A_{label} = (\{N, Y\}, \Sigma_f, f_{A_{label}}, \{N\})$, *sendo* $f_{A_{label}}(N, \sigma_f) = Y$ *e* $f_{A_{label}}(Y, \sigma_f) = Y$, *para todo* $\sigma_f \in \Sigma_f$ *(as únicas transições definidas nesse autômato).*

- *Passo 2: Calcule o diagnosticador* $Diag(G) = (X_{Diag}, \Sigma_o, f_{Diag}, x_{0,Diag}) = Obs(G \| A_{label}, \Sigma_o)$.

- *Passo 3: Verifique se existe em* $Diag(G)$ *um ciclo* $cl = (x^k_{Diag}, \sigma_k, x^{k+1}_{Diag}, \ldots, x^l_{Diag}, \sigma_l, x^k_{Diag})$, $l \geq k \geq 0$, *com* $x^i_{Diag} \in X_{Diag}$, *tal que cl é indeterminado. Se cl existe, então L é não diagnosticável com relação a* Σ_f *e* P_o. *Caso contrário, L é dita ser diagnosticável.* □

Em [28] é apresentado também um método para a codiagnose de falhas baseado no método de SAMPATH *et al.* [10]. Esse método é baseado na construção de diagnosticadores locais, $Diag_i(G)$, $i = 1, \ldots, m$, sendo que cada diagnosticador possui um conjunto de eventos observáveis Σ_{o_i}. Assim, cada diagnosticador local $Diag_i(G)$ é criado de acordo com os passos 1 e 2 do algoritmo 2.2, porém considerando como conjunto de eventos observáveis o conjunto Σ_{o_i} no lugar de Σ_o, ou seja, $Diag_i(G) = Obs(G \| A_{label}, \Sigma_{o_i})$. Assim como no caso centralizado, é possível verificar as condições necessárias e suficientes para a diagnosticabilidade descentralizada por meio de um diagnosticador *off-line*. Esse autômato de teste é criado da seguinte forma:

$$CoDiag(G) = [\|_{i=1}^{m} Diag_i(G)] \, \| Diag(G) \tag{2.12}$$

A verificação da codiagnosticabilidade de um SED por meio do autômato $CoDiag(G)$ também se baseia na busca por ciclos indeterminados, porém esses ciclos devem pertencer a $CoDiag(G)$ e estarem associados aos diagnosticadores locais. Essa busca por ciclos é vista no algoritmo 2.3, que apresenta os passos necessários para a verificação da codiagnosticabilidade utilizando o autômato de teste $CoDiag(G)$ apresentado em (2.12).

Algoritmo 2.3 *Seja* $G = (X, \Sigma, f, x_0)$ *o autômato que modela um SED e considere m diagnosticadores locais* $Diag_i(G)$, *sendo que cada um é associado a um conjunto de eventos observáveis* Σ_{o_i}. *Além disso, considere que L é diagnosticável com relação a* Σ_f *e* $P_o : \Sigma^\star \to \Sigma_o^\star$, *sendo* $\Sigma_o = \bigcup_{i=1}^{m} \Sigma_{o_i}$.

- *Passo 1: Construa o autômato A_{label} de acordo com o passo 1 do algoritmo 2.2.*

- *Passo 2: Construa os m diagnosticadores locais $Diag_i(G) = Obs(G \| A_{label}, \Sigma_{o_i})$, para $i = 1, \ldots, m$, considerando Σ_{o_i} como o conjunto de eventos observáveis.*

- *Passo 3: Construa o autômato codiagnosticador $CoDiag(G) = (X_{CoDiag}, \Sigma_o, f_{CoDiag}, x_{0,CoDiag})$, realizando as composições paralelas $CoDiag(G) = [\|_{i=1}^{m} Diag_i(G)] \| Diag(G)$.*

- *Passo 4: Verifique se existe em $CoDiag(G)$ um ciclo $cl = (x^k_{CoDiag}, \sigma_k, x^{k+1}_{CoDiag}, \ldots, x^l_{CoDiag}, \sigma_l, x^k_{CoDiag})$, $l \geq k \geq 0$, com $x^i_{CoDiag} \in X_{CoDiag}$, tal que cl possa ser associado a um ciclo indeterminado em cada diagnosticador local $Diag_i(G)$ e o último componente de pelo menos um estado x^i_{CoDiag} (a componente referente ao autômato $Diag(G)$) seja positivo. Se cl existe, então L é não codiagnosticável com relação a P_{o_i} e Σ_f. Caso contrário, L é codiagnosticável.* □

O algoritmo 2.3 mostra como verificar a codiagnosticabilidade de um SED baseado na existência de ciclos indeterminados nos diagnosticadores locais. Para tanto, deve-se primeiro encontrar um ciclo $cl = (x^k_{CoDiag}, \sigma_k, x^{k+1}_{CoDiag}, \ldots, x^l_{CoDiag}, \sigma_l, x^k_{CoDiag})$, $l \geq k \geq 0$, com $x^i_{CoDiag} \in X_{CoDiag}$, tal que, para algum dos estados pertencentes a cl, a coordenada referente ao autômato G (a última coordenada de cada estado do conjunto X_{CoDiag}) é positiva. Isso garante que o sistema executou uma sequência $st \in L \setminus L_N$ arbitrariamente longa após a ocorrência de um evento de falha. Em seguida, deve-se verificar se é possível associar cl a um ciclo indeterminado em cada um dos diagnosticadores locais. Se for possível, então a falha ocorreu para determinada sequência observada de eventos pertencente ao ciclo de $CoDiag(G)$ e não foi detectada por nenhum diagnosticador local, ou seja, existem sequências $s_i \in L_N$, com $i = 1, \ldots, m$, tais que, para todo i, $P_{o_i}(st) = P_{o_i}(s_i)$, o que, de acordo com a definição 2.17, caracteriza uma linguagem ser não codiagnosticável. De forma similar ao que ocorre ao utilizar o diagnosticador para verificar a diagnosticabilidade centralizada de um SED, para verificar a codiagnosticabilidade também é necessário considerar as hipóteses $H1$ e $H2$, garantindo

que a não existência de um ciclo *cl* com as características apresentadas no passo 4 do algoritmo 2.3 seja um condição necessária e suficiente para a verificação da codiagnosticabilidade. Esse fato é formalizado no teorema a seguir.

Teorema 2.2 *Suponha que as hipóteses $H1$ e $H2$ sejam válidas. Então, uma linguagem L gerada por um autômato G é codiagnosticável com relação à projeção $P_{o_i} : \Sigma^\star \rightarrow \Sigma_{o_i}^\star$, sendo $\Sigma_o = \bigcup_{i=1}^{m} \Sigma_{o_i}$, e ao conjunto de eventos de falha Σ_f, se, e somente se, em $CoDiag(G)$ não possui nenhum ciclo $cl = (x_{CoDiag}^k, \sigma_k, x_{CoDiag}^{k+1}, \ldots, x_{CoDiag}^l, \sigma_l, x_{CoDiag}^k)$, $l \geq k \geq 0$, com $x_{CoDiag}^i \in X_{CoDiag}$, tal que cl possa ser associado a um ciclo indeterminado em cada diagnosticador local $Diag_i(G)$ e a última componente de pelo menos um estado x_{CoDiag}^i seja positiva.* □

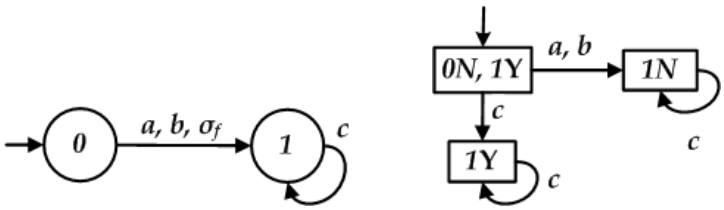
Demonstração Ver [12]. □

A seguir são apresentados dois exemplos que ilustram a obtenção dos diagnosticadores propostos por SAMPATH *et al.* [10], e como utilizá-los para testar as propriedades de diagnosticabilidade de um SED. No exemplo é obtido o diagnosticador para o caso centralizado, enquanto que no exemplo é obtido o codiagnosticador utilizado para o caso descentralizado.

Exemplo 2.9 *Considere o autômato G da figura 2.9(a) e a partição do seu conjunto de eventos $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, sendo $\Sigma_o = \{a, b, c\}$ e $\Sigma_{uo} = \{\sigma_f\}$. Seja $\Sigma_f = \{\sigma_f\}$ o conjunto de eventos de falha. Seguindo o algoritmo 2.2, o primeiro passo para a verificação da diagnosticabilidade de L com relação a Σ_f e P_o é criar o autômato A_{label} mostrado na figura 2.8. O segundo passo do algoritmo 2.3 é calcular o diagnosticador $Diag(G) = Obs(G \| A_{label}, \Sigma_o)$, resultando no autômato mostrado na figura 2.9(b). Por fim, no passo 3 é feito o teste de diagnosticabilidade. Para tanto, é realizada uma busca por ciclos indeterminados no autômato $Diag(G)$. Como nesse autômato não existe nenhum ciclo indeterminado, então a linguagem gerada por G é diagnosticável em relação a P_o e Σ_f.* □

Exemplo 2.10 *Considere novamente o autômato G da figura 2.9(a), e suponha que se deseja verificar a codiagnosticabilidade de L com relação a Σ_f e P_o com dois diagnosticadores locais, cujos os conjuntos de eventos observáveis são $\Sigma_{o_1} = \{a, c\}$ e*

$\Sigma_{o_2} = \{b, c\}$. *De acordo com o algoritmo 2.3, o passo* 1 *é criar o autômato* A_{label} *(ver figura 2.8). O segundo passo é a criação dos dois diagnosticadores locais* $Diag_i(G)$, *com* $i = 1, 2$. *Para tanto, deve-se realizar a operação* $Diag_i(G) = Obs(G \| A_{label}, \Sigma_{o_i})$. *Os diagnosticadores locais* $Diag_1(G)$ *e* $Diag_2(G)$ *podem ser vistos nas figuras 2.10(a) e 2.10(b), respectivamente. O terceiro passo é a criação do autômato de teste para a codiagnosticabilidade* $CoDiag(G) = [\|_{i=1}^{2} Diag_i(G)] \| Diag(G)$, *mostrado na figura 2.10(c). O autômato* $CoDiag(G)$ *é utilizado no passo* 4 *para o teste da codiagnosticabilidade, procurando nesse autômato um ciclo que possa ser associado a um ciclo indeterminado em cada diagnosticador local, sendo que, pelo menos um estado desse ciclo em* $CoDiag(G)$ *deve possuir o último componente positivo. Note que existe um ciclo com essas características no autômato da figura 2.10(c). Esse ciclo é* $cl = (\{\{1Y, 1N\}, \{1Y, 1N\}, \{1Y\}\}, c, \{\{1Y, 1N\}, \{1Y, 1N\}, \{1Y\}\})$, *que está associado aos ciclos indeterminados* $cl_1 = (\{1Y, 1N\}, c, , \{1Y, 1N\})$ *e* $cl_2 = (\{1Y, 1N\}, c, , \{1Y, 1N\})$ *nos diagnosticadores locais* $Diag_1(G)$ *e* $Diag_2(G)$, *respectivamente. Assim, a linguagem gerada por* G *é não codiagnosticável em relação a* P_{o_i} *e* Σ_f. □



(a) (b)

Figura 2.9: (a) Autômato G, (b) diagnosticador de G, $Diag(G)$.

Uma das desvantagens do método proposto por SAMPATH *et al.* [10] é o possível crescimento exponencial do espaço de estados do diagnosticador, que pode ocorrer durante a criação desse diagnosticador *off-line* devido à utilização de observadores. Para suprir a necessidade de obter o diagnosticador *off-line* para verificar a propriedade de diagnosticabilidade de um SED, pode-se utilizar verificadores. O uso de verificadores é interessante devido à complexidade computacional polinomial que eles apresentam. O conceito de complexidade computacional é apresentado na próxima seção.

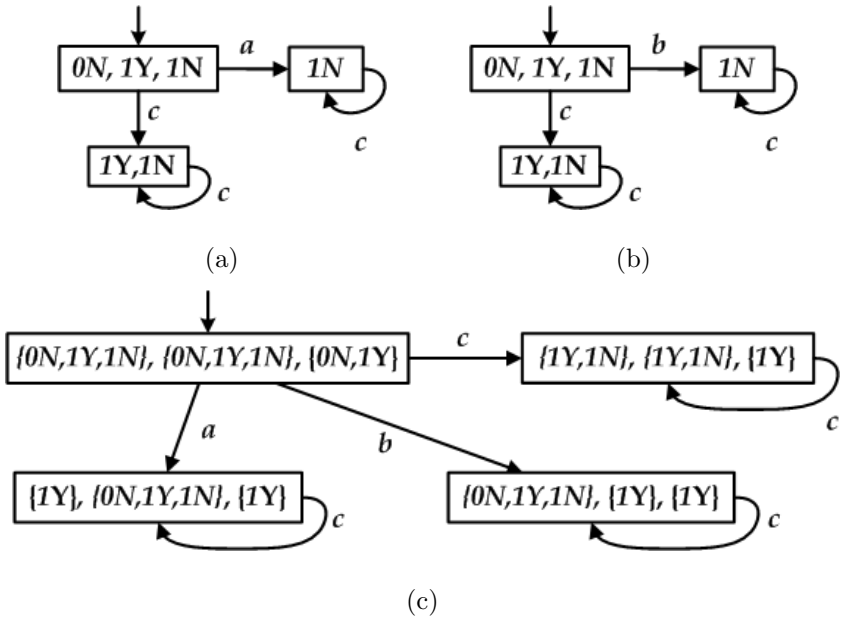


Figura 2.10: (a) Diagnosticador local 1, $Diag_1(G)$, (b) diagnosticador local 2, $Diag_2(G)$, e (c) codiagnosticador, $CoDiag(G)$.

2.8 Complexidade computacional de algoritmos

Algoritmos são comparados tomando por base sua eficiência computacional, que é determinada pela *ordem de crescimento* de um algoritmo, também chamada de *complexidade computacional*. Essa complexidade é uma abstração que concede uma estimativa de quanto tempo um algoritmo levará processando uma entrada qualquer de tamanho n. O tempo de processamento é calculado por uma função matemática que caracteriza a ordem de crescimento do algoritmo [31].

O fato de que a eficiência de um algoritmo está relacionada a uma função matemática permite afirmar que, para um valor de n elevado, as constantes multiplicativas e os termos de mais baixa ordem dessa função são dominados pelos efeitos do próprio tamanho da entrada, tornando relevante para a análise apenas os termos de maior ordem [31]. Essa é a idéia da análise de eficiência *assintótica* de um algoritmo. A seguir é apresentada a notação assintótica usada neste trabalho para avaliar a complexidade dos algoritmos apresentados.

Definição 2.18 (Notação O) *Para uma dada função $g(n)$, denotamos por*

$O(g(n))$ (lê-se "O de g de n") o conjunto de funções

$$O(g(n)) = \{f(n) : \textit{existem constantes positivas } c \in \mathbb{R} \textit{ e } n_0 \in \mathbb{N} \textit{ tais que } 0 \leq f(n) \leq cg(n) \textit{ para todo } n \geq n_0\}.$$

□

A notação assintótica O é usada para dar um limitante superior a uma função. Quando dizemos que uma função $f(n)$ (ou um algoritmo que cresce à taxa de $f(n)$) é $O(g(n))$, estamos dizendo que, no pior caso e para valores de n suficientemente elevados, $f(n)$ *cresce tanto quanto* $g(n)$. Além disso, é comum referenciar a ordem de crescimento de um algoritmo apenas pela categoria da função $g(n)$. Por exemplo, se um algoritmo possuir complexidade $O(g(n))$ e $g(n)$ for exponencial, diz-se que esse algoritmo tem complexidade exponencial. Da mesma forma, se $g(n)$ for um polinômio, diz-se que esse algoritmo tem complexidade polinomial.

No contexto de diagnose de falhas em SED, os algoritmos para determinar se uma linguagem é diagnosticável ou codiagnosticável baseiam-se na construção de autômatos e na verificação da existência de ciclos nesses autômatos. De acordo com CORMEN *et al.* [31], a busca por ciclos em um autômato é linear no número de estados e transições, ou seja, $O(|X| \times |\Sigma|)$. Isso porque um autômato é um grafo direcionado, e a busca por componentes fortemente conectados em um grafo direcionado (que é o mesmo que ciclos em um autômato) tem complexidade $O(V + E)$, sendo V o número de vértices e E o números de arestas em um grafo. Em autômatos, os vértices são os estados e as arestas são as transições. Como no pior caso um autômato (determinístico) possui $|X| \times |\Sigma|$ transições, temos que $V + E$ corresponde a $|X| + |X| \times |\Sigma|$. Porém, como $|X| \times |\Sigma|$ é dominante sobre $|X|$, temos que $O(V + E)$ é equivalente a $O(|X| \times |\Sigma|)$. Assim, a eficiência de um algoritmo de verificação das propriedades de diagnosticabilidade de um SED é determinada pelo número de estados e transições dos autômatos verificadores ou diagnosticadores nos quais são realizadas as buscas por ciclos.

2.9 Conclusão

Neste capítulo foram apresentadas as principais definições acerca do estudo de sistemas a eventos discretos, bem como as ferramentas necessárias para o entendi-

mento deste trabalho, enfatizando principalmente a representação de linguagens por autômatos finitos e a manipulação dessas linguagens. Esses conceitos foram utilizados para definir o problema da diagnose de falhas em SED, que pode ser tratado utilizando uma arquitetura centralizada ou descentralizada. Foi mostrado ainda que verificar a diagnosticabilidade de um SED pode ser uma tarefa de complexidade exponencial. Muitos algoritmos propostos na literatura para a verificação da diagnosticabilidade de um SED são de complexidade polinomial. No próximo capítulo é mostrado como realizar essa verificação em tempo polinomial por meio de autômatos verificadores. Também é apresentada uma breve discussão sobre a complexidade computacional dos algoritmos abordados.

Capítulo 3

Verificação em tempo polinomial da diagnosticabilidade de SED

No capítulo 2 o problema de diagnose de falhas em um sistema a eventos discretos foi formulado e foi apresentada uma forma de resolver esse problema utilizando-se diagnosticadores [10]. Porém a verificação da diagnosticabilidade por meio de diagnosticadores pode possuir complexidade exponencial, uma vez que requer a busca por ciclos com determinadas características no autômato diagnosticador que, no pior caso, apresenta um crescimento exponencial na cardinalidade do espaço de estados do sistema. Para contornar esse problema, autômatos verificadores que crescem polinomialmente com a cardinalidade do espaço de estados do sistema podem ser utilizados para a verificação da diagnosticabilidade tanto para o caso centralizado [13–15, 24, 25] quanto para o caso descentralizado [15, 16, 24, 25].

Neste capítulo quatro métodos para a verificação em tempo polinomial da diagnosticabilidade de SED propostos na literatura são revistos [13–16]. Os métodos propostos por JIANG *et al.* [13] e YOO e LAFORTUNE [14] são capazes de realizar apenas a verificação da diagnosticabilidade utilizando a arquitetura centralizada de um SED, enquanto que o método proposto por WANG *et al.* [16] é utilizado apenas para a verificação da codiagnosticabilidade de um SED. O método proposto por QIU e KUMAR [15], pode ser utilizado para verificar a diagnosticabilidade nos casos centralizado e descentralizado.

Neste trabalho um novo método de verificação [24] que, assim como o método proposto por QIU e KUMAR [15], pode ser utilizado tanto para a verificação da

diagnosticabilidade no caso centralizado quanto para a verificação da codiagnosticabilidade de um SED, é apresentado. Esse novo método possui complexidade computacional menor do que os demais métodos encontrados na literatura. A análise da complexidade computacional do novo método é apresentada no final deste capítulo.

3.1 Verificador proposto por Jiang *et al.* [13]

O método apresentado em [13] propõe a criação de um autômato verificador não determinístico que mapeia os pares de sequências pertencentes a L que possuem a mesma projeção $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$. Para tanto, deve ser criado um autômato intermediário cuja linguagem gerada seja $P_o(L)$ e que possua em cada estado a informação de ocorrência de falha em seu rótulo. Por fim, deve ser feito um produto desse autômato intermediário com ele mesmo, gerando assim o autômato verificador. Em [13] são supostas as hipóteses de vivacidade ($H1$) e não existência de ciclos de eventos não observáveis ($H2$). O algoritmo 3.1 descreve detalhadamente esse método.

Algoritmo 3.1 *Seja $G = (X, \Sigma, f, x_0)$ o autômato que modela um SED e considere a partição do conjunto de eventos $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, e a projeção $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$. Seja $\mathcal{F} = \{F_i, i = 1, \ldots, p\}$ o conjunto de tipos de falha e $\psi : \Sigma \rightarrow \mathcal{F} \cup \{\emptyset\}$ a função que associa a cada evento pertencente a Σ um tipo de falha ou o conjunto vazio.*

- *Passo 1: Crie o autômato não determinístico $V_{J_o} = (X_{J_o}, \Sigma_o, f_{J_o}, x_{0,J_o})$ da seguinte forma:*
  - *$X_{J_o} = \{(x, \Lambda) : x \in X_1 \cup \{x_0\}, \Lambda \subseteq \mathcal{F}\}$ é um conjunto finito de estados, sendo $X_1 = \{x \in X : (\exists x' \in X)(\exists \sigma \in \Sigma_o)[f(x', \sigma) = x]\}$ o conjunto de estados em G que podem ser alcançados por meio de um evento observável, e Λ é o conjunto formado pelos tipos de falha associados aos eventos que ocorreram em um determinado caminho de x_0 até x* [1]*. Caso*

[1] Note que, se existirem dois caminhos distintos de x_0 até x, tais que os conjuntos de tipos de falha associados a esses caminhos também sejam distintos, então existirá dois estados pertencentes a X_1, ambos associados ao estado x, porém com informações distintas de ocorrência de tipos de falha.

nenhuma falha tenha ocorrido em determinado caminho de x_0 *até* x, $\Lambda = \emptyset$.

- Σ_o *é o conjunto de eventos observáveis.*
- $f_{J_o} : X_{J_o} \times \Sigma_o \rightarrow X_{J_o}$ *é a função de transição de estados tal que* $f_{J_o}((x, \Lambda), \sigma) = (x', \Lambda')$ *se, e somente se, existe um caminho* $(x, \sigma_1, x_1, \ldots, \sigma_n, x_n, \sigma, x')$ *(*$n \geq 1$*) em* G *tal que* $P_o(\sigma_i) = \epsilon$, $\forall i \in \{1, 2, \ldots, n\}$, $P_o(\sigma) = \sigma$ *e* $\Lambda' = \{\psi(\sigma_i) : \psi(\sigma_i) \neq \emptyset, 1 \leq i \leq n\} \cup \Lambda$.
- $x_{0,J_o} = (x_0, \emptyset) \in X_{J_o}$ *é o estado inicial.*

- *Passo 2: Calcule o autômato* $V_J = V_{J_o} \times V_{J_o} = (X_J, \Sigma_o, f_J, x_{0,J})$.
- *Passo 3: Verifique se existe em* V_J *um ciclo* $cl = (x_J^k, \sigma_k, x_J^{k+1}, \ldots, x_J^l, \sigma_l, x_J^k)$, $l \geq k \geq 0$, *tal que para todo* $x_J^i = ((x_i^1, \Lambda_i^1), (x_i^2, \Lambda_i^2))$, $i = k, k+1, \ldots, l$, *tem-se que* $\Lambda_i^1 \neq \Lambda_i^2$. *Se cl existe, então o sistema é não diagnosticável em relação a* P_o *e* Σ_f. *Caso contrário, o sistema é diagnosticável.* □

De acordo com o algoritmo 3.1, a linguagem gerada por um autômato G é diagnosticável se, e somente se, para todo ciclo existente no autômato V_J, a informação de ocorrência de falha em cada estado pertencente ao ciclo é a mesma para todos os estados do ciclo. Isso garante que não existe em L uma sequência st de comprimento arbitrariamente longo após a ocorrência de um evento de falha de um determinado tipo, e uma sequência s' que não possua um evento de falha desse mesmo tipo, tal que $P_o(s') = P_o(st)$. Desta forma, de acordo com a definição 2.16, pode-se afirmar que L é diagnosticável em relação a P_o e Σ_f.

Assim como no método proposto por SAMPATH *et al.* [10], é importante notar a necessidade das hipóteses $H1$ e $H2$. Como, pela definição de V_{J_o}, $\mathcal{L}(V_{J_o}) = P_o(L)$, e pela definição de V_J, $\mathcal{L}(V_J) = \mathcal{L}(V_{J_o}) = P_o(L)$, então, ao ser desconsiderada a hipótese $H1$, poderia existir uma sequência $u \in L$ de comprimento limitado que leva a um estado de bloqueio em G, tal que $\sigma_f \in u$, e uma sequência $s' \in L_N$, tais que $P_o(u) = P_o(s')$, o que, de acordo com a definição 2.16, caracteriza L como uma linguagem não diagnosticável em relação a P_o e Σ_f. Porém, uma vez que $\mathcal{L}(V_J) = \mathcal{L}(V_{J_o}) = P_o(L)$, então a sequência $P_o(u) = P_o(s')$ está associada a um caminho em V_J, entretanto, esse caminho não forma um ciclo, uma vez que u é

uma sequência de comprimento limitado e, consequentemente, a sequência $P_o(u)$ não possui comprimento arbitrariamente longo. Desta forma, se a hipótese $H1$ for falsa, a não existência de um ciclo com as mesmas características que o ciclo cl definido no passo 3 do algoritmo 3.1 não é uma condição necessária e suficiente para a diagnosticabilidade do SED. No caso de ser desconsiderada a hipótese $H2$, também tem-se que a não existência de um ciclo com as mesmas características que o ciclo cl definido no passo 3 do algoritmo 3.1 passa a não ser condição necessária e suficiente para a diagnosticabilidade. Nesse caso, pode existir uma sequência st, sendo s uma sequência de comprimento limitado, $\sigma_f \in s$ e $t \in \Sigma_{uo}^{\star}$ uma sequência arbitrariamente longa, e pode existir uma sequência $s' \in L_N$, tais que $P_o(st) = P_o(s')$, o que, de acordo com a definição 2.16, caracteriza L como uma linguagem não diagnosticável em relação a P_o e Σ_f. Entretanto, como t é uma sequência formada apenas por eventos não observáveis, então $P_o(st) = P_o(s)$ e $P_o(st)$ possui comprimento limitado. Assim, como a sequência $P_o(st) \in \mathcal{L}(V_J)$, ela está associada a um caminho em V_J, porém esse caminho não forma um ciclo, fazendo com que a não existência de um ciclo com as mesmas características que o ciclo cl definido no passo 3 do algoritmo 3.1 não seja uma condição necessária e suficiente para a diagnosticabilidade do SED.

Teorema 3.1 *Suponha que as hipóteses $H1$ e $H2$ sejam válidas. Então, uma linguagem L gerada por um autômato G é diagnosticável com relação à projeção $P_o : \Sigma^{\star} \rightarrow \Sigma_o^{\star}$ e ao conjunto de tipos de falha $\mathcal{F} = \{F_i, i = 1, \ldots, p\}$, se, e somente se, para todo ciclo $cl = (x_J^k, \sigma_k, x_J^{k+1}, \ldots, x_J^l, \sigma_l, x_J^k)$, $l \geq k \geq 0$, em V_J, tem-se que, para todo $x_J^i = ((x_i^1, \Lambda_i^1), (x_i^2, \Lambda_i^2))$, $i = k, k+1, \ldots, l$, $\Lambda_i^1 = \Lambda_i^2$.* □

Demonstração Ver [13]. □

Em [13] é demonstrado que a complexidade do algoritmo 3.1 é $O(|X|^4 \times 2^{4|\mathcal{F}|} \times |\Sigma_o|)$, que é polinomial no número de estados e eventos observáveis de G e é exponencial no número de tipos de falha do sistema. Para contornar o problema do crescimento exponencial com o número de tipos de falha do sistema, pode-se criar um verificador para cada tipo de falha e fazer o teste de diagnosticabilidade de um tipo de falha por vez [13–15, 24]. Isso também se aplica para o caso de realizar a verificação da codiagnosticabilidade [15, 16, 24]. Desta forma, a complexidade do método de JIANG *et al.* [13] passa a ser $O(|X|^4 \times |\Sigma_o| \times |\mathcal{F}|)$, ou seja, polinomial

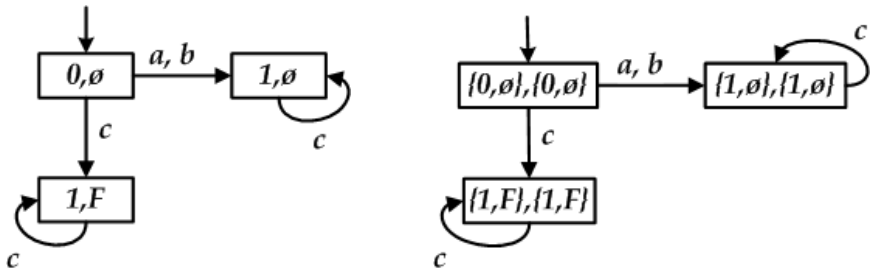
tanto no número de estados e eventos observáveis de G, como no número de tipos de falha. Os métodos propostos por YOO e LAFORTUNE [14], QIU e KUMAR [15], WANG *et al.* [16] e o novo método para a verificação da codiagnosticabilidade [24] apresentado a seguir neste trabalho, também realizam a verificação de cada tipo de falha por vez, considerando os eventos de falha de outros tipos sendo eventos não observáveis comuns. Assim, neste trabalho, sem perda de generalidade, é considerado que existe apenas um tipo de falha, isto é, $l = 1$. Isso implica que a complexidade do método de JIANG *et al.* [13], para fins de comparações com os demais métodos apresentados neste capítulo, passa a ser $O(|X|^4 \times |\Sigma_o|)$.

A seguir é apresentado um exemplo para ilustrar o método descrito no algoritmo 3.1.

Exemplo 3.1 *Considere o autômato G da figura 2.9(a), assim como sua partição do conjunto de eventos $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, com $\Sigma_o = \{a, b, c\}$, $\Sigma_{uo} = \{\sigma_f\}$ e $\Sigma_f = \{\sigma_f\}$. Assim, há apenas um tipo de falha denotada aqui por F, ou seja, $\mathcal{F} = \{F\}$.*

De acordo com o algoritmo 3.1, o primeiro passo é a criação do autômato V_{J_o} (apresentado na figura 3.1(a)). Esse autômato é criado eliminando-se as transições não observáveis de G e adicionando-se a cada estado a informação de ocorrência de falha durante a evolução do autômato. Note que a informação de falha contida em um estado $x_{J_o} \in X_{J_o}$ consiste no conjunto de tipos de falha que ocorreram desde o estado inicial x_0 até o estado de G associado a x_{J_o}. No passo 2 é criado o autômato $V_J = V_{J_o} \times V_{J_o}$. Esse autômato é mostrado na figura 3.1(b). Por fim, no passo 3 é feito o teste de diagnosticabilidade. Esse teste baseia-se em verificar se cada ciclo cl pertencente a V_J possui todos os estados $x_J = ((x^1, \Lambda^1), (x^2, \Lambda^2))$ que o compõe com a informação $\Lambda^1 = \Lambda^2$, garantido assim que a linguagem gerada pelo sistema é diagnosticável. Observando o autômato V_J da figura 3.1(b), nota-se que existem apenas dois ciclos: $cl_1 = (\{\{1, \emptyset\}, \{1, \emptyset\}\}, c, \{\{1, \emptyset\}, \{1, \emptyset\}\})$ e $cl_2 = (\{\{1, F\}, \{1, F\}\}, c, \{\{1, F\}, \{1, F\}\})$, sendo que para todo estado de cl_1, $\Lambda^1 = \Lambda^2 = \emptyset$ e, para todo estado de cl_2, $\Lambda^1 = \Lambda^2 = F$.

Assim, de acordo com o teorema 3.1, a linguagem gerada por G é diagnosticável em relação a P_o e Σ_f. □



(a) (b)

Figura 3.1: (a) Autômato V_{J_o}, (b) autômato V_J.

3.2 Verificador proposto por Yoo e Lafortune [14]

Assim como o método de JIANG *et al.* [13], o método proposto por YOO e LAFORTUNE [14] também baseia-se na construção de um autômato verificador não determinístico. Entretanto, o método de YOO e LAFORTUNE [14] apresenta uma complexidade computacional de ordem inferior em relação ao número de estados do sistema do que o método de JIANG *et al.* [13].

Os passos necessários para a construção do autômato verificador proposto por YOO e LAFORTUNE [14] é apresentado no algoritmo 3.2. Vale ressaltar que em [14] também são supostas as hipóteses de vivacidade ($H1$) e não existência de ciclos de eventos não observáveis ($H2$).

Algoritmo 3.2 *Seja $G = (X, \Sigma, f, x_0)$ o autômato que modela um SED e considere a partição $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. Considere também o conjunto de eventos de falha $\Sigma_f \subseteq \Sigma_{uo}$.*

- *Passo 1: Construa o autômato verificador V_Y, como segue:*

$$V_Y = CoAc\left[Ac(X_Y, \Sigma, f_Y, x_{0,Y})\right], \tag{3.1}$$

sendo

$$X_Y = X \times L \times X \times L,$$

$$x_{0,Y} = (x_0, N, x_0, N),$$

com o conjunto de rótulos para informação de ocorrência de falha dado por $L = \{N, F\}$, e as seguintes regras para a função de transição f_Y:

Para $\sigma \in \Sigma_f$

$$f_Y((x^1, l^1, x^2, l^2), \sigma) = \begin{cases} (f(x^1, \sigma), F, x^2, l^2) \\ (x^1, l^1, f(x^2, \sigma), F) \\ (f(x^1, \sigma), F, f(x^2, \sigma), F) \end{cases} . \tag{3.2}$$

Para $\sigma \in \Sigma_{uo} \setminus \Sigma_f$

$$f_Y((x^1, l^1, x^2, l^2), \sigma) = \begin{cases} (f(x^1, \sigma), l^1, x^2, l^2) \\ (x^1, l^1, f(x^2, \sigma), l^2) \\ (f(x^1, \sigma), l^1, f(x^2, \sigma), l^2) \end{cases} . \tag{3.3}$$

Para $\sigma \in \Sigma_o$

$$f_Y((x^1, l^1, x^2, l^2), \sigma) = \left(f(x^1, \sigma), l^1, f(x^2, \sigma), l^2\right). \tag{3.4}$$

- *Passo 2: Verifique se existe em* V_Y *um ciclo* $cl = (x_Y^k, \sigma_k, x_Y^{k+1}, \ldots, x_Y^l, \sigma_l, x_Y^k)$, $l \geq k \geq 0$, *tal que para todo* $x_Y^j \in X_Y$, $x_Y^j = (x_j^1, l_j^1, x_j^2, l_j^2)$, $j = k, k+1, \ldots, l$, $l_j^1 = F$ *e* $l_j^2 = N$ *ou vice-versa. Se cl existe, então o sistema é não diagnosticável em relação a* P_o *e* Σ_f. *Caso contrário, o sistema é diagnosticável.*

□

De acordo com o algoritmo 3.2, a linguagem gerada por um autômato G é diagnosticável se, e somente se, todo estado pertencente a um ciclo no autômato V_Y, possui a informação de ocorrência de falha igual aos demais estados do ciclo. Isso garante que não existe uma sequência $st \in L$, $\sigma_f \in s$, e t sendo uma sequência de comprimento arbitrariamente longo, e uma sequência sem falha s', tais que $P_o(s') = P_o(st)$, uma vez que não existe nenhum ciclo no qual cada estado indica tanto que uma falha ocorreu, quanto indica que essa falha não ocorreu. Desta forma, de acordo com a definição 2.16, pode-se afirmar que L é diagnosticável em relação a P_o e Σ_f.

Assim como nos métodos propostos por SAMPATH *et al.* [10] e por JIANG *et al.* [13], é importante notar a necessidade das hipóteses $H1$ e $H2$. Suponha que a hipótese $H2$ não seja verdadeira. Nesse caso, a existência de um ciclo com as mesmas características de um ciclo cl descrito no passo 2 do algoritmo 3.2 deixa de ser condição necessária e suficiente para a não diagnosticabilidade. Isso pode

ser constatado pela construção de V_Y, que garante, sem perda de generalidade, que para cada estado $x_Y^j = (x_j^1, F, x_j^2, N)$ em um ciclo $cl = (x_Y^k, \sigma_k, x_Y^{k+1}, \ldots, x_Y^l, \sigma_l, x_Y^k)$ no verificador V_Y, existe uma sequência $s_j t_j$ contendo pelo menos um evento de falha, e a uma sequência $s'_j t'_j \in L_N$ tais que $P_o(s_j t_j) = P_o(s'_j t'_j)$, $f(x_0, s_j t_j) = x_j^1$ e $f(x_0, s'_j t'_j) = x_j^2$. Entretanto, como pode haver ciclos de eventos não observáveis em G, e a função de transição $f_Y((x_j^1, F, x_j^2, N), \sigma)$ é definida para $\sigma \in \Sigma_{uo} \setminus \Sigma_f$ quando $f(x_j^2, \sigma)$ é definida, então pode ocorrer que $s_j t_j$ seja de comprimento limitado e que $s'_j t'_j$ seja arbitrariamente longa, nesse caso, com $t'_j \in \Sigma_{uo}^\star$, o que, de acordo coma definição 2.16, não caracteriza uma linguagem não diagnosticável em relação a P_o e Σ_f, o que era esperado devido à existência de cl. Suponha agora que a hipótese $H1$ não seja verdadeira. Nesse caso é possível existir uma sequência $st \in L$ tal que $\sigma_f \in s$ e st leva a um estado de bloqueio, e uma sequência $s' \in L$, tais que $P_o(st) = P_o(s')$, e que não estão associadas a um caminho de comprimento arbitrariamente longo em V_Y, uma vez que st é limitada. Portanto, não existe um ciclo com as características apresentadas no passo 2 do algoritmo 3.2 que viola as condições de diagnosticabilidade de L em relação a P_o e Σ_f.

O teorema a seguir formaliza a condição necessária e suficiente para a diagnosticabilidade utilizando o método proposto por YOO e LAFORTUNE [14].

Teorema 3.2 *Suponha que as hipóteses $H1$ e $H2$ sejam válidas. Então, uma linguagem L gerada por um autômato G é diagnosticável com relação à projeção $P_o : \Sigma^\star \to \Sigma_o^\star$ e ao conjunto de eventos de falha Σ_f, se, e somente se, para todo ciclo $cl = (x_Y^k, \sigma_k, x_Y^{k+1}, \ldots, x_Y^l, \sigma_l, x_Y^k)$, $l \geq k \geq 0$, em V_Y, tem-se que, para todo $x_Y^j = (x_j^1, l_j^1, x_j^2, l_j^2)$, $j = k, k+1, \ldots, l$, $l_j^1 = l_j^2 = F$ ou $l_j^1 = l_j^2 = N$.* □

Demonstração Ver [14]. □

YOO e LAFORTUNE [14] demonstram que a complexidade do algoritmo 3.2 é $O(|X|^2 \times |\Sigma|)$, sendo, portanto, polinomial no número de estados e eventos do sistema G. O exemplo a seguir ilustra o método proposto por YOO e LAFORTUNE [14].

Exemplo 3.2 *Considere novamente o autômato G da figura 2.9(a), juntamente com sua partição do conjunto de eventos $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, com $\Sigma_o = \{a, b, c\}$, $\Sigma_{uo} = \{\sigma_f\}$ e $\Sigma_f = \{\sigma_f\}$.*

Para realizar o teste de diagnosticabilidade de um SED a partir do método proposto em [14], deve-se executar o algoritmo 3.2. No passo 1 *desse algoritmo é criado o autômato verificador* V_Y *a partir das regras de transição expressas nas equações (3.2), (3.3) e (3.4). O autômato* V_Y *pode ser visto na figura 3.2. No passo* 2 *o teste de diagnosticabilidade é feito nesse autômato, procurando por algum ciclo cl no qual cada estado* $x_Y = (x^1, l^1, x^2, l^2) \in X^Y$ *pertencente a cl é definido com* $l^1 = F$ *e* $l^2 = N$*, ou vice-versa, indicando assim que o sistema é não diagnosticável. Como no autômato* V_Y *da figura 3.1(b) todos os ciclos existentes (*$cl_1 = (1N1N, c, 1N1N)$ *e* $cl_2 = (1F1F, c, 1F1F)$*) possuem todos estados em cada ciclo com* $l^1 = l^2 = N$ *ou* $l^1 = l^2 = F$*, pode-se afirmar de acordo com o teorema de 3.2 que a linguagem gerada por G é diagnosticável em relação a* P_o *e* Σ_f*.* □

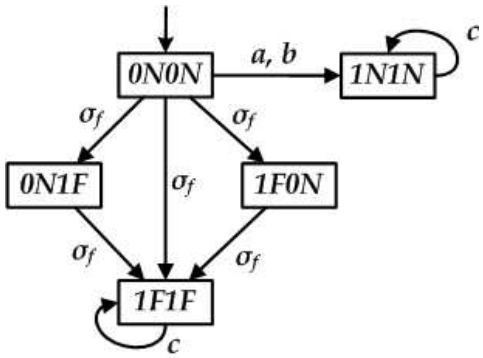


Figura 3.2: Verificador proposto por YOO e LAFORTUNE [14] referente ao exemplo 3.2.

3.3 Verificador proposto por Qiu e Kumar [15]

Outro método para a verificação da diagnosticabilidade de um SED em tempo polinomial é proposto em [15]. Esse método pode ser utilizado para os casos centralizado e descentralizado. Em [15] as hipóteses de vivacidade da linguagem gerada pelo sistema (hipótese $H1$) e de não existência de ciclos de eventos não observáveis (hipótese $H2$) são removidas.

O método proposto por QIU e KUMAR [15] é apresentado no algoritmo 3.3. Esse algoritmo mostra como realizar a verificação da codiagnosticabilidade. Em seguida é apresentada uma forma de adaptar esse algoritmo para a verificação da

diagnosticabilidade no caso centralizado. Sem perda de generalidade, o algoritmo é apresentado considerando a existência de apenas dois diagnosticadores locais, ou seja, $m = 2$.

Algoritmo 3.3 *Seja G o autômato determinístico que modela o sistema, $H = (X_H, \Sigma, f_H, \Gamma_H, x_{0,H})$ o subautômato de G que gera a linguagem normal $L_N \subseteq L$ e Σ_f o conjunto de eventos de falha. Considere ainda as seguintes partições de Σ sendo que cada uma é associada a um diagnosticador local $\Sigma = \Sigma_{o_i} \dot{\cup} \Sigma_{uo_i}$, para $i = 1, 2$, sendo Σ_{o_i} e Σ_{uo_i} os conjuntos de eventos observáveis e não observáveis para cada diagnosticador local, respectivamente.*

- *Passo 1: Obtenha o autômato aumentado $\bar{H} = (X_{\bar{H}}, \Sigma, f_{\bar{H}}, x_{0,\bar{H}})$ como segue:*
  - *Passo 1.1: Defina $x_{0,\bar{H}} = x_{0,H}$.*
  - *Passo 1.2: Adicione um novo estado F ao espaço de estados de H para indicar a ocorrência do evento de falha. Assim, $X_{\bar{H}} = X_H \cup \{F\}$.*
  - *Passo 1.3: Para cada $x_{\bar{H}} \in X_H$ defina:*

$$f_{\bar{H}}(x_{\bar{H}}, \sigma) = \begin{cases} f_H(x_{\bar{H}}, \sigma), \text{ se } \sigma \in \Gamma_H(x_{\bar{H}}) \\ F, \text{ se } \sigma \in \Sigma \setminus \Gamma_H(x_{\bar{H}}) \end{cases},$$

    e para $x_{\bar{H}} = F$ defina:

$$f_{\bar{H}}(x_{\bar{H}}, \sigma) = F, \text{ para todo } \sigma \in \Sigma.$$

- *Passo 2: Construa o autômato de teste da codiagnosticabilidade $V_Q = \left(X_Q, \Sigma^Q, f_Q, x_{0,Q}\right)$ sendo:*
  - $X_Q = X \times X_{\bar{H}} \times X_H \times X_H$
  - $x_{0,Q} = (x_0, x_{0,\bar{H}}, x_{0,H}, x_{0,H})$
  - $\Sigma^Q = \overline{\Sigma}^3$, *com* $\overline{\Sigma} = \Sigma \cup \{\epsilon\}$
  - $f_Q : X_Q \times \Sigma^Q \to X_Q$ *definida como:*

    $\forall x_Q = (x, x_{\bar{H}}, x_H^1, x_H^2) \in X_Q,$
    $\sigma^Q = (\sigma, \sigma^1, \sigma^2) \in \Sigma^Q - \{\epsilon, \epsilon, \epsilon\},$
    $f_Q(x_Q, \sigma^Q) = (f(x, \sigma), f_{\bar{H}}(x_{\bar{H}}, \sigma), f_H(x_H^1, \sigma^1), f_H(x_H^2, \sigma^2)),$
    se, e somente se,
    $[P_{o_1}(\sigma) = P_{o_1}(\sigma^1), P_{o_2}(\sigma) = P_{o_2}(\sigma^2)] \wedge [f(x, \sigma), f_H(x_H^1, \sigma^1), f_H(x_H^2, \sigma^2) \neq \emptyset]$.

- *Passo 3: Verifique se existe em* V_Q *um ciclo* $cl = (x_Q^k, \sigma_k^Q, x_Q^{k+1}, \ldots, x_Q^l, \sigma_l^Q, x_Q^k)$, $l \geq k \geq 0$, *tal que* $\sigma_i^Q = (\sigma_i, \sigma_i^1, \sigma_i^2)$, $x_Q^i = (x_i, x_{\bar{H}_i}, x_{H_i}^1, x_{H_i}^2)$, *em que* $\exists i \in [k, l]$ *tal que* $x_{\bar{H}_i} = F$ *e* $\sigma_i \neq \epsilon$. *Se cl existe, então o sistema é não codiagnosticável em relação a* P_{o_i} *e* Σ_f. *Caso contrário, o sistema é codiagnosticável.* □

O algoritmo 3.3 propõe a criação do autômato aumentado $\bar{H}$ no passo 1. Esse autômato é utilizado para rotular as ocorrências de falha. Isso é feito através do estado F criado no passo 1.2. Assim, toda sequência de falha pertencente a L executada em $\bar{H}$ leva ao estado F.

No passo 2, o verificador é criado, considerando que cada estado pertencente a X_Q é formado por uma componente associada a G, uma componente associada a $\bar{H}$ e duas componentes associadas a H. Isso é feito para que, para cada sequência st pertencente a L, possa ser verificada a ocorrência de um evento de falha através da evolução de estados em $\bar{H}$, e possa ser verificada a existência de sequências sem falha s_1 e s_2 pertencentes a L, não necessariamente distintas, tais que $P_{o_1}(s_1) = P_{o_1}(st)$ e $P_{o_2}(s_2) = P_{o_2}(st)$, através da evolução de estados em H para cada um dos dois diagnosticadores locais, sendo necessário então duas componentes associadas a H. Esse rastreamento feito observando-se estados pode ser feito observando-se eventos. Assim, é definido também no passo 2 um conjunto de eventos Σ^Q sendo que cada evento $\sigma^Q \in \Sigma^Q$ é da forma $\sigma^Q = (\sigma, \sigma^1, \sigma^2)$, indicando o evento σ, que é executado tanto em G quanto em $\bar{H}$, e os eventos σ^1 e σ^2 que são executados em H, de forma que $P_{o_1}(\sigma) = P_{o_1}(\sigma^1)$, $P_{o_2}(\sigma) = P_{o_2}(\sigma^2)$ e $f(x, \sigma) \neq \emptyset$, $f_H(x_H^1, \sigma^1) \neq \emptyset$, $f_H(x_H^2, \sigma^2) \neq \emptyset$.

Por fim, o passo 3 do algoritmo 3.3 mostra como realizar o teste de codiagnosticabilidade. Nesse caso, a linguagem gerada por um autômato G é não diagnosticável se, e somente se existir no autômato V_Q um ciclo $cl = (x_Q^k, \sigma_k^Q, x_Q^{k+1}, \ldots, x_Q^l, \sigma_l^Q, x_Q^k)$, sendo $l \geq k \geq 0$, $\sigma_i^Q = (\sigma_i, \sigma_i^1, \sigma_i^2)$ e $x_Q^i = (x_i, x_{\bar{H}_i}, x_{H_i}^1, x_{H_i}^2)$, em que $\exists i \in [k, l]$ tal que $x_{\bar{H}_i} = F$ e $\sigma_i \neq \epsilon$. Uma vez que existe pelo menos um estado pertencente a cl que possui a coordenada $x_{\bar{H}} = F$, então cl é um ciclo associado a uma sequência $st \in L$ contendo algum evento de falha. Além disso, como existe em cl pelo menos um evento $\sigma^Q = (\sigma, \sigma^1, \sigma^2)$ tal que $\sigma \neq \epsilon$, então st também está associada a um ciclo em G, ou seja, st é uma sequência arbitrariamente longa após a falha. Como V_Q representa apenas as sequências de L que possuem mesma projeção

P_{o_1} e mesma projeção P_{o_2} que alguma sequência pertencente a L_N, então existem sequências sem falha s_1, s_2, não necessariamente distintas, tais que $P_{o_1}(s_1) = P_{o_1}(st)$ e $P_{o_2}(s_2) = P_{o_2}(st)$. Desta forma, de acordo com a definição 2.17, pode-se afirmar que L é não codiagnosticável em relação a P_{o_i} e Σ_f.

É interessante observar que, diferente dos métodos propostos por SAMPATH *et al.* [10], por JIANG *et al.* [13] e por YOO e LAFORTUNE [14], o método proposto por QIU e KUMAR [15] não supõe que as hipóteses $H1$ e $H2$ são verdadeiras, uma vez que isso não afeta a condição necessária e suficiente para a codiagnosticabilidade apresentada nesse método. A hipótese $H2$ pode ser relaxada pelo fato de que, diferente dos métodos apresentados em [10] e [13], o verificador apresentado em [15] não gera a linguagem $P_o(L)$, não escondendo assim as transições rotuladas por eventos não observáveis (incluindo os ciclos de eventos não observáveis). Por outro lado, diferente do método proposto em [14], o verificador proposto por QIU e KUMAR [15] diferencia os ciclos associados a sequências $st \in L$ e s_1t_1, $s_2t_2 \in L \setminus L_N$, tais que st é arbitrariamente longa após a falha, $P_{o_1}(s_1t_1) = P_{o_1}(st)$ e $P_{o_2}(s_2t_2) = P_{o_2}(st)$, dos ciclos associados a sequências $s't' \in L$, $s'_1t'_1$, $s'_2t'_2 \in L \setminus L_N$, tais que $s't'$ é de comprimento limitado, $t'_1, t'_2 \in \Sigma^{\star}_{uo}$, $P_{o_1}(s'_1t'_1) = P_{o_1}(s't')$ e $P_{o_2}(s'_2t'_2) = P_{o_2}(s't')$, fazendo com que a existência de ciclos de eventos não observáveis em G (ciclos associados a t'_1 e t'_2) não seja um problema para a verificação da codiagnosticabilidade. A hipótese $H1$ é desnecessária porque, se L não for viva, pode-se adicionar autolaços rotulados por eventos não observáveis aos estados de bloqueio, fazendo com que L passe a ser uma linguagem viva.

O teorema a seguir formaliza a condição necessária e suficiente para realizar a verificação da codiagnosticabilidade de um SED utilizando o verificador proposto por QIU e KUMAR [15].

Teorema 3.3 *Uma linguagem L gerada por um autômato G é codiagnosticável com relação às projeções P_{o_i}, com $i = 1, 2, \ldots, m$, e ao conjunto de falhas Σ_f, se, e somente se, não existe em V_Q um ciclo $cl = (x_Q^k, \sigma_k^Q, x_Q^{k+1}, \ldots, x_Q^l, \sigma_l^Q, x_Q^k)$, $l \geq k \geq 0$, tal que $\sigma_i^Q = (\sigma_i, \sigma_i^1, \sigma_i^2)$, $x_Q^i = (x_i, x_{\bar{H}_i}, x_{H_i}^1, x_{H_i}^2)$, em que $\exists i \in [k, l]$ tal que $x_{\bar{H}_i} = F$ e $\sigma_i \neq \epsilon$.* □

Demonstração Ver [15]. □

Observação 3 *O método proposto por QIU e KUMAR [15] apresentado no algoritmo 3.3 realiza a verificação da codiagnosticabilidade de um SED. Entretanto pode-se utilizá-lo para a verificação da diagnosticabilidade no caso centralizado. Para tanto, basta considerar que as observações de eventos são feitas por um único diagnosticador, ou seja,* $m = 1$*. Portanto, um autômato verificador para o caso centralizado é dado por* $V_{Q,c} = \left(X_{Q,c}, \Sigma^{Q,c}, f_{Q,c}, x_{0,Q,c}\right)$*, sendo*

- $X_{Q,c} = X \times X_{\bar{H}} \times X_H$
- $x_{0,Q} = (x_0, x_{0,\bar{H}}, x_{0,H})$
- $\Sigma^{Q,c} = \overline{\Sigma}^2$
- $f_{Q,c} : X_{Q,c} \times \Sigma^{Q,c} \rightarrow X_{Q,c}$ *definida como:*

$$
\begin{aligned}
&\forall x_{Q,c} = (x, x_{\bar{H}}, x_H^1) \in X_{Q,c},\\
&\sigma^{Q,c} = (\sigma, \sigma^1) \in \Sigma^{Q,c} - \{\epsilon, \epsilon\},\\
&f_{Q,c}(x_{Q,c}, \sigma^{Q,c}) = (f(x,\sigma), f_{\bar{H}}(x_{\bar{H}}, \sigma), f_H(x_H^1, \sigma^1)),\\
&\textit{se, e somente se,}\\
&[P_o(\sigma) = P_o(\sigma^1)] \wedge [f(x,\sigma), f_H(x_H^1, \sigma^1) \neq \emptyset]\,,
\end{aligned}
$$

e a condição necessária e suficiente para a não diagnosticabilidade de L *é a existência de um ciclo de estados de falha em* $V_{Q,c}$ *tal que, para pelo menos um evento* $\sigma^{Q,c} = (\sigma, \sigma^1)$ *no ciclo, tem-se* $\sigma \neq \epsilon$*.* □

O algoritmo 3.3 tem complexidade computacional $O(|X| \times |X_H|^{m+1} \times |\Sigma|^{m+1})$, ou seja, esse algoritmo é de ordem $(m+1)$ no número de estados e eventos, e no caso centralizado $(m = 1)$ o algoritmo apresenta complexidade de segunda ordem no número de estados e eventos do sistema.

Os exemplos 3.3 e 3.4 ilustram a utilização do algoritmo 3.3 para o caso descentralizado e para o caso centralizado, respectivamente.

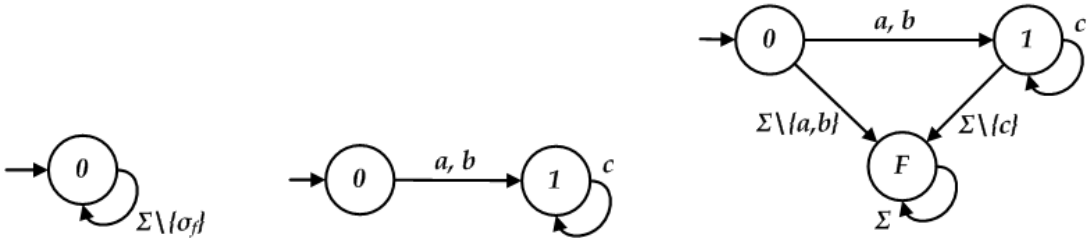
Exemplo 3.3 *Considere novamente o autômato* G *da figura 2.9(a), juntamente com sua partição do conjunto de eventos* $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$*, com* $\Sigma_o = \{a, b, c\}$*,* $\Sigma_{uo} = \{\sigma_f\}$ *e* $\Sigma_f = \{\sigma_f\}$*, e suponha que se deseja realizar a verificação de diagnosticabilidade descentralizada. Considere então a existência de dois diagnosticadores*

locais no sistema (ou seja, $m = 2$) e as projeções $P_{o_i} : \Sigma^\star \rightarrow \Sigma_{o_i}^\star$, $i = 1, 2$, em que $\Sigma_{o_1} = \{a, c\}$ e $\Sigma_{o_2} = \{b, c\}$.

O algoritmo 3.3 supõe a existência de um autômato H que gera a linguagem normal do sistema, ou seja, a linguagem considerada, nesse caso, como linguagem de não falha do sistema. Vamos sempre supor neste trabalho que o autômato H exigido pelo método de QIU e KUMAR [15] é um autômato que gera todas as sequências de $\mathcal{L}(G)$ que não possuem nenhum evento de falha. Assim, pode-se criar o autômato H pelo produto $H = G \times H_0$. O autômato H_0 é um autômato de único estado mostrado na figura 3.3(a), e o autômato H para este exemplo pode ser visto na figura 3.3(b).

Aplicando o algoritmo 3.3, no passo 1 deve-se criar o autômato aumentado $\bar{H}$ adicionando ao autômato H um novo estado F. Esse autômato é mostrado na figura 3.3(c). O autômato V_Q apresentado na figura 3.4, cuja finalidade é a realização do teste da diagnosticabilidade, é criado no passo 2, a partir da junção de estados dos autômatos G, $\bar{H}$ e H, e da definição de uma função de transição f_Q que considera um conjunto aumentado de eventos Σ^Q. Por fim, no passo 3 é realizado o teste para a diagnosticabilidade no autômato V_Q, procurando nele algum ciclo cl que possua pelo menos um estado cuja segunda componente seja F e que possua pelo menos um evento cuja primeira componente seja diferente de ϵ. No autômato V_Q da figura 3.4 existe um ciclo com essas características, e é $cl = (1F11, ccc, 1F11)$. Assim, a linguagem gerada por G é não codiagnosticável em relação a P_{o_i} e a Σ_f, de acordo com o teorema de 3.3. □

Exemplo 3.4 *Considere novamente o autômato G da figura 2.9(a), juntamente com sua partição do conjunto de eventos $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, com $\Sigma_o = \{a, b, c\}$, $\Sigma_{uo} = \{\sigma_f\}$ e $\Sigma_f = \{\sigma_f\}$, e suponha que agora se deseja realizar a verificação de diagnosticabilidade para o caso centralizado (ou seja, $m = 1$) utilizando o método proposto por QIU e KUMAR [15]. Assim, aplicando o algoritmo 3.3 é obtido o autômato de teste $V_{Q,c}$ da figura 3.5. Como nesse autômato não existe nenhum ciclo com estados rotulados por F, pode-se afirmar que a linguagem gerada por G é diagnosticável em relação a P_o e a Σ_f, de acordo com o teorema de 3.3.* □



(a) (b) (c)

Figura 3.3: (a) Autômato H_0, (b) autômato H que modela o comportamento normal de G, e (c) autômato estendido, $\bar{H}$.

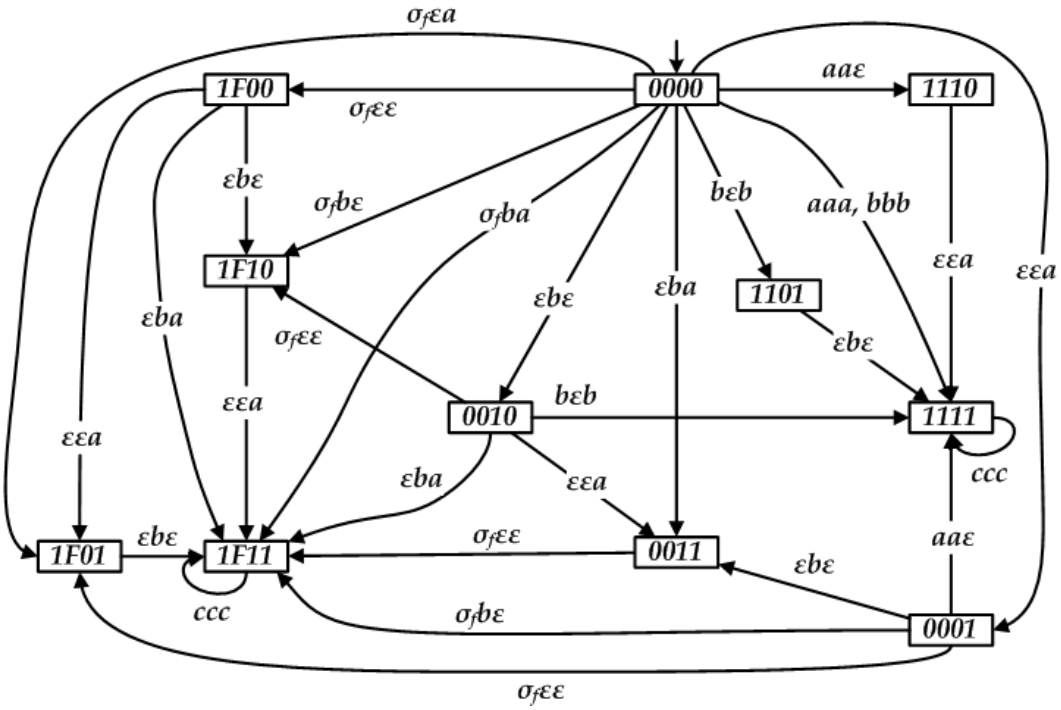


Figura 3.4: Autômato de Teste V_Q para o caso descentralizado

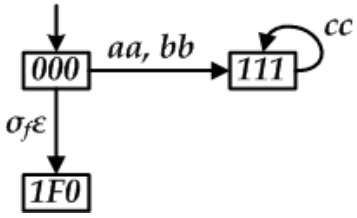


Figura 3.5: Autômato de Teste $V_{Q,c}$ para o caso centralizado

3.4 Verificador proposto por Wang *et al.* [16]

O método proposto por WANG *et al.* [16] também aborda o problema da verificação da codiagnosticabilidade, porém, diferente do método proposto por QIU e KUMAR

[15], esse método não pode ser utilizado para a verificação da diagnosticabilidade centralizada. É importante ressaltar que em [16] também são removidas as hipóteses de vivacidade da linguagem gerada pelo sistema (hipótese $H1$) e de não existência de ciclos de eventos não observáveis (hipótese $H2$), assim como é feito em [15] e pelas mesmas razões.

Algoritmo 3.4 *Seja $G = (X, \Sigma, f, x_0)$ um autômato determinístico e o conjunto de eventos falhas Σ_f. Suponha que, para cada diagnosticador local, Σ seja particionado como $\Sigma_o = \Sigma_{o_i} \dot{\cup} \Sigma_{uo_i}$, $i = 1, 2$, em que Σ_{o_i} e Σ_{uo_i} são os conjuntos de eventos observáveis e não observáveis para cada diagnosticador local, respectivamente.*

- *Passo 1: Construa o autômato verificador*

$$V_W = (X_W, \Sigma^W, f_W, x_{0,W}) \tag{3.5}$$

sendo

$$\Sigma^W = (\Sigma \cup \{\epsilon\})^3,$$

$$X_W = X \times \{N, P\} \times X \times \{N, P\} \times X \times \{N, P\},$$

$$x_{0,W} = (x_0, N, x_0, N, x_0, N)\,,$$

com as seguintes regras para a função de transição f_W (para facilitar a leitura, considere $f(x^i, \sigma) = \hat{x}^i$):

Para $\sigma \in \Sigma_{o_1} \cap \Sigma_{o_2}$

$$f_W((x^1, l^1, x^2, l^2, x^3, l^3), \sigma\sigma\sigma) = \left(\hat{x}^1, l^1, \hat{x}^2, l^2, \hat{x}^3, l^3\right). \tag{3.6}$$

Para $\sigma \in \Sigma_{o_1} \setminus \Sigma_{o_2}$

$$\begin{cases} f_W((x^1, l^1, x^2, l^2, x^3, l^3), \sigma\epsilon\sigma) = (\hat{x}^1, l^1, x^2, l^2, \hat{x}^3, l^3) \\ f_W((x^1, l^1, x^2, l^2, x^3, l^3), \epsilon\sigma\epsilon) = (x^1, l^1, \hat{x}^2, l^2, x^3, l^3) \end{cases}. \tag{3.7}$$

Para $\sigma \in \Sigma_{o_2} \setminus \Sigma_{o_1}$

$$\begin{cases} f_W((x^1, l^1, x^2, l^2, x^3, l^3), \epsilon\sigma\sigma) = (x^1, l^1, \hat{x}^2, l^2, \hat{x}^3, l^3) \\ f_W((x^1, l^1, x^2, l^2, x^3, l^3), \sigma\epsilon\epsilon) = (\hat{x}^1, l^1, x^2, l^2, x^3, l^3) \end{cases}. \tag{3.8}$$

Para $\sigma \in \Sigma_{uo} \setminus \Sigma_f$

$$\begin{cases} f_W((x^1,l^1,x^2,l^2,x^3,l^3),\sigma\epsilon\epsilon) = (\hat{x}^1,l^1,x^2,l^2,x^3,l^3) \\ f_W((x^1,l^1,x^2,l^2,x^3,l^3),\epsilon\sigma\epsilon) = (x^1,l^1,\hat{x}^2,l^2,x^3,l^3) \\ f_W((x^1,l^1,x^2,l^2,x^3,l^3),\epsilon\epsilon\sigma) = (x^1,l^1,x^2,l^2,\hat{x}^3,l^3) \end{cases} . \tag{3.9}$$

Para $\sigma \in \Sigma_f$

$$\begin{cases} f_W((x^1,l^1,x^2,l^2,x^3,l^3),\sigma\epsilon\epsilon) = (\hat{x}^1,P,x^2,l^2,x^3,l^3) \\ f_W((x^1,l^1,x^2,l^2,x^3,l^3),\epsilon\sigma\epsilon) = (x^1,l^1,\hat{x}^2,P,x^3,l^3) \\ f_W((x^1,l^1,x^2,l^2,x^3,l^3),\epsilon\epsilon\sigma) = (x^1,l^1,x^2,l^2,\hat{x}^3,P) \end{cases} . \tag{3.10}$$

- *Passo 2: Verifique se existe em* V_W *um ciclo* $cl = (x_W^k, \sigma_k^W, x_W^{k+1}, \ldots, x_W^l, \sigma_l^W, x_W^k)$, *sendo* $l \geq k \geq 0$ *e* $\sigma_i^W = (\sigma_i^1\sigma_i^2\sigma_i^3)$, *tal que para todo* $x_W^i = (x_i^1, l_i^1, x_i^2, l_i^2, x_i^3, l_i^3)$, $l_i^1 = N$, $l_i^2 = N$, $l_i^3 = P$ *e* $\sigma_i^3 \neq \epsilon$. *Se cl existe, então o sistema é não codiagnosticável em relação a* P_{o_i} *e* Σ_f. *Caso contrário, o sistema é codiagnosticável.* □

O algoritmo 3.4 utiliza a mesma idéia do método proposto em QIU e KUMAR [15], que é representar os estados do autômato verificador sendo compostos pelos estados de cada diagnosticador local e pelos estados G, que são alcançados por sequências de mesma projeção P_{o_i}, $i = 1, 2$. Por sua vez, os eventos do verificador são formados pelos eventos ocorridos em G e em cada diagnosticador local. Assim, cada estado $x_W = (x^1, l^1, x^2, l^2, x^3, l^3) \in X_W$ pode ser associado a um estado no primeiro diagnosticador local, um estado no segundo diagnosticador local e a um estado em G pelos estados x^1, x^2 e x^3, respectivamente, assim como as ocorrências de um evento de falha nesses autômatos estão associadas aos ao rótulos l^1, l^2 e l^3, respectivamente. Ainda no passo 1 do algoritmo 3.4 é definida a função de transição f_W para eventos $\sigma^W = (\sigma^1, \sigma^2, \sigma^3) \in \Sigma^W$, sendo σ^1, σ^2 e σ^3 associados ao primeiro diagnosticador local, ao segundo diagnosticador local e a G, respectivamente, de forma que $P_{o_1}(\sigma^1) = P_{o_1}(\sigma^3)$ e $P_{o_2}(\sigma^2) = P_{o_2}(\sigma^3)$.

Por fim, o passo 2 do algoritmo 3.4 mostra como realizar o teste de co-diagnosticabilidade. De acordo com o algoritmo 3.4, a linguagem gerada por um autômato G é diagnosticável se, e somente se, não existir um ciclo $cl = (x_W^k, \sigma_k, x_W^{k+1}, \ldots, x_W^l, \sigma_l, x_W^k)$, sendo $l \geq k \geq 0$ e $\sigma_i^W = (\sigma_i^1\sigma_i^2\sigma_i^3)$, tal que para

todo $x_W^i = (x_i^1, l_i^1, x_i^2, l_i^2, x_i^3, l_i^3)$, $l_i^1 = N$, $l_i^2 = N$, $l_i^3 = P$ e $\sigma_i^3 \neq \epsilon$. Se *cl* não existe, então não existem sequências s_1, $s_2 \in L \setminus L_N$ associadas a *cl* tais que $f(x_0, s_1) = x_i^1$ e $f(x_0, s_2) = x_i^2$, e não existe uma sequência falha $st \in L$ arbitrariamente longa tal que $f(x_0, st) = x_i^3$, sendo $P_{o_1}(s_1) = P_{o_1}(st)$ e $P_{o_2}(s_2) = P_{o_2}(st)$. O fato de que $l_i^1 = l_i^2 = N$ garante que s_1 e s_2 não contêm algum evento de falha, da mesma forma que $l_i^3 = P$ indica que $\sigma_f \in st$. A garantia de *st* ser arbitrariamente longa após a falha é devido ao fato de $\sigma_i^3 \neq \epsilon$. Assim, como o ciclo *cl* não existe, então não existe sequência *st* de comprimento arbitrariamente longo após a ocorrência de uma falha, que possua a mesma projeção P_{o_i}, $i = 1, 2$, que sequências $s_1, s_2 \in L$, não necessariamente distintas. Desta forma, se *cl* não existe, de acordo com a definição 2.17, pode-se afirmar que L é codiagnosticável em relação a P_{o_i} e Σ_f. Esse fato é apresentado no teorema a seguir.

Teorema 3.4 *Uma linguagem L gerada por um autômato G é codiagnosticável com relação às projeções P_{o_i}, com $i = 1, 2, \ldots, m$, e ao conjunto de falhas Σ_f, se, e somente se, não existe em V_W um ciclo $cl = (x_W^k, \sigma_k, x_W^{k+1}, \ldots, x_W^l, \sigma_l, x_W^k)$, sendo $l \geq k \geq 0$ e $\sigma_i^W = (\sigma_i^1 \sigma_i^2 \sigma_i^3)$, tal que para todo $x_W^i = (x_i^1, l_i^1, x_i^2, l_i^2, x_i^3, l_i^3)$, $l_i^1 = N$, $l_i^2 = N$, $l_i^3 = P$ e $\sigma_i^3 \neq \epsilon$.* □

Demonstração Ver [16]. □

O algoritmo proposto por WANG *et al.* [16] gera um autômato verificador com no máximo $2^{m+1} \times |X|^{m+1}$ estados e $2^{m+1} \times |X|^{m+1} \times |\Sigma| \times (m+1)$ transições.

O exemplo 3.5 ilustra o algoritmo 3.4 proposto por WANG *et al.* [16].

Exemplo 3.5 *Considerando o autômato G da figura 2.9(a) para a realização da verificação de diagnosticabilidade descentralizada, com a observação do sistema sendo feita por dois diagnosticadores locais, ou seja, $m = 2$. Assim, sejam os conjuntos de eventos observáveis referente a cada um dos diagnosticadores locais, $\Sigma_{o_1} = \{a, c\}$ e $\Sigma_{o_2} = \{b, c\}$. Aplicando o algoritmo 3.4, deve-se criar no passo 1 o autômato verificador V_W a partir das regras de transição expressas nas equações (3.6), (3.7), (3.8), (3.9) e (3.10). O autômato V_W pode ser visto na figura 3.6. No passo 2, o teste de codiagnosticabilidade é feito nesse autômato, procurando por algum ciclo cl no qual cada estado $x_W = (x^1, l^1, x^2, l^2, x^3, l^3) \in X_W$ pertencente ao ciclo é definido*

com $l^1 = N$, $l^2 = N$, $l^3 = P$ *e que pelo menos um evento* $\sigma^W = (\sigma^1\sigma^2\sigma^3)$ *de cl seja tal que* $\sigma^3 \neq \epsilon$, *indicando assim que o sistema é não codiagnosticável. Note que no autômato* V_W *da figura 3.6 existe um ciclo com as características descritas. Esse ciclo é* $cl = (1N1N1P, ccc, 1N1N1P)$. *Assim, de acordo com o teorema 3.4, a linguagem gerada por* G *não é codiagnosticável em relação a* P_{o_i} *e a* Σ_f. □

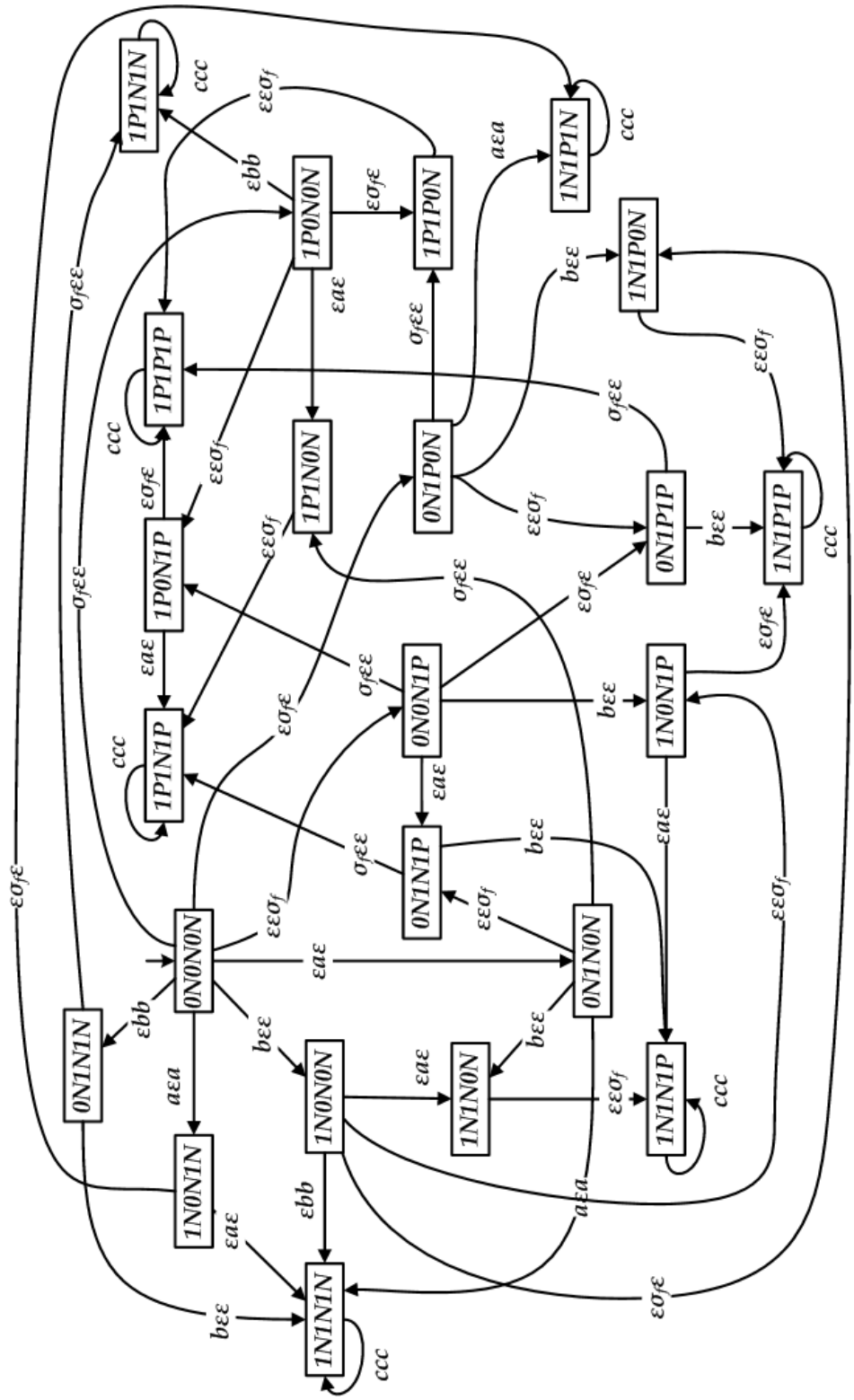


Figura 3.6: Verificador proposto por WANG *et al.* [16]

3.5 Um novo algoritmo para a verificação da diagnosticabilidade de SED

Assim como os métodos propostos por QIU e KUMAR [15] e WANG *et al.* [16], o novo método proposto neste seção pode ser utilizado para realizar a verificação da diagnosticabilidade de um SED em tempo polinomial para o caso descentralizado, porém pode ser utilizado também para o caso centralizado, como o método proposto por QIU e KUMAR [15]. Esse método segue a idéia apresentada na definição 2.17, que afirma que uma linguagem L é não codiagnosticável em relação a P_{o_i} e Σ_f se, e somente se, existe uma sequência st de comprimento arbitrariamente longo após a ocorrência de um evento de falha, e sequências $s_i \in L_N$, não necessariamente distintas, tais que $P_{o_i}(s_i) = P_{o_i}(st)$ para todo $i = 1, \ldots, m$. É importante observar que esse o novo método não faz nenhuma consideração sobre a necessidade de L ser viva. Isso se deve ao fato de que o método de MOREIRA *et al.* [24] contorna o problema da existência de um bloqueio no sistema inserindo um autolaço rotulado por um evento não observável $\sigma_u \in \Sigma_{uo}$ a cada estado de bloqueio. Isso leva a uma linguagem viva L tal que todas as sequências em L têm as mesmas projeções P_{o_i} que antes, e assim, não afeta a propriedade de codiagnosticabilidade do sistema. Esse é o mesmo procedimento descrito em [15], com a diferença que em [15] o evento não observável é a sequência vazia ϵ. Uma vez que estamos considerando o autômato G sendo determinístico, os autolaços adicionados são rotulados por eventos não observáveis σ_u. Dessa forma, sem perda de generalidade, é considerado deste ponto em diante que a linguagem L é viva. Assim, o algoritmo de verificação proposto por MOREIRA *et al.* [24] é baseado na procura por sequências $s_i \in L_N$, para $i = 1, \ldots, m$, e $st \in L \setminus L_N$ que violam as condições de codiagnosticabilidade apresentadas na definição 2.17.

A seguir, um algoritmo de verificação da codiagnosticabilidade de um sistema é apresentado considerando, sem perda de generalidade, $m = 2$. Em seguida, um teorema que mostra sua veracidade é demonstrado.

Algoritmo 3.5 *Seja G o autômato determinístico que modela o sistema e Σ_f o conjunto de eventos de falha. Considere ainda as seguintes partições de Σ sendo que cada uma é associada a um diagnosticador local $\Sigma = \Sigma_{o_i} \dot{\cup} \Sigma_{uo_i}$, para $i = 1, 2$,*

sendo Σ_{o_i} *e* Σ_{uo_i} *os conjuntos de eventos observáveis e não observáveis para cada diagnosticador local, respectivamente.*

- *Passo 1: Construa o autômato* G_N *que modela o comportamento normal de* G*, como segue:*
  - *Passo 1.1: Defina* $\Sigma_N = \Sigma \setminus \Sigma_f$*.*
  - *Passo 1.2: Construa o autômato* A_N *composto de um único estado* N *(que também é o estado inicial) com um autolaço rotulado com todos os eventos pertencentes a* Σ_N*.*
  - *Passo 1.3: Construa o autômato de não falha* $G_N = G \times A_N = (X_N, \Sigma, f_N, \Gamma_N, x_{0,N})$[2]*.*
  - *Passo 1.4: Redefina o conjunto de eventos de* G_N *como* Σ_N*, i.e.,* $G_N = (X_N, \Sigma_N, f_N, \Gamma_N, x_{0,N})$*.*
- *Passo 2: Construa o autômato* G_F *que modela o comportamento de falha do sistema, como segue:*
  - *Passo 2.1: Defina* $A_l = (X_l, \Sigma_f, f_l, x_{0,l})$*, sendo* $X_l = \{N, F\}$*,* $x_{0,l} = \{N\}$*,* $f_l(N, \sigma_f) = F$ *e* $f_l(F, \sigma_f) = F$*, para todo* $\sigma_f \in \Sigma_f$*.*
  - *Passo 2.2: Construa* $G_l = G \| A_l = (X_{G_l}, \Sigma_{G_l}, f_{G_l}, \Gamma_{G_l}, x_{0,G_l})$ *e marque todos os estados de* G_l *cuja segunda coordenada é igual a* F*.*
  - *Passo 2.3: Construa o autômato de falha* $G_F = CoAc(G_l) = (X_F, \Sigma_F, f_F, \Gamma_F, x_{0,F})$ [3]*.*
- *Passo 3: Defina a função* $R_i : \Sigma_N \rightarrow \Sigma_{R_i}$ [4] *como:*

$$R_i(\sigma) = \begin{cases} \sigma, \ \textit{se } \sigma \in \Sigma_{o_i} \\ \sigma_{R_i}, \ \textit{se } \sigma \in \Sigma_{uo_i} \setminus \Sigma_f \end{cases} . \tag{3.11}$$

[2]Note que, como G_N é o subautômato de G que representa o comportamento de não falha do sistema, então $\mathcal{L}(G_N) = L_N$

[3]Note que todas as sequências de G que contém um evento de falha pertencem à linguagem gerada de G_F, *i.e.*, $\mathcal{L}(G_F) = \overline{L \setminus L_N}$.

[4]Note que a função R_i apenas renomeia os rótulos dos eventos pertencentes a $\Sigma_{uo_i} \setminus \Sigma_f$. A notação $R_i(\Sigma_N)$ é usada neste trabalho para representar a renomeação dos eventos pertencentes a Σ_N como descrito por (3.11). Assim, $\Sigma_{R_i} = R_i(\Sigma_N)$.

Construa os autômatos $G_{N,i} = (X_N, \Sigma_{R_i}, f_{N,i}, x_{0,N})$, *para* $i = 1, 2$, *com* $f_{N,i}(x_N, R_i(\sigma)) = f_N(x_N, \sigma)$ *para todo* $\sigma \in \Sigma_N$.

- *Passo 4: Construa o autômato verificador* $G_V = G_{N,1} \| G_{N,2} \| G_F = (X_V, \Sigma_{R_1} \cup \Sigma_{R_2} \cup \Sigma, f_V, x_{0,V})$. *Note que um estado de* G_V *é dado por* $x_V = (x_{N,1}, x_{N,2}, x_F)$, *sendo que* $x_{N,1}$, $x_{N,2}$, *e* x_F *são estados de* $G_{N,1}$, $G_{N,2}$, *e* G_F, *respectivamente, e* $x_F = (x, x_l)$, *sendo que* x *e* x_l *são estados de* G *e* A_l, *respectivamente.*

- *Passo 5: Verifique a existência de um ciclo* $cl := (x_V^k, \sigma_k, x_V^{k+1}, ..., x_V^l, \sigma_l, x_V^k)$, *sendo* $l \geq k \geq 0$, *em* G_V *satisfazendo as seguintes condições:*

$$\exists j \in \{k, k+1, \ldots, l\} \text{ tal que, para algum } x_V^j, (x_l^j = F) \wedge (\sigma_j \in \Sigma).$$

Se cl existe, então L é não codiagnosticável com respeito a P_{o_i} *e* Σ_f. *Caso contrário, L é codiagnosticável.* □

Observação 4 *É importante ressaltar que o verificador proposto neste capítulo é diferente dos verificadores propostos em [15] e [16], uma vez que o verificador obtido no passo* 5 *do algoritmo 3.5 procura apenas sequências de* $\mathcal{L}(G_N)$ *que tenham a mesma projeção que uma sequência de* $\mathcal{L}(G_F)$, *e o verificador proposto em [15] procura por todas as sequências de* $\mathcal{L}(G_N)$ *que tem a mesma projeção que uma sequência de* $\mathcal{L}(G)$. *Isso reduz o número de estados e transições do verificador, reduzindo a complexidade do algoritmo 3.5. Outra característica importante é que o algoritmo 3.5 usa basicamente a composição paralela para a criação do verificador, não criando nenhuma nova operação para autômatos, como feito em [15].* □

O seguinte teorema demonstra a veracidade do algoritmo 3.5.

Teorema 3.5 *Sejam* L *e* L_N ($L_N \subset L$) *linguagens prefixo-fechadas geradas, por* G *e pelo autômato de não falha* G_N, *respectivamente. Considere dois módulos locais com projeções* $P_{o_i} : \Sigma^\star \to \Sigma_{o_i}^\star$ ($i = 1, 2$) *e seja* Σ_f *o conjunto de eventos de falha. Então, L é não codiagnosticável em relação a* P_{o_i} *e* Σ_f *se, e somente se, existe um ciclo em* G_V, $cl := (x_V^k, \sigma_k, x_V^{k+1}, ..., x_V^l, \sigma_l, x_V^k)$ *(como definido no passo* 5 *do algoritmo 3.5), sendo* $l \geq k \geq 0$, *que satisfaz as seguintes condições:*

$$\exists j \in \{k, k+1, \ldots, l\} \text{ tal que, para algum } x_V^j, (x_l^j = F) \wedge (\sigma_j \in \Sigma). \tag{3.12}$$

□

Demonstração ($\Leftarrow$) Suponha que exista um ciclo $cl := (x_V^k, \sigma_k, x_V^{k+1}, ..., x_V^l, \sigma_l, x_V^k)$ satisfazendo as condições (3.12). Uma vez que $x_l^j = F$ para algum $j \in \{k, k+1, \ldots, l\}$, então, pela construção de A_l e G_V, tem-se que $x_l^j = F$ para todo $j \in \{k, k+1, \ldots, l\}$. Logo, existe uma sequência $s_V t_V \in \mathcal{L}(G_V)$, tal que $\sigma_f \in s_V$, sendo $\sigma_f \in \Sigma_f$, e $t_V = (\sigma_k \sigma_{k+1} \ldots \sigma_l)^p$, $p \in \mathbb{N}$, sendo que $|t_V| > n$, $\forall n \in \mathbb{N}$. Defina agora as seguintes projeções:

$$\begin{aligned} P_{R_1} &: (\Sigma \cup \Sigma_{R_1} \cup \Sigma_{R_2})^\star \to \Sigma_{R_1}^\star, \\ P_{R_2} &: (\Sigma \cup \Sigma_{R_1} \cup \Sigma_{R_2})^\star \to \Sigma_{R_2}^\star, \\ P &: (\Sigma \cup \Sigma_{R_1} \cup \Sigma_{R_2})^\star \to \Sigma^\star. \end{aligned}$$

Como $G_V = G_{N,1} \| G_{N,2} \| G_F$, então $\mathcal{L}(G_V) = P_{R_1}^{-1}[\mathcal{L}(G_{N,1})] \cap P_{R_2}^{-1}[\mathcal{L}(G_{N,2})] \cap P^{-1}[\mathcal{L}(G_F)]$, o que implica que $s_V t_V \in P^{-1}[\mathcal{L}(G_F)]$. Seja $st = P(s_V t_V)$, em que $s = P(s_V)$ e $t = P(t_V)$. Assim, uma vez que $P\left[P^{-1}(\mathcal{L}(G_F))\right] = \mathcal{L}(G_F)$, então $st \in \mathcal{L}(G_F)$. Note que, como $t_V = (\sigma_k \sigma_{k+1} \ldots \sigma_l)^p$, $p \in \mathbb{N}$, sendo que $|t_V| > n$, $\forall n \in \mathbb{N}$, e, por hipótese, existe um evento $\sigma_j \in \Sigma$ para $j \in \{k, k+1, \ldots, l\}$, então a sequência $t = P(t_V)$ pode ser feita arbitrariamente longa. Isso define uma sequência $st \in L$ arbitrariamente longa após a ocorrência de um evento de falha.

Seja $s_{R_1} = P_{R_1}(s_V t_V)$. Uma vez que $s_V t_V \in \mathcal{L}(G_V)$, então $s_V t_V \in P_{R_1}^{-1}[\mathcal{L}(G_{N,1})]$. Além disso, como $P_{R_1}\left[P_{R_1}^{-1}[\mathcal{L}(G_{N,1})]\right] = \mathcal{L}(G_{N,1})$, então $s_{R_1} \in \mathcal{L}(G_{N,1})$. Note que $G_{N,1}$ é obtido a partir de G_N após a renomeação dos eventos do conjunto $\Sigma \setminus \Sigma_f$ usando a função R_1. Assim, existe uma sequência $s_1 \in \mathcal{L}(G_N)$ tal que $P_{o_1}(s_1) = P(s_{R_1})$. Usando o mesmo raciocínio pode ser mostrado que existem sequências $s_{R_2} \in \mathcal{L}(G_{N,2})$ e $s_2 \in \mathcal{L}(G_N)$ tais que $P_{o_2}(s_2) = P(s_{R_2})$. Para concluir a demonstração, note que

$$P(s_{R_1}) = P[P_{R_1}(s_V t_V)] = P_{R_1}[P(s_V t_V)] = P_{R_1}(st)$$

e assim

$$P_{o_1}(s_1) = P_{R_1}(st).$$

Como $st \in \mathcal{L}(G_F) \subseteq \Sigma^\star$,

$$P_{R_1}(st) = P_{o_1}(st),$$

o que leva a

$$P_{o_1}(s_1) = P_{o_1}(st).$$

O mesmo raciocínio pode ser usado para mostrar que $P_{o_2}(s_2) = P_{o_2}(st)$. Esses fatos mostram que existem sequências $s_1, s_2 \in L_N$ e $st \in L \setminus L_N$ que violam a condição de codiagnosticabilidade da definição 2.17.

($\Rightarrow$) Suponha que L não é codiagnosticável em relação a P_{o_i} e Σ_f. Assim, existe uma sequência $st \in \mathcal{L}(G_F)$, sendo $\sigma_f \in s$ e $|t| > n$ para todo $n \in \mathbb{N}$, e $s_1, s_2 \in \mathcal{L}(G_N)$, tais que $P_{o_1}(st) = P_{o_1}(s_1)$ e $P_{o_2}(st) = P_{o_2}(s_2)$. Vamos mostrar que existe um ciclo de estados de falha em G_V com sequências st, s_1 e s_2, que violam a condição de codiagnosticabilidade da definição 2.17 e, para esse propósito, vamos dividir a demonstração em duas partes como segue: (i) na primeira parte vamos mostrar que existe uma sequência de comprimento arbitrariamente longo $v \in \mathcal{L}(G_V)$ tal que $P(v) = st$, $P_{R_1}(v) = s_{R_1}$, e $P_{R_2}(v) = s_{R_2}$, sendo $s_{R_1} = R_1(s_1)$ e $s_{R_2} = R_2(s_2)$[5]; (ii) na segunda parte nós demonstramos que existe um ciclo cl, associado à sequência v, satisfazendo a condição (3.12).

Para demonstrar a parte (i), suponha que existe um estado em G_V, $x_V = (x_{N,1}, x_{N,2}, x_F)$, alcançável a partir do estado inicial $x_{0,V}$ depois da execução da sequência $u \in \mathcal{L}(G_V)$, sendo que $P(u)$ pertence ao prefixo fechamento de $\{st\}$, *i.e.*, $P(u) \in \overline{\{st\}}$. Note que esse estado x_V sempre existe, uma vez que u pode ser a sequência vazia e, nesse caso, $x_V = x_{0,V}$. Agora, seja $\sigma \in \Sigma$ um evento ativo de x_F, tal que $P(u)\sigma \in \overline{\{st\}}$, e considere o problema de encontrar o estado de G_V, $\hat{x}_V$, alcançável a partir de x_V, que tenha σ como um evento ativo. Três casos são possíveis: (a) σ é observável por apenas um diagnosticador local; (b) σ é observável por ambos diagnosticadores locais; (c) σ é um evento não observável, *i.e.*, $\sigma \in \Sigma_{uo}$.

Considere primeiro o caso (a) e suponha, por exemplo, que $\sigma \in \Sigma_{o_1} \setminus \Sigma_{o_2}$. Portanto, para obter $G_V = G_{N,1} \| G_{N,2} \| G_F$, um autolaço rotulado por σ deve ser introduzido em todos os estados de $G_{N,2}$. Assim, σ pode ocorrer se, e somente se, ele é um evento ativo do estado associado a $G_{N,1}$. Uma vez que $P_{o_1}(s_1) = P_{o_1}(st)$, é possível ver que, depois da ocorrência de uma sequência finita de eventos não observáveis pertencentes a $\Sigma^{\star}_{R_1}$, um estado de $G_{N,1}$ ($\hat{x}_{N,1}$) que possui σ como um evento ativo deve ser alcançado. Quando esse estado for alcançado, σ será um evento ativo de $\hat{x}_V = (\hat{x}_{N,1}, x_{N,2}, x_F)$ como desejado.

[5]A extensão da função R_i para o domínio $\Sigma^{\star}$ é feita de modo usual, *i.e.*, $R_i(s\sigma) = R_i(s)R_i(\sigma)$, para todo $s \in \Sigma^{\star}$ e $\sigma \in \Sigma$, e $R_i(\epsilon) = \epsilon$.

Considere agora o caso (b), *i.e.*, $\sigma \in \Sigma_{o_1} \cap \Sigma_{o_2}$. Nesse caso, σ será um evento ativo de x_V se, e somente se, ele é ativo para os estados correspondentes de $G_{N,1}$ e $G_{N,2}$. Uma vez que $P_{o_1}(s_1) = P_{o_1}(st)$ e $P_{o_2}(s_2) = P_{o_2}(st)$, então σ será ativo para o estado de G_V, $\hat{x}_V = (\hat{x}_{N,1}, \hat{x}_{N,2}, x_F)$, depois da ocorrência de uma sequência finita de eventos pertencente a $(\Sigma_{R_1} \cup \Sigma_{R_2})^\star$.

Finalmente, considere o caso (c), *i.e.*, $\sigma \in \Sigma_{uo}$. Nesse caso, um autolaço rotulado por todos os eventos pertencentes ao conjunto Σ_{uo} é adicionado a cada estado de $G_{N,1}$ e $G_{N,2}$, o que implica que σ é ativo para $x_V = (x_{N,1}, x_{N,2}, x_F)$. Portanto, pode-se ver que existe uma sequência v associada a st tal que $v \in P_{R_1}^{-1}(s_{R_1}) \cap P_{R_2}^{-1}(s_{R_2}) \cap P^{-1}(st)$, o que implica que $P(v) = st$, $P_{R_1}(v) = s_{R_1}$, e $P_{R_2}(v) = s_{R_2}$.

Para demonstrar a parte (ii), *i.e.* que existe um ciclo cl em G_V, associado a v, sendo que pelo menos um dos eventos pertencentes a esse ciclo pertence a Σ, note que uma vez que G_F e G_V são autômatos finitos, então associado a st existe um ciclo de estados de falha cl_F em G_F que pode ser associado a um caminho em G_V sendo que os eventos de cl_F estão contidos nesse caminho. Uma vez que G_V é um autômato de estados finitos e $st = P(v)$, esse caminho deve incluir um ciclo cl sendo que pelo menos um dos eventos em cl pertence a Σ. Além disso, uma vez que cl_F é um ciclo de estados de falha, pode-se ver pela construção do autômato verificador G_V que cl é um ciclo de estados de falha que satisfaz a condição (3.12).

□

A demonstração do Teorema 3.5 fornece uma maneira simples para encontrar as sequências s_1, s_2 e st que levam à violação da codiagnosticabilidade de L, como segue:

(1) Identifique um ciclo que satisfaça a condição para não codiagnosticabilidade imposta no passo 5 do algoritmo 3.5 e obtenha uma sequência v que leva $x_{0,V}$ para esse ciclo.

(2) Obtenha $s_{R_1} = P_{R_1}(v)$, $s_{R_2} = P_{R_2}(v)$ e $st = P(v)$.

(3) Defina a função de renomeação inversa

$$\begin{aligned} R_i^{-1}: \quad \Sigma_{R_i} &\to \Sigma \\ \sigma_{R_i} &\mapsto \sigma \end{aligned}$$

em que $\sigma_{R_i} = R_i(\sigma)$, com a seguinte extensão para o domínio $\Sigma_{R_i}^{\star}$: $R_i^{-1}(s_{R_i}\sigma_{R_i}) = R_i^{-1}(s_{R_i})R_i^{-1}(\sigma_{R_i})$ para todo $s_{R_i} \in \Sigma_{R_i}^{\star}$ e $\sigma_{R_i} \in \Sigma_{R_i}$, e $R_i^{-1}(\epsilon) = \epsilon$.

(4) Obtenha $s_1 = R_1^{-1}(s_{R_1})$ e $s_2 = R_2^{-1}(s_{R_2})$.

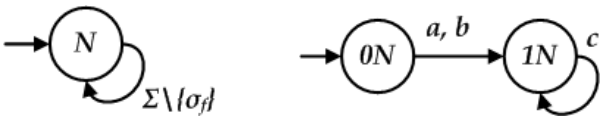
Outra observação importante é que um teste para a diagnosticabilidade de L em relação à projeção P_o e ao conjunto de eventos de falha Σ_f, no caso centralizado, pode ser facilmente obtido fazendo-se $m = 1$ no algoritmo 3.5. Portanto, um autômato verificador para o caso centralizado é dado por $G_{V,c} = G_{N,1} \| G_F$ e a condição necessária e suficiente para a não diagnosticabilidade de L é a existência de um ciclo de estados de falha em $G_{V,c}$ tal que pelo menos um evento no ciclo é um evento pertencente a Σ.

O exemplo 3.6 ilustra a utilização do algoritmo 3.5 para a verificação da codiagnosticabilidade, e o exemplo 3.7 ilustra a utilização desse algoritmo para a verificação da diagnosticabilidade centralizada.

Exemplo 3.6 *Considere o sistema modelado pelo autômato G mostrado na figura 2.9(a), que foi utilizado nos exemplos 2.9, 3.1, 3.2, 3.3, 3.4, 3.5, e suponha que a observação do sistema é realizada por dois diagnosticadores locais, ou seja, $m = 2$. Assim, sejam os conjuntos de eventos observáveis referente a cada um dos diagnosticadores locais, $\Sigma_{o_1} = \{a, c\}$ e $\Sigma_{o_2} = \{b, c\}$. O conjunto de eventos não observáveis neste exemplo é $\Sigma_{uo} = \{\sigma_f\}$ e o conjunto de eventos de falha é $\Sigma_f = \{\sigma_f\}$. Para verificar se a linguagem L gerada por G é codiagnosticável em relação a P_{o_i}, para $i = 1, 2$, e Σ_f, um autômato verificador G_V para o caso descentralizado pode ser obtido usando o algoritmo 3.5. O primeiro passo é a obtenção do autômato A_N e do autômato de não falha G_N. Esses autômatos são mostrados na figura 3.7. O próximo passo é obter o autômato G_l, que é apresentado na figura 3.8(b) e é o resultado da composição paralela entre G e A_l (apresentado na figura 3.8(a)), assim como calcular o autômato de falha G_F apresentado na figura 3.8(c), obtido tomando a parte coacessível de G_l. De acordo com o algoritmo 3.5, é preciso obter os autômatos $G_{N,1}$ e $G_{N,2}$ (mostrados na figura 3.9) a partir de G_N pela renomeação dos eventos não observáveis nos conjuntos $\Sigma_{uo_1} \setminus \Sigma_f = \{b\}$ e $\Sigma_{uo_2} \setminus \Sigma_f = \{a\}$, respectivamente. Finalmente, o autômato verificador $G_V = G_{N,1} \| G_{N,2} \| G_F$, apresentado na figura 3.10,*

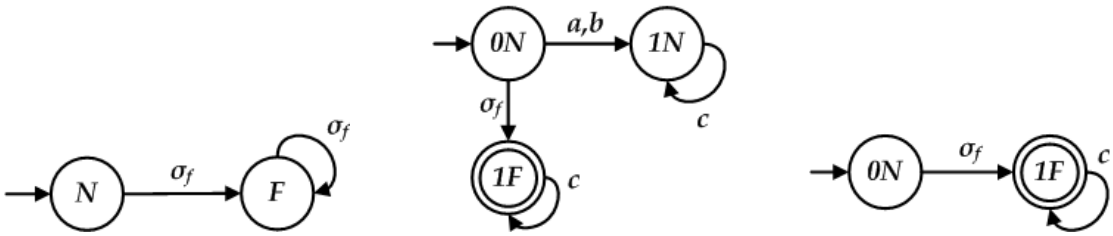
é calculado. Note que existe um autolaço no estado $1N1N1F$ *rotulado pelo evento* b*. Como esse é um ciclo de estados de falha e* $b \in \Sigma$*, de acordo com o teorema 3.5 a linguagem gerada por* G *é não codiagnosticável em relação a* P_{o_i} *e* Σ_f*.*

Como citado, as sequências que levam à violação da codiagnosticabilidade podem ser obtidas a partir do verificador proposto por MOREIRA et al. *[24]. É possível ver no verificador da figura 3.10 que as sequências que levam à violação da diagnosticabilidade são: as sequências de falha* $st = \sigma_f c^n$*; as sequências de não falha* $s_1 = bc^p$ *em relação a* P_{o_1}*; e as sequências de não falha* $s_2 = ac^q$ *em relação a* P_{o_2}*.* □



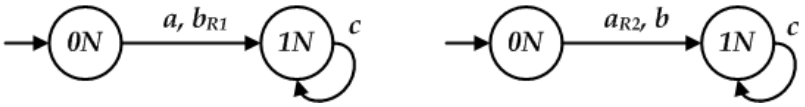
(a) (b)

Figura 3.7: (a) Autômato A_N e (b) autômato de não falha, G_N.



(a) (b) (c)

Figura 3.8: (a) Autômato A_l, (b) autômato G_l, e (c) autômato de falha, G_F.



(a) (b)

Figura 3.9: (a) Autômato de não falha para o módulo 1, $G_{N,1}$ e (b) autômato de não falha para o módulo 2, $G_{N,2}$.

Exemplo 3.7 *Aplicando o algoritmo 3.5 no autômato* G *mostrado na figura 2.9(a), e considerando uma arquitetura centralizada, (ou seja,* $m = 1$*), os conjuntos de eventos* $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$*,* $\Sigma_o = \{a, b, c\}$*,* $\Sigma_{uo} = \{\sigma_f\}$ *e* $\Sigma_f = \{\sigma_f\}$*, é obtido o autômato verificador para o caso centralizado, denotado neste trabalho por* $G_{V,c}$*, que é apresentado*

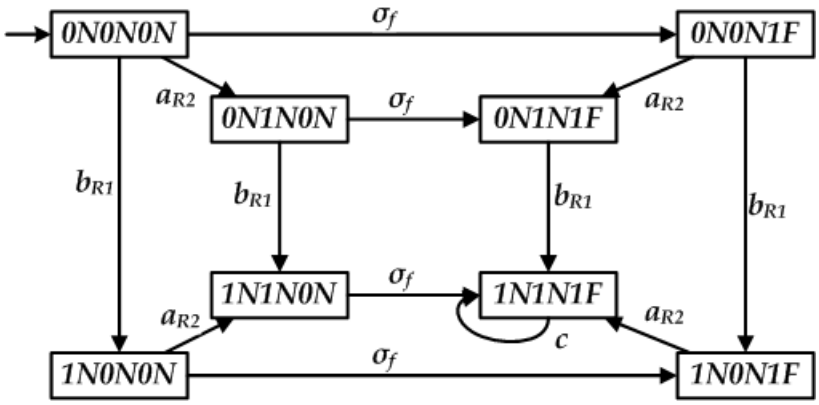


Figura 3.10: Autômato verificador para o caso descentralizado, G_V.

na figura 3.11. Como nesse autômato não existe nenhum ciclo com estados de falha, então, de acordo com o teorema 3.5, a linguagem gerada por G é diagnosticável em relação a P_o *e a* Σ_f*.* □

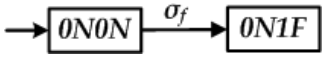


Figura 3.11: Autômato verificador para o caso centralizado, $G_{V,c}$.

3.6 Análise de complexidade do algoritmo de Moreira *et al.* [24]

A complexidade computacional do algoritmo 3.5 é determinada unicamente baseada na análise dos passos necessários para obter o autômato verificador G_V, uma vez que o passo 5 (a busca por ciclos em G_V que violam a codiagnosticabilidade) é sabido ser linear no número de estados e transições de G_V [31].

Na tabela 3.1 é mostrado o número máximo de estados e transições de todos os autômatos que devem ser construídos de acordo com o algoritmo 3.5 para obter o autômato G_V supondo m diagnosticadores locais. O primeiro passo do algoritmo 3.5 é a construção do autômato A_N, composto por um único estado e com transições rotuladas por todos os eventos do conjunto $\Sigma_N = \Sigma \setminus \Sigma_f$, e do autômato de não falha $G_N = G \times A_N$. Isso implica que o número máximo de estados e o número máximo de transições de G_N são $|X|$ e $|\Sigma| - |\Sigma_f|$, respectivamente. O segundo passo do algoritmo 3.5 é a construção do autômato G_F. Para tanto, é necessário primeiro construir o

Tabela 3.1: Complexidade computacional do algoritmo 3.5.

	No. Estados	No. Transições
G	$\|X\|$	$\|X\| \times \|\Sigma\|$
A_N	1	$\|\Sigma\| - \|\Sigma_f\|$
G_N	$\|X\|$	$\|X\| \times (\|\Sigma\| - \|\Sigma_f\|)$
A_l	2	$2\|\Sigma_f\|$
G_l	$2\|X\|$	$2\|X\| \times \|\Sigma\|$
G_F	$2\|X\|$	$2\|X\| \times \|\Sigma\|$
H_i	$\|X\|$	$\|X\| \times (\|\Sigma\| - \|\Sigma_f\|)$
V	$2\|X\|^{m+1}$	$2\|X\|^{m+1} \times [\|\Sigma\| + m(\|\Sigma\| - \|\Sigma_f\|)]$
Complexidade		$O\left(m \times \|X\|^{m+1} \times (\|\Sigma\| - \|\Sigma_f\|)\right)$

autômato A_l com dois estados, N e F, cujas transições são rotuladas somente por eventos de falha e, em seguida, fazer $G_l = G \| A_l$. Note que $\mathcal{L}(G_l) = \mathcal{L}(G)$ e que os estados de G_l são da forma (x, N) ou (x, F), com $x \in X$. Portanto, o número máximo de estados de G_l é $2 \times |X|$. O autômato de falha G_F é calculado tomando a parte coacessível de G_l cujo conjunto de estados marcados é formado pelos estados da forma (x, F), $x \in X$. Isso leva a um autômato que gera a linguagem $\mathcal{L}(G_F) = \overline{L \setminus L_N}$. Uma vez que $G_F = CoAc(G_l)$, então, no pior caso, G_F tem o mesmo número de estados e transições que G_l. No passo 3 os autômatos $G_{N,i}$, para $i = 1, \ldots, m$, são obtidos a partir de G_N pela renomeação dos eventos não observáveis do conjunto $\Sigma_{uo_i} \setminus \Sigma_f$ para cada diagnosticador local de acordo com a função R_i definida em (3.11). Isso leva a m autômatos com o mesmo número de estados e transições que G_N. Finalmente, no passo 4, o autômato verificador G_V é obtido a partir da operação $G_V = G_{N,1} \| G_{N,2} \| \ldots \| G_{N,m} \| G_F$. Como os números de estados de $G_{N,i}$ e G_F são no máximo iguais a $|X|$ e $2 \times |X|$, respectivamente, então o número de estados de G_V é no pior caso igual a $2 \times |X|^{m+1}$. Além disso, o número máximo de transições de G_V é igual a $2 \times |X|^{m+1} \times [|\Sigma| + m \times (|\Sigma| - |\Sigma_f|)]$ uma vez que, para a construção de cada autômato $G_{N,i}$, $(|\Sigma| - |\Sigma_f|)$ novos eventos devem ser criados. Portanto, a complexidade do algoritmo 3.5 é $O\left(m \times |X|^{m+1} \times (|\Sigma| - |\Sigma_f|)\right)$.

Note que a complexidade computacional do algoritmo 3.5 é

Tabela 3.2: Complexidade computacional dos métodos de verificação de diagnosticabilidade.

Método	Complexidade
JIANG *et al.* [13]	$O(\|X\|^4 \times \|\Sigma_o\|))$
YOO e LAFORTUNE [14]	$O(\|X\|^2 \times \|\Sigma\|))$
QIU e KUMAR [15]	$O(\|X\|^2 \times \|\Sigma\|^2)$
MOREIRA *et al.* [24]	$O\left(\|X\|^2 \times (\|\Sigma\| - \|\Sigma_f\|)\right)$

Tabela 3.3: Complexidade computacional dos métodos de verificação de codiagnosticabilidade.

Método	Complexidade
QIU e KUMAR [15]	$O(\|X\|^{m+1} \times \|\Sigma\|^{m+1})$
WANG *et al.* [16]	$O(m \times 2^m \times \|X\|^{m+1} \times \|\Sigma\|)$
MOREIRA *et al.* [24]	$O\left(m \times \|X\|^{m+1} \times (\|\Sigma\| - \|\Sigma_f\|)\right)$

$O\left(m \times |X|^{m+1} \times (|\Sigma| - |\Sigma_f|)\right)$, que é menor do que a complexidade dos algoritmos propostos em [15] e [16] que são $O(|X|^{m+1} \times |\Sigma|^{m+1})$ e $O(m \times 2^m \times |X|^{m+1} \times |\Sigma|)$, respectivamente. Isso acontece porque apenas sequências $s \in \mathcal{L}(G_F)$ e $s_1, s_2 \in \mathcal{L}(G_N)$ (s_1 não necessariamente distinta de s_2) que satisfazem $P_{o_1}(s) = P_{o_1}(s_1)$ e $P_{o_2}(s) = P_{o_2}(s_2)$ são representadas no verificador G_V. É importante observar ainda que o tamanho do autômato verificador G_V é, em geral, menor do que o pior caso apresentado na tabela 3.1 uma vez que o algoritmo procura apenas pelas sequências em $L \setminus L_N$ e L_N que têm as mesmas projeções.

A tabela 3.2 mostra um comparativo entre as complexidades computacionais dos métodos de verificação de diagnosticabilidade centralizada de um SED em tempo polinomial. Por sua vez, a tabela 3.3 mostra a comparação entre as complexidades computacionais dos métodos de verificação de codiagnosticabilidade.

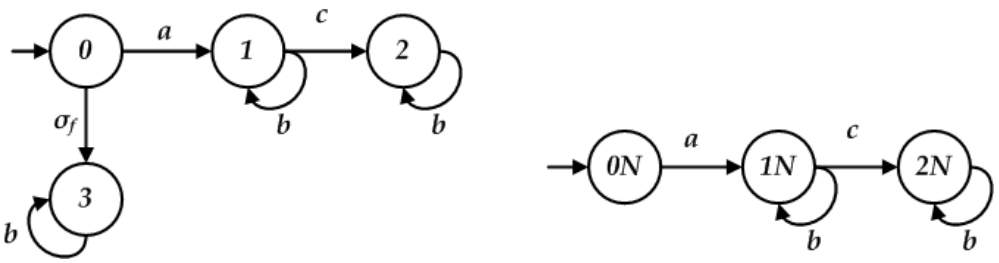
Observação 5 *É importante ressaltar que, para sistemas com número de eventos elevado, tem-se que* $|\Sigma| \gg |\Sigma_f|$*, tornando a complexidade computacional no método proposto por MOREIRA* et al. *[24]* $O\left(m \times |X|^{m+1} \times |\Sigma|\right)$*, no caso descentralizado,*

e $O(|X|^2 \times |\Sigma|)$*, no caso centralizado.* □

Para ilustrar o crescimento dos autômatos analisando-se suas ordens de complexidade, são apresentados a seguir os exemplos 3.8, 3.9, 3.10 e 3.11. O exemplo 3.8 apresenta uma comparação entre o tamanho dos autômatos verificadores para a codiagnosticabilidade a codiagnose apresentados neste trabalho, enquanto que o exemplo 3.9 apresenta essa comparação de tamanho entre os verificadores para o caso centralizado apresentados neste trabalho. Os exemplos 3.10 e 3.11 ilustram o quão grande pode ser o crescimento de um autômato verificador utilizado para a verificação da codiagnosticabilidade de um SED.

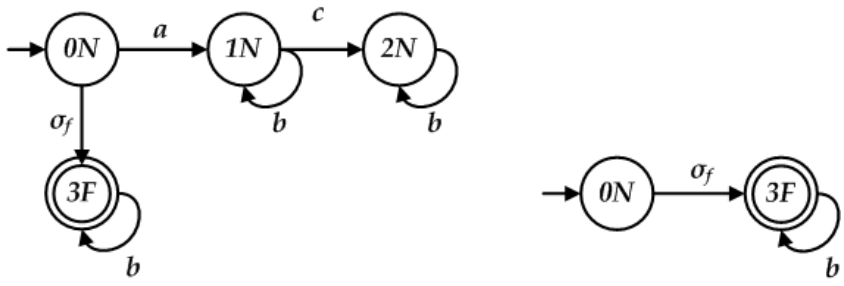
Exemplo 3.8 *Considere o autômato G apresentado na figura 3.12(a) e considere que existam dois diagnosticadores locais cujos conjuntos de eventos observáveis sejam* $\Sigma_{o_1} = \{a, b\}$ *e* $\Sigma_{o_2} = \{b, c\}$*. O conjunto dos eventos de falha é* $\Sigma_f = \{\sigma_f\}$*. Assim, seguindo os passos do algoritmo 3.5 e construindo os autômatos* G_N*,* G_l*,* G_F*,* $G_{N,1}$ *e* $G_{N,2}$*, mostrados nas figuras 3.12(b), 3.13(a), 3.13(b), 3.14(a) e 3.14(b), respectivamente, pode-se obter o autômato verificador* G_V *da figura 3.15. Note que* G_V *não possui ciclos de estados de falha e, assim, de acordo com o teorema 3.5, a linguagem L gerada por G é codiagnosticável em relação a* P_{o_i} *e* Σ_f*. Vamos comparar esse verificador com os demais métodos existentes na literatura. De acordo com o método de QIU e KUMAR [15], é necessário criar os autômatos H e* $\bar{H}$ *(mostrados na figura 3.8) para então obter o autômato de teste* V_Q *que está representado na figura 3.17. Esse autômato também indica que L é codiagnosticável em relação a* P_{o_i} *e* Σ_f *devido à ausência de ciclos de estados de falha, de acordo com o teorema 3.3. Nesse ponto já é possível ver o ganho obtido utilizando o método de MOREIRA* et al. *[24]. Enquanto* G_V *apresenta apenas* 4 *estados e* 4 *transições, o autômato* V_Q *possui* 10 *estados e* 21 *transições. Crescimento ainda maior é constatado no verificador* V_W *do método de WANG* et al. *[16], que é exibido na figura 3.18. Esse autômato possui* 20 *estados e* 41 *transições. Analisando o autômato* V_W *também podemos constatar que, de acordo com o teorema 3.4, L é codiagnosticável em relação a* P_{o_i} *e* Σ_f*, uma vez que não existe nenhum ciclo de estados com os rótulos* $l^1 = N$*,* $l^2 = N$ *e* $l^3 = P$*. Vamos analisar também o autômato proposto por SAMPATH* et al. *[10], mesmo não sendo um método baseado na construção de verificadores em tempo polinomial. De acordo com o algoritmo 2.3 é preciso criar os diagnosticadores*

locais $Diag_1(G)$ *e* $Diag_2(G)$ *(apresentados nas figuras 3.19(a) e 3.19(b), respectivamente) para então obter o codiagnosticador* $CoDiag(G)$ *usado para a verificação da codiagnosticabilidade. Também pode-se afirmar, de acordo com o teorema 2.2, que* L *é codiagnosticável em relação a* P_{o_i} *e* Σ_f *a partir de* $CoDiag(G)$*, já que nesse autômato não existe nenhum ciclo que possa ser associado a um ciclo indeterminado em cada diagnosticador local. Mesmo possuindo uma complexidade exponencial, o método de* SAMPATH et al. *[10] gerou um autômato com apenas* 5 *estados e* 8 *transições. Esses números se devem ao fato do autômato* G *deste exemplo possuir poucas sequências ambíguas (uma sequência de falha de comprimento arbitrariamente longo que tenha a mesma projeção de uma sequência que não possui nenhum evento de falha), o que impede o crescimento exponencial característico do método de* SAMPATH et al. *[10].* □



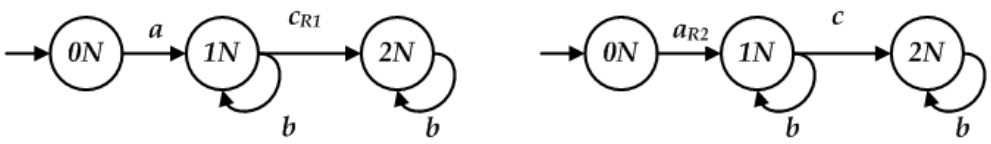
(a) (b)

Figura 3.12: (a) Autômato G e (b) autômato de não falha, G_N.



(a) (b)

Figura 3.13: (a) autômato G_l, e (b) autômato de falha, G_F.



(a) (b)

Figura 3.14: (a) Autômato de não falha para o módulo 1, $G_{N,1}$ e (b) autômato de não falha para o módulo 2, $G_{N,2}$.

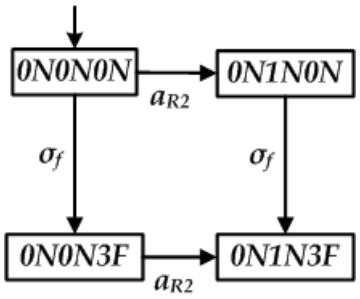
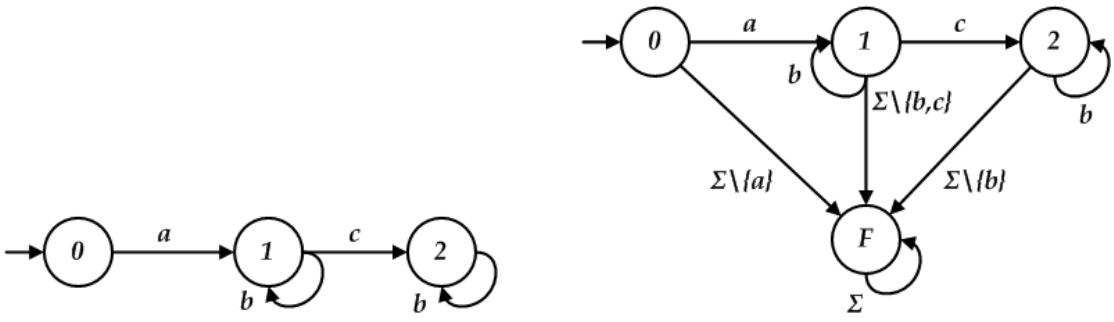


Figura 3.15: Autômato verificador para o caso descentralizado, G_V.



(a) (b)

Figura 3.16: (a) autômato H que modela o comportamento normal de G, e (b) autômato estendido, $\bar{H}$.

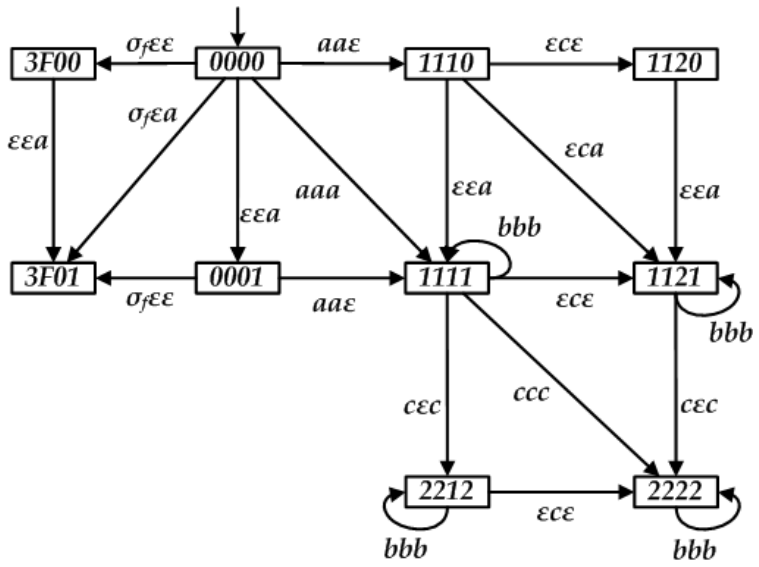


Figura 3.17: Autômato de teste V_Q para o caso descentralizado.

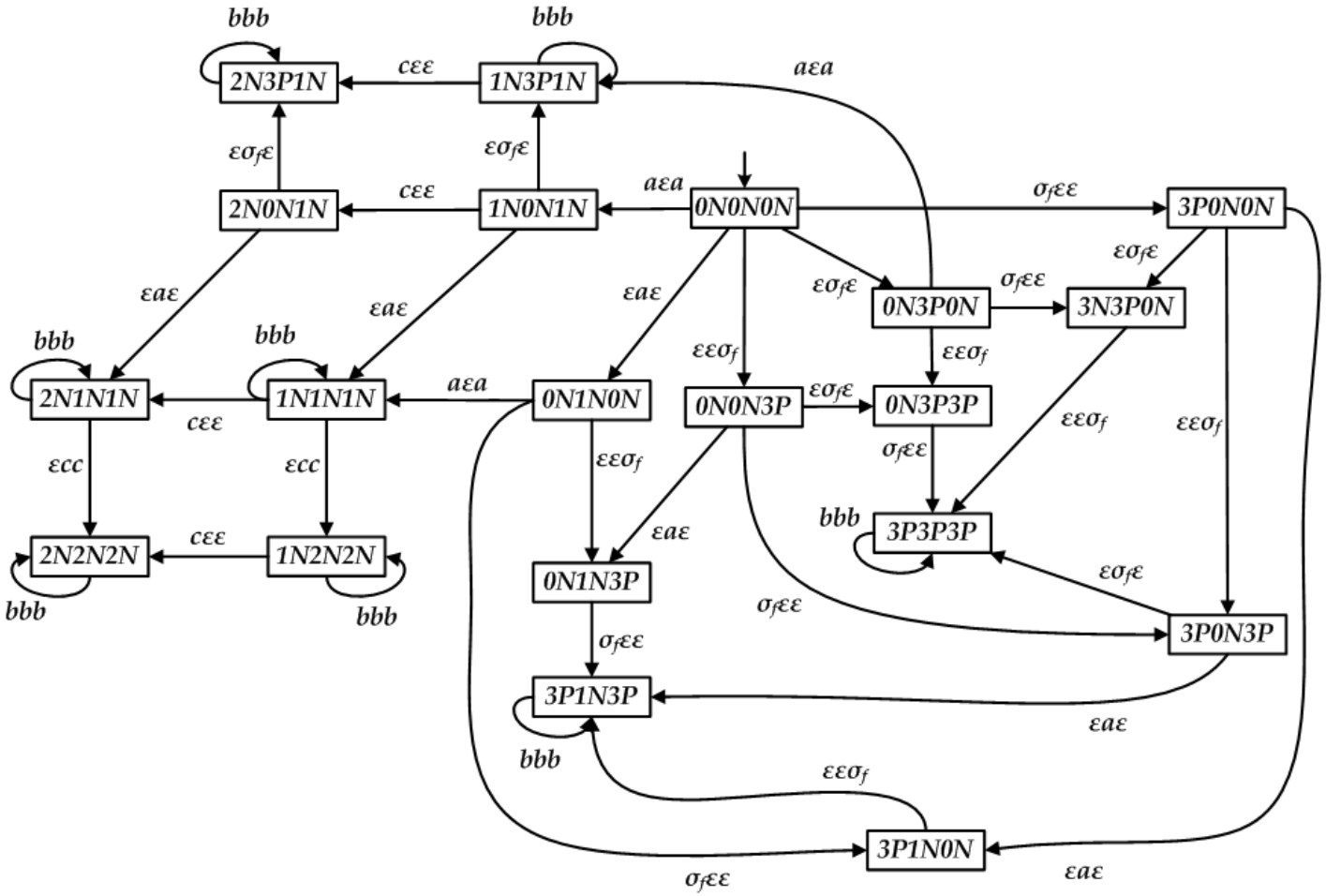


Figura 3.18: Verificador V_W de WANG *et al.* [16].

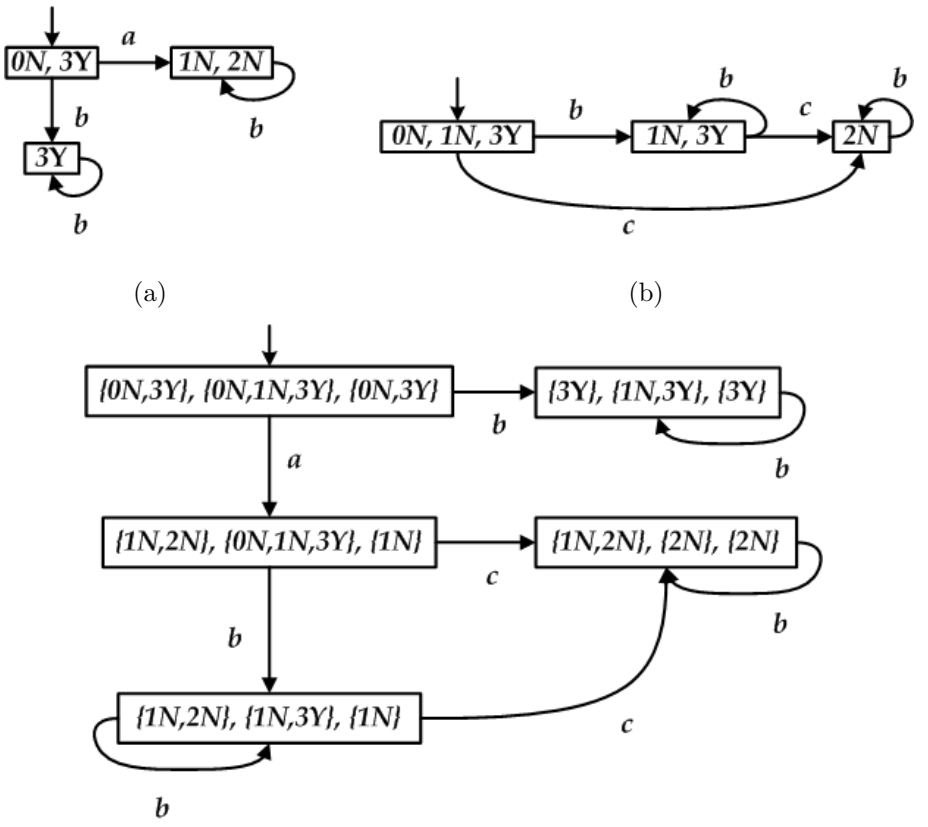


Figura 3.19: (a) Diagnosticador local 1, $Diag_1(G)$, (b) diagnosticador local 2, $Diag_2(G)$, e (c) codiagnosticador, $CoDiag(G)$.

Exemplo 3.9 *Considere novamente o autômato G apresentado na figura 3.12(a), e suponha que se deseje realizar a verificação da diagnosticabilidade centralizada por meio dos métodos apresentados em [24], [13], [14], [15] e [10].*

Assim, considere $\Sigma_o = \{a, b, c\}$ *e* $\Sigma_{uo} = \Sigma_f = \{\sigma_f\}$. *Aplicando o algoritmo 3.5 proposto por* MOREIRA et al. *[24] obtemos o autômato* $G_{V,c}$ *da figura 3.20 que possui apenas* 2 *estados e* 1 *transição, tendo assim* 2 *estados e* 5 *transições a menos em relação ao verificador* V_J *da figura 3.21(b) proposto por* JIANG et al. *[13] e que possui* 4 *estados e* 6 *transições. Esse autômato foi criado através do algoritmo 3.1, com o auxílio do autômato* V_{J_o} *(apresentado na figura 3.21(a)). Aplicando o método proposto por* YOO *e* LAFORTUNE *[14], é obtido o verificador* V_Y *apresentado na figura 3.22, que possui* 6 *estados e* 10 *transições. Assim como o método de* MOREIRA et al. *[24], o método de* QIU *e* KUMAR *[15] apresentado no algoritmo 3.3 pode ser utilizado tanto para em uma arquitetura descentralizada quanto centralizada. Na arquitetura centralizada esse método gerou o autômato da figura 3.23, que*

possui 4 *estados e* 5 *transições. Por fim, temos o autômato* $Diag(G)$ *mostrado na figura 3.9, obtido pelo algoritmo 2.2 proposto por* SAMPATH et al. *[10].* $Diag(G)$ *possui* 4 *estados e* 6 *transições. Note que todos os métodos indicam, de acordos com seus respectivos teoremas, que a linguagem* L *é diagnosticável em relação ao conjunto de eventos observáveis* Σ_o*, à projeção* P_o *e ao conjunto de eventos de falhas* Σ_f*. Isso era esperado, uma vez que o* L *é codiagnosticável.* □

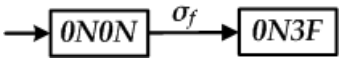
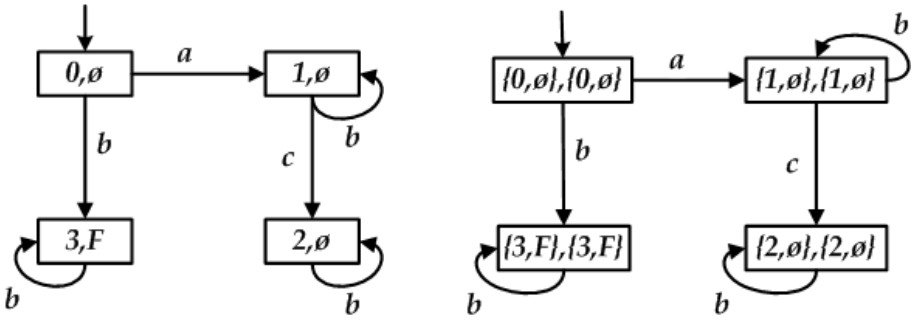


Figura 3.20: Autômato verificador para o caso centralizado, $G_{V,c}$.



(a) (b)

Figura 3.21: (a) Autômato V_{J_o}, (b) autômato V_J.

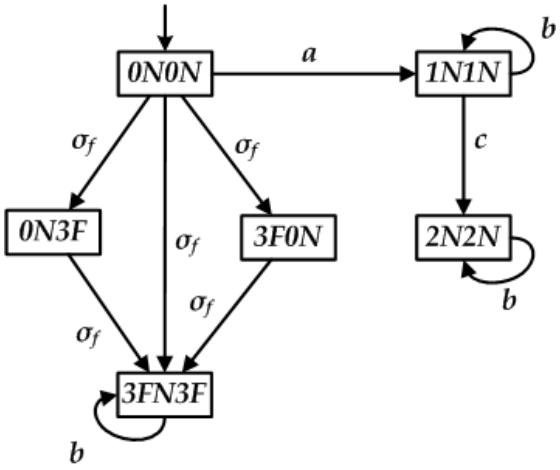


Figura 3.22: Verificador V_Y proposto por YOO e LAFORTUNE [14]

Exemplo 3.10 *Para este exemplo, considere o autômato* G *apresentado na figura 3.25 e suponha que os conjuntos de eventos observáveis dos diagnosticadores locais são* $\Sigma_{o_1} = \{a, b, c\}$ *e* $\Sigma_{o_2} = \{a, c, d\}$*, e que o conjunto dos eventos de falha é*

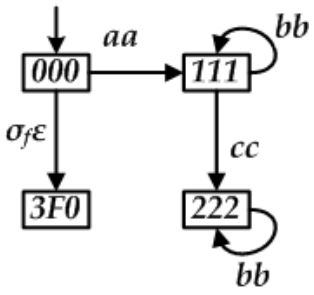


Figura 3.23: Autômato de Teste $V_{Q,c}$ para o caso centralizado

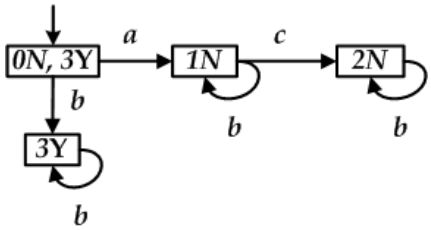


Figura 3.24: Diagnosticador de G, $Diag(G)$

$\Sigma_f = \{\sigma_f\}$. *Assim, pode-se obter o autômato* G_V *da figura 3.26 pelo método de MOREIRA* et al. *[24]. Esse autômato apresenta* 15 *estados e* 21 *transições. Note que, analisando o autômato* G_V *de acordo com o teorema 3.5, conclui-se que a linguagem gerada por* G *é não codiagnosticável em relação a* P_{o_i} *e* Σ_f*, uma vez que existe um ciclo de estados de falha* $cl = (2N4N4F, d_{R_1}, 1N4N4F, c, 2N4N4F)$*, no qual pelo menos um dos eventos pertencentes ao ciclo, pertence a* Σ*, que é o caso do evento c. Aplicando o método de QIU e KUMAR [15] é obtido um autômato de* 115 *estados e* 558 *transições que, devido ao tamanho, não é apresentado. Aqui fica claro que, embora as ordens de complexidade dos métodos proposto por MOREIRA* et al. *[24] e por QIU e KUMAR [15] sejam próximas, a diferença no tamanho dos verificadores pode ser muito grande. Nesse caso, o verificador de QIU e KUMAR [15] possui aproximadamente sete vezes mais estados e aproximadamente vinte e seis vezes mais transições em relação o verificador de MOREIRA* et al. *[24]. O autômato obtido pelo método de WANG* et al. *[16] para este exemplo também não é apresentado, uma vez que esse autômato possui* 192 *estados e* 465 *transições, o que resulta em aproximadamente doze vezes mais estados e aproximadamente vinte e duas vezes mais transições em relação o verificador de MOREIRA* et al. *[24].*

□

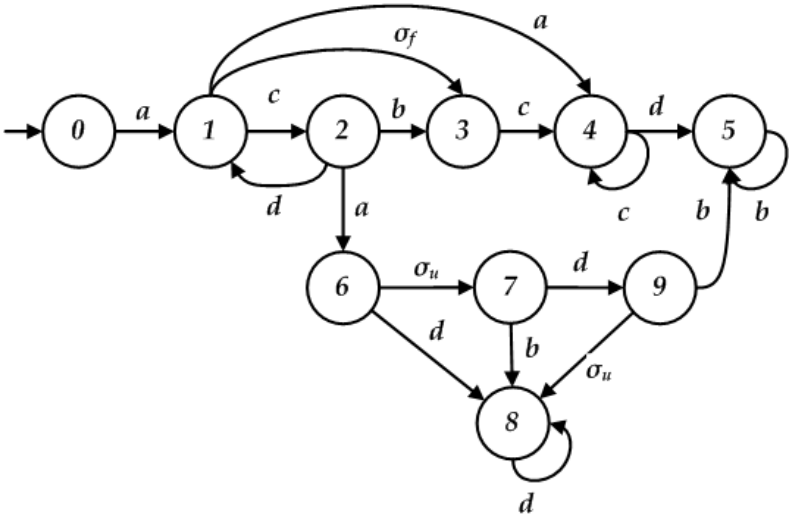


Figura 3.25: Autômato G referente ao exemplo 3.10.

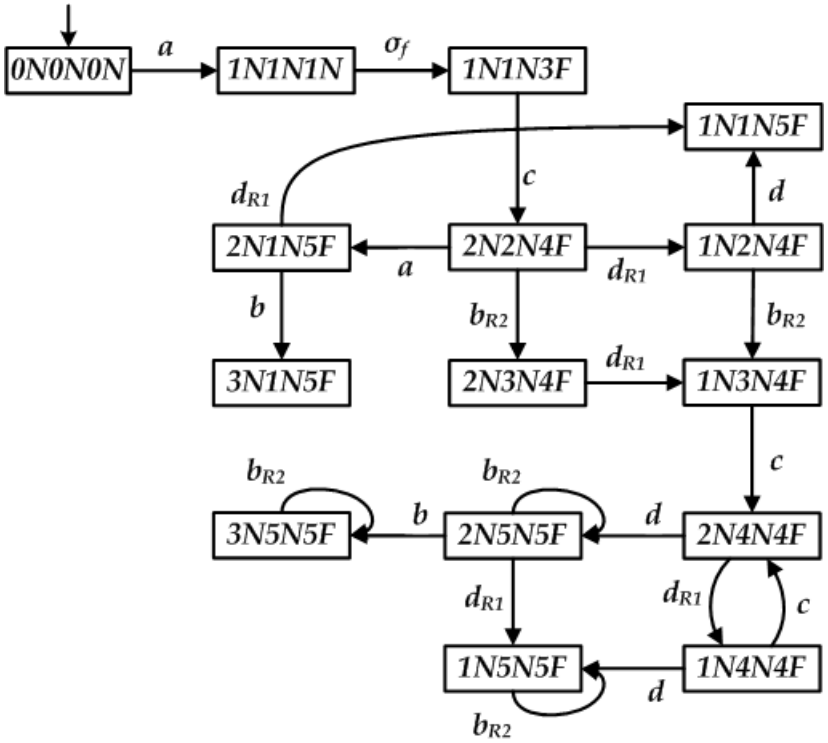


Figura 3.26: Autômato verificador para o caso descentralizado, G_V.

Exemplo 3.11 *O exemplo 3.10 ilustra o fato de que, embora as ordens de complexidade dos métodos proposto por MOREIRA* et al. *[24], QIU e KUMAR [15] e WANG* et al. *[16] sejam próximas, a diferença no tamanho dos verificadores pode ser muito grande. Essa diferença tende a crescer à medida que o tamanho do sistema cresce, como mostram as tabelas 3.4, 3.5 e 3.6. Essas tabelas mostram o número de estados e de transições de três autômatos distintos, G_1, G_2 e G_3, res-*

pectivamente, e dos autômatos verificadores da codiagnosticabilidade associados a esses autômatos. Nesse caso, são comparados os métodos propostos por MOREIRA et al. *[24],* QIU e KUMAR *[15] e* WANG et al. *[16], considerando os seguintes conjuntos de eventos:* $\Sigma_{o_1} = \{a, b, c, d, e, f, g, h, m\}$, $\Sigma_{o_2} = \{a, c, d, f, g, i, j, l, n\}$, $\Sigma_{uo} = \{\sigma_f, \sigma_u\}$, $\Sigma_f = \{\sigma_f\}$. *Devido ao grande número de estados e transições, tanto os autômatos* G_1, G_2 *e* G_3, *quanto o autômato verificador proposto por* MOREIRA et al. *[24],* G_V, *o autômato verificador proposto por* QIU e KUMAR *[15],* V_Q, *e o autômato verificador proposto por* WANG et al. *[16],* V_W, *não são apresentados.* □

Tabela 3.4: Análise comparativa de número de estados e transições dos verificadores para o autômato G_1.

Autômato	G_1	G_V	V_Q	V_W
No. de Estados	100	1840	2450	17317
No. de Transições	199	3511	7173	35898

Tabela 3.5: Análise comparativa de número de estados e transições dos verificadores para o autômato G_2.

Autômato	G_2	G_V	V_Q	V_W
No. de Estados	150	2498	3557	87528
No. de Transições	300	4481	9345	177781

Tabela 3.6: Análise comparativa de número de estados e transições dos verificadores para o autômato G_3.

Autômato	G_3	G_V	V_Q	V_W
No. de Estados	215	6413	7681	132500
No. de Transições	400	11055	18450	251468

3.7 Conclusão

Neste capítulo foram apresentados métodos para a verificação das propriedades de diagnosticabilidade em um SED. Essa verificação pode ser feita utilizando verificadores polinomiais existentes na literatura, como nos métodos propostos por JIANG *et al.* [13], YOO e LAFORTUNE [14], QIU e KUMAR [15] e WANG *et al.* [16]. Foi ainda proposto um novo verificador para realizar a verificação da diagnosticabilidade descentralizada de um SED, que também possui complexidade polinomial, porém de ordem inferior ao algoritmos já apresentados. Esse novo algoritmo não requer as hipóteses de vivacidade da linguagem gerada pelo sistema ou a não existência de ciclos de eventos não observáveis. O algoritmo pode também ser utilizado para verificar a diagnosticabilidade centralizada de SED. No próximo capítulo é mostrado como realizar a diagnose *on-line* de SED a partir do verificador proposto neste capítulo.

Capítulo 4

Diagnose *on-line* centralizada de SED

No capítulo anterior foram apresentados diversos métodos de verificação em tempo polinomial da diagnosticabilidade de sistemas a eventos discretos, além de ser proposto um novo método para a verificação da diagnosticabilidade de SED [24]. Neste capítulo é apresentado um algoritmo para a diagnose de falhas *on-line* de SED para o caso centralizado [26]. Esse algoritmo utiliza o autômato verificador proposto na seção 3.5 [24] e baseia-se na idéia de que somente as sequências de eventos observáveis estritamente necessárias para a diagnose de falhas devem ser percorridas pelo diagnosticador, ou seja, o processo de diagnose é interrompido caso a falha seja diagnosticada ou então se a sequência observada indicar que a falha não ocorreu e não pode mais ocorrer no sistema.

Para ilustrar uma situação na qual o processo de diagnose pode continuar desnecessariamente mesmo que a falha não tenha ocorrido e não possa mais ocorrer, considere o autômato G da figura 4.1, que foi primeiramente apresentado em [21]. Nesse autômato os conjuntos de eventos observáveis, não observáveis, e de falha são, respectivamente, $\Sigma_o = \{a, b\}$, $\Sigma_{uo} = \{\sigma_f, \sigma_u\}$ e $\Sigma_f = \{\sigma_f\}$. Nesse caso, se no estado inicial o evento a é observado, então é fácil ver que o estado 4 é alcançado e, uma vez que nenhum outro estado pode ser alcançado por uma sequência de eventos sem falha cujo único evento observável é a, então a falha ocorreu com certeza e o processo de diagnose pode ser interrompido. Por outro lado, se o evento b é observado, então o estado 1 é alcançado e a falha não somente não ocorreu como também

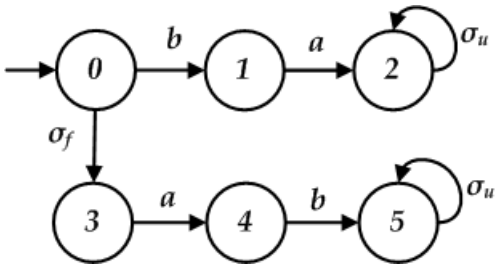


Figura 4.1: Autômato G proposto por CASSEZ *et al.* [21].

não pode mais ocorrer. Continuar o processo de diagnose neste caso é totalmente desnecessário levando a um gasto de energia com o uso de sensores e de esforço computacional para a leitura e processamento das informações fornecidas pelos sensores. Um exemplo real de sistemas nos quais essa economia é muito importante e significativa, são as redes de sensores sem fio [23].

4.1 Um novo diagnosticador para realizar o processo de diagnose *on-line*

Nesta seção, utilizando o verificador $G_{V,c}$ obtido a partir do algoritmo 3.5, um autômato diagnosticador não determinístico é apresentado. Nesse diagnosticador apenas as projeções das sequências estritamente necessárias para o processo de diagnose são representadas, possibilitando que a informação de que uma falha já ocorreu ou que uma falha não ocorreu e não irá mais ocorrer seja obtida.

Antes de apresentar o algoritmo para construção do diagnosticador e o algoritmo para a realização do processo de diagnose *on-line*, é preciso fazer algumas observações sobre o verificador proposto por MOREIRA *et al.* [24].

Observação 6 *Em relação ao algoritmo 3.5, é importante observar que uma outra forma de calcular o autômato G_N é eliminar as transições rotuladas por eventos de falha de G_l e tomar a parte acessível do autômato resultante. Isso mostra que todos os estados de G_N e de G_F podem ser associados aos estados de G_l, ou seja, $X_F \cup X_N = X_{G_l}$.*

Outro resultado importante é apresentado no teorema a seguir.

Teorema 4.1 *Seja G o autômato que modela um SED e sejam G_N, G_F e $G_{V,c}$, os autômatos de não falha, falha e verificador, respectivamente, obtidos seguindo os passos do algoritmo 3.5. Considere a partição $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ e defina a projeção $P : (\Sigma \cup \Sigma_R)^\star \rightarrow \Sigma_o^\star$. Então, uma sequência s_V pertence à linguagem gerada por $G_{V,c}$, i.e., $s_V \in \mathcal{L}(G_{V,c})$, se e somente se existem sequências $s_N \in \mathcal{L}(G_N)$ e $s_F \in \mathcal{L}(G_F)$ tais que $P(s_V) = P(s_N) = P(s_F)$.* □

Demonstração A demonstração pode ser facilmente obtida pela construção do autômato verificador $G_{V,c}$ e é, portanto, omitida. □

Note que todas as sequências de G que possuem um evento de falha pertencem à linguagem gerada por G_F, isto é, $\mathcal{L}(G_F) = \overline{L \setminus L_N}$. Assim, de acordo com o teorema 4.1, a principal característica do algoritmo 3.5 é gerar um autômato $G_{V,c}$ cujas sequências $s_V \in \mathcal{L}(G_{V,c})$ estão sempre associadas a sequências $s_N \in \mathcal{L}(G_N)$ e $s_F \in \mathcal{L}(G_F)$ que tenham a mesma projeção no conjunto de eventos observáveis.

Feitas essas considerações, é possível então apresentar o algoritmo contendo os passos necessários para a construção do diagnosticador, que é mostrado a seguir.

Algoritmo 4.1 *Seja G um autômato determinístico que modela o sistema e considere a partição $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. Considere também que a linguagem gerada por G é diagnosticável com relação à projeção P_o e Σ_f.*

- *Passo 1: Construa o verificador $G_{V,c} = (X_{V,c}, \Sigma_{R_1} \cup \Sigma, f_{V,c}, x_{0,V})$ de acordo com o algoritmo 3.5.*
- *Passo 2: Construa o autômato não determinístico $G_{V'} = (X_{V'}, \Sigma_o, \Gamma_{V'}, f_{V'}, x_{0,V'})$ da seguinte forma:*
  - *Passo 2.1: Defina $x_{0,V'} = UR(x_{0,V})$ e faça $X_{V'} = \{x_{0,V'}\}$ e $\hat{X}_{V'} = X_{V'}$.*
  - *Passo 2.2: Faça $\overline{X}_{V'} = \hat{X}_{V'}$ e $\hat{X}_{V'} = \emptyset$.*
  - *Passo 2.3: Para cada $B \in \overline{X}_{V'}$,*
    - *Passo 2.3.1: Defina*

$$\Gamma_{V'}(B) = \left(\bigcup_{x_{V,c} \in B} \Gamma_{V,c}(x_{V,c}) \right) \cap \Sigma_o.$$

* *Passo 2.3.2: Para cada* $\sigma \in \Gamma_{V'}(B)$

$$f_{V'}(B,\sigma) = \bigcup_{x_{V,c} \in B} \{UR(f_{V,c}(x_{V,c},\sigma))\}.$$

* *Passo 2.3.3:* $\hat{X}_{V'} \leftarrow \hat{X}_{V'} \cup \{f_{V'}(B,\sigma)\}$.

– *Passo 2.4:* $X_{V'} \leftarrow X_{V'} \cup \hat{X}_{V'}$.

– *Passo 2.5: Repita os passos* 2.2 *a* 2.4 *até que toda a parte acessível de* $G_{V'}$ *tenha sido construída.*

- *Passo 3: Defina a função* $S : X_N \times X_F \rightarrow 2^{X_{G_l}}$ *como:*

$$S[(x_N, x_F)] = \begin{cases} \{x_N\}, \text{ se } x_N = x_F \\ \{x_N, x_F\}, \text{ caso contrário} \end{cases}. \tag{4.1}$$

Construa o autômato diagnosticador $G_D = (X_D, \Sigma_o, f_D, x_{0,D})$*, em que cada estado* $x_{V'} \in X_{V'}$ *possui um estado correspondente* $x_D \in X_D$ *obtido fazendo*

$$x_D = \bigcup_{x_{V,c} \in x_{V'}} S(x_{V,c}), \tag{4.2}$$

o estado inicial é dado por

$$x_{0,D} = \bigcup_{x_{V,c} \in x_{V_0'}} S(x_{V,c})$$

e a função de transição de estados é dada por

$$f_D(x_D, \sigma) = f_{V'}(x_{V'}, \sigma), \forall \sigma \in \Sigma_o,$$

sendo x_D *o estado correspondente a* $x_{V'}$ *obtido a partir de (4.2).* □

Observação 7 *Note que* $f_{V'}$ *é uma função não determinística e, portanto, o diagnosticador* G_D *é um autômato não determinístico. É importante ressaltar também que como o autômato verificador* $G_{V,c}$ *possui no pior caso um número máximo de estados igual a* $2|X|^2$*, então* G_D *possui no máximo o mesmo número de estados. Além disso, pode ser verificado que um limitante superior para o número de transições de* G_D *é igual a* $4|X|^4 \times |\Sigma_o|$ *e, portanto, o diagnosticador* G_D *pode ser construído em tempo polinomial.* □

Observação 8 *Note que* $\mathcal{L}(G_{V'}) = P(\mathcal{L}(G_{V,c}))$ *e, como* $G_{V'}$ *e* G_D *possuem estados equivalentes e as mesmas transições, então* $\mathcal{L}(G_D) = P(\mathcal{L}(G_{V,c}))$. □

A seguir é apresentado o procedimento para a diagnose de falhas *on-line* utilizando o diagnosticador G_D obtido a partir do algoritmo 4.1.

Algoritmo 4.2

- *Passo 1: Defina* $x_{on} = x_{0,D}$.
- *Passo 2: Aguarde até que um evento observável* $\sigma \in \Sigma_o$ *ocorra.*
- *Passo 3: Após a observação do evento* $\sigma \in \Sigma_o$ *faça:* (i) *se* $\sigma \notin \Gamma_D(x_D), \forall x_D \in x_{on}$,

$$x_{on} = \begin{cases} F, \text{ se } (\exists x_D \in x_{on}, \text{ tal que } \exists x_{G_l} \in x_D, \\ \quad x_{G_l} = (x, F), \sigma \in \Gamma(x)) \\ N, \text{ se } (\exists x_D \in x_{on}, \text{ tal que } \exists x_{G_l} \in x_D, \\ \quad x_{G_l} = (x, N), \sigma \in \Gamma(x)) \end{cases}$$

  (ii) *Caso contrário:*

$$x_{on} \leftarrow \bigcup_{x_D \in x_{on}} \{f_D(x_D, \sigma)\} \tag{4.3}$$

- *Passo 4: Se* $x_{on} = N$ *então a falha não ocorreu e não pode mais ocorrer. Se* $x_{on} = F$ *a falha é identificada. Assim, se* $x_{on} \in \{N, F\}$*, interrompa o processo e sinalize* x_{on}*. Caso contrário, volte para o passo* 2. □

Note que em cada passo do processo de diagnose é necessário apenas armazenar a estimativa de estado x_{on} de G_D. Após a ocorrência de um evento observável $\sigma \in \Sigma_o$, que seja ativo para pelo menos um dos estados da estimativa x_{on}, uma nova estimativa x_{on} é calculada a partir da estimativa de estado anterior utilizando a equação (4.3). Como, de acordo com o teorema 4.1 para toda sequência s_V estão associadas sequências s_N e s_F tais que $P(s_V) = P(s_N) = P(s_F)$ e, de acordo com a observação 8, $\mathcal{L}(G_D) = P(\mathcal{L}(G_{V,c}))$, então associado a cada uma das sequências $s_D \in \mathcal{L}(G_D)$ existem sequências $s_N \in \mathcal{L}(G_N)$ e $s_F \in \mathcal{L}(G_F)$ tais que as projeções $P(s_N)$ e $P(s_F)$ são iguais a s_D. Uma vez que $s_N, s_F \in \Sigma^\star$, $P(s_N) = P_o(s_N)$ e $P(s_F) = P_o(s_F)$, o que implica que para toda sequência $s_D \in \mathcal{L}(G_D)$ tem-se que $s_D = P_o(s_N) = P_o(s_F)$. Portanto, neste caso, não é possível ter certeza da ocorrência de um evento de falha ou então afirmar que a falha não ocorreu e não pode mais ocorrer. Contudo, se um evento observável não ativo para todos os estados de $x_D \in x_{on}$ ocorre, isso significa que não existem sequências $s_N \in \mathcal{L}(G_N)$ e $s_F \in \mathcal{L}(G_F)$

cujas projeções são iguais à sequência observada. Logo, ou a sequência s gerada pela planta G pertence a $\mathcal{L}(G_N) \setminus \mathcal{L}(G_F)$ ou a $\mathcal{L}(G_F) \setminus \mathcal{L}(G_N)$. Se $s \in \mathcal{L}(G_N) \setminus \mathcal{L}(G_F)$, então para todas as sequências t na pós linguagem de L após s, L/s, tem-se que st pertence a $\mathcal{L}(G_N) \setminus \mathcal{L}(G_F)$ e, portanto, correspondem somente a caminhos cujos estados pertencem a G_l com a segunda coordenada igual a N, o que significa que a falha não ocorreu e não pode mais ocorrer. Por outro lado, se $s \in \mathcal{L}(G_F) \setminus \mathcal{L}(G_N)$, então todos os estados alcançados em G_l após a sequência s possuem a segunda coordenada igual a F indicando a ocorrência da falha.

O algoritmo 4.2 leva a um processo de diagnose de falhas em SED em que os critérios de parada são: (i) a falha ocorreu e não existe nenhuma sequência sem falha que tenha projeção P_o igual a da sequência gerada pelo sistema; (ii) a falha não ocorreu e não existe nenhuma sequência com falha que tenha projeção P_o igual a da sequência gerada pelo sistema. Portanto, a metodologia proposta neste trabalho leva a um processo de diagnose em que os eventos do sistema são observados somente enquanto há dúvida com relação à ocorrência da falha, permitindo que os sensores sejam desligados quando não há mais dúvida. Isso permite uma redução no consumo de energia para ativar os sensores e, principalmente, permite uma redução no esforço computacional para processar as informações enviadas pelos sensores.

É importante notar que a existência de um SED no qual um determinado estado pode ser alcançado sem a ocorrência de um evento de falha, sendo que desse estado em diante uma falha não poderá mais ocorrer, é bastante razoável. Um exemplo disso, são sistemas compostos por diversos ciclos de operação e cuja linguagem gerada seja diagnosticável. Assim, suponha que o sistema entra em um determinado ciclo de operação no qual uma falha é possível ocorrer uma falha, executa os eventos desse ciclo por um tempo e em seguida executa um evento que leva o sistema a mudar de ciclo de operação, não sendo possível retornar ao primeiro ciclo. Se o sistema executou o primeiro ciclo sem a ocorrência de um evento de falha, então, nesse caso é possível afirmar que uma falha não ocorreu e nem poderá mais ocorrer. O diagnosticador proposto neste trabalho é bastante útil para SED nos quais seja possível ter a certeza que uma falha não ocorrerá mais. Para os demais sistemas, esse diagnosticador proposto contém a mesma informação contida no diagnosticador proposto por SAMPATH *et al.* [10].

A seguir, um exemplo é usado para ilustrar a construção do diagnosticador e o processo de diagnose apresentados nos algoritmos 4.1 e 4.2, respectivamente.

Exemplo 4.1 Considere o sistema G apresentado na figura 4.2 em que $\Sigma_o = \{a, b, c, d\}$, $\Sigma_{uo} = \{\sigma_u, \sigma_f\}$ e $\Sigma_f = \{\sigma_f\}$. Para a obtenção do diagnosticador G_D, o primeiro passo é obter o verificador $G_{V,c}$ de acordo com o algoritmo 3.5. Para tanto, primeiro são construídos os autômatos G_l, G_N e G_F, mostrados nas figuras 4.3, 4.4 e 4.5, respectivamente. Uma vez obtidos esses autômatos, o verificador $G_{V,c}$, apresentado na figura 4.6, é obtido fazendo-se $G_{V,c} = G_{N,1} \| G_F$, em que $G_{N,1}$ é semelhante ao autômato G_N, substituindo-se o rótulo da transição não observável σ_u por $\sigma_{u_{R_1}}$.

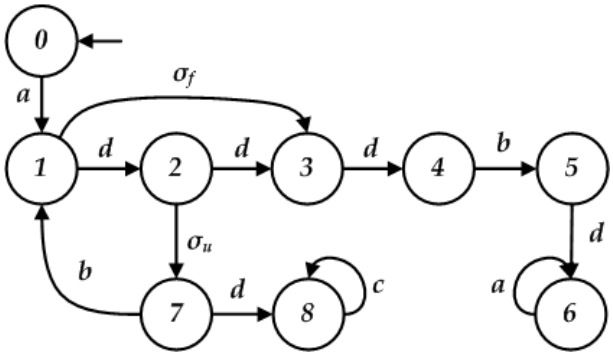


Figura 4.2: Autômato G referente ao exemplo 4.1.

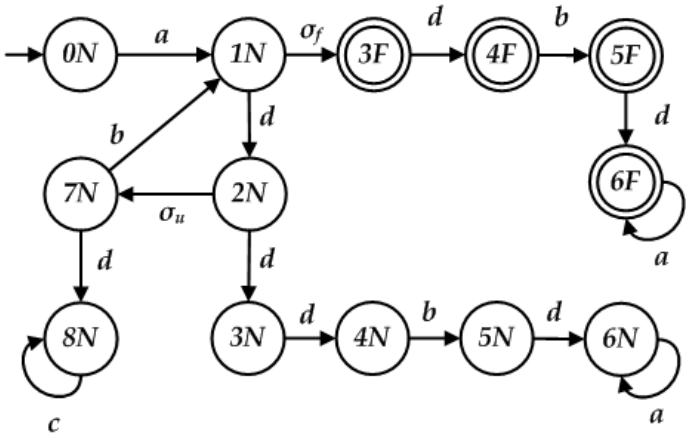


Figura 4.3: Autômato G_l.

Uma vez obtido o verificador $G_{V,c}$, o autômato não determinístico $G_{V'}$ é calculado seguindo o passo 2 do algoritmo 4.1. Na figura 4.7 é mostrado o autômato $G_{V'}$. Note que os estados de $G_{V'}$ são conjuntos formados por estados de $G_{V,c}$ que são agrupados

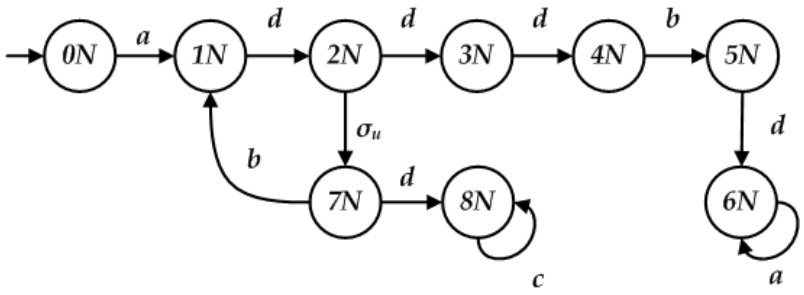


Figura 4.4: Autômato de não falha G_N.

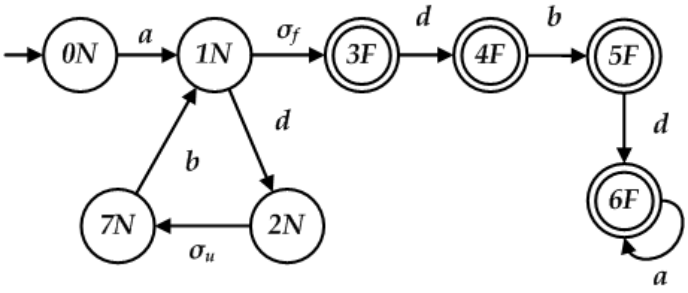


Figura 4.5: Autômato de falha G_F.

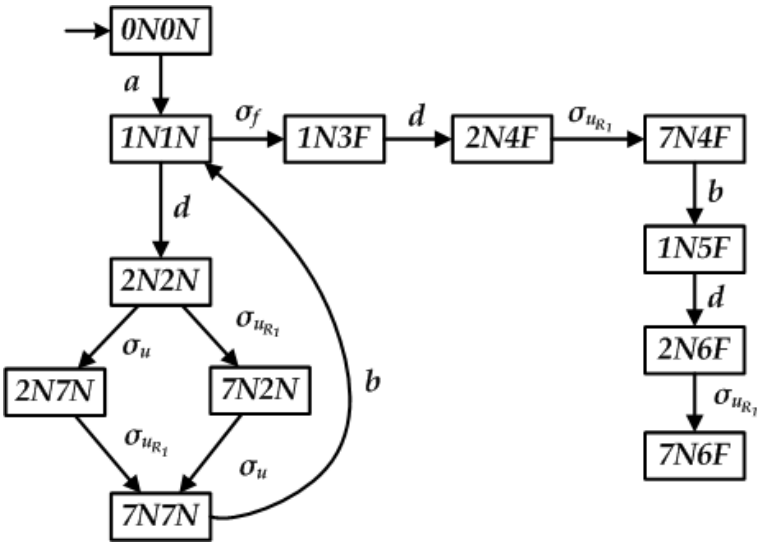


Figura 4.6: Autômato verificador $G_{V,c}$.

de modo que todas as transições rotuladas por eventos não observáveis em $G_{V,c}$ não apareçam em $G_{V'}$. No passo 3 do algoritmo 4.1, o autômato G_D, mostrado na figura 4.8, é construído renomeando os estados de $G_{V'}$ visando manter em cada estado de G_D a estimativa dos estados alcançados em G_l após a observação de uma sequência

de eventos.

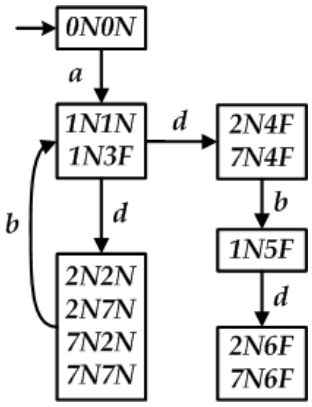


Figura 4.7: Autômato $G_{V'}$.

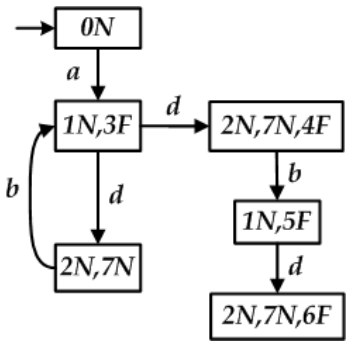


Figura 4.8: Autômato diagnosticador G_D.

O processo de diagnose *on-line* de falhas pode ser feito utilizando-se o diagnosticador G_D como descrito no algoritmo 4.2. Para entender como o algoritmo funciona, vamos considerar três exemplos de sequências observadas pelo sistema, $s_{D,1}$, $s_{D,2}$ e $s_{D,3}$. No primeiro exemplo suponha que a sequência que o sistema observa seja $s_{D,1} = add$. No passo 1 do algoritmo 4.2, o estado x_{on} é feito igual ao estado inicial de G_D, $x_{0,D}$, ou seja, $x_{on} = \{0N\}$ e o sistema aguarda a ocorrência de um evento observável $\sigma \in \Sigma_o$. Após o evento a ser observado pelo sistema, é feita a verificação do passo 3 para a atualização do estado x_{on}. Como, $a \in \Gamma_D(\{0N\})$, o estado x_{on} é atualizado utilizando-se (4.3), levando a $x_{on} = \{1N, 3F\}$ e o algoritmo volta ao passo 2 em que é aguardada a ocorrência de um novo evento $\sigma \in \Sigma_o$. Após a ocorrência do evento d, o algoritmo alcança o passo 3 e como $d \in \Gamma_D(\{1N, 3F\})$, o novo estado $x_{on} = \{\{2N, 7N\}, \{2N, 7N, 4F\}\}$ é obtido. Por fim, na segunda

ocorrência de d, como pode ser visto na figura 4.8, d não é um evento ativo, *i.e.*, $d \notin \Gamma_D(\{2N, 7N\}) \cup \Gamma_D(\{2N, 7N, 4F\})$, o que implica que ou a falha poderá ser diagnosticada ou a falha não ocorreu e não ocorrerá mais. Como $d \in \Gamma_{G_l}(2N)$, então $x_{on} = \{N\}$ indicando que a falha não ocorreu e não pode mais ocorrer e o processo é interrompido.

O fato de que a falha não pode mais ocorrer pode ser facilmente comprovado analisando o autômato G. Nesse autômato, existem apenas dois caminhos que levam à observação de $s_{D,1}$: $c_1 = (0, a, 1, d, 2, d, 3)$ e $c_2 = (0, a, 1, d, 2, \sigma_u, 7, d, 8)$. Nesse caso, tanto c_1 quanto c_2 não possuem qualquer evento de falha e ambos levam a estados a partir dos quais todos os caminhos existentes também não possuem um evento de falha. O diagnosticador ainda poderia observar uma sequência $s_{D,2} = adbdbdbdd$. Nesse caso, fazendo a mesma análise realizada para a observa,ão de $s_{D,1}$, tem-se que $s_{D,2}$ também leva à afirmação de que a falha não ocorreu e não pode mais ocorrer. Porém, note que a observação de $s_{D,2}$ implica que o sistema executou por um determinado tempo o ciclo de operação $cl = (1, d, 2, \sigma_u, 7, b, 1)$, porém sem executar o evento de falha σ_f, uma vez que a sequência $s_{D,2} = adbdbdbdd$ não pode ser observada se a falha ocorrer. Como a observação de $s_{D,2}$ leva o sistema aos estados 3 ou 8, a partir dos quais todos os caminhos existentes também não possuem um evento de falha, então é possível afirmar que a falha não ocorreu e não pode mais ocorrer.

Suponha agora que a sequência de eventos observada seja $s_{D,3} = adbda$. Neste caso, seguindo os passos do algoritmo 4.2, pode-se observar que o prefixo de $s_{D,3}$, $adbd$, pode ser percorrido em G_D alcançando o estado $x_{on} = \{\{2N, 7N\}, \{2N, 7N, 6F\}\}$. Contudo, a não é um evento ativo de nenhum dos estados $x_D \in x_{on}$, o que implica que ou a falha ocorreu, ou a falha não ocorreu e não ocorrerá mais. Neste caso é fácil ver que $a \in \Gamma_{G_l}(6F)$ o que leva a $x_{on} = \{F\}$ indicando que a falha ocorreu e o processo é interrompido. □

4.2 Conclusão

Neste capítulo foi apresentado um novo algoritmo para a diagnose de falhas *on-line* de sistemas a eventos discretos para o caso centralizado. Esse método é baseado na

construção do verificador apresentado em [24], que tem como principal característica representar somente as sequências de falha e de não falha que possuem a mesma projeção. Essa característica permite a construção em tempo polinomial de um diagnosticador não determinístico que permite inferir se uma falha ocorreu ou se nenhuma falha ocorreu e não pode mais ocorrer. Assim, o método para a diagnose *on-line* proposto neste capítulo permite o desligamento de sensores sempre que as informações fornecidas por eles forem inúteis para o processo de diagnose.

Capítulo 5

Conclusões e trabalhos futuros

Este trabalho trata do problema de diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos. Esse problema é abordado considerando tanto uma arquitetura centralizada, quanto uma arquitetura descentralizada, e propõe soluções para a verificação da propriedade de diagnosticabilidade de um SED e para a diagnose *on-line* centralizada.

Neste trabalho um novo algoritmo em tempo polinomial para a verificação da diagnosticabilidade descentralizada de um sistema a eventos discretos é proposto. Esse algoritmo possui menor complexidade computacional do que outros algoritmos propostos na literatura, uma vez que esse método gera um verificador que representa apenas a sequências do sistema que são estritamente necessárias para a verificação da codiagnosticabilidade. O algoritmo não requer que as hipóteses de vivacidade da linguagem gerada pelo sistema e de a não existência de ciclos de eventos não observáveis sejam verdadeiras, e pode também ser utilizado para verificar a diagnosticabilidade centralizada de SED.

Além disso, um novo algoritmo para a diagnose de falhas *on-line* de sistemas a eventos discretos é proposto. Esse método é baseado na construção do verificador proposto neste trabalho, que tem como principal característica representar somente as sequências de falha e de não falha que possuem a mesma projeção. Essa característica permite a construção em tempo polinomial de um diagnosticador não determinístico que permite inferir se uma falha ocorreu ou se nenhuma falha ocorreu e não pode mais ocorrer. Assim, o método para a diagnose *on-line* proposto neste trabalho permite o desligamento de sensores sempre que as informações for-

necidas por eles forem inúteis para o processo de diagnose. Esse diagnosticador se mostra eficiente quando utilizado em sistemas cuja estrutura possibilite alcançar um estado através de uma sequência que não possua um evento de falha, e que, a partir desse estado, todos os demais estados que possam ser alcançados também não possuam um evento de falha sendo um evento ativo. Um exemplo de sistemas com essas características são os sistemas que possuem vários ciclos de operação. Caso contrário, o diagnosticador obtido pelo método proposto neste trabalho possuirá a mesma informação contida no diagnosticador proposto por SAMPATH *et al.* [10].

Um possível trabalho futuro é utilizar o verificador proposto no capítulo 3 para estudar a codiagnosticabilidade robusta, que consiste na propriedade de um sistema continuar sendo codiagnosticável, mesmo após ocorrer uma falha permanente se sensor. Nesse caso, o conjunto de eventos observáveis do sistema Σ_o passa a ser um conjunto menor $\Sigma'_o \subset \Sigma_o$, e mesmo com menos eventos observáveis, o sistema ainda assim é codiagnosticável, porém em relação a esse novo conjunto eventos observáveis e ao conjunto de eventos falhas. Outra linha de pesquisa é estender os resultados obtidos neste trabalho para SED modelados por redes de Petri.

Referências Bibliográficas

[1] CORONA, D., GIUA, A., SEATZU, C. "Marking estimation of petri nets with silent transitions", *Proceedings of the 43rd IEEE Conf. on Decision and Control*, pp. 966–971, Dec 2004.

[2] GIUA, A., SEATZU, C. "Fault detection for discrete event systems using petri nets with unobservable transitions", *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 6323–6328, Dec 2005.

[3] RAMIREZ-TREVIÑO, A., RUIZ-BELTRÁN, E., LÓPEZ-MELLADO, E. "Online fault diagnosis of discrete event systems. A Petri net-based approach", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 1, pp. 31–39, 2007.

[4] LEFEBVRE, D., DELHERM, C. "Diagnosis of des with petri net models", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 1, pp. 114–118, 2007.

[5] BASILE, F., CHIACCHIO, P., DE TOMMASI, G. "An efficient approach for online diagnosis of discrete event systems", *IEEE Transactions on Automatic Control*, v. 54, n. 4, pp. 748–759, 2009.

[6] WILLSKY, A. "A survey of design methods for failure detection in dynamic systems", *Automatica*, v. 12, n. 6, pp. 601–611, 1976.

[7] VISWANADHAM, N., JOHNSON, T. "Fault detection and diagnosis of automated manufacturing systems", *Proceedings of the IEEE Conference on Decision and Control*, pp. 2301–2306, 1988.

[8] FRANK, P. "Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy. A survey and some new results", *Automatica*, v. 26, n. 3, pp. 459–474, 1990.

[9] LIN, F. "Diagnosability of discrete event systems and its applications", *Discrete Event Dynamic Systems: Theory and Applications*, v. 4, n. 2, pp. 197–212, 1994.

[10] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. "Diagnosability of discrete-event systems", *IEEE Transactions on Automatic Control*, v. 40, n. 9, pp. 1555–1575, 1995.

[11] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. "Failure diagnosis using discrete-event models", *IEEE Transactions on Control Systems Technology*, v. 4, n. 2, pp. 105–124, Mar 1996.

[12] DEBOUK, R., LAFORTUNE, S., TENEKETZIS, D. "Coordinated decentralized protocols for failure diagnosis of discrete event systems", *Discrete Event Dynamic Systems: Theory and Applications*, v. 10, n. 1, pp. 33–86, 2000.

[13] JIANG, S., HUANG, Z., CHANDRA, V., et al. "A polynomial algorithm for testing diagnosability of discrete-event systems", *IEEE Transactions on Automatic Control*, v. 46, n. 8, pp. 1318–1321, Aug 2001.

[14] YOO, T.-S., LAFORTUNE, S. "Polynomial-time verification of diagnosability of partially observed discrete-event systems", *IEEE Transactions on Automatic Control*, v. 47, n. 9, pp. 1491–1495, Sep 2002.

[15] QIU, W., KUMAR, R. "Decentralized failure diagnosis of discrete event systems", *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, v. 36, n. 2, pp. 384–395, 2006.

[16] WANG, Y., YOO, T.-S., LAFORTUNE, S. "Diagnosis of discrete event systems using decentralized architectures", *Discrete Event Dynamic Systems: Theory and Applications*, v. 17, n. 2, pp. 233–263, 2007.

[17] BASILIO, J. C., LAFORTUNE, S. "Robust codiagnosability of discrete event systems", *American Control Conference*, pp. 2202–2209, 2009.

[18] DEBOUK, R., LAFORTUNE, S., TENEKETZIS, D. "On an optimization problem in sensor selection for failure diagnosis", *Proceedings of the 38th IEEE Conference on Decision and Control*, pp. 4990–4995, 1999.

[19] YOO, T.-S., LAFORTUNE, S. "On the computational complexity of some problems arising in partially-observed discrete-event systems", *American Control Conference, 2001. Proceedings of the 2001*, pp. 307–312, 2001.

[20] JIANG, S., KUMAR, R., GARCIA, H. E. "Optimal sensor selection for discrete-event systems with partial observation", *IEEE Transactions on Automatic Control*, v. 48, n. 3, pp. 369–381, 2003.

[21] CASSEZ, F., TRIPAKIS, S., ALTISEN, K. "Sensor Minimization Problems with Static or Dynamic Observers for Fault Diagnosis", *7th International Conference on Application of Concurrency to System Design, 2007*, pp. 90–99, Jul 2007.

[22] CASSEZ, F., TRIPAKIS, S. "Fault diagnosis with dynamic observers", *9th International Workshop on Discrete Event Systems, 2008*, pp. 212–217, May 2008.

[23] AKYILDIZ, I. F., SU, W., SANKARASUBRAMANIAM, Y., et al. "Wireless sensor networks: a survey", *Computer Networks*, v. 38, n. 4, pp. 393 – 422, 2002.

[24] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C. "Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems", *Accepted for publication, IEEE Transactions on Automatic Control*, 2010.

[25] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C. "Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems", *Proceedings of 2010 American Control Conference*, pp. 3353–3358, 2010.

[26] JESUS, T. C., MOREIRA, M. V., BASILIO, J. C. "Diagnóstico de falhas em tempo real de sistemas a eventos discretos descritos por autômatos

finitos", *Anais do 18º Congresso Brasileiro de Automática*, pp. 712–719, 2010.

[27] BASILIO, J. C., CARVALHO, L. K., MOREIRA, M. V. "Diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos", *Sba Controle e Automação*, v. 21, pp. 510 – 533, Out 2010.

[28] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2 ed. Secaucus, NJ, Springer, 2008.

[29] HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D. *Introduction to automata theory, languages, and computation*. 3 ed. Boston, Addison Wesley, 2006.

[30] RAMADGE, P., WONHAM, W. "The control of discrete event systems", *Proceedings of the IEEE*, v. 77, n. 1, pp. 81–98, Jan 1989.

[31] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L. *Introduction to Algorithms*. 3 ed. Cambridge, Mass., MIT Press, 2001.