OBTENÇÃO DE BASES MÍNIMAS PARA O DIAGNÓSTICO DE FALHAS DE SISTEMAS A EVENTOS DISCRETOS UTILIZANDO VERIFICADORES

Leonardo Pascoal Motta Santoro

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
Setembro de 2013

OBTENÇÃO DE BASES MÍNIMAS PARA O DIAGNÓSTICO DE FALHAS DE SISTEMAS A EVENTOS DISCRETOS UTILIZANDO VERIFICADORES

Leonardo Pascoal Motta Santoro

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. João Carlos dos Santos Basilio, Ph.D.

Prof. Antonio Eduardo Carrilho da Cunha, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2013

Santoro, Leonardo Pascoal Motta

Obtenção de bases mínimas para o diagnóstico de falhas de sistemas a eventos discretos utilizando verificadores/Leonardo Pascoal Motta Santoro. – Rio de Janeiro: UFRJ/COPPE, 2013.

XI, 109 p.: il.; 29, 7cm.

Orientador: Marcos Vicente de Brito Moreira

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2013.

Referências Bibliográficas: p. 106 – 109.

1. Diagnóstico de falhas. 2. Bases mínimas para diagnóstico. 3. Verificador centralizado. I. Moreira, Marcos Vicente de Brito. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Agradecimentos

Agradeço aos meus pais, Cristina Maria Motta Santoro e Pascoal Santoro, por toda educação, amor, e também, pelos valores morais ensinados.

Agradeço à minha irmã, Marcelle Motta Santoro, pela dedicação e esforços prestados a família e pelo exemplo que sempre foi para mim.

Agradeço aos meus amigos, Eduardo Lopes Gonçalves Filho, Luís Filipe Azevedo Carvalho e Rodrigo Cochrane Esteves pelo companheirismo e incentivo dado ao longo desses anos.

Agradeço aos meus amigos e companheiros de mestrado, Lucas Shigueoka, Bernardo Bouzan e Elly Bouzan pela ajuda e incentivos dados nesse período.

Agradeço à minha namorada e futura esposa, Marina Rangel Margem, pela compreensão, incentivo, carinho e amor dados nesse período.

Agradeço ao professor João Carlos dos Santos Basilio pelos ensinamentos e incentivos dados, motivos pelos quais escolhi essa área de estudo para minha tese de mestrado.

Agradeço também ao meu professor e orientador Marcos Vicente de Brito Moreira pelos ensinamentos, compreensão, paciência e esforço para que fosse possível levar esse estudo até o fim.

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

OBTENÇÃO DE BASES MÍNIMAS PARA O DIAGNÓSTICO DE FALHAS DE SISTEMAS A EVENTOS DISCRETOS UTILIZANDO VERIFICADORES

Leonardo Pascoal Motta Santoro

Setembro/2013

Orientador: Marcos Vicente de Brito Moreira

Programa: Engenharia Elétrica

A verificação da diagnosticabilidade de um sistema a eventos discretos (SED) pode ser realizada utilizando-se autômatos verificadores. Verificadores são autômatos determinísticos cujos estados possuem rótulos indicando sobre a ocorrência ou não dos eventos de falha. Uma vez verificada a diagnosticabilidade da linguagem de um sistema com relação a um conjunto de eventos observáveis, uma outra questão pode ser levantada: dado que o sistema é diagnosticável considerando-se o conjunto de eventos observáveis, seria possível que o sistema permanecesse diagnosticável para um subconjunto desse conjunto? Caso a resposta a essa questão seja positiva para um determinado sistema, é possível reduzir o número de sensores utilizados no diagnóstico, diminuindo também o custo financeiro da planta em questão. Outra possibilidade seria aproveitar a redundância de alguns desses sensores para tornar o diagnóstico mais robusto e confiável diante de uma situação de falha dos mesmos. Neste trabalho são propostos dois algoritmos que retornam, de maneira sistemática, todos os subconjuntos de eventos essenciais para o diagnóstico de falhas de um sistema (bases mínimas para o diagnóstico), sendo que um deles possui menor complexidade computacional que os demais métodos existentes atualmente na literatura.

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

OBTAINING MINIMUM BASES FOR FAULT DIAGNOSIS OF DISCRETE-EVENT SYSTEMS USING VERIFIERS

Leonardo Pascoal Motta Santoro

September/2013

Advisor: Marcos Vicente de Brito Moreira

Department: Electrical Engineering

The verification of diagnosability of discrete-event systems (DES) could be done by using verifiers automata. Verifiers are deterministic automata whose states have labels that inform about the occurrence or not of failure events. Once the language of a system is diagnosable with respect to a set of observable events, another question could be posed: since the system is diagnosable considering the set of observable events, would it be possible to ensure the system diagnosability using a subset of this set? If the answer for this question is positive for a given system, then it is possible to reduce the number of sensors used in the diagnosis, therefore reducing the cost of the system. Another possibility could be to use the redundancy of some of these sensors in order to obtain a more reliable and robust diagnosis. In this work two algorithms are proposed to find, in a systematic way, all essential events subsets (minimum bases for diagnosability). One of them has lower computational complexity than other methods proposed in the literature.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Um sistema a eventos discretos (SED) é um sistema no qual o conjunto de estados é discreto e sua evolução não é norteada pelo tempo, mas sim pela ocorrência assíncrona de eventos. Em muitas situações, a ocorrência de alguns desses eventos pode fazer com que o sistema se comporte de maneira indesejável. Nesse caso, diz-se que ocorreu uma falha e o evento que levou o sistema a esse comportamento é dito ser um evento de falha. Então, de forma a assegurar o correto funcionamento de um sistema, diversos estudos têm sido realizados nos últimos anos sobre diagnóstico de falhas em sistemas a eventos discretos (SEDs) [1–14]. Outra razão para esse interesse se deve ao fato de modelos a eventos discretos poderem ser aplicados não só a sistemas cuja essência é de natureza discreta, onde os modelos discretos serão os mais apropriados, tais como sistemas de computação, redes de comunicação e de manufatura, mas também a diversos outros sistemas dinâmicos de variáveis contínuas, uma vez que esses podem ser modelados como SEDs dependendo do grau de abstração desejado.

Os estudos sobre diagnóstico de falhas em SEDs são fundamentados em dois pontos principais: (i) a falha que se deseja diagnosticar é um evento não observável, ou seja, um evento cuja ocorrência não pode ser detectada e comunicada a um diagnosticador; (ii) a ocorrência de uma falha altera o comportamento do sistema, contudo, não necessariamente leva o sistema a uma parada imediata. Dessa forma, o desafio no estudo de diagnóstico de falhas de um sistema consiste em inferir, após a ocorrência de um número finito de eventos, a ocorrência de uma falha utilizando somente eventos observáveis. Para tal, deve-se, primeiramente, construir um modelo a eventos discretos que represente tanto o comportamento normal quanto o comportamento do sistema após a ocorrência da falha.

Embora os autômatos sejam os modelos mais utilizados atualmente [2–14], redes de Petri [15–20] também podem ser utilizadas para esse fim, e diversos trabalhos têm sido apresentados para o diagnóstico de falhas utilizando propriedades das redes de Petri. O problema na abordagem via redes de Petri é que esses métodos não atendem

a todas as classes de redes [21] e uma generalização precisa ainda ser alcançada. Neste trabalho, apenas sistemas modelados por autômatos serão considerados. Uma vez obtido o modelo do sistema, deve-se definir um conjunto de regras a serem seguidas para que o diagnóstico de falhas seja realizado.

Inicialmente, o problema de diagnóstico de falhas foi abordado no contexto de SEDs por LIN [22], que introduziu o conceito da capacidade de se diagnosticar a ocorrência de uma falha em um sistema. Em seguida, SAMPATH *et al.* [3] apresentaram condições necessárias e suficientes para o diagnóstico de falhas de SEDs, assim como a construção de um autômato diagnosticador que permite, além de verificar se o sistema é diagnosticável, realizar o diagnóstico de falhas em tempo real.

O diagnosticador proposto por SAMPATH *et al.* [3] possui a premissa de que todos os eventos observáveis do sistema estejam acessíveis em um determinado ponto. Essa abordagem é conhecida na literatura como diagnóstico de falhas centralizado. Contudo, na prática, devido à natureza descentralizada de alguns sistemas e grandes diferenças nas distâncias entre o diagnosticador e os dispositivos que registram a ocorrência de eventos, nem sempre é possível utilizar a abordagem de diagnóstico centralizado proposto por SAMPATH *et al.* [3]. Para contornar essas dificuldades, DEBOUK *et al.* [6] propuseram uma arquitetura descentralizada com coordenação, denominada codiagnose, que consiste em módulos locais capazes de observar a ocorrência de parte dos eventos observáveis do sistema. Esses módulos locais se comunicam com um coordenador, que é responsável pelo diagnóstico das falhas que venham a ocorrer no sistema. Essa arquitetura proposta por DEBOUK *et al.* [6] nos leva ao conceito de diagnóstico descentralizado. Nesta dissertação, apenas diagnósticos centralizados serão abordados.

O diagnosticador proposto por SAMPATH *et al.* [3], apresenta a deficiência de ter complexidade computacional exponencial para o cálculo do espaço de estados do diagnosticador em relação à cardinalidade do espaço de estados do autômato cuja linguagem gerada se deseja diagnosticar. Para contornar esse problema, JIANG *et al.* [23] e YOO e LAFORTUNE [24] propuseram uma nova forma de se verificar a diagnosticabilidade de SEDs baseado na construção de autômatos não determinísticos denominados verificadores, cujos espaços de estados possuem crescimento polinomial com relação a cardinalidade do espaço de estados do modelo do sistema. QIU e KUMAR [11] e WANG *et al.* [25] estenderam esses verificadores para a verificação da codiagnosticabilidade, levando aos chamados verificadores descentralizados. Mais recentemente, um novo algoritmo para obtenção de autômatos verificadores foi desenvolvido por MOREIRA *et al.* [26]. O verificador proposto em [26], além de ser determinístico e apropriado tanto para a verificação da diagnosticabilidade no caso centralizado quanto para a verificação da codiagnosticabilidade, possui a vantagem

de exibir apenas as sequências normais e de falha que possuem a mesma projeção. Devido a esse fator, o algoritmo para obtenção de verificadores desenvolvido em [26] é, atualmente, o de menor complexidade computacional existente na literatura e, portanto, será o método utilizado nesta dissertação.

Os trabalhos sobre diagnósticos de falhas em sistemas a eventos discretos apontam, atualmente, no sentido de estudar e elevar a robustez e confiabilidade das plantas em operação. O contínuo aumento nas exigências de desempenho, mesmo com variações ambientais, e sistemas com modelos cada vez mais complexos vem estimulando a publicação de diversas pesquisas sobre o tema. O conceito de diagnosticabilidade robusta em SEDs foi introduzido por BASILIO E LAFORTUNE [27] no contexto de diagnosticabilidade descentralizada, supondo que a comunicação entre um módulo e o coordenador não é confiável. Outra definição de diagnosticabilidade robusta foi proposta por TAKAI [28], na qual o sistema é descrito por um conjunto de possíveis modelos que possuem o mesmo conjunto de eventos observáveis.

Dentro do contexto de confiabilidade e robustez, algumas questões podem ser realizadas: sendo uma falha diagnosticada em tempo finito a partir de um conjunto de eventos observáveis, seria possível realizar o mesmo diagnóstico utilizando apenas parte desse conjunto? Caso um ou mais eventos pertencentes ao conjunto de eventos observáveis deixem de ser observados, será possível permanecer realizando corretamente o diagnóstico dessa falha? As respostas a essas questões estão diretamente relacionadas a fatores econômicos, de segurança e de robustez do sistema. Observe que, caso a falha possa ser diagnosticada com um número menor de eventos, é possível utilizar menos sensores, o que resulta em um menor tempo de processamento, dado que o número de sinais de entrada a serem processados é menor, e também em uma redução do custo total dos instrumentos utilizados na planta em questão. Contudo, caso desejado, todos os sensores podem ser mantidos, mas, nesse caso, existirá redundância entre alguns deles que pode ser aproveitada no sentido de tornar o diagnóstico mais robusto, oferecendo também a possibilidade de não interromper o funcionamento do sistema caso um dos sensores redundantes necessite de manutenção.

As questões levantadas acima podem ser interpretadas da seguinte forma: dado que uma falha pode ser diagnosticada considerando um conjunto de eventos observáveis, quais os subconjuntos de menor cardinalidade mantêm a linguagem do sistema diagnosticável? Em [12], [29], [30] e [31] o problema de se encontrar os subconjuntos do conjunto de eventos observáveis que permitam o diagnóstico de falhas é considerado. DEBOUK *et al.* [30] e JIANG *et al.* [12] abordaram esse problema no sentido de obter o conjunto de sensores que minimiza uma função custo mantendo uma certa propriedade do sistema, como por exemplo, a diagnosticabilidade da linguagem gerada pelo mesmo. Em TRAVÉ-MASSUYÈS *et al.* [31],

o objetivo é encontrar um conjunto de sensores redundantes que somados ao conjunto de sensores já existentes levem a um grau desejado de diagnosticabilidade e de discriminabilidade, que é a capacidade de distinguir entre duas falhas distintas. O trabalho desenvolvido por BASILIO *et al.* [29] tem por objetivo explorar a estrutura do sistema para, utilizando autômatos diagnosticadores, obter todos os conjuntos de eventos observáveis que mantêm inalterada a diagnosticabilidade da linguagem do sistema (bases para o diagnóstico). A busca exaustiva, que consiste em testar, para cada subconjunto possível de eventos observáveis se a linguagem é ou não diagnosticável, é evitada. Em [29], além do conceito de base para o diagnóstico, é introduzido também o conceito de bases mínimas para o diagnóstico, que consiste em conjuntos de eventos que mantêm inalterada a diagnosticabilidade da linguagem do sistema mas, caso qualquer um dos eventos pertencentes a esses conjuntos não seja observável, a linguagem torna-se não diagnosticável.

Uma vez obtidas as bases mínimas para o diagnóstico, é possível utilizá-las na construção de um diagnosticador robusto. Para tal, deve-se construir, para cada uma das bases mínimas para o diagnóstico, um diagnosticador que considere a observação de seus eventos. Então, a falha será reconhecida quando um dos diagnosticadores apontar a ocorrência do evento de falha. Observe que a obtenção de um diagnosticador robusto envolve a construção de diagnosticadores que não observam todos os eventos do conjunto de eventos observáveis. Contudo, note que esse problema é diferente do problema de codiagnose, uma vez que cada diagnosticador é obtido considerando a observação dos eventos essenciais para o diagnóstico da falha, e não devido à natureza distribuída do sistema. A obtenção de diagnosticadores robustos foi abordada, recentemente, por CARVALHO *et al.* [32], introduzindo o conceito de diagnosticadores robustos generalizados.

O problema de se obter as bases mínimas de um sistema é classificado na literatura como *NP-completo* [8], [33] e [34]. Conforme será visto nos próximos capítulos, nenhuma solução polinomial é conhecida para um problema *NP-completo*. Em [29], foi proposto um algoritmo, baseado em autômatos diagnosticadores, em que, a partir das informações e propriedades dos sistemas, é possível obter, de forma sistemática, todos os subconjuntos formado pelos sensores que mantêm a propriedade de diagnosticabilidade do sistema. Contudo, a complexidade computacional do algoritmo proposto é, no pior caso, pior que o uso do método de busca exaustiva. Esse resultado pode ser explicado pois, além de utilizar diagnosticadores, cujo espaço de estados pode crescer exponencialmente com relação ao número de estados do sistema, existe a necessidade de se encontrar todos os ciclos que possuem determinada característica o que, de acordo com [33] e [35], possui ordem de crescimento pior que exponencial com relação ao número de estados do grafo.

O objetivo desta dissertação, assim como em [29], é estudar e propor uma forma

sistemática de se encontrar todas as bases mínimas para o diagnóstico de falhas de um determinado SED. A busca exaustiva será evitada, pois os métodos a serem propostos são baseados apenas nas propriedades do sistema e no conjunto de condições necessárias e suficientes para que a diagnosticabilidade seja garantida. Ao contrário do método proposto em [29], serão utilizados verificadores determinísticos, em detrimento dos autômatos diagnosticadores. Verificadores possuem a vantagem de serem, no pior caso, calculados em tempo polinomial.

O uso de verificadores no problema de se encontrar todos os subconjuntos de menor cardinalidade que garantam a diagnosticabilidade do sistema permite que os algoritmos propostos neste trabalho possuam menor complexidade computacional que os demais apresentados na literatura. O primeiro algoritmo desenvolvido, apesar de menor custo computacional que o proposto em [29], pode ter, no pior caso, complexidade maior que o uso do método de busca exaustiva, uma vez que o mesmo exige a busca por todos os caminhos cíclicos com determinadas características em um autômato verificador. Já o segundo, por utilizar uma maneira inédita de obtenção desses subconjuntos, possuirá complexidade, em geral, menor que a utilização da busca exaustiva, caracterizando, dessa forma, o algoritmo de menor complexidade computacional existente para o problema em questão.

Os capítulos subseqüentes são organizados conforme indicado a seguir. O capítulo 2 apresenta alguns conceitos preliminares que serão necessários para completo entendimento desta dissertação e faz uma breve revisão sobre SEDs e o conceito de diagnosticabilidade de falhas. Além disso, são apresentadas as condições necessárias e suficientes para o diagnóstico de falhas centralizado utilizando diagnosticadores e verificadores. No capítulo 3, o trabalho realizado em [29] para obter todas as bases mínimas para o diagnóstico de falhas utilizando autômatos diagnosticadores é apresentado. Os capítulos 4 e 5 introduzem novos algoritmos para obtenção das bases mínimas utilizando verificadores, de forma que a cada ocorrência de um evento classificado como de falha seja detectado em tempo finito. Ao fim de cada capítulo, as complexidades computacionais dos algoritmos propostos são analisadas. Exemplos também são apresentados para ilustrar os resultados obtidos. Finalmente, no capítulo 6, conclusões e propostas de trabalhos futuros sobre o tema são realizadas.

Capítulo 2

Fundamentos teóricos sobre diagnóstico de falhas em sistemas a eventos discretos

Neste capítulo serão apresentados os fundamentos teóricos e definições preliminares necessários para o entendimento deste trabalho. Dentre os principais conceitos, as definições de autômato e linguagem gerada por um sistema a eventos discretos (SED) serão revistas. Serão relembradas ainda as operações realizadas entre dois ou mais autômatos e, em seguida, o problema do diagnóstico de falhas em sistemas a eventos discretos será formulado. Além disso, serão revistos os conceitos e passos necessários para a construção de autômatos diagnosticadores e verificadores, que são amplamente utilizados para verificar a diagnosticabilidade de falhas em SEDs, e apresentadas as condições necessárias e suficientes para que a linguagem de um sistema seja diagnosticável. Diversos exemplos serão utilizados ao longo deste capítulo para ilustrar os conceitos apresentados. Maiores detalhes podem ser obtidos em [2], [26] e [36].

2.1 Sistemas a eventos discretos

Sistemas a eventos discretos são sistemas dinâmicos cujo espaço de estados é um conjunto discreto e o mecanismo de transição de estados é dirigido por eventos, ou seja, o sistema, em geral, não está sincronizado com o tempo. Nesse caso, as transições são realizadas no momento da ocorrência de algum evento. Como consequência das características dessa classe de sistemas, os estados de um SED podem representar, além de qualquer valor discreto (como por exemplo valores numéricos pertencentes ao conjunto de N ou Z), valores simbólicos, tais como {válvula aberta, válvula fechada} ou {porta trancada, porta destravada}. Já os eventos podem estar

relacionados a ações específicas (acender uma lâmpada, acionar uma válvula, desligar uma máquina, etc) ou ser uma combinação de condições satisfeitas (chegada de uma peça a uma máquina, a vazão de uma linha em uma planta química que atinge determinado valor, etc). Pode-se observar que, a depender do grau de abstração considerado, praticamente qualquer sistema físico pode ser modelado como um SED, contudo, alguns sistemas são intrinsecamente discretos e com evolução dirigida por eventos.

O objetivo da modelagem de um sistema, seja ele de variáveis contínuas ou discretas, é a criação de um modelo que descreva o comportamento do mesmo, repeitando-se os limites de tolerância estabelecidos. Enquanto nos sistemas de varáveis contínuas as trajetórias de estados são traçadas em função do tempo, nos SEDs elas são definidas em função de uma sequência de eventos. O conjunto formado por todas as sequências de comprimento finito possíveis de serem geradas por um SED caracterizam a linguagem gerada por esse SED, sendo essa definida sobre o conjunto de eventos desse sistema. Dessa forma, o modelo de um SED é baseado basicamente em dois componentes: estados e eventos. A partir de ambos, é possível descrever a evolução ocorrida em um determinado sistema. O exemplo 2.1 ilustra um exemplo de sistema a eventos discretos e a importância de seus estados e eventos.

Exemplo 2.1. *Considere um cruzamento simples de veículos controlado por um semáforo, conforme ilustrado na figura 2.1. Existem quatro trajetórias possíveis para os veículos que pertencem ao sistema dado:*

- *(1,2) Veículos que vem de 1 e viram à direita para 2*
- *(1,3) Veículos que vem de 1 e viram à esquerda para 3*
- *(2,3) Veículos em trajetória retilínea de 2 para 3*
- *(3,2) Veículos em trajetória retilínea de 3 para 2*

O semáforo funciona de forma que estará vermelho ou verde para os veículos que vem de 1, e consequentemente verde ou vermelho para os veículos vindos de 2 e 3, respectivamente. Dessa forma, o conjunto de eventos do sistema é dado por: $\Sigma = \{a_{12}, a_{13}, a_{23}, a_{32}, d_{12}, d_{13}, d_{23}, d_{32}, g, r\}$*, sendo:*

- $a_{12}, a_{13}, a_{23}, a_{32}$*: a chegada de um veículo para cada uma da trajetórias possíveis,*
- $d_{12}, d_{13}, d_{23}, d_{32}$*: a partida de um veículo para cada uma da trajetórias possíveis,*
- g*: indica que o semáforo está verde para os veículos de trajetória (1,2) e (1,3),*

- *r: indica que o semáforo está vermelho para os veículos de trajetória (1,2) e (1,3).*

O espaço de estados pode ser definido como $X = \{(x_{12}, x_{13}, x_{23}, x_{32}, y) : x_{12}, x_{13}, x_{23}, x_{32} \geq 0, y \in \{G, R\}\}$, *sendo* x_{12}, x_{13}, x_{23}, x_{32} *a quantidade de veículos aguardando em fila para realizar cada uma das quatro trajetórias possíveis e y o estado do semáforo (verde ou vermelho).*

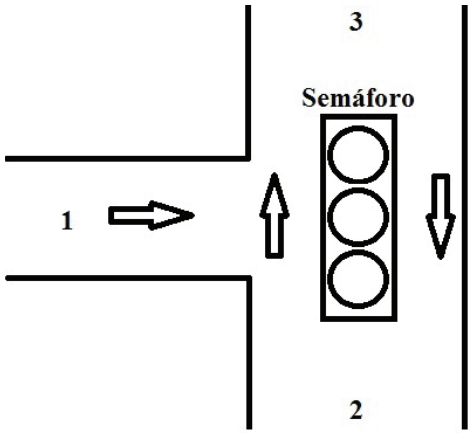


Figura 2.1: Cruzamento simples de veículos controlado por um semáforo.

Note que, conforme mostrado no exemplo 2.1, o conjunto de eventos Σ é formado por ações que ocorrem em instantes de tempo desconhecidos e são responsáveis por modificar o estado atual, como por exemplo, a chegada de um veículo para realizar a trajetória (1,2) (evento a_{12}), que acrescenta em uma unidade a quantidade de veículos aguardando em fila para realizar tal trajetória (estado x_{12}). Dessa forma, é evidente que o sistema modelado no exemplo 2.1 é um SED.

Observe que a modelagem mostrada no exemplo 2.1 poderia ser realizada de outras formas, a depender do interesse e grau de abstração desejado ao sistema de estudo.

2.2 Linguagens

Todo SED possui um conjunto de eventos Σ, cujos componentes formam o alfabeto da linguagem do sistema. Sequências realizadas a partir desses elementos devem ser interpretadas como as palavras de uma linguagem. Dessa forma, uma linguagem modela e representa o comportamento de um SED. A definição formal de linguagem é dada a seguir [2].

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.*

Diversas operações podem ser realizadas sobre o conjunto de eventos Σ ou sobre sequências existentes para formação de novas sequências. Uma delas é a concatenção, que consiste em agrupar duas ou mais sequências afim de gerar apenas uma. A sequência ba, por exemplo, é obtida após a concatenação dos eventos b e a, respectivamente. O elemento neutro da operação de concatenação é a sequência vazia ε. Dessa forma, $s\varepsilon = \varepsilon s = s$, em que s denota uma sequência de eventos.

A linguagem de um SED está contida em um conjunto que contem todas as sequências finitas formadas pelos elementos de Σ, incluindo a sequência vazia. Esse conjunto, denotado por Σ^*, é chamado de $Fecho\ de\ Kleene$ de Σ .

As operações de concatenação e *fecho de Kleene* podem também ser definidas sobre linguagens [2].

Definição 2.2. *(Concatenação) Sejam L_a, $L_b \subseteq \Sigma^*$, então a concatenação L_aL_b é definida como:*

$$L_aL_b = \{s \in \Sigma^* : (s = s_as_b)[s_a \in L_a, s_b \in L_b]\}.$$

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

$$L^* = \{\varepsilon\} \cup L \cup LL \cup LLL \ldots$$

Além das operações de concatenação e fecho de Kleene, outras operações podem ser realizadas sobre linguagens. Tais operações são definidas a seguir.

Definição 2.4. *(Fecho de prefixo) Seja $L \subseteq \Sigma^*$, então o fecho de prefixo de L, denotado por $\bar{L}$, é definido como:*

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

Uma linguagem L, tal que $L = \bar{L}$, é dita ser prefixo-fechada.

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

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

Por definição, $L/s = \emptyset$ se $s \notin \bar{L}$.

Definição 2.6. *(Projeção) A projeção* $P_s : \Sigma_l^\star \to \Sigma_s^\star$*, sendo* $\Sigma_s \subset \Sigma_l$*, é definida da seguinte forma:*

$$P_s(\varepsilon) = \varepsilon$$

$$P_s(\sigma) = \begin{cases} \sigma, \text{ se } \sigma \in \Sigma_s \\ \varepsilon, \text{ se } \sigma \in \Sigma_l \setminus \Sigma_s \end{cases}$$

$P_s(s\sigma) = P_s(s)P_s(\sigma)$*, para todo* $s \in \Sigma_l^\star$*,* $\sigma \in \Sigma_l$*,*
em que $\setminus$ *denota a diferença entre conjuntos.*

Note que, de acordo com a definição 2.6, a operação de projeção consiste em apagar das sequências de s $\in \Sigma_l^\star$, todos os eventos $\sigma \in \Sigma_l \setminus \Sigma_s$.

Uma das aplicações da operação de projeção é representar a linguagem observada de um sistema, isto é, somente eventos observados serão visualizados nas sequências existentes nesse sistema. Dessa forma, é possível que duas sequências distintas tenham a mesma projeção, e passem a ser idênticas do ponto de vista de algum observador, gerando ambiguidade entre ambas.

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

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

2.3 Autômatos

Autômatos são dispositivos capazes de representar uma linguagem conforme regras definidas. A definição formal de um autômato determinístico é dada a seguir [2].

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

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

sendo X *o conjunto de estados,* Σ *o conjunto finito de eventos de G,* $f : X \times \Sigma \to X$*, a função de transição de estados,* $\Gamma : X \to 2^\Sigma$*, a função de eventos ativos,* x_0*, o estado inicial do sistema e* X_m *o conjunto de estados marcados.*

Observação 2.1. *Por conveniência,* f *é sempre estendida do domínio* $X \times \Sigma$ *para o domínio* $X \times \Sigma^*$ *de forma recursiva, como mostrado abaixo:*

$$f(x, \varepsilon) = x,$$

$$f(x, s\sigma) = f[f(x, s), \sigma], \forall x \in X, \sigma \in \Sigma, s \in \Sigma^*.$$

Observação 2.2. *Um autômato é denominado determinístico quando para todo estado $x \in X$ e para todo evento $\sigma \in \Gamma(x)$, existe um único estado $y \in X$ tal que $f(x, \sigma) = y$. Note que, se um autômato for não-determinístico, $f(x, \sigma)$ poderá levar não só a um novo estado, mas sim a um conjunto de estados.*

Autômatos são representados graficamente por grafos direcionados. Nesses grafos, círculos representam os estados, que são conectados por arcos rotulados. Os símbolos que identificam os arcos de transição representam os eventos de G. O estado inicial é indicado através de uma seta orientada para ele que não provem de qualquer outro estado. Estados marcados são representados classicamente por duas circunferências concêntricas e, geralmente, estão associados ao cumprimento de determinada tarefa ou objetivo.

A definição de linguagem gerada e marcada por um autômato é apresentada a seguir.

Definição 2.9. *(Linguagem gerada) A linguagem gerada por um autômato $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ é definida como*

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

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

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

As definições de caminho e caminho cíclico também são importantes para o desenvolvimento dos conceitos desta dissertação, e são apresentadas a seguir.

Definição 2.11. *(Caminho) Em um autômato, um caminho (x_k, σ_1, x_{k+1}, σ_2, ..., σ_l, x_{k+l}), $l > 0$, é a sequência de estados e eventos tais que $x_{k+i} = f(x_{k+i-1}, \sigma_i)$, $\forall i \in \{1, 2, 3, \ldots, l\}$.*

Definição 2.12. *(Caminho cíclico) Um caminho (x_k, σ_1, x_{k+1}, σ_2, ..., σ_l, x_{k+l}) é dito ser cíclico se $x_{k+l} = x_k$.*

A sequência associada a um caminho é dada pela concatenação dos rótulos das transições pertencentes a esse caminho. Dessa forma, uma sequência $s \in \mathcal{L}(G)$ se, e somente se, s está associada a um caminho possível no grafo orientado de G, ou seja, se $f(x_0, s)$ for definida. Já a linguagem marcada é formada pelas sequências de eventos associadas a caminhos possíveis a partir do estado inicial que alcançam estados marcados.

É evidente que a linguagem marcada $\mathcal{L}_m(G)$ será sempre um subconjunto da linguagem gerada $\mathcal{L}(G)$, uma vez que $\mathcal{L}_m(G)$ é formada por todas as sequências s tais que $f(x_0, s) \in X_m$.

Em algumas situações, é necessário analisar apenas parte de um autômato. Em outras, pode ser necessário criar um novo autômato que represente a composição de dois ou mais sistemas. Para essas situações, devem ser realizadas operações com autômatos, que serão mostradas na próxima seção.

2.4 Operações com autômatos

Conforme será visto ainda neste capítulo e também nos capítulos 3, 4 e 5, diversas operações entre autômatos serão fundamentais para o diagnóstico de falhas em SEDs. Serão apresentadas, a seguir, as operações necessárias para proporcionar a compreensão dos algoritmos que serão expostos nos capítulos posteriores.

A parte acessível de um autômato G é a operação unária que tem por obetivo eliminar todos os estados de G que não são alcançáveis a partir do estado inicial x_0. Obviamente, as transições associadas a estados não acessíveis também são apagadas. A definição formal de parte acessível de um autômato é dada abaixo.

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

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

sendo $X_{ac} = \{x \in X : (\exists s \in \Sigma^*)\ [f(x_0, s) = x]\}$; $X_{ac,m} = X_m \cap X_{ac}$; $f_{ac} : X_{ac} \times \Sigma^* \rightarrow X_{ac}$, a função de transição obtida após a restrição do domínio de f para o domínio dos estados acessíveis X_{ac}, isto é, os estados alcançáveis a partir de x_0.

A parte coacessível de um autômato G, também uma operação unária, tem por objetivo eliminar todos os estados de G a partir dos quais não se pode alcançar um estado marcado.

Definição 2.14. *(Parte coacessível) Dado o autômato $G = (X, \Sigma, f, x_0, X_m)$, a parte coacessível de G, denotada por $CoAc(G)$, é o subautômato*

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

sendo $X_{coac} = \{x \in X : (\exists s \in \mathcal{L}(G))[f(x, s) \in X_m]\}$; $x_{0,coac} = x_0$, se $x_0 \in X_{coac}$ e $x_{0,coac}$ é indefinido, se $x_0 \notin X_{coac}$; $f_{coac} : X_{coac} \times \Sigma^* \rightarrow X_{coac}$, denota a nova função de transição obtida após a restrição do domínio de f para o domínio dos estados coacessíveis X_{coac}, isto é, os estados a partir dos quais se alcança um estado marcado.

Exemplo 2.2. *Considere o autômato determinístico G, mostrado na figura 2.2. As figuras 2.3(a) e (b) mostram os autômatos resultantes após as operações de parte acessível e coacessível, respectivamente.*

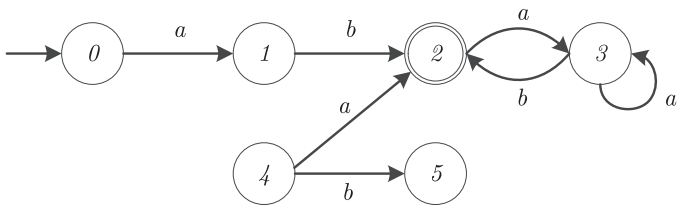
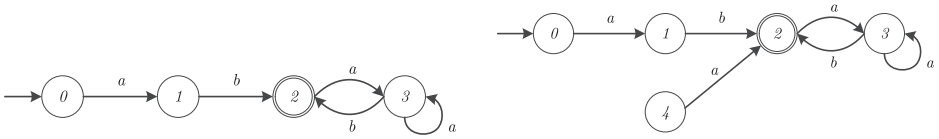


Figura 2.2: Autômato determinístico G.



(a) Parte acessível de G.

(b) Parte coacessível de G.

Figura 2.3: Exemplos de operações unárias em um autômato.

As operações de projeção e projeção inversa de uma linguagem L são operações unárias que podem ser aplicadas sobre um autômato G que gere ou marque L. A projeção $P : \Sigma_l^* \to \Sigma_s^*$ aplicada sobre a linguagem L é implementada no autômato G substituindo-se todas as transições rotuladas por eventos de $\Sigma_l \setminus \Sigma_s$ pela sequência vazia ε, gerando um novo autômato não determinístico G_{nd}. É possível obter um autômato determinístico a partir de G_{nd}, porém, para tal, é necessário o cálculo de um observador, que será mostrado mais adiante neste capítulo.

A operação de projeção inversa $P^{-1} : \Sigma_s^* \to 2^{\Sigma_l^*}$ aplicada sobre uma linguagem $L \subseteq \Sigma_s^*$ pode ser obtida a partir da adição de um autolaço rotulado por todos os eventos de $\Sigma_l \setminus \Sigma_s$ a cada estado do autômato G que marca L e, em seguida, tomando-se a linguagem marcada pelo autômato resultante.

A seguir, serão vistas operações que envolvem dois ou mais autômatos gerando um único autômato.

Definição 2.15. *(Produto) Sejam os autômatos* $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{01}, X_{m1})$ *e* $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{02}, X_{m2})$*, então, o produto* $G_1 \times G_2$ *é dado por:*

$$G_1 \times G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\times2}, \Gamma_{1\times2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}),$$

sendo

$$f_{1\times2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), \text{ se } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2); \\ \text{indefinido, caso contrário.} \end{cases}$$

$$\Gamma_{1\times2}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2).$$

Observe que uma transição ocorrerá em $G_1 \times G_2$ se, e somente se, a transição é possível em ambos os autômatos, ou seja, a evolução de estados do autômato $G_1 \times G_2$ será totalmente sincronizada com a evolução de estados dos autômatos G_1 e G_2. Como consequência, pode-se afirmar que $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$ e $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$.

Definição 2.16. *(Composição paralela) Sejam os autômatos* $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{01}, X_{m1})$ *e* $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{02}, X_{m2})$*. Então, a composição paralela, denotada por* $G_1 \| G_2$*, é dada por:*

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

sendo

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

$$\Gamma_{1\|2}(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus \Sigma_2] \cup [\Gamma_2(x_2) \setminus \Sigma_1].$$

Utilizando-se a operação de projeção, é possível definir as linguagens gerada e marcada do autômato resultante $G_1 \| G_2$ como $\mathcal{L}(G_1 \| G_2) = P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)]$ e $\mathcal{L}_m(G_1 \| G_2) = P_1^{-1}[\mathcal{L}_m(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)]$, respectivamente, em que $P_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^*$, para $i = 1, 2$.

Observe que a composição paralela permite a ocorrência de um evento comum a Σ_1 e Σ_2 se, e somente se, esse evento estiver ativo simultaneamente nos estados atuais de G_1 e G_2. Já eventos particulares pertencentes a $\Sigma_1 \setminus \Sigma_2$ ou $\Sigma_2 \setminus \Sigma_1$ podem ser executados sempre que possível.

Exemplo 2.3. *Considere os autômatos* G_1 *e* G_2 *mostrados na figura 2.4. As figuras 2.5(a) e (b) mostram o resultado do produto e composição paralela, respectivamente, entre* G_1 *e* G_2*.*

2.5 SEDs com observação reduzida

Até o momento, considerou-se que todos os eventos de Σ são eventos cuja ocorrência é conhecida de alguma forma, ou seja, os mesmos podem ser observados. Entretanto,

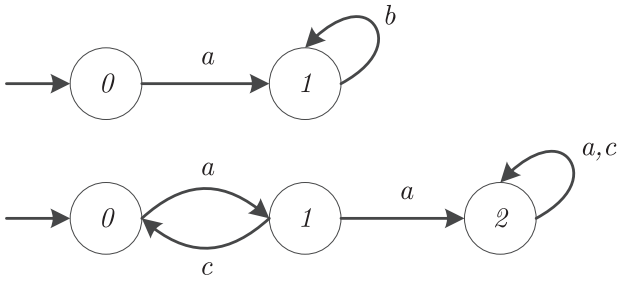
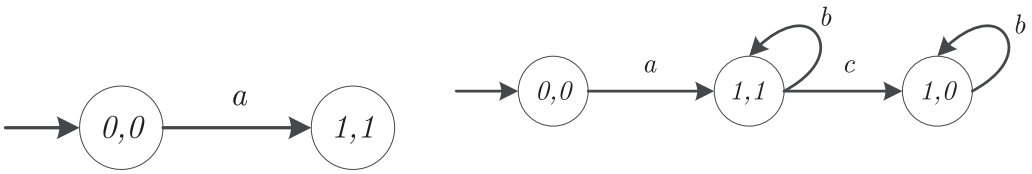


Figura 2.4: Autômatos G_1 e G_2 do exemplo 2.3.



(a) Produto entre G_1 e G_2.

(b) Composição paralela entre G_1 e G_2.

Figura 2.5: Exemplos de operações com dois autômatos.

nem sempre isso é verdade. SEDs com observação reduzida são sistemas cujo conjunto de eventos pode ser particionado em dois subconjuntos: Σ_o e Σ_{uo}, ou seja, $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. O conjunto de eventos observáveis Σ_o consiste no conjunto de eventos cuja ocorrência pode ser registrada por um observador externo, enquanto Σ_{uo} é formado por todos os eventos que não podem ser registrados. Eventos de falha, objeto de grande interesse neste trabalho, também podem fazer parte do conjunto de eventos não observáveis, conforme será considerado daqui em diante, uma vez que, caso as falhas pudessem ser registradas, as mesmas seriam trivialmente diagnosticadas.

No caso de sistemas determinísticos contendo eventos não observáveis, a linguagem gerada observada por G será composta por todas as sequências de $\mathcal{L}(G)$ apagando-se os eventos não observáveis. Essa linguagem pode ser obtida por intermédio da operação de projeção, isto é, a linguagem gerada observada de G será $P_o[\mathcal{L}(G)]$, sendo $P_o : \Sigma^* \rightarrow \Sigma_o^*$. O mesmo pode ser realizado para se obter a linguagem marcada observada de G, que será, dessa forma, igual a $P_o[\mathcal{L}_m(G)]$. É evidente que sequências distintas poderão ter a mesma projeção, o que nos remete a autômatos não determinísticos. De forma a obter um autômato determinístico que gere e marque as linguagens observadas do autômato com observação reduzida, será necessário introduzir o conceito de observador. Entretanto, outro conceito é necessário para o completo entendimento do cálculo do autômato observador: o alcance não observável.

Definição 2.17. *(Alcance não observável) O alcance não observável de um estado $x \in X$, denotado por $UR(x)$, é definido como:*

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

O alcance não observável pode ser definido também para um conjunto $B \in 2^X$ da seguinte forma:

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

Observe que o alcance não observável de um estado x retorna todos os estados alcançáveis a partir de x através de transições rotuladas por eventos não observáveis. A seguir, será apresentado o conceito de observador, que utiliza a definição 2.17 de alcance não observável.

Definição 2.18. *(Observador) O observador de um autômato G com relação a um conjunto de eventos observáveis Σ_o, denotado por $Obs(G, \Sigma_o)$, é dado por*

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

sendo $X_{Obs} \subseteq 2^X$ e $X_{mObs} = \{B \in X_{Obs} : B \cap X_m \neq \emptyset\}$. f_{Obs}, Γ_{Obs} e $x_{0,Obs}$ são definidos de acordo com o algoritmo 2.1 de obtenção do observador.

Algoritmo 2.1. *(Observador)*

Passo 1) *Defina $x_{0,Obs} = UR(x_0)$. Faça $X_{Obs} = \{x_{0,Obs}\}$ e $\widetilde{X}_{Obs} = X_{Obs}$.*

Passo 2) $\overline{X}_{Obs} = \widetilde{X}_{Obs}$ *e* $\widetilde{X}_{Obs} = \emptyset$.

Passo 3) *Para cada $B \in \overline{X}_{Obs}$:*

- Passo 3.1) $\Gamma_{Obs}(B) = (\bigcup_{x \in B} \Gamma(x)) \cap \Sigma_o$.
- Passo 3.2) *Para cada $\sigma \in \Gamma_{Obs}(B)$:*

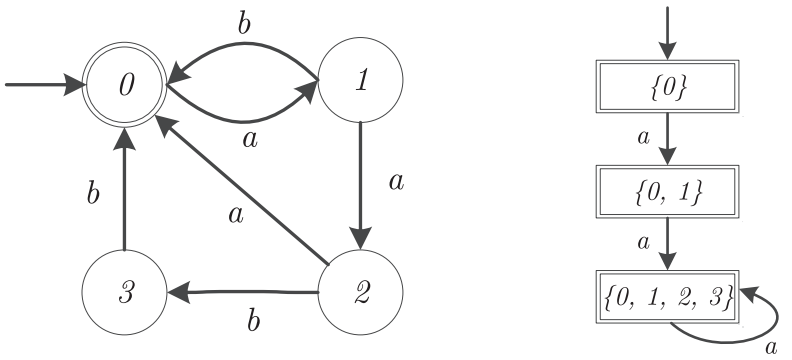
$$f_{Obs}(B, \sigma) = UR(\{x \in X : (\exists y \in B)[x = f(y, \sigma)]\}).$$

- Passo 3.3) $\widetilde{X}_{Obs} \leftarrow \widetilde{X}_{Obs} \cup \{f_{Obs}(B, \sigma)\}$.

Passo 4) $X_{Obs} \leftarrow X_{Obs} \cup \widetilde{X}_{Obs}$.

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

Passo 6) $X_{mObs} = \{B \in X_{Obs} : B \cap X_m \neq \emptyset\}$.



(a) Autômato G. (b) Observador de G.

Figura 2.6: Autômato G e seu observador apresentados no exemplo 2.4.

Exemplo 2.4. *Seja G o autômato mostrado na figura 2.6(a) e suponha que o conjunto de eventos Σ possa ser particionado como $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ em que $\Sigma_o = \{a\}$ e $\Sigma_{uo} = \{b\}$. A figura 2.6(b) mostra o observador obtido ao se executar o algoritmo 2.1. Note que, ao ser observada a sequência de eventos $s = aa$, a estimativa de estado alcançado é $\{0, 1, 2, 3\}$, ou seja, existe uma sequência v_i para cada estado $x_i \in X$ de G cuja projeção $P_o(v_i) = s$ e que satisfaz $f(x_0, v_i) = x_i$.*

Os conceitos e definições apresentados nesta seção formam a base teórica necessária para que o problema de diagnóstico de falhas em SEDs possa ser formulado.

2.6 Diagnóstico de falhas

Ao introduzir eventos não observáveis na modelagem de sistemas, pode ser necessário inferir a ocorrência de um ou mais eventos desse conjunto, como por exemplo, os eventos de falha. Os casos nos quais as falhas podem ser detectadas por sensores, o diagnóstico é realizado imediatamente, não sendo necessário qualquer estudo para tal. Contudo, em muitos cenários, não existirá qualquer sensor que torne essa tarefa possível e, portanto, a falha deverá ser modelada como um evento não observável e sua ocorrência deverá ser inferida utilizando a teoria de diagnóstico de falhas em SEDs.

A ideia de diagnosticabilidade da linguagem de um SED está relacionada com a capacidade de se inferir a ocorrência do(s) evento(s) de falha não observável, após um número finito de observações de eventos. Nos trabalhos a respeito de diagnóstico de falhas em SED, as seguintes hipóteses são usualmente realizadas [27], [29]:

- **H1** - A linguagem gerada por G é viva, ou seja, $\Gamma(x_i) \neq \emptyset$ para todo $x_i \in X$.

- **H2** - O autômato G não possui caminhos cíclicos compostos somente por eventos não observáveis.
- **H3** - Existe um único evento de falha, ou seja, $\Sigma_f = \{\sigma_f\}$.

A hipótese **H1** pode ser entendida do ponto de vista prático uma vez que se deseja que os sistemas estejam em contínua operação. A premissa **H2** é realizada inicialmente para evitar que a ocorrência do evento de falhas seja sucedida por um ciclo de estados não observáveis, o que impediria que a detecção dessa falha fosse realizada. Essa hipóstese será removida posteriormente, ao definirmos os chamados ciclos escondidos [27]. A premissa **H3** foi realizada apenas por questões de simplificação de análise. Em caso de existência de mais de um evento de falha, bastaria analisar a diagnosticabilidade do sistema para cada um desses eventos individualmente, isto é, considerar um deles como sendo de falha e os demais como eventos comuns não observáveis do sistema, e repetir o procedimento até que todo o conjunto que representa as falhas do sistema tenha sido verificado.

Seja L a linguagem gerada por um autômato e seja $L_N \subseteq L$ a linguagem que representa o comportamento normal do sistema, isto é, L_N é formada por todas as sequências de L que não contém qualquer evento do conjunto de eventos de falha Σ_f. Observe que L_N é necessariamente prefixo fechada. Utilizando essa definição, é possível introduzir a definição de linguagem diagnosticável.

Definição 2.19. *(Linguagem diagnosticável) Seja L a linguagem prefixo fechada gerada por um autômato G e seja $L_N \subset L$ a linguagem prefixo fechada que representa o comportamento normal do sistema. Considere o conjunto de eventos Σ particionado da seguinte forma: $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, sendo $\Sigma_f \subseteq \Sigma_{uo}$ o conjunto de eventos de falha. Então, L é diagnosticável com relação a $P_o : \Sigma^* \to \Sigma_o^*$ e Σ_f, se, e somente se,*

$$(\exists n \in \mathbb{N})(\forall s \in L \setminus L_N)\ (\forall st \in L \setminus L_N, \|t\| \geq n) \Rightarrow$$
$$(\forall \omega \in P_o^{-1}(P_o(st)) \cap L, \omega \in L \setminus L_N).$$

em que $||.||$ denota o comprimento de uma sequência.

Conforme a definição acima, L é diagnosticável com relação a P_o e Σ_f se, e somente se, para todas as sequências st de comprimento arbitrariamente longo contendo o evento de falha, não existe qualquer sequência $s' \in L_N$ tal que $P_o(s') = P_o(st)$.

2.7 Autômato diagnosticador

Com o objetivo de se realizar o diagnóstico de falhas a partir da observação do comportamento do sistema em tempo real e/ou para verificar se a linguagem gerada

por um autômato G é diagnosticável, pode-se utilizar um autômato determinístico denominado diagnosticador. Um diagnosticador $G_d = (X_d, \Sigma_o, f_d, \Gamma_d, x_{o_d})$ é um autômato cujo conjunto de eventos é igual ao conjunto dos eventos observáveis de G e cujos estados possuem rótulos para indicar a ocorrência do evento de falha. São utilizados dois rótulos distintos na construção dos diagnosticadores: N, que indica a não ocorrência de eventos de falha, e Y, que indica a ocorrência de σ_f.

A seguir, é apresentado um algoritmo para obtenção de G_d [2].

Algoritmo 2.2. *(Diagnosticador)*

Passo 1 *Faça a composição $G\|A_{label}$, sendo G o autômato que modela o sistema e A_{label} o autômato que incluirá os rótulos aos estados G, vide figura 2.7.*

Passo 2 *Calcule $G_d = Obs(G\|A_{label})$.*

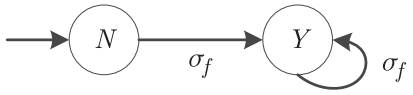


Figura 2.7: Autômato rotulador, A_{label}.

Note que, $\mathcal{L}(G\|A_{label}) = \mathcal{L}(G)$ e que $\mathcal{L}(G_d) = P_o[\mathcal{L}(G\|A_{label})] = P_o[\mathcal{L}(G)]$.

É importante observar que o diagnosticador é um estimador de estados de G. Quando G_d alcança um estado cujos rótulos são todos iguais a N, tem-se a certeza que a falha não ocorreu, caracterizando um estado *negativo* ou *normal*. Por outro lado, se todos os rótulos dos estados que compõem o estado vigente de G_d são iguais a Y, tem-se a certeza da ocorrência da falha, o que caracteriza um estado *positivo* ou *certo*. Caso existam estados em G_d 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. Além disso, deve ser observado que, uma vez que o diagnosticador alcance um estado *positivo*, todos os estados futuros serão também *positivos*. Contudo, é possível para um diagnosticador mudar de um estado *normal* para *incerto* ou *certo*.

Definição 2.20. *Um estado $x_d \in X_d$ é denominado certo, se $l = Y$ para todo $(x, l) \in x_d$, e normal se $l = N$ para todo $(x, l) \in x_d$. Caso exista $(x, l), (y, \widetilde{l}) \in x_d$, x não necessariamente distinto de y tal que $l = Y$ e $\widetilde{l} = N$, então x_d é um estado incerto de G_d.*

Observação 2.3. *Seja x_d um estado incerto de G_d, então, existem $s_1 \in L \setminus L_N$ e $s_2 \in L_N$ tais que $P_o(s_1) = P_o(s_2) = \nu$ e $f_d(x_{0_d}, \nu) = x_d$.*

Observação 2.4. *Seja $x_d = f_d(x_{0_d}, \nu)$. Se x_d é um estado certo de G_d, então $\forall w \in (P_o^{-1}(\nu) \cap L)$, $w \in L \setminus L_N$.*

Uma consequência das observações 2.3 e 2.4 é que a linguagem gerada por G será diagnosticável com relação a Σ_f e P_o se, e somente se, o diagnosticador G_d sempre alcançar um estado certo para toda sequência arbitrariamente longa de L que contém o evento σ_f. As definições a seguir (vide [3]) são importantes para se estabelecer a condição necessária e suficiente para o diagnóstico de um sistema utilizando-se diagnosticadores.

Definição 2.21. *(Ciclo) Um conjunto de estados $\{x_k, x_{k+1}, \ldots, x_{k+l}\}$ forma um ciclo em um autômato G se existir uma sequência $s = \sigma_1\sigma_2\ldots\sigma_l$ tal que (x_k, σ_1, x_{k+1}, σ_2, …, σ_l, x_{k+l}), $l > 0$, forme um caminho cíclico em G. Além disso, um ciclo é dito simples se, além das condições anteriores, os estados x_k, x_{k+1}, …, x_{k+l-1} são distintos.*

Definição 2.22. *(Ciclo indeterminado) Um conjunto de estados incertos $\{x_{d_1}, x_{d_2}, \ldots, x_{d_p}\} \subseteq X_d$ forma um ciclo indeterminado se as seguintes condições forem satisfeitas:*

1. *$\{x_{d_1}, x_{d_2}, \ldots, x_{d_p}\}$ forma um ciclo em G_d;*
2. *$\exists (x_l^{k_l}, Y), (\widetilde{x}_l^{r_l}, N) \in x_{d_l}$, sendo $x_l^{k_l}$ não necessariamente distinto de $\widetilde{x}_l^{r_l}$, $l = 1, 2, \ldots, p, k_l = 1, 2, \ldots, m_l$, e $r_l = 1, 2, \ldots, \widetilde{m}_l$ de tal forma que os estados $\{x_l^{k_l}\}$, $l = 1, 2, \ldots, p, k_l = 1, 2, \ldots, m_l$ e $\{\widetilde{x}_l^{r_l}\}$, $l = 1, 2, \ldots, p, r_l = 1, 2, \ldots, \widetilde{m}_l$ podem ser rearrumados para formar ciclos em G.*

Utilizando as definições de ciclos, ciclos indeterminados e a de linguagens diagnosticáveis, pode-se enunciar a seguinte condição necessária e suficiente para o diagnóstico de uma linguagem.

Teorema 2.1. *Considere que as hipóteses **H1**, **H2** e **H3** sejam satisfeitas. Então, uma linguagem L gerada por um autômato G será diagnosticável com relação a projeção P_o e $\Sigma_f = \{\sigma_f\}$ se, e somente se, o seu diagnosticador G_d não tiver ciclos indeterminados.*

Prova. A prova pode ser obtida em [3]. □

Assim, após a obtenção do diagnosticador do sistema, $G_d = Obs(G \| A_{label})$, basta verificar a existência de algum ciclo indeterminado para determinar a diagnosticabilidade da linguagem gerada por G. Caso haja algum ciclo indeterminado, então L é não diagnosticável com relação a Σ_f e P_o. Caso contrário, L é dita ser diagnosticável.

Exemplo 2.5. *Para ilustrar os conceitos vistos sobre diagnosticadores, considere o autômato G, mostrado na figura 2.8, e a partição de seu conjunto de eventos $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, sendo $\Sigma_o = \{a, b, c\}$ e $\Sigma_{uo} = \{\sigma_f\}$. Considere também $\Sigma_f = \{\sigma_f\}$. As figuras 2.9(a) e 2.9(b) mostram, respectivamente, a composição paralela $G \| A_{label}$ e o diagnosticador $G_d = Obs(G \| A_{label})$. Para que a diagnosticabilidade do sistema com relação a P_o e Σ_f seja determinada, deve-se procurar por ciclos indeterminados. Neste exemplo, como não existem ciclos indeterminados, então a linguagem gerada por G é diagnosticável com relação a P_o e Σ_f.*

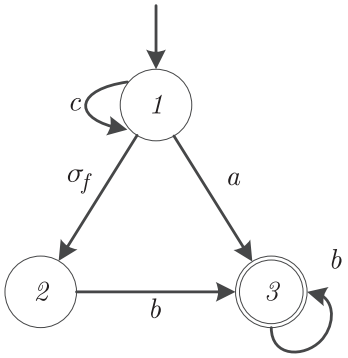
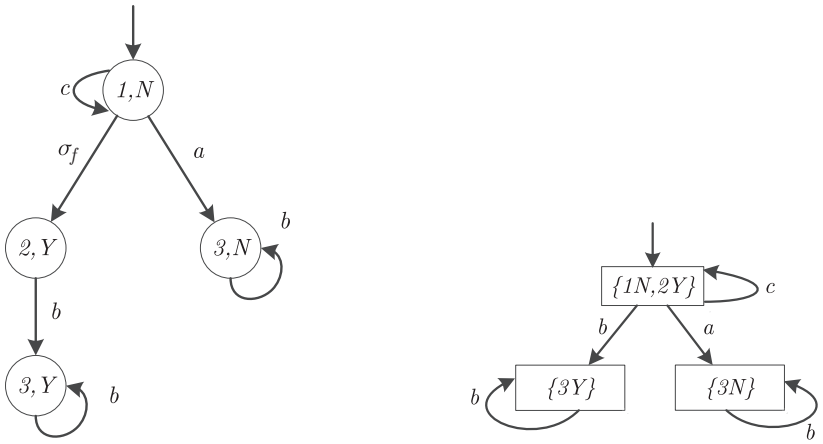


Figura 2.8: Diagrama de transição de estados do autômato do exemplo 2.5.



(a) Composição paralela $G \| A_{label}$.

(b) Diagnosticador G_d.

Figura 2.9: Obtenção do diagnosticador centralizado G_d.

Observação 2.5. *É importante destacar que a simples presença de ciclos em G_d formados somente por estados incertos não implica necessariamente que a linguagem L seja não diagnosticável com relação a P_o e Σ_f. Para que L seja não diagnosticável, é necessário que exista em G ao menos um ciclo após a ocorrência da falha e esse ciclo seja correspondente ao ciclo de estados incertos em G_d.*

Autômatos diagnosticadores possuem como vantagem a possibilidade de serem utilizados tanto para verificação da diagnosticabilidade de uma linguagem quanto para diagnóstico *on-line*. Como desvantagem, apresentam o possível crescimento exponencial de seu número de estados, devido à utilização de observadores em seu algoritmo de construção (vide algoritmo 2.2). Para contornar esse problema, pode-se verificar a propriedade de diagnosticabilidade da linguagem de um sistema por intermédio de autômatos verificadores. O uso de verificadores possui como vantagem a complexidade polinomial com relação ao espaço de estados do SED original. A seguir, será apresentado o conceito e as características de autômatos verificadores, assim como seu algoritmo de construção.

2.8 Autômato verificador

Na seção anterior, o problema de detecção de falhas em SEDs foi formulado utilizando-se diagnosticadores e, como foi ressaltado, o uso de diagnosticadores pode resultar em uma verificação de complexidade exponencial, devido à possibilidade de crescimento do espaço de estados dos diagnosticadores se comparado ao sistema original. Tal fato estimulou pesquisas por um dispositivo que pudesse realizar essa tarefa com complexidade polinomial, resultando nos autômatos verificadores. Diversos trabalhos foram realizados, apresentando diferentes algoritmos para a obtenção de verificadores para verificar a diagnosticabilidade do sistema, tanto para cenários com informação centralizada [23], [24] como em situações de sistemas distribuídos [11], [25]. Dentre os métodos existentes para o cálculo do autômato verificador, o que apresenta menor complexidade computacional atualmente foi desenvolvido por MOREIRA *et al.* [26] e, por essa razão, será o método utilizado neste trabalho.

Algoritmo 2.3. *(Verificador) Seja G o autômato determinístico que modela o sistema, Σ_f o conjunto de eventos de falha e suponha que Σ seja particionado de forma que $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$.*

Passo 1) *Construa o autômato G_N que modela o comportamento normal do sistema, como indicado abaixo:*

- *Defina $\Sigma_N = \Sigma \setminus \Sigma_f$.*
- *Construa o autômato A_N composto por um único estado N contendo um autolaço rotulado por todos os eventos de Σ_N.*
- *Construa o autômato $G_N = G \times A_N = (X_N, \Sigma, f_N, \Gamma_N, x_{0,N})$.*
- *Redefina o conjunto de eventos de G_N como Σ_N, isto é, $G_N = (X_N, \Sigma_N, f_N, \Gamma_N, x_{0,N})$.*

Passo 2) *Construa o autômato G_F que modela o comportamento de falha do sistema, conforme mostrado abaixo:*

- *Obtenha $G_l = G \| A_{label}$ e marque todos os estados de G_l cuja segunda componente seja Y.*
- *Calcule o autômato que modela o comportamento de falha $G_F = C_o A_c(G_l)$.*

Passo 3) *Defina a função $R : \Sigma_N \to \Sigma_R$ como:*

$$R(\sigma) = \begin{cases} \sigma, \text{ se } \sigma \in \Sigma_o, \\ \sigma_R, \text{ se } \sigma \in \Sigma_{uo} \setminus \Sigma_f. \end{cases}$$

Em seguida, construa o autômato $G_{N,1} = (X_N, \Sigma_R, f_{N,1}, x_{0,N})$, sendo $f_{N,1}(x_N, R(\sigma)) = f_N(x_N, \sigma)$, $\forall \sigma \in \Sigma_N$.

Passo 4) *Construa o autômato verificador $G_V = G_{N,1} \| G_F = (X_V, \Sigma_R \cup \Sigma, f_V, x_{0,V})$. Note que um estado de G_V é dado por $x_V = (x_N, x_F)$, sendo x_N e x_F estados de G_N e G_F, respectivamente, e $x_F = (x, x_l)$, sendo x e x_l estados de G e A_{label}, respectivamente.*

Passo 5) *Verifique a existência de um caminho cíclico em G_V tal que*

$$cl := (x_V^k, \sigma_k, x_V^{k+1}, \sigma_{k+1}, \ldots, x_V^l, \sigma_l, x_V^k),$$

sendo $l \geq k > 0$, que satisfaça as seguintes condições:

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

Conforme mostrado no algoritmo 2.3, a linguagem L_N gerada pelo autômato G_N contém todas as sequências de G que não possuem eventos de falha do conjunto Σ_f. Por outro lado, G_F é o autômato que representa o comportamento de falha do sistema, isto é, $\mathcal{L}(G_F) = L \setminus L_N$. Note que a função de renomeação dada no passo 3 tem por objetivo alterar o rótulo dos eventos que pertencem a $\Sigma_{uo} \setminus \Sigma_f$ para que, ao realizar a composição paralela no passo seguinte, somente os eventos observáveis sejam comuns a G_N e G_F. Isso fará com que o verificador exiba apenas as sequências de G_F que possuam as mesmas projeções das sequências do autômato G_N, uma vez que apenas os eventos observáveis podem ocorrer simultaneamente, o que nos leva à seguinte definição.

Definição 2.23. *(Sequência ambígua) Uma sequência $s_F \in L \setminus L_N$ é dita ser uma sequência ambígua se, e somente se, existe $s_N \in L_N$ tal que $P_o(s_F) = P_o(s_N)$, em que L_N é a linguagem gerada pelo autômato G_N.*

Dessa forma, pode-se afirmar que as sequências existentes em G_V que levam a estados rotulados por Y estão associadas às sequências ambíguas de L.

O teorema a seguir, dado em [26], enuncia uma condição necessária e suficiente para a verificação da propriedade de diagnosticabilidade de uma linguagem utilizando-se verificadores.

Teorema 2.2. *Sejam L e L_N linguagens prefixo-fechadas geradas por G e G_N, respectivamente. Considere a projeção $P_o : \Sigma^* \to \Sigma_o^*$ e o conjunto de eventos de falhas Σ_f. Então, L não será diagnosticável com relação a P_o e Σ_f se, e somente se, existir um caminho cíclico em G_V, $cl := (x_V^k, \sigma_k, x_V^{k+1}, \sigma_{k+1}, \ldots, x_V^l, \sigma_l, x_V^k)$, sendo $l \geq k > 0$, que satisfaz à seguinte condição:*

$$\exists j \in \{k, k+1, \ldots, l\} \text{ tal que, para algum } x_V^j, (x_l^j = Y) \wedge (\sigma_j \in \Sigma). \tag{2.1}$$

Prova. Vide [26]. □

Observação 2.6. *Note que a hipótese **H1** pode ser removida ao utilizar o verificador apresentado no algoritmo 2.3, uma vez que, caso o sistema possua estados de bloqueio, bastaria acrescentar autolaços rotulados por um evento não observável $\sigma_u \in \Sigma_{uo}$ a esses estados, gerando um novo autômato determinístico. Dessa forma, o resultado será uma nova linguagem, viva, mas tal que sua projeção seja idêntica a da linguagem existente antes da inclusão dos autolaços, não afetando, assim, a propriedade de diagnosticabilidade do sistema.*

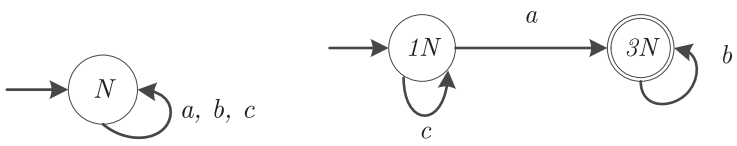
*Observe que a hipótese **H2** também pode ser removida pois, diferentemente do diagnosticador, o verificador apresentado no algoritmo 2.3 não calcula o observador do autômato que modela o sistema. Assim, as transições rotuladas por eventos não observáveis não são escondidas. Note ainda que, de acordo com o teorema 2.2, ciclos formados apenas por eventos não observáveis e renomeados não alteram a diagnosticabilidade do sistema.*

Definição 2.24. *(Caminho cíclico ambíguo) Um caminho cíclico $cl := (x_V^k, \sigma_k, \ldots, x_V^k)$ é dito ser um caminho cíclico ambíguo se satisfaz a condição dada na equação 2.1.*

O exemplo a seguir ilustra a construção de verificadores e a sua utilização na análise do diagnóstico de falhas em um SED.

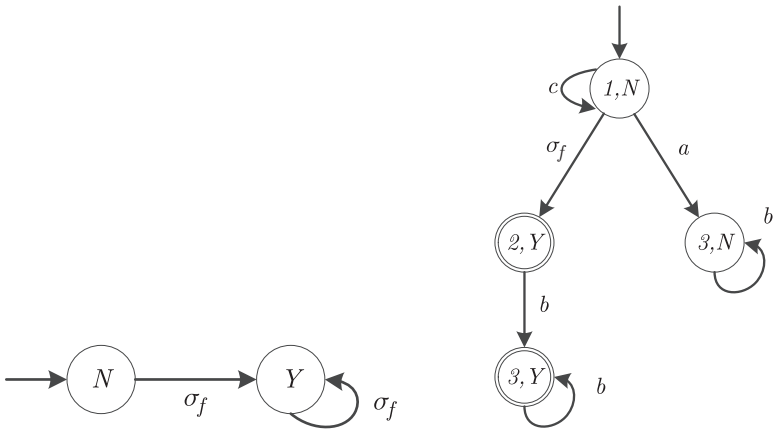
Exemplo 2.6. *Considere o autômato G, cujo diagrama de transição de estados é apresentado na figura 2.8. Considere também que os conjuntos de eventos observáveis e não observáveis são dados por $\Sigma_o = \{a, b, c\}$ e $\Sigma_{uo} = \{\sigma_f\}$, respectivamente. Seja $\Sigma_f = \{\sigma_f\}$. O passo 1 do algoritmo 2.3 consiste na construção do*

autômato G_N que modela o comportamento normal do sistema. Para tanto, deve-se criar o autômato A_N, mostrado na figura 2.10(a) e realizar a operação produto $G_N = G \times A_N$, cujo resultado é mostrado na figura 2.10(b). O passo seguinte requer o cálculo de G_F. Para tanto, deve-se obter os autômatos A_{label} (figura 2.11(a)) e G_l (figura 2.11(b)), e tomar a parte coacessível de G_l. Neste exemplo, como todos os eventos de G_N são observáveis, o autômato $G_{N,1} = G_N$. Seguindo o passo 4, o verificador é obtido ao realizar-se a composição paralela entre $G_{N,1}$ e G_F, mostrado na figura 2.13. Como G_V não possui caminhos cíclicos ambíguos, então é possível afirmar que a linguagem L gerada por G é diagnosticável com relação a $P_o : \Sigma^* \to \Sigma_o^*$ e Σ_f, o que já era esperado uma vez que a mesma conclusão foi obtida no exemplo 2.5 utilizando-se um diagnosticador.



(a) Autômato A_N. (b) Autômato G_N.

Figura 2.10: Obtenção do autômato G_N que modela a parte normal de G.



(a) Autômato rotulador A_l. (b) Autômato G_l.

Figura 2.11: Passos para a construção do autômato G_F.

Até o momento, foram apresentadas condições para a diagnosticabilidade de linguagens regulares utilizando-se diagnosticadores e verificadores. Contudo, uma questão pode ser levantada: caso o sistema seja diagnosticável considerando-se todos os eventos do conjunto Σ_o como observáveis, seria possível que o mesmo sistema

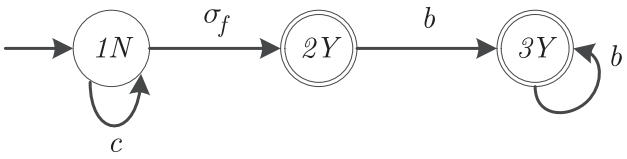


Figura 2.12: Autômato G_F que modela a parte de falha de G.

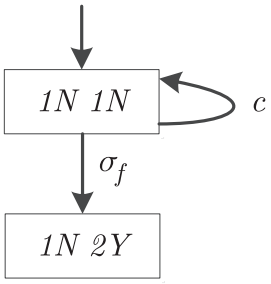


Figura 2.13: Verificador G_V.

continuasse sendo diagnosticável com relação a uma projeção $P_o{}' : \Sigma^* \rightarrow \Sigma_o'^*$ e Σ_f, sendo $\Sigma_o{}' \subset \Sigma_o$? Nos próximos capítulos será visto que a resposta pode ser positiva e, nesses casos, será possível realizar o diagnóstico de falhas em um sistema a eventos discretos utilizando um número menor de sensores, ou ainda utilizar diferentes conjuntos de sensores para se realizar o mesmo diagnóstico, o que pode agregar maior confiabilidade e robustez ao sistema de diagnóstico de falhas. O capítulo 3 apresenta uma breve revisão do trabalho realizado em [29], que propõe um algoritmo, baseado em autômatos diagnosticadores, para se obter todos os conjuntos mínimos de eventos observáveis (sem redundâncias) que permitem que uma falha seja diagnosticada. Antes de encerrar este capítulo, faz-se necessário apresentar uma breve descrição dos algoritmos de busca em grafos orientados e complexidade e análise de algoritmos.

2.9 Algoritmos de busca e complexidade computacional

No contexto de computação, um grafo orientado $G = (V, E)$, em que V denota o conjunto de vértices do grafo e E denota o conjunto de arcos orientados, pode ser representado utilizando listas de adjacências ou matriz de adjacências. A representação por listas de adjacências é usualmente preferida devido à forma compacta de se representar grafos esparsos. Porém, a matriz de adjacências pode ser preferível quando o grafo G for muito denso, ou seja, quando $|E|$ for suficientemente próxima

de $|V|^2$ [37].

A representação de um grafo G por lista de adjacências consiste em um *array* contendo todos os vértices adjacentes para cada um dos vértices do grafo. Para cada $u \in V$, a lista de adjacências contém todos os vértices v tais que existe um arco $(u, v) \in E$. A figura 2.15 mostra a representação do grafo G dado na figura 2.14.

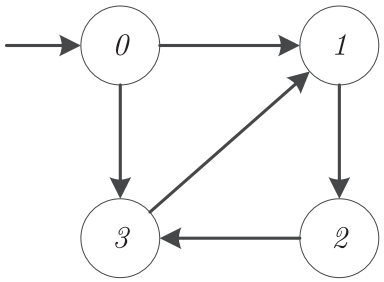


Figura 2.14: Grafo orientado G.

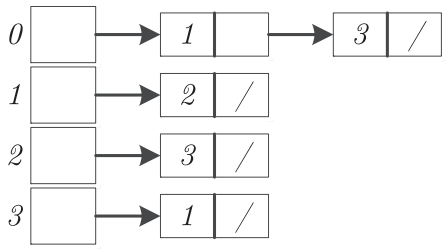


Figura 2.15: Representação do grafo G por lista de adjacências.

Os algoritmos utilizados para determinar se uma linguagem é ou não diagnosticável com relação a uma projeção P_o e um conjunto de falhas Σ_f baseiam-se na construção de autômatos, seguindo algumas regras, e na busca de caminhos cíclicos que violem as condições de diagnosticabilidade do sistema. Essa busca por caminhos cíclicos é realizada percorrendo-se todo o autômato verificador ou diagnosticador. Existem duas formas usuais de se percorrer um autômato: busca em largura e busca em profundidade.

2.9.1 Algoritmos de busca em um grafo orientado

Dado um grafo $G = (V, E)$ e um vértice de partida v_s, o algoritmo de busca em largura explorará cada arco de G afim de encontrar todos os vértices alcançáveis a partir de v_s. Para cada vértice v alcançável a partir de v_s, o caminho encontrado pelo algoritmo de busca em largura corresponde ao menor caminho de v_s até v em G, ou seja, o caminho de v_s até v que contém o menor número de arcos.

A busca em largura, como o próprio nome sugere, tem por objetivo descobrir todos os vértices a uma distância k de v_s antes de descobrir qualquer vértice a uma distância $k+1$. Segue abaixo o algoritmo de busca em largura, apresentado na forma de pseudo-código [37].

Algoritmo 2.4. *(Busca em largura)*

Passo 1) *Para cada vértice* $u \in V[G] - \{v_s\}$*:*

Passo 1.1) *Faça* $cor[u] \leftarrow BRANCO$.

Passo 1.2) $d[u] \leftarrow \infty$.

Passo 1.3) $\pi[u] \leftarrow NIL$.

Passo 2) $cor[v_s] \leftarrow CINZA$.

Passo 3) $d[v_s] \leftarrow 0$.

Passo 4) $\pi[v_s] \leftarrow NIL$.

Passo 5) $Q \leftarrow \emptyset$.

Passo 6) $ENQUEUE(Q, v_s)$.

Passo 7) *Enquanto* $Q \neq \emptyset$*:*

Passo 7.1) *Faça* $u \leftarrow DEQUEUE(Q)$.

Passo 7.2) *Para cada* $v \in Adj[u]$*:*

Passo 7.2.1) *Faça se* $cor[v] = BRANCO$*:*

Passo 7.2.1.1) *Então* $cor[v] \leftarrow CINZA$.

Passo 7.2.1.2) $d[v] \leftarrow d[u] + 1$.

Passo 7.2.1.3) $\pi[v] \leftarrow u$.

Passo 7.2.1.4) $ENQUEUE(Q, v)$.

Passo 7.3) $cor[u] \leftarrow PRETO$.

Observação 2.7. *A distância entre o vértice de partida* v_s *até o vértice* u *é armazenada na variável* $d[u]$*, enquanto seu ancestral é armazenado na variável* $\pi[u]$*. Já a variável* $cor[u]$ *tem como objetivo indicar se um vértice já foi visitado anteriormente. Observe que todos os vértices são iniciados com a cor branca e, quando um vértice é descoberto, o mesmo deverá ter cor cinza ou preta. Se* $(u, v) \in E$ *e o vértice* u *está com sua cor preta, então o vértice* v *e todos os demais adjacentes de* u *já foram descobertos. Vértices em cinza podem ter nós adjacentes ainda não descobertos, ou seja, de cor branca.*

O algoritmo de busca em profundidade, diferentemente do algoritmo de busca em largura, progride a partir da expansão do vértice de partida v_s até encontrar um vértice que não possua arcos novos a serem explorados. Então, a busca retrocede até que um vértice com arcos não explorados seja encontrado, reiniciando o processo até que todos os vértices sejam encontrados. O algoritmo de busca em profundidade e um algoritmo auxiliar, denominado $VISITE$, são apresentados a seguir na forma de pseudo-código [37].

Algoritmo 2.5. *(Busca em profundidade)*

Passo 1) *Para cada vértice* $u \in V[G]$*:*

Passo 1.1) *Faça* $cor[u] \leftarrow BRANCO$.

Passo 1.2) $\pi[u] \leftarrow NIL$.

Passo 2) $tempo \leftarrow 0$.

Passo 3) *Para cada vértice* $u \in V[G]$*:*

Passo 3.1) *Faça se* $cor[u] = BRANCO$*:*

Passo 3.1.1) *Então* $VISITE(u)$.

Algoritmo 2.6. *(Algoritmo auxiliar do algoritmo de busca em profundidade)*

$VISITE(u)$

Passo 1) $cor[u] \leftarrow CINZA$.

Passo 2) $tempo \leftarrow tempo + 1$.

Passo 3) $d[u] \leftarrow tempo$.

Passo 4) *Para cada vértice* $v \in Adj[u]$*:*

Passo 4.1) *Faça se* $cor[v] = BRANCO$*:*

Passo 4.1.1) *Então* $\pi[v] \leftarrow u$.

Passo 4.1.2) $VISITE(v)$.

Passo 5) $cor[u] \leftarrow PRETO$.

Passo 6) $f[u] \leftarrow tempo \leftarrow tempo + 1$.

Observação 2.8. *Note que o algoritmo de busca em profundidade armazena o instante no qual um vértice u foi descoberto na variável d[u] e também o instante no qual o vértice u não possui mais qualquer arco ativo e é finalizado na variável f[u]. Então, o vértice u será branco nos instantes anteriores a d[u], será cinza entre os instantes d[u] e f[u] e será preto após f[u].*

Observação 2.9. *Observe que a função $VISITE(u)$ é chamada recursivamente até que não haja vértices adjacentes a serem descobertos.*

Os exemplos a seguir mostram a evolução passo a passo de cada um dos dois algoritmos de busca apresentados.

Exemplo 2.7. *Seja G o grafo orientado mostrado na figura 2.16. As figuras 2.17 e 2.18 mostram o desenvolvimento dos algoritmos de busca em largura e profundidade, respectivamente.*

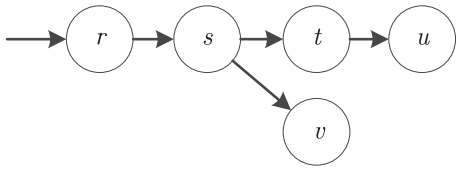


Figura 2.16: Grafo orientado G.

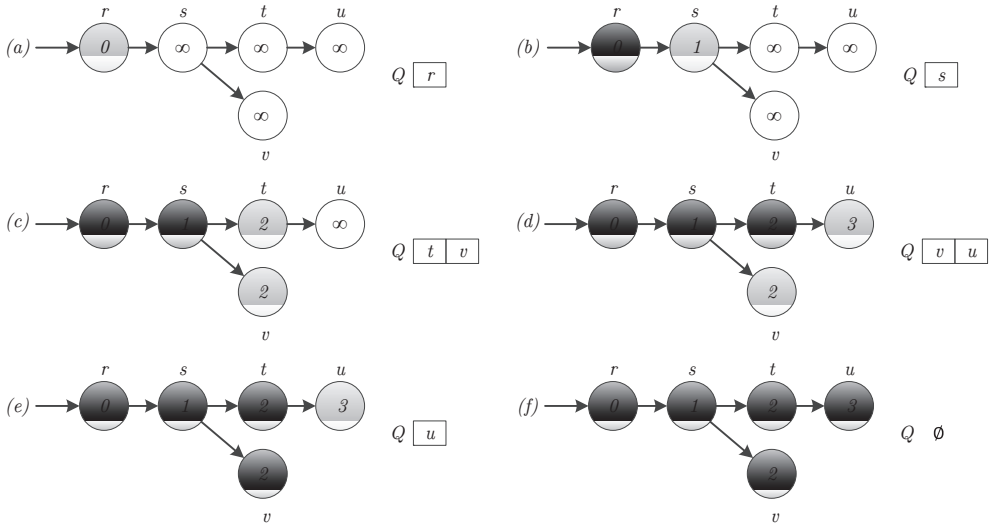


Figura 2.17: Algoritmo de busca em largura aplicado ao grafo G da figura 2.16.

Observe que, ao utilizar o algoritmo de busca em largura, todos os vértices a menos do vértice de partida v_s são inicializados na cor branca e com distância infinita,

uma vez que os mesmos ainda não foram alcançados. Em seguida, o vértice r é pintado de cinza e colocado na fila Q, como mostrado na figura 2.17(a). Como r possui apenas um arco, que o liga ao vértice s, então s é posto na fila e r é colorido de preto, indicando que o mesmo já não possui vértices adjacentes não encontrados, mostrado na figura 2.17(b). Como existem dois vértices adjacentes a s, ambos são colocados na fila Q, e o vértice s pode ser marcado com a cor preta, conforme mostra a figura 2.17(c). Posteriormente, um dos vértices adjacentes a s deve ser visitado. Ao visitar t, o vértice u é pintado de cinza e inserido na fila Q após v, como mostrado na figura 2.17(d). Então, os vértices v e u são visitados (figuras 2.17(e) e (f), respectivamente) e pintados de preto e, como a fila Q se torna vazia, o algoritmo de busca em largura é finalizado.

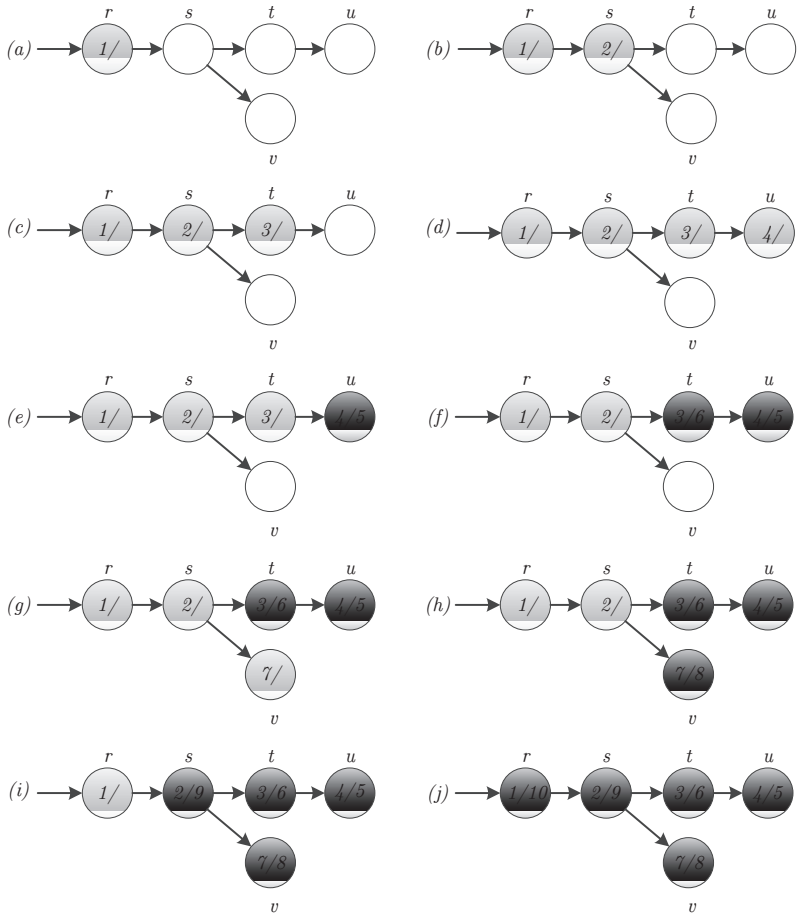


Figura 2.18: Algoritmo de busca em profundidade aplicado ao grafo G da figura 2.16.

Ao executar o algoritmo de busca em profundidade, todos os vértices são inicializados na cor branca e, em seguida, o vértice de partida é visitado e pintado de cinza, conforme mostrado na figura 2.18(a). Como o vértice r possui apenas um arco, que

o conecta ao vértice s, então s é visitado e colorido de cinza, como apresentado na figura 2.18(b). As figuras 2.18(c) e (d) mostram que após s, os vértices t e u são visitados e, uma vez que u não possui vértices adjacentes, a busca retrocede até o vértice v ser encontrado, conforme mostrado nas figuras 2.18(e), (f) e (g).

Note que, ao alcançar o vértice s, o algoritmo de busca em largura inclui na fila todos os seus vértices adjacentes, t e v, enquanto o algoritmo de busca em profundidade seleciona apenas um de seus arcos, t, e o explora até que não hajam mais vértices adjacentes a serem visitados.

Outro ponto muito importante no contexto de autômatos e diagnóstico de falhas trata-se da eficiência com que um algoritmo é capaz de verificar a diagnosticabilidade da linguagem de um sistema. Para tal, uma breve descrição sobre complexidade computacional é apresentada a seguir.

2.9.2 Complexidade computacional de algoritmos

Para determinar se um determinado algoritmo realiza uma tarefa com melhor eficiência que outro existente, é necessário conhecer a possível taxa de crescimento de cada um deles, também chamada de complexidade computacional. Essa complexidade representa uma estimativa de quanto tempo um algoritmo precisará para processar uma entrada de tamanho n. O tempo de processamento é calculado por uma função matemática que caracteriza a ordem de crescimento do algoritmo [37].

Tradicionalmente, a eficiência de um algoritmo é determinada considerando-se apenas os termos de maior ordem presentes na expressão matemática que define a eficiência de um algoritmo [37]. Uma vez que a análise é realizada para o pior cenário possível, as constantes e termos de baixa ordem tornam-se irrelevantes ao atribuir-se um valor elevado para o tamanho da entrada (n). Essa é a idéia da análise de eficiência assintótica de um algoritmo. A seguir, é apresentada a notação assintótica usada nesta dissertação para avaliar a complexidade dos algoritmos apresentados.

Definição 2.25. *(Notação O) Para uma dada função $g(n)$, denotamos por $O(g(n))$ o conjunto de funções*

$$O(g(n)) = \{f(n) : \textit{existe } c \in \mathbb{R}_+^* \textit{ e } n_0 \in \mathbb{N} : 0 \leq f(n) \leq cg(n),\ \forall n \geq n_0\}.$$

A notação assintótica O é usada para atribuir um limitante superior a uma função. Diz-se que uma função $f(n) = O(g(n))$ quando deseja-se indicar que a função $f(n)$ faz parte do conjunto $O(g(n))$. Isso implica que $cg(n) \geq f(n), \forall n \geq n_0$, podendo $f(n)$ estar suficientemente próxima da função $g(n)$ ou não. Note que $(an^2 + bn + c) = O(n^2)$, assim como $(an^2 + b) = O(n^2)$. Tecnicamente, quando se diz que a complexidade computacional de um algoritmo é da ordem de $O(g(n))$,

por exemplo, quer dizer que, no pior caso, independentemente da forma da entrada de tamanho n, o tempo de execução desse algoritmo está limitado superiormente pelo valor de $g(n)$ e de sua taxa de crescimento. 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.

Note que, tanto para grafos orientados quanto para os não orientados, o consumo de memória ao utilizar a lista de adjacências para representar um grafo G é da ordem de $O(|V|+|E|)$, enquanto a matriz de adjacências representa um consumo da ordem de $O(|V|^2)$, independentemente do número de arcos do grafo.

Os algoritmos de busca em largura e em profundidade possuem, de acordo com [37], complexidade de ordem $O(|V|+|E|)$. 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 $|X| + (|X| \times |\Sigma|)$. Como $|X| + (|X| \times |\Sigma|) = |X|(1 + |\Sigma|) \leq 2|X| \times |\Sigma|$, a complexidade dos algoritmos de busca apresentados é da ordem $O(|X| \times |\Sigma|)$. Então, como a verificação da diagnosticabilidade de uma linguagem exige a busca por caminhos cíclicos com determinadas características em um diagnosticador ou verificador, vide teoremas 2.1 e 2.2, a eficiência de um algoritmo de verificação das propriedades de diagnosticabilidade da linguagem de um SED está relacionada à cardinalidade do número de estados e eventos dos autômatos nos quais são realizadas essas buscas. Uma vez que o número de estados dos diagnosticadores pode apresentar crescimento exponencial, é esperado que um algoritmo para verificar a diagnosticabilidade da linguagem de um sistema utilizando verificadores, cujo número de estados apresenta, no pior caso, crescimento polinomial, possua menor custo computacional se comparado a um baseado em diagnosticadores.

O problema de se encontrar todos os subconjuntos (bases para o diagnóstico de falhas) do conjunto de eventos observáveis que são essenciais para o diagnóstico de falha em um sistema a eventos discretos é classificado como um problema NP-Completo [8], [33]. Diz-se que um problema é NP-Completo quando não existe um algoritmo de complexidade polinomial capaz de solucioná-lo. Problemas NP-Completos são, atualmente, um dos maiores desafios da área de computação, uma vez que sequer existem provas de que não existem algoritmos polinomiais capazes de solucioná-los. O motivo pelo qual o problema de se encontrar todos os subconjuntos formados por eventos observáveis que garantam a diagnosticabilidade do sistema é NP-Completo é o possível crescimento exponencial do número de ciclos nos autômatos utilizados para o diagnóstico de falhas. Como os algoritmos são analisados para o pior cenário possível, as soluções exibidas até hoje são de ordem

exponencial ou ainda piores. Isso significa que realizar a força bruta, que consiste em verificar cada subconjunto possível de ser formado pelos elementos do conjunto de eventos observáveis, pode ser, em determinados casos, mais vantajoso que implementar os algoritmos existentes atualmente. Esta dissertação tem por objetivo apresentar dois novos algoritmos para tratar o problema mencionado acima, sendo ambos mais eficientes que os demais apresentados na literatura.

Capítulo 3

Bases mínimas para o diagnóstico de falhas em SEDs utilizando diagnosticadores

No capítulo 2, o problema de diagnóstico de falhas em um sistema a eventos discretos foi formulado e foram apresentadas duas maneiras de se determinar se a linguagem de um sistema é ou não diagnosticável: utilizando-se autômatos diagnosticadores ou verificadores. Uma questão, entretanto, foi levantada: sendo a linguagem L gerada pelo sistema diagnosticável com relação a $P_o : \Sigma \rightarrow \Sigma_o$ e Σ_f, seria possível obter um subconjunto $\Sigma_o^{'} \subset \Sigma_o$ tal que L seja diagnosticável com relação a $P_o^{'} : \Sigma \rightarrow \Sigma_o^{'}$ e Σ_f? Em caso positivo, é possível planejar o sistema para realizar o diagnóstico de falhas utilizando um menor número de sensores, ou então tirar proveito dos sensores redundantes para agregar confiabilidade e robustez ao diagnóstico realizado. Neste capítulo, serão mostradas as condições necessárias e suficientes para que isso seja possível, assim como mostrado o algoritmo desenvolvido em [29] para a obtenção de todos os subconjuntos de Σ_o que garantem que a diagnosticabilidade da linguagem do sistema é mantida. Tal algoritmo é baseado em autômatos diagnosticadores e utiliza apenas as propriedades estruturais do sistema em análise, evitando assim a utilização da força bruta. Antes da apresentação do algoritmo em si, é necessária a definição de diagnosticadores com observação reduzida e diagnosticadores teste, além de outras definições apresentadas em [29].

3.1 Diagnóstico de falhas com observação reduzida

O problema de se obter subconjuntos do conjunto de eventos observáveis Σ_o que mantêm inalterada a diagnosticabilidade da linguagem do sistema será denominado

neste trabalho de diagnóstico de falhas com observação reduzida. Para abordar esse problema, deve-se considerar, juntamente com as hipóteses **H1**-**H3** realizadas no capítulo 2, a seguinte hipótese:

- **H4** - A linguagem L gerada por G é diagnosticável com relação a $P_o : \Sigma^* \rightarrow \Sigma_o^*$ e Σ_f.

Seja $G_d^{'} = (X_d^{'}, \Sigma_o^{'}, f_d^{'}, \Gamma_d^{'}, x_{0_d}^{'})$ o diagnosticador obtido supondo observação reduzida, isto é, $G_d^{'}$ pode observar apenas os eventos pertencentes a $\Sigma_o^{'} \subset \Sigma_o$. Então, o seguinte resultado pode ser enunciado.

Teorema 3.1. *O diagnosticador com observação reduzida* $G_d^{'}$ *e* $\widehat{G}_d^{'} = Obs(G_d, \Sigma_o^{'})$ $= (\widehat{X}_d^{'}, \Sigma_o^{'}, \widehat{f}_d^{'}, \widehat{\Gamma}_d^{'}, \widehat{x}_{0_d}^{'})$ *(o observador de* G_d *com relação a projeção* $P_{oo'} : \Sigma_o^* \rightarrow \Sigma_o^{'*}$*) são iguais considerando-se a seguinte correlação de estados:*

$$\widehat{x}_d^{'} = \{x_{d_1}, x_{d_2}, \ldots, x_{d_n}\} \in \widehat{X}_d^{'},\ x_{d_i} \in X_d \Leftrightarrow x_d^{'} = \bigcup_{i=1}^{n} x_{d_i} \in X_d^{'}.$$

Prova. Vide [29]. □

Observação 3.1. *Note que, o resultado do teorema 3.1 implica que, caso o diagnosticador* G_d *já tenha sido obtido, então não é necessário construir* $G_d^{'}$ *a partir de* G*, mas diretamente a partir de* G_d*. Para tal, deve-se fazer a fusão dos estados de* G_d *conectados por transições rotuladas por eventos de* $\Sigma_o \setminus \Sigma_o^{'}$ *em um único estado, ou seja, implementar um novo alcance não-observável.*

Embora a hipótese **H1** estabeleça que a linguagem gerada por G seja viva, é possível que, ao se fazer a fusão de estados de G_d, a linguagem gerada por $G_d^{'}$ não seja viva. Isso ocorre quando um caminho cíclico em G_d é formado a partir de transições rotuladas por eventos que se tornaram não observáveis para o diagnosticador com observação reduzida, produzindo um único estado em $G_d^{'}$. Nesses casos, embora o diagnosticador com observação reduzida não detecte alteração do estado atual de G, sabe-se que o estado atual do mesmo está mudando ciclicamente, o que caracteriza a existência de um ciclo escondido em $G_d^{'}$, cuja definição será formalizada a seguir [29].

Definição 3.1. *(Ciclos escondidos) Seja* $x_d^{'} \in X_d^{'}$ *um estado obtido agrupando-se os estados* $x_{d_1}, x_{d_2}, \ldots, x_{d_n} \in X_d$*. Então, existe um ciclo escondido em* $x_d^{'}$ *se para algum* $\{i_1, i_2, \ldots, i_k\} \subset \{1, 2, \ldots, n\}$*,* $\{x_{d_{i_1}}, x_{d_{i_2}}, \ldots, x_{d_{i_k}}\}$ *forma um ciclo em* G_d*.*

Definição 3.2. *(Ciclos escondidos indeterminados) Seja* $x_d^{'} \in X_d^{'}$ *e suponha que existe um ciclo escondido* $\{x_{d_{i_1}}, x_{d_{i_2}}, \ldots, x_{d_{i_n}}\}$ *em* $x_d^{'}$*. Se* $x_{d_{i_1}}, x_{d_{i_2}}, \ldots, x_{d_{i_k}}$ *são estados certos e* $x_d^{'}$ *é um estado incerto, então, o ciclo escondido é indeterminado.*

Observação 3.2.

(*a*) *Ao considerar a existência de ciclos escondidos, as hipóteses* ***H1*** *e* ***H2*** *podem ser relaxadas.*

(*b*) *Observe que os estados* $x_{d_{i_k}}$ *que formam um ciclo escondido em* $x_d^{'} \in X_d^{'}$ *podem ser certos, normais ou incertos. Entretanto, ciclos escondidos formados por estados incertos não são indeterminados, conforme a definição dada acima.*

(*c*) *Ciclos indeterminados que não são escondidos serão referidos daqui em diante como ciclos observados indeterminados.*

Os ciclos escondidos serão representados nos diagramas de transição de estados dos diagnosticadores com observação reduzida por laços tracejados: os ciclos escondidos indeterminados serão rotulados como *ihc* (do inglês indeterminate hidden cycle) e os demais ciclos escondidos serão identificados simplesmente como *hc*.

O teorema a seguir, apresentado em [29], provê uma condição necessária e suficiente para o diagnóstico de falhas sob observação reduzida.

Teorema 3.2. *Suponha que a linguagem* L *seja diagnosticável com relação a projeção* P_o *e* Σ_f*. Então,* L *também será diagnosticável com relação a projeção* $P_o^{'} : \Sigma^* \rightarrow \Sigma_o^{'}$, $\Sigma_o^{'} \subset \Sigma_o$ *e* $\Sigma_f = \{\sigma_f\}$ *se, e somente se,* $G_d^{'}$ *não tiver nenhum ciclo indeterminado (escondido ou observado).*

Prova. Vide [29]. □

3.2 Bases para o diagnóstico de falhas

O resultado apresentado no teorema 3.2 mostra que é possível que exista um subconjunto de Σ_o tal que a diagnosticabilidade da linguagem do sistema seja preservada, porém, para que isso ocorra, algumas condições têm de ser satisfeitas. As definições de bases e bases mínimas para o diagnóstico se fazem necessárias nesse momento.

Definição 3.3. *(Bases para o diagnóstico) O conjunto* $\Sigma_o^{'} \subseteq \Sigma_o$ *é uma base para o diagnóstico de* L *se* L *é diagnosticável com relação a projeção* $P_o^{'} : \Sigma^* \rightarrow \Sigma_o^{'*}$ *e* $\Sigma_f = \{\sigma_f\}$.

Definição 3.4. *(Bases mínimas para o diagnóstico) O conjunto* $\Sigma_o^{'} \subseteq \Sigma_o$ *é uma base mínima para o diagnóstico se* $\Sigma_o^{'}$ *é uma base para o diagnóstico e, para todo subconjunto não vazio* $\Sigma_o^{''} \subset \Sigma_o^{'}$*,* L *não é diagnosticável com relação a projeção* $P_o^{''} : \Sigma^* \rightarrow \Sigma_o^{''*}$ *e* $\Sigma_f = \{\sigma_f\}$.

Note que a diferença entre as definições 3.3 e 3.4 está na relevância da presença dos eventos na composição de cada conjunto. Os eventos presentes em uma base mínima são estritamente essenciais para que a falha seja diagnosticada, isto é, a remoção de qualquer um deles resulta na não diagnosticabilidade da falha. Em contrapartida, uma base não mínima possui eventos redundantes, ou seja, nem todos os eventos que a formam são de fato necessários para o diagnóstico da falha em questão.

Com as definições 3.3 e 3.4 dadas acima, o problema de se encontrar todos os subconjuntos de Σ_o que garantam a diagnosticabilidade da linguagem do sistema pode ser formulado da seguinte forma: dado um autômato $G = (X, \Sigma, f, \Gamma, x_0)$, sendo $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, e supondo que Σ_o é uma base para o diagnóstico, encontre todos os subconjuntos $\Sigma_o' \in 2^{\Sigma_o} \setminus \{\Sigma_o, \emptyset\}$ que também sejam bases para o diagnóstico de L.

3.2.1 Conjunto de eventos elementares para o diagnóstico

O trabalho apresentado em [29] tem por objetivo explorar a estrutura do sistema, utilizando as propriedades do autômato diagnosticador, de forma a obter sistematicamente as bases mínimas para o diagnóstico de L. Conforme a premissa **H4**, a linguagem L é diagnosticável considerando-se o conjunto Σ_o e, dessa forma, o diagnosticador G_d não possui ciclos indeterminados. Isso implica que todos os estados incertos de G_d se tornam certos ou normais após um número limitado de transições. Então, ao menos um evento que compõe cada caminho que leva um estado incerto de G_d a um estado certo deverá fazer parte do conjunto de bases mínimas para o diagnóstico, pois, caso contrário, existirá um ciclo escondido indeterminado que fará com que a linguagem L seja não diagnosticável. Essa idéia será formalizada no algoritmo 3.1, mas antes, é necessário apresentar algumas definições e notações que serão utilizadas no decorrer do trabalho.

Suponha que $x_{d_{YN}}$, x_{d_Y} e $x_{d_N} \in X_d$ denotem, respectivamente, estados incertos, certos e normais de G_d. Devido à hipótese **H4**, é possível definir o seguinte subconjunto de X_d:

$$X_{YN}^Y = \{x_{d_{YN}} \in X_d : (\exists (x_{d_Y}, \sigma) \in X_d \times \Sigma_o)[f_d(x_{d_{YN}}, \sigma) = x_{d_Y}]\}.$$

Note que, para cada estado de X_{YN}^Y, é sempre possível definir, ao menos um caminho $P_Y = (x_{d_{YN}}, \sigma_0, x_{d_{Y,1}}, \sigma_1, \ldots, \sigma_{n-1}, x_{d_{Y,n}})$ que satisfaz às seguintes condições: *(i)* $x_{d_{Y,n}} = x_{d_{Y,i}}$ para algum $i \in \{1, 2, \ldots, n-1\}$, isto é, os estados $\{x_{d_{Y,i}}, x_{d_{Y,i+1}}, \ldots, x_{d_{Y,n}}\}$ formam um ciclo; *(ii)* $\{x_{d_{Y,i}}, x_{d_{Y,i+1}}, \ldots, x_{d_{Y,n}}\}$ é o único ciclo existente no caminho. O conjunto X_{YN}^Y será referido como conjunto de estados-origem de caminhos de falha e o caminho P_Y como caminho de falha. Os elementos de X_{YN}^Y são chamados estados-origem de caminhos de falha.

Definição 3.5. *(Evento de um caminho de falha e conjunto de eventos de um caminho de falha)*

A. *Um evento $\sigma \in \Sigma_o$ é um evento de um caminho de falha se ele pertence a qualquer caminho de falha definido para algum estado X_{YN}^{Y}.*

B. *Um conjunto de eventos de um caminho de falha, denotado por Σ_{fpes}, é um conjunto formado por todos os eventos de um caminho de falha.*

A definição 3.5 permite que se estabeleça uma condição necessária para que um conjunto $\Sigma_o^{'} \subset \Sigma_o$ seja uma base para o diagnóstico.

Teorema 3.3. *Seja N_{fpes} o número de caminhos de falha de G_d. Então, uma condição necessária para que $\Sigma_o^{'} \subset \Sigma_o$ seja uma base para o diagnóstico da linguagem gerada por G e Σ_f é*

$$\Sigma_o^{'} \cap \Sigma_{fpes,i} \neq \emptyset, i = 1, 2, \ldots, N_{fpes},$$

em que $\Sigma_{fpes,i}$ denota o conjunto de eventos do i-ésimo caminho de falha de G_d.

Prova. Vide [29]. □

Observação 3.3. *Note que a condição dada no teorema 3.3 é apenas necessária. Então, é possível que, mesmo que ela seja atendida, $\Sigma_o^{'}$ não seja uma base para o diagnóstico de falhas de L. A condição necessária e suficiente para que $\Sigma_o^{'}$ seja uma base para diagnóstico foi apresentada no teorema 3.2.*

Conforme mostrado no teorema 3.3, para que a ocorrência da falha seja diagnosticável, pelo menos um evento de cada caminho de falha deve ser observável. Esse fato será utilizado para se construir um conjunto que será o ponto de partida na busca das bases de diagnosticabilidade do sistema, denominado de conjunto de eventos elementares para o diagnóstico. A seguir, será realizada a definição da operação produto união, que possibilitará definir os conjuntos de eventos elementares para o diagnóstico de falhas.

Definição 3.6. *(Produto união) O produto união dos conjuntos Σ_i, $i = 1, 2, \ldots, n$, denotado por $\Sigma_1 \dot{\times} \ldots \dot{\times} \Sigma_n$, é realizado como mostrado abaixo:*

$$\Sigma_1 \dot{\times} \ldots \dot{\times} \Sigma_n = \begin{cases} \{\Sigma_e = \Sigma_{e,1} \cup \ldots \cup \Sigma_{e,n} : \Sigma_{e,i} \in \Sigma_i,\ i = 1,\ 2,\ \ldots,\ n\},\ \textit{se os} \\ \qquad\qquad\qquad\qquad\qquad\qquad \textit{elementos de } \Sigma_i \textit{ são conjuntos,} \\ 2_1^{\Sigma_1} \dot{\times} 2_1^{\Sigma_2} \dot{\times} \ldots \dot{\times} 2_1^{\Sigma_n},\ \textit{caso contrário,} \end{cases}$$

sendo $2_1^{\Sigma} = \{\widetilde{\Sigma} \in 2^{\Sigma} : |\widetilde{\Sigma}| = 1\}$.

Definição 3.7. *(Conjunto de eventos elementares para o diagnóstico) Suponha que* $\Sigma_{fpes,i}$, $i = 1, 2, \ldots, N_{fpes}$ *sejam conjuntos de eventos dos caminhos de falha de* G_d. *Então, o conjunto formado pelos conjuntos de eventos elementares para o diagnóstico de* G_d *é definido como:*

$$\Sigma_{edes} = \Sigma_{fpes,1} \dot{\times} \Sigma_{fpes,2} \dot{\times} \ldots \dot{\times} \Sigma_{fpes,N_{fpes}}. \tag{3.1}$$

O algoritmo a seguir sugere uma forma sistemática de se encontrar todos os conjuntos de eventos elementares para o diagnóstico de G_d.

Algoritmo 3.1. *(Algoritmo para encontrar todos os conjuntos de eventos elementares para o diagnóstico de* G_d*)*

Passo 1) *Construa o diagnosticador* G_d *e encontre o conjunto de estados-origem de caminhos de falha* (X_{YN}^Y) *de* G_d. *Defina* $|X_{YN}^Y| = N_{YN}$.

Passo 2) *Para cada estado-origem* $x_{d_{YN,i}} \in X_{YN}^Y$, $i = 1, 2, \ldots, N_{YN}$ *construa uma árvore a partir de* $x_{d_{YN,i}}$, *da seguinte forma:*

Passo2.1) *Defina* $\Gamma_d^Y(x_{d_{YN,i}}) = \{\sigma \in \Gamma_d(x_{d_{YN,i}}) : f_d(x_{d_{YN,i}}, \sigma) = x_{d_Y}\}$ *e suponha que* $|\Gamma_d^Y(x_{d_{YN,i}})| = \eta_{YN,i}$. *Crie* $\eta_{YN,i}$ *descendentes de* $x_{d_{YN,i}}$ *e rotule-os como* x_{d_Y}, *sendo* $x_{d_Y} = f_d(x_{d_{YN,i}}, \sigma)$, $\sigma \in \Gamma_d^Y(x_{YN,i})$. *Rotule os ramos* $(x_{d_{YN,i}}, x_{d_Y})$*com* σ*;*

Passo2.2) *Um nó rotulado como* x_{d_Y}, *definido na árvore, será uma folha se o estado* x_{d_Y} *já tiver rotulado algum estado no caminho de* $x_{d_{YN,i}}$ *a* x_{d_Y}. *Caso contrário, defina* $|\Gamma_d(x_{d_Y})| = \eta_Y$. *Crie* η_Y *descendentes de* x_{d_Y} *e rotule-os como* $x_{d_{Y,new}}$, *sendo* $x_{d_{Y,new}} = f_d(x_{d_Y}, \sigma)$, $\sigma \in \Gamma_d(x_{d_{YN,i}})$. *Rotule os ramos* $(x_{d_Y}, x_{d_{Y,new}})$ *com* σ. *Repita esse passo até que todos os estados* $x_{d_{Y,new}}$ *sejam folhas;*

Passo 3) *Para cada árvore* T_i, $i = 1, 2, \ldots, N_{YN}$, *identifique as folhas* $x_{d_{Y,i}}^l$, $l = 1, \ldots, l_{T_i}$, *em que* l_{T_i} *é o número de folhas da árvore* T_i. *Forme caminhos* $P_{Y,i}^l$, $l = 1, \ldots, l_{T_i}$, *iniciando em* $x_{d_{YN,i}}$ *e terminando em* $x_{d_{Y,i}}^l$, $l = 1, \ldots, l_{T_i}$ *(esses caminhos são os caminhos de falha que se iniciam em* $x_{d_{YN,i}}$*).*

Passo 4) *Forme os conjuntos de eventos de um caminho de falha* $\Sigma_{fpes,i}^l$, $i = 1, \ldots, N_{YN}$, $l = 1, \ldots, l_{T_i}$ *utilizando os caminhos* $P_{Y,i}^l$ *obtidos no passo anterior.*

Passo 5) *Com os conjuntos obtidos no passo 4, construa o conjunto formado pelos conjuntos de eventos elementares para o diagnóstico* Σ_{edes}, *de acordo com a equação (3.1).*

Observação 3.4. *Observe que, ao final do algoritmo 3.1, é provável que* Σ_{edes} *possua conjuntos tais que* $\widetilde{\Sigma}_{edes} \in \Sigma_{edes}$, $\widehat{\Sigma}_{edes} \in \Sigma_{edes}$ *e* $\widehat{\Sigma}_{edes} \subseteq \widetilde{\Sigma}_{edes}$. *Assim, caso o interesse seja pelos conjuntos de eventos elementares de menor cardinalidade, deve-se acrescentar o seguinte passo ao algoritmo 3.1:*

Passo 6) *Remova de* Σ_{edes} *todos os conjuntos* $\widetilde{\Sigma}_{edes} \in \Sigma_{edes}$ *caso exista* $\widehat{\Sigma}_{edes} \in \Sigma_{edes}$ *tal que* $\widehat{\Sigma}_{edes} \subseteq \widetilde{\Sigma}_{edes}$. *Forme o conjunto* Σ'_{edes} *com os conjuntos restantes de* Σ_{edes}.

Após a execução do algoritmo 3.1, pode-se afirmar que a condição necessária estabelecida no teorema 3.3 foi atendida. Entretanto, de acordo com o teorema 3.2, L será diagnosticável com relação a P'_o e Σ_f se, e somente se, G'_d não tiver nenhum ciclo indeterminado (escondido ou observado). Dessa forma, caso a linguagem do sistema seja diagnosticável considerando-se a observação dos eventos que pertencem a $\Sigma'_o = \Sigma'_{edes}$, obtido no algoritmo 3.1, então Σ'_o é uma base mínima para o diagnóstico de falhas. Caso contrário, será necessário adicionar novos eventos a Σ'_o. Como o interesse está na busca por bases mínimas para o diagnóstico, a inserção de eventos deve ser feita de forma criteriosa, de forma a evitar que Σ'_o possua eventos redundantes.

3.3 Busca pelas bases mínimas para o diagnóstico de falhas

A ideia básica é formar novos conjuntos $\Sigma''_o = \Sigma'_o \cup \{\sigma\}$, sendo σ um evento pertencente a uma sequência ambígua (s_a) ou a uma sequência normal (s_N) de L ($P'_o(s_a) = P'_o(s_N)$) de forma a tornar $P''_o(s_a) \neq P''_o(s_N)$, sendo $P''_o : \Sigma^*_o \rightarrow \Sigma''^*_o$. Entretanto, sequências ambíguas não aparecem claramente em G'_d e, portanto, uma outra forma de se identificar essas sequências deve ser obtida.

Considere o autômato G'_{teste} definido como:

$$G'_{teste} = G'_d \| G_d = (X_t, \Sigma_o, f_t, \Gamma_t, x_{0_t}).$$

Note que um estado x_t de G'_{teste} possui a seguinte estrutura:

$$x_t = (x'_d, x_d),$$

sendo $x'_d \in X'_d$ e $x_d \in X_d$. Um estado x_t de G'_{teste} é dito ser certo se x'_d e x_d são ambos certos e incerto se x_d é certo mas x'_d é incerto. Além disso, um ciclo em G'_{teste} é dito indeterminado se o ciclo correspondente em G'_d (observado ou escondido) for indeterminado.

Teorema 3.4. *Seja L diagnosticável com relação à projeção $P_o : \Sigma^* \rightarrow \Sigma_o^*$ e $\Sigma_f = \{\sigma_f\}$. Então, L será diagnosticável com relação à projeção $P_o^{'} : \Sigma^* \rightarrow \Sigma_o^{'*}$, $\Sigma_o^{'} \subset \Sigma_o$, e $\Sigma_f = \{\sigma_f\}$ se, e somente se, $G_{teste}^{'}$ não possuir ciclos indeterminados.*

Prova. Vide [29]. □

De acordo com o teorema 3.4, a linguagem L não será diagnosticável se $G_{teste}^{'}$ possui um ou mais ciclos indeterminados. Nesse caso, existirão, pelo menos, duas sequências (uma de falha e outra de não falha) em G tais que as suas projeções serão idênticas, considerando-se $\Sigma_o^{'}$ como o conjunto de eventos observáveis. Note que a não-diagnosticabilidade de L com relação a $P_o^{'}$ e Σ_f é devido à existência de ciclos indeterminados observados ou escondidos em $G_d^{'}$. Na sequência, será mostrada uma ligação entre esses ciclos de $G_d^{'}$ e suas projeções inversas em $G_{teste}^{'}$, conforme estabelecido em [29].

De acordo com os teoremas 3.2 e 3.4, a não diagnosticabilidade de L com relação a $P_o^{'}$ e Σ_f implica que ambos, $G_d^{'}$ e $G_{teste}^{'}$ possuem um ou mais ciclos indeterminados. Serão considerados, inicalmente, os ciclos indeterminados observados de $G_d^{'}$. Nesse caso, existem duas sequências t_Y, $t_N \in L$, tais que $t_Y \in L \setminus L_N$ e $t_N \in L_N$ ($s_Y = P_o(t_Y)$ e $s_N = P_o(t_N)$), que satisfazem as seguintes condições:

CO1. $f_d(x_{0_d}, s_Y) = x_{d_Y}$ e $f_d(x_{0_d}, s_N) = x_{d_N}$, sendo x_{d_Y} um estado certo de G_d que pertence a um ciclo de estados certos e x_{d_N} um estado normal ou incerto de G_d, sendo que, caso seja formado por estados incertos, os mesmos não formam um ciclo inderterminado em G_d.

CO2. $P_o^{'}(t_Y) = P_o^{'}(t_N) = s_{YN}^{'}$, sendo $s_{YN}^{'}$ tal que $f_d^{'}(x_{0_d}^{'}, s_{YN}^{'}) = x_{d_{YN}}^{'}$ e $x_{d_{YN}}^{'}$ pertence a um ciclo indeterminado de $G_d^{'}$.

Portanto, para cada ciclo observado indeterminado de $G_d^{'}$ é possível associar pelo menos dois ciclos em $G_{teste}^{'}$: *(i)* um ciclo formado por estados cujas primeiras componentes são estados $x_{d_{YN}}^{'}$ de $G_d^{'}$ que são alcançados através de $s_{YN}^{'}$ e cujas segundas componentes são estados certos x_{d_Y} de G_d alcançados através de s_Y; *(ii)* e outro ciclo formado por estados cujas primeiras componentes são os mesmos estados $x_{d_{YN}}^{'}$ do ciclo anterior, mas cujas segundas componentes são estados normais ou incertos de G_d que não são parte de ciclos indeterminados e são alcançados através de s_N.

Considere agora a existência de ciclos escondidos indeterminados em $G_d^{'}$. Nesse caso, sempre existirão dois traços t_Y e $t_N \in L$, tais que $t_Y \in L \setminus L_N$ e $t_N \in L_N$ ($s_Y = P_o(t_Y)$ e $s_N = P_o(t_N)$), que satisfazem as seguintes condições:

CE1. s_Y é uma sequência arbitrariamente longa e s_N possui comprimento limitado.

CE2. $P_o'(t_Y) = P_o'(t_N) = s'$ também é limitada.

Pode-se concluir que, se G_d' possui um ou mais ciclos indeterminados (escondidos ou observados), a condição necessária para que L seja diagnosticável com relação a $P_o'' : \Sigma^* \rightarrow \Sigma_o''^*$ e Σ_f, sendo $\Sigma_o'' = \Sigma_o' \cup \Sigma_{ies}$ ($\Sigma_{ies} \subseteq \Sigma_o \setminus \Sigma_o'$) é que Σ_{ies} possua pelo menos um evento de s_Y ou s_N tais que $P_o''(t_Y) \neq P_o''(t_N)$.

3.3.1 Caminhos primos e cobertura para um caminho com ciclos internos

Embora a ideia de se adicionar eventos a Σ_o' pareça simples, os eventos devem ser adicionados de maneira ordenada, evitando-se a adição de eventos redundantes, de forma a se garantir que as bases obtidas sejam mínimas. Em [29], foi mostrado que um ciclo pode conter outros ciclos dentro dele. Como consequência, podem existir diversas sequências s_Y e s_N que violam as condições de diagnosticabilidade, mesmo no caso em que existe uma única sequência que conecta o estado inicial de G_{teste}' ao primeiro estado do ciclo. Assim, a escolha dos eventos que serão incluídos no conjunto de eventos observáveis não é uma tarefa direta, uma vez que todos os ciclos de G_{teste}' devem ser considerados. Essa dificuldade será contornada pela substituição de sequências arbitrariamente longas por caminhos de tamanho finito, como mostrado em [29], através da definição de caminhos primos.

Seja $P_l^c = (x_l, \sigma_l, x_{l+1}, \sigma_{l+1}, \ldots, \sigma_{n-1}, x_n, \sigma_n, x_l)$ um caminho do autômato G que possui um ou mais caminhos cíclicos internos, ou seja, x_i não necessariamente é diferente de x_j para $i \neq j$, $i, j \in \{l, l+1, \ldots, n\}$. Considere também um caminho $P_0 = (x_0, \sigma_0, x_1, \sigma_1, \ldots, x_{l-1}, \sigma_{l-1}, P_l^c)$, sendo x_0 o estado inicial de G.

Definição 3.8. *(Caminhos primos) Um caminho P_0 é dito ser primo se $x_i \neq x_j$ para todo $i \neq j$, i, $j \in \{0, 1, 2, \ldots, n\}$.*

A obtenção de todos os caminhos primos de um autômato G pode ser realizado utilizando-se o algoritmo 3.2 desenvolvido em [29], através da construção de uma árvore T com raíz x_0, como mostrado a seguir.

Algoritmo 3.2. *(Algoritmo para obtenção de todos os caminhos primos de um autômato)*

Passo 1) *Rotule a raíz de T com x_0.*

Passo 2) *Defina $|\Gamma(x_0)| = \eta_0$ e $x = f(x_0, \sigma)$, $\sigma \in \Gamma(x_0)$. Crie η_0 descendentes de x_0 e rotule-os com x e os correspondentes ramos (x_0, x) com σ.*

Passo 3) *Um nó rotulado com x, definido na árvore, será uma folha se o estado x já tiver rotulado algum estado no caminho de x_0 a x. Caso contrário, defina*

$|\Gamma(x)| = n$ *e* $x_{new} = f(x, \sigma)$, $\sigma \in \Gamma(x)$. *Crie* n *descendentes de* x *e rotule-os com* x_{new} *e os respectivos ramos* (x, x_{new}) *com* σ. *Repita esse passo até que todos os estados* x_{new} *sejam folhas.*

Passo 4) *Identifique todas folhas* x_l *de* T *e forme todos os possíveis caminhos que se iniciam na raíz e terminam em* x_l.

Note que, quando um caminho P_0 possui ciclos internos, já não existe um único caminho primo. Nesses casos, será necessário dividir P_0 para obter todos os seus respectivos caminhos primos, de forma a não perder informações a respeito dos eventos que aparecem nos ciclos internos. Para tal, seguem as definições de caminhos primos de cobertura e cobertura para um caminho com ciclos internos.

Definição 3.9. *(Caminhos primos de cobertura) Considere o caminho com ciclos internos* $P_0 = (x_0, \sigma_0, x_1, \sigma_1, \ldots, x_{l-1}, \sigma_{l-1}, P_l^c)$. *Um caminho primo de cobertura para* P_0 *é qualquer caminho primo que pode ser obtido de* P_0.

Definição 3.10. *(Cobertura para um caminho com ciclos internos) Seja* $C(P_0) = \{P_{0,1}, P_{0,2}, \ldots, P_{0,\eta}\}$ *o conjunto formado por* η *caminhos primos obtidos a partir de* P_0. *Então,* $C(P_0)$ *será uma cobertura para* P_0 *se e somente se toda transição definida em* P_0 *aparece em pelo menos um caminho primo de* $C(P_0)$.

Voltando ao contexto de diagnóstico de falhas e considerando o autômato G'_{teste}, deve-se considerar a seguinte definição.

Definição 3.11. *(Caminhos primos-Y) Um caminho primo-Y de* G'_{teste} *é um caminho primo cujos estados do seu único ciclo formam um ciclo indeterminado em* G'_{teste}.

Dessa forma, pode-se afirmar que um caminho P_0 de G'_{teste} com caminhos cíclicos internos não possuirá ciclos indeterminados se, e somente se, P_0 não possui caminhos primos-Y. Assim, pode-se concluir que, a existência de ciclos indeterminados em G'_{teste} pode ser evitada se houver garantia de que não existam caminhos primos cujo único ciclo é indeterminado. Esse resultado possibilita que a busca por caminhos com ciclos internos indeterminados seja substituída pela busca por caminhos primos cujo único ciclo é indeterminado.

3.3.2 Análise da diagnosticabilidade para diagnosticadores com ciclos observados indeterminados

Em [29], a análise dos ciclos indeterminados observados são realizadas separadamente dos ciclos indeterminados escondidos. Nesta seção, serão analisados apenas os ciclos observados indeterminados. Como destacado anteriormente, uma vez que

L é diagnosticável com relação a P_o e Σ_f, a existência de ciclos observados indeterminados em $G_d^{'}$ está atrelada à existência de pelo menos dois traços arbitrariamente longos s_Y e $s_N \in L_d$ que satisfazem as condições **CO1** e **CO2**. Conforme definido anteriormente, nenhum ciclo de $G_{teste}^{'}$ possuirá ciclos internos indeterminados se, e somente se, não houverem caminhos primos-Y de cobertura.

Considere os seguintes conjuntos:

$$S_Y = \{s : s \text{ é um traço associado a um caminho primo-Y de } G_{teste}^{'}\}. \tag{3.2}$$

$$S_N = \{s : s \text{ é um traço associado a um caminho primo de } G_{teste}^{'} \text{ que não é um caminho primo-Y e cujas primeiras componentes são estados incertos de um ciclo indeterminado de } G_d^{'}\}. \tag{3.3}$$

O seguinte resultado pode ser apresentado [29].

Teorema 3.5. *Seja L não diagnosticável com relação a $P_o^{'}$ e Σ_f e $\Sigma_o^{''} = \Sigma_o^{'} \cup \Sigma_{ies}$, $\Sigma_{ies} \subseteq \Sigma_o \setminus \Sigma_o^{'}$. Suponha que $G_d^{''}$ denote o diagnosticador com observação reduzida considerando-se $\Sigma_o^{''}$ como o conjunto de eventos observáveis. Além disso, considere $(s_Y, s_N) \in S_Y \times S_N$ satisfazendo $P_o^{'}(s_Y) = P_o^{'}(s_N)$ e o traço $s^{'}$ associado ao caminho primo de $G_d^{'}$ cujo único ciclo é indeterminado (mas não escondido) tal que $s^{'} \in \overline{P_o^{'}(s_Y)}$ e $s^{'} \in \overline{P_o^{'}(s_N)}$. A condição necessária para que $s^{'}$ não seja um traço associado a um caminho primo de $G_d^{''}$ cujo único ciclo é indeterminado é que $\Sigma_{ies} \cap [(\Sigma_{s_Y} \cup \Sigma_{s_N}) \setminus \Sigma_o^{'}] \neq \emptyset$, em que Σ_{s_Y} e Σ_{s_N} denotam, respectivamente, os conjuntos formados com os eventos pertencentes aos traços s_Y e s_N.*

Observação 3.5. *Note que a condição apresentada no teorema 3.5 é apenas necessária, uma vez que, mesmo após a inclusão de um evento em comum entre s_Y e s_N em Σ_{ies}, é possível que $P_o^{''}(s_Y) = P_o^{''}(s_N) = s^{''}$,o que significa que $s^{''}$ está associado a um caminho com ciclos internos indeterminados.*

De acordo com o teorema 3.5, a condição necessária para que um par de traços $(s_Y, s_N) \in S_Y \times S_N$, que satisfaz $P_o^{'}(s_Y) = P_o^{'}(s_N)$, não leve a um caminho com ciclos internos indeterminados em $G_d^{''}$ é que pelo menos um evento de s_Y ou de s_N pertença a Σ_{ies}. Portanto, essa exigência deve ser satisfeita para todos os pares de traços (s_Y, s_N) de $G_{teste}^{'}$ que levam a algum traço $s^{'}$ associado a um caminho primo cujo único ciclo é indeterminado e observado. O algoritmo a seguir, desenvolvido em [29], retorna um conjunto Σ_{ies}^{o} cujos elementos são conjuntos de $\Sigma_o \setminus \Sigma_o^{'}$ que devem ser adicionados a $\Sigma_o^{'}$ para criar candidatos a serem bases mínimas para o diagnóstico. A exigência para utilizar o algoritmo a seguir é que todos os caminhos primos de $G_d^{'}$ e $G_{teste}^{'}$ tenham sido calculados.

Algoritmo 3.3. *(Algoritmo para obtenção dos eventos associados a ciclos indeterminados observados de $G_d^{'}$)*

Passo 1) *Forme os seguintes conjuntos:*

$$S_d^{'} = \{s^{'} \in \Sigma_o^{'*} \, : \, s^{'} \text{ é um traço associado a um caminho primo de } G_d^{'} \text{ formado a partir de ciclos indeterminados observados}\}.$$

Se $S_d^{'} = \emptyset$, então $\Sigma_{ies}^o = \emptyset$ e pare. Caso contrário, vá para o passo 2.

Passo 2) *Forme conjuntos S_Y e S_N como mostrado nas equações (3.2) e (3.3). Para todos os caminhos primos associados a traços de S_Y e S_N, identifique os estados $x_{t,Y}^*$ e $x_{t,N}^*$ e seus correspondentes traços $s_Y = u_Y v_Y$ e $s_N = u_N v_N$ tais que $x_{t,Y}^* = f_t(x_{0_t}, u_Y)$ e $f_t(x_{t,Y}^*, v_Y) = x_{t,Y}^*$ e $x_{t,N}^* = f_t(x_{0_t}, u_N)$ e $f_t(x_{t,N}^*, v_N) = x_{t,N}^*$. A seguir, forme os seguintes conjuntos:*

$$S_Y^o = \{s_Y = u_Y v_Y \in S_Y \, : \, v_Y \text{ possui pelo menos um evento em } \Sigma_o^{'}\},$$
$$S_N^o = \{s_N = u_N v_N \in S_N \, : \, v_N \text{ possui pelo menos um evento em } \Sigma_o^{'}\}.$$

Passo 3) *Seja $S_d^{'} = \{s_1^{'}, s_2^{'}, \ldots, s_p^{'}\}$, em que $p = |S_d^{'}|$. Para cada $s_i^{'} \in S_d^{'}$, $i = 1, \ldots, p$, forme os seguintes conjuntos:*

$$S_{Y,i}^o = \{s_Y \in S_Y^o \, : \, s_i^{'} \in \overline{P_o^{'}(s_Y)}\},$$
$$S_{N,i}^o = \{s_N \in S_N^o \, : \, s_i^{'} \in \overline{P_o^{'}(s_N)}\}.$$

Passo 4) *Para cada traço $s_{Yi}^k \in S_{Y,i}^o$, forme um conjunto $\Sigma_{Y,i}^k$ com os eventos de s_{Yi}^k que pertencem a $\Sigma_o \setminus \Sigma_o^{'}$. Para cada traço $s_{Ni}^l \in S_{N,i}^o$, forme um conjunto $\Sigma_{N,i}^l$ com os eventos de s_{Ni}^l que pertencem a $\Sigma_o \setminus \Sigma_o^{'}$.*

Passo 5) *Seja $l_{Yi} = |S_{Y,i}^o|$ e $l_{Ni} = |S_{N,i}^o|$. Para $i = 1, \ldots, p$, obtenha:*

$$\Sigma_{ies,Yi}^o = \begin{cases} \emptyset, \text{ se } l_{Yi} = 1 \text{ e } \Sigma_{Y,i} = \emptyset \\ 2_1^{\Sigma_{Y,i}}, \text{ se } l_{Yi} = 1 \text{ e } \Sigma_{Y,i} \neq \emptyset \\ \Sigma_{Y,i}^1 \dot{\times} \Sigma_{Y,i}^2 \dot{\times} \ldots \dot{\times} \Sigma_{Y,i}^{l_{Yi}}, \text{ se } l_{Yi} \geq 1, \end{cases}$$

$$\Sigma_{ies,Ni}^o = \begin{cases} \emptyset, \text{ se } l_{Ni} = 1 \text{ e } \Sigma_{N,i} = \emptyset \\ 2_1^{\Sigma_{N,i}}, \text{ se } l_{Ni} = 1 \text{ e } \Sigma_{N,i} \neq \emptyset \\ \Sigma_{N,i}^1 \dot{\times} \Sigma_{N,i}^2 \dot{\times} \ldots \dot{\times} \Sigma_{N,i}^{l_{Ni}}, \text{ se } l_{Ni} \geq 1, \end{cases}$$

$$\Sigma_{ies,i}^o = \Sigma_{ies,Yi}^o \cup \Sigma_{ies,Ni}^o.$$

Passo 6) *Calcule* $\Sigma^o_{ies} = \Sigma^o_{ies,1} \dot{\times} \Sigma^o_{ies,2} \dot{\times} \ldots \dot{\times} \Sigma^o_{ies,p}$.

Passo 7) *Remova de* Σ^o_{ies} *todos os conjuntos* $\widetilde{\Sigma}^o_{ies} \in \Sigma^o_{ies}$ *para os quais existe outro conjunto* $\widehat{\Sigma}^o_{ies} \in \Sigma^o_{ies}$ *tal que* $\widetilde{\Sigma}^o_{ies} \supseteq \widehat{\Sigma}^o_{ies}$.

3.3.3 Análise da diagnosticabilidade para diagnosticadores com ciclos escondidos indeterminados

Nesta seção serão analisados apenas os ciclos escondidos indeterminados. De acordo com as condições **CE1** e **CE2**, a existência de ciclos escondidos indeterminados em G'_d implica na existência de um traço arbitrariamente longo $s_Y \in L_d$ associado a um caminho com ciclos internos formado com estados certos de G_d, e um traço de comprimento limitado $s_N \in L_d$ que leva G_d de seu estado inicial a um estado normal ou incerto que satisfazem $P'_o(s_Y) = P'_o(s_N) = s'$. Além disso, dada a premissa **H4**, G_d não possui ciclos indeterminados escondidos ou observados e, como $G'_{teste} = G'_d \parallel G_d$, pode-se afirmar que os ciclos escondidos indeterminados de G'_d serão ciclos indeterminados observados em G'_{teste}. A existência de ciclos escondidos está relacionada a ciclos cujas transições são rotuladas por eventos que pertencem a $\Sigma_o \setminus \Sigma'_o$. A idéia apresentada em [29] consiste em relacionar os caminhos primos-Y de G'_{teste} aos caminhos de G'_d que possuem ciclos indeterminados escondidos.

Considere a definição dos seguintes conjuntos [29]:

$$S'_t = \{s : s \text{ é um traço associado a um caminho primo de } G'_{teste}\}, \tag{3.4}$$

$$S_{YY} = \{s \in S'_t : (\text{existe uma folha rotulada por } (x'_d, x_d), \text{ sendo } x'_d \text{ e } x_d \text{ ambos estados certos})[f_t(x_{t_0}, s) = (x'_d, x_d)]\}, \tag{3.5}$$

$$S_{NN} = \{s \in S'_t : (\text{existe uma folha rotulada por } (x'_d, x_d), \text{ sendo } x'_d \text{ e } x_d \text{ ambos estados normais})[f_t(x_{t_0}, s) = (x'_d, x_d)]\}. \tag{3.6}$$

Teorema 3.6. *Seja* $x^*_t = (x'^*_d, x^*_d)$ *o único estado revisitado de um caminho primo-Y de* G'_{teste} *e* s_Y *o traço formado com os eventos desse caminho. Suponha* $s_Y = uv$, *sendo* $v \in (\Sigma_o \setminus \Sigma'_o)^*$, $x^*_t = f_t(x_{0_t}, u)$ *e* $f_t(x^*_t, v) = x^*_t$. *Então,* $s' = P'_o(s_Y)$ *é um traço formado pelos eventos do caminho com ciclos internos indeterminados escondidos de* G'_d *no estado* $x'^*_d = f'_d(x'_{0_d}, s')$. *Considere, ainda, as seguintes definições:*

$$S^h_Y(s') = \{s_Y \in S^h_Y : P'_o(s_Y) = s'\},$$

em que $S^h_Y = \{s_Y \in S_Y : (s_Y = uv \in \Sigma^*_o)(v \in (\Sigma_o \setminus \Sigma'_o))\}$.

$$S_N^h(s') = S_Y^{YN}(s') \cup S_{YY}^{YN} \cup S_N \cup S_{NN}, \tag{3.7}$$

sendo S_N e S_{NN} conforme as equações (3.3) e (3.6), respectivamente, e

$$S_{YY}^{YN} = S_{YY,1}^{YN} \setminus S_{YY,2}^{YN}, \tag{3.8}$$

em que

$S_{YY,1}^{YN} = \{s \in \overline{S_{YY}} : (\exists x_d \text{ incerto}) [f_t(x_{t_0}, s) = (x_d', x_d)]\}$,
$S_{YY,2}^{YN} = \{s \in S_{YY,1}^{YN} : (\exists s_{max} \in S_{YY,1}^{YN}) [(s \in \overline{s_{max}}) \wedge (s \neq s_{max})]\}$,

$$S_Y^{YN}(s') = S_{Y,1}^{YN}(s') \setminus S_{Y,2}^{YN}(s'), \tag{3.9}$$

sendo

$S_{Y,1}^{YN}(s') = \{s \in \overline{S_Y(s')} : (\exists x_d \text{ incerto}) [f_t(x_{t_0}, s) = (x_d', x_d)]\}$,
$S_{Y,2}^{YN}(s') = \{s \in S_{Y,1}^{YN}(s') : (\exists s_{max} \in S_{Y,1}^{YN}(s'))[(s \in \overline{s_{max}}) \wedge (s \neq s_{max})]\}$,

$$S_Y(s') = S_Y^h \setminus S_Y^h(s'). \tag{3.10}$$

Então existe pelo menos um traço $s_P \in S_N^h(s')$ que possui prefixo s_N ($s_N \in \overline{s_P}$) tal que $P_o'(s_N) = s'$ e $f_d(x_{0_d}, s_P) = x_d$, onde x_d é um estado normal ou incerto (que não possua um ciclo indeterminado associado) de G_d.

Teorema 3.7. *Seja L uma linguagem não diagnosticável com relação a P_o' e Σ_f e seja $\Sigma_o'' = \Sigma_o' \cup \Sigma_{ies}$, $\Sigma_{ies} \subseteq \Sigma_o \setminus \Sigma_o'$. Seja (s_Y, s_N) um par de traços em G_{teste}', em que s_Y é um traço associado a um caminho primo-Y de G_{teste}' que corresponde a um caminho contendo um ciclo interno indeterminado escondido de G_d' e s_N é um prefixo do traço $s_P \in S_N^h(s')$ ($s' = P_o'(s_Y)$), cujo último evento $s_{N_f} \in \Sigma_o'$. A condição necessária para que o caminho que contem um ciclo interno indeterminado escondido em s' não seja um caminho com ciclo interno indeterminado escondido em G_{teste}' é que $\Sigma_{ies} \cap [(\Sigma_{s_Y} \cup \Sigma_{s_N}) \setminus \Sigma_o'] \neq \emptyset$, sendo Σ_{s_Y} e Σ_{s_N} os conjuntos formados pelos eventos de traços s_Y e s_N, respectivamente.*

Prova. Vide [29]. □

De acordo com o teorema 3.7, para um par de traços (s_Y, s_P), em que s_Y é formado por eventos do caminho primo-Y de G_{teste}' associado ao caminho com ciclos internos indeterminados escondidos em G_d', e $s_P \in S_N^h(s')$, que possui um prefixo s_N cujo último evento pertence a Σ_o' e satisfaz $P_o'(s_Y) = P_o'(s_N) = s'$, não resultar na ocorrência de ciclos escondidos indeterminados, é necessário incluir em Σ_{ies} pelo menos um evento de s_Y ou pelo menos um evento de s_N que pertença a $\Sigma_o \setminus \Sigma_o'$. Esse

requerimento deve ser satisfeito para todos os pares de traços (s_Y, s_N) de G'_{teste} que satisfaçam as condições acima. O algoritmo a seguir, desenvolvido em [29], retorna todos os eventos que devem ser adicionados a Σ'_o para criar candidatos a serem bases mínimas de diagnóstico. A exigência para utilizar o algoritmo abaixo é que todos os caminhos primos de G'_d e G'_{teste} tenham sido calculados previamente.

Algoritmo 3.4. *(Algoritmo para obtenção dos eventos associados a ciclos indeterminados escondidos de G'_d)*

Passo 1) *Para cada caminho primo-Y de G'_{teste}, identifique o estado revisitado $x^*_{t,Y}$ e o correspondente traço $s_Y = u_Y v_Y$ tal que $x^*_{t,Y} = f_t(x_{0_t}, u_Y)$ e $f_t(x^*_{t,Y}, v_Y) = x^*_{t,Y}$ e forme o seguinte conjunto:*

$$S^h_Y = \{s_Y = u_Y v_Y \in \Sigma^*_o : v_Y \in (\Sigma_o \setminus \Sigma'_o)\},$$

Em seguida, calcule $P'_o(S^h_Y)$. Defina $p = |P'_o(S^h_Y)|$ $(p \leq |P_{oo'}(S^h_Y)|)$, e escreva $P'_o(S^h_Y)$ como

$$P'_o(S^h_Y) = \{s'_1, s'_2, \ldots, s'_p\}.$$

Passo 2) *Forme conjuntos S_N, S_{NN} e S^{YN}_{YY} conforme as equações (3.3), (3.6) e (3.8), respectivamente.*

Passo 3) *Para cada $s'_i \in P_{oo'}(S^h_Y)$, $i = 1, \ldots, p$, forme conjuntos $S^h_{Y,i} = \{s_Y \in S^h_Y = s'_i\}$.*

Passo 4) *Para cada $s'_i \in P_{oo'}(S^h_Y)$, $i = 1, \ldots, p$, forme conjuntos $S^h_{N,i}$ como mostrado abaixo:*

- *Forme conjuntos $S^h_N(s'_i)$ conforme equação (3.7), trocando s' por s'_i e $S_Y(s'_i) = S^h_Y \setminus S^h_{Y,i}$.*
- *Forme o conjunto*

$$S^h_{N,i} = \{s_N \in \overline{S^h_N(s'_i)} : (\exists s_{N_f} \in \Sigma'_o)[P_{oo'}(s_N) = s'_i]\},$$

em que s_{N_f} denota o último evento de s_N.

Passo 5) *Construa conjuntos $\Sigma^k_{Y,i}$ e $\Sigma^l_{N,i}$ da seguinte forma:*

- *Para cada $s^k_{Y,i} \in S^k_{Y,i}$ forme o conjunto $\Sigma^k_{Y,i}$ com os eventos de $s^k_{Y,i}$ que pertencem a $\Sigma_o \setminus \Sigma'_o$.*
- *Para cada $s^l_{N,i} \in S^l_{N,i}$ forme o conjunto $\Sigma^l_{N,i}$ com os eventos de $s^l_{N,i}$ que pertencem a $\Sigma_o \setminus \Sigma'_o$.*

Passo 6) *Considere* $l_{Yi} = |S^h_{Y,i}|$ *e* $l_{Ni} = |S^h_{N,i}|$. *Para* $i = 1, \ldots, p$, *faça:*

$$\Sigma^h_{ies,Yi} = \begin{cases} \{\emptyset\}, \text{ se } l_{Yi} = 1 \text{ e } \Sigma_{Y,i} = \emptyset \\ 2_1^{\Sigma_{Y,i}}, \text{ se } l_{Yi} = 1 \text{ e } \Sigma_{Y,i} \neq \emptyset \\ \Sigma^1_{Y,i} \dot{\times} \Sigma^2_{Y,i} \dot{\times} \ldots \dot{\times} \Sigma^{l_{Yi}}_{Y,i}, \text{ se } l_{Yi} \geq 1, \end{cases}$$

$$\Sigma^h_{ies,Ni} = \begin{cases} \{\emptyset\}, \text{ se } l_{Ni} = 1 \text{ e } \Sigma_{N,i} = \emptyset \\ 2_1^{\Sigma_{N,i}}, \text{ se } l_{Ni} = 1 \text{ e } \Sigma_{N,i} \neq \emptyset \\ \Sigma^1_{N,i} \dot{\times} \Sigma^2_{N,i} \dot{\times} \ldots \dot{\times} \Sigma^{l_{Ni}}_{N,i}, \text{ se } l_{Ni} \geq 1, \end{cases}$$

$\Sigma^h_{ies,i} = \Sigma^h_{ies,Yi} \cup \Sigma^h_{ies,Ni}$.

Passo 7) *Calcule* $\Sigma^h_{ies} = \Sigma^h_{ies,1} \dot{\times} \Sigma^h_{ies,2} \dot{\times} \ldots \dot{\times} \Sigma^h_{ies,p}$.

Passo 8) *Remova de* Σ^h_{ies} *todos os conjuntos* $\widetilde{\Sigma}^h_{ies} \in \Sigma^h_{ies}$ *para os quais existe outro conjunto* $\widehat{\Sigma}^h_{ies} \in \Sigma^h_{ies}$ *tal que* $\widetilde{\Sigma}^h_{ies} \supseteq \widehat{\Sigma}^h_{ies}$.

3.3.4 Algoritmo de busca das bases mínimas para diagnóstico proposto em [29]

Será apresentado, a seguir, o algoritmo proposto em [29] para encontrar as bases mínimas para o diagnóstico de falhas. O algoritmo possui a premissa que, dado um autômato G, a linguagem L gerada por G é diagnosticável com relação a $P_o : \Sigma \to \Sigma_o^*$ e σ_f, em que Σ_o representa o conjunto de eventos observáveis de G.

Algoritmo 3.5. *(Algoritmo de busca das bases mínimas para diagnóstico de falhas utilizando diagnosticadores)*

Passo 1) *Utilizando o algoritmo 3.1, obtenha* Σ_{edes}.

Passo 2) *Defina* $\Sigma_{mdbc} = \Sigma_{edes}$ *e* $\Sigma_{mdb} = \emptyset$.

Passo 3) *Escolha o elemento de* Σ_{mdbc} *de menor cardinalidade e denote-o por* $\Sigma_o^{'}$. *A seguir, atualize* $\Sigma_{mdbc} \leftarrow \Sigma_{mdbc} \setminus \{\Sigma_o^{'}\}$ *e calcule* $G_d^{'}$.

Passo 3.1) *Se* $G_d^{'}$ *não tiver ciclos indeterminados:*

Passo 3.1.1) *Defina* $\Sigma_{mdb} \leftarrow \Sigma_{mdb} \cup \{\Sigma_o^{'}\}$. *[O algoritmo pode ser interrompido aqui se desejado].*

Passo 3.1.2) *Se* $\Sigma_{mdbc} = \emptyset$, *então pare. Caso contrário, volte ao início do passo 3.*

Passo 3.2) *Se $G_d^{'}$ tiver ciclos indeterminados:*

Passo 3.2.1) *Utilize o algoritmo 3.2 para obter todas os caminhos primos de $G_{teste}^{'}$ e $G_d^{'}$.*

Passo 3.2.2) *Utilize o algoritmo 3.3 para obter Σ_{ies}^{o}.*

Passo 3.2.3) *Utilize o algoritmo 3.4 para obter Σ_{ies}^{h}.*

Passo 3.2.4) *Faça $\Sigma_{ies} = \Sigma_{ies}^{o} \dot{\times} \Sigma_{ies}^{h}$.*

Passo 3.2.5) *Remova de Σ_{ies} todos os conjuntos $\widetilde{\Sigma}_{ies}$ para os quais existe outro conjunto $\widehat{\Sigma}_{ies}$ tal que $\widetilde{\Sigma}_{ies} \supseteq \widehat{\Sigma}_{ies}$.*

Passo 3.2.6) *Faça $\Sigma_{mdbc} \leftarrow \Sigma_{mdbc} \cup (\{\Sigma_o^{'}\} \dot{\times} \Sigma_{ies})$.*

Passo 3.2.7) *Remova de Σ_{mdbc} todos os conjuntos $\widetilde{\Sigma}_{mdbc}$ para os quais existe outro conjunto $\widehat{\Sigma}_{mdbc}$ tal que $\widetilde{\Sigma}_{mdbc} \supseteq \widehat{\Sigma}_{mdbc}$ ou existe um conjunto $\widehat{\Sigma}_{mdb} \in \Sigma_{mdb}$ tal que $\widetilde{\Sigma}_{mdbc} \supseteq \widehat{\Sigma}_{mdb}$.*

Passo 3.2.8) *Se $\Sigma_{mdbc} = \{\Sigma_o\}$, então defina $\Sigma_{mdb} = \Sigma_o$ e pare. Caso contrário, volte ao início do passo 3.*

Será apresentado agora um exemplo para ilustrar o uso do algoritmo 3.5.

Exemplo 3.1. *Considere o autômato G mostrado na figura 3.1, cujo conjunto de eventos observáveis é $\Sigma_o = \{a, b, c\}$ e o conjunto de eventos de falha é $\Sigma_f = \{\sigma_f\}$. Conforme a premissa **H4**, a linguagem L gerada por G é diagnosticável com relação a P_o e Σ_f. O algoritmo 3.5 será então aplicado de forma a se obter todas as bases mínimas para o diagnóstico de L.*

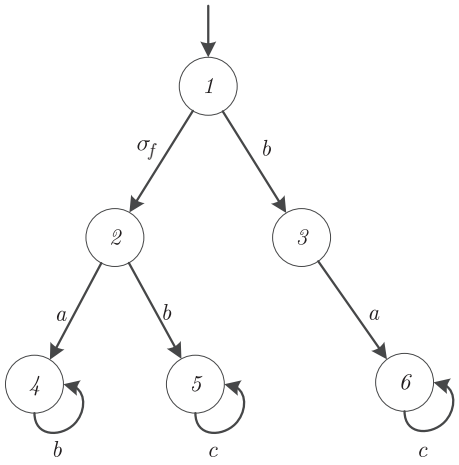


Figura 3.1: Autômato G do exemplo 3.1.

O primeiro passo do algoritmo 3.5 requer o uso do algoritmo 3.1 para obtenção dos conjuntos elementares para diagnóstico Σ_{edes}. Para tal, deve-se calcular G_d

(mostrado na figura 3.2) e obter todas as árvores formadas a partir de estados incertos de G_d que alcançam estados certos da ocorrência da falha (mostradas na figura 3.3).

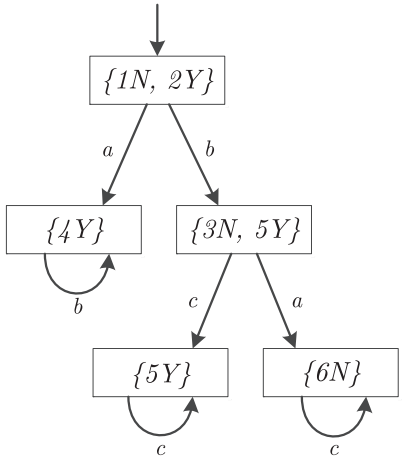
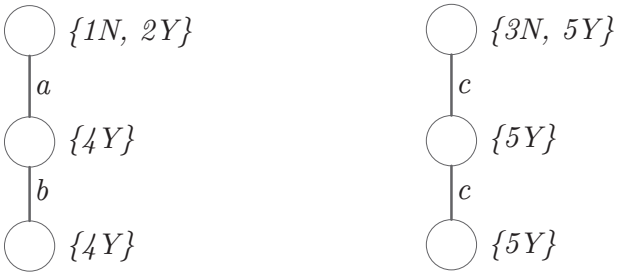


Figura 3.2: Diagnosticador centralizado G_d do exemplo 3.1.

Em particular, as árvores mostradas na figura 3.3 possuem, cada uma, um único caminho que liga suas raízes às suas folhas. São eles: $P^1_{Y,1} = \{\{1N, 2Y\}, a, \{4Y\}, b, \{4Y\}\}$ *e* $P^2_{Y,1} = \{\{3N, 5Y\}, c, \{5Y\}, c, \{5Y\}\}$*. Dessa forma, pode-se afirmar que* $\Sigma^1_{fpes,1} = \{a, b\}$ *e* $\Sigma^1_{fpes,2} = \{c\}$*, resultando em* $\Sigma_{edes} = \{\{a, c\}, \{b, c\}\}$*.*

De acordo com o passo 2 do algoritmo 3.5, $\Sigma_{mdbc} = \Sigma_{edes} = \{\{a, c\}, \{b, c\}\}$ *e* $\Sigma_{mdb} = \emptyset$*.*

O passo 3 estabelece que deve-se escolher um elemento de Σ_{mdbc} *e denotá-lo por* Σ'_o *e, em seguida, deve-se atualizar* Σ_{mdbc} *retirando o conjunto selecionado. Nesse exemplo, será considerado* $\Sigma'_o = \{a, c\}$ *e* $\Sigma_{mdbc} = \{\{b, c\}\}$*. Após a definição de* Σ'_o*, deve-se obter* G'_d *e verificar se o mesmo possui um ou mais ciclos indeterminados (observados ou escondidos). O autômato diagnosticador* G'_d *é mostrado na figura*



(a) Árvore com raiz $x_{d_{YN,1}} = \{1N, 2Y\}$. (b) Árvore com raiz $x_{d_{YN,2}} = \{3N, 5Y\}$.

Figura 3.3: Árvores rotuladas criadas a partir de estados $x_{d_{YN,i}}$ de G_d.

3.4. Note que existe um ciclo indeterminado escondido no estado $\{4Y, 6N\}$*, fazendo com que a linguagem* L *seja não diagnosticável com relação a* $P_o^{'}$ *e* Σ_f*. Dessa forma, será necessário obter o conjunto* Σ_{ies} *para adicionar eventos a* $\Sigma_o^{'}$*.*

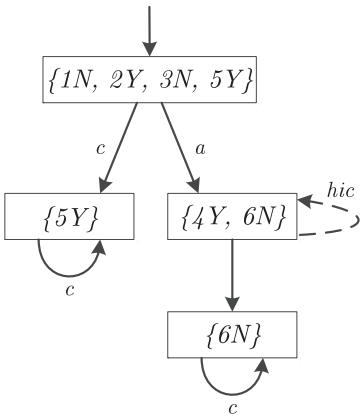


Figura 3.4: Diagnosticador com observação reduzida $G_d^{'}$ sendo $\Sigma_o^{'} = \{a, c\}$.

Deve-se então calcular $G_{teste}^{'}$ *(mostrado na figura 3.5) e obter todos os caminhos primos de* $G_d^{'}$ *e* $G_{teste}^{'}$ *a partir do algoritmo 3.2.*

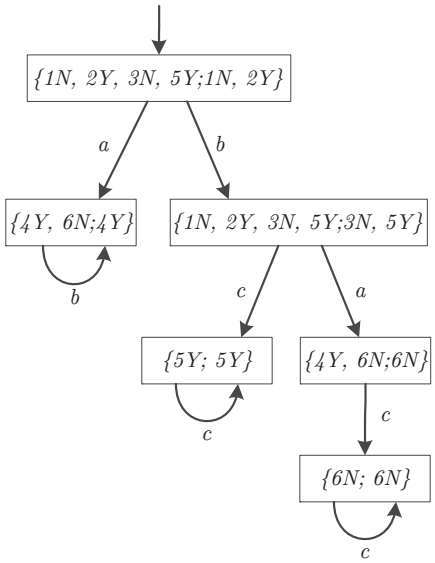


Figura 3.5: Autômato teste $G_{teste}^{'}$ assumindo $\Sigma_o^{'} = \{a, c\}$.

Note que, uma vez que não existem ciclos indeterminados observados em $G_d^{'}$*, o algoritmo 3.3 retorna* $\Sigma_{ies}^{o} = \emptyset$*.*

Em seguida, deve-se utilizar o algoritmo 3.4 para obter Σ_{ies}^{h}*. A partir da figura 3.6, deve-se obter os caminhos primos-*Y *de* $G_{teste}^{'}$ *associados a ciclos indeterminados*

escondidos de $G_d^{'}$. *Assim ,* $S_Y^h = \{ab\}$. *Será necessário conhecer, também, a projeção realizada a partir de* $\Sigma_o^{'}$ *dos traços contidos em* S_Y^h, *que é dada por* $P_o^{'}(S_Y^h) = \{a\}$.

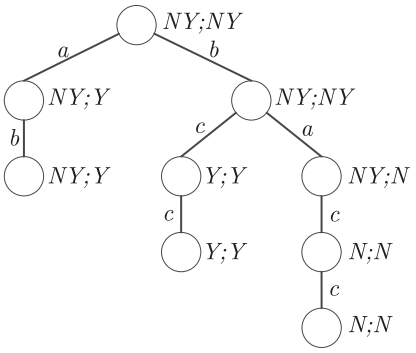


Figura 3.6: Árvore correspondete ao autômato teste $G_{teste}^{'}$ em que $\Sigma_o^{'} = \{a, c\}$.

O próximo passo do algoritmo 3.4 requer a construção dos conjuntos S_N, S_{NN}, S_{YY} *e* S_{YY}^{YN}. *Novamente, utilizando a figura 3.6, pode-se obter* $S_N = \emptyset$, $S_{NN} = \{bacc\}$ *e* $S_{YY} = \{bcc\}$. *O conjunto* S_{YY}^{YN} *é formado tomando-se o maior prefixo de cada traço de* S_{YY} *que leva do estado inicial de* G_d *a um estado incerto. Assim, com base na figura 3.6,* $S_{YY}^{YN} = \{b\}$.

A seguir, devem ser formados os conjuntos $S_{Y,1}^h$ *e* $S_{N,i}^h$ *associados a* $s_1^{'} \in P_o^{'}(S_Y^h)$. *Dessa forma, como* $s_1^{'} = a$, $S_{Y,1}^h = \{ab\}$ *e* $S_N^h(s_1^{'}) = \{bacc, b\}$, *que resulta em* $S_{N,1}^h = \{ba\}$.

Para finalizar o algoritmo 3.4, deve-se verificar os eventos que pertencem a $\Sigma_o \setminus \Sigma_o^{'}$ *e se encontram nos conjuntos* $S_{Y,1}^h$ *e* $S_{N,1}^h$, *respectivamente. Assim, tem-se que* $\Sigma_{Y,1}^h = \{b\}$ *e* $\Sigma_{N,1}^h = \{b\}$, *o que resulta em* $\Sigma_{ies}^h = \Sigma_{ies,1}^h = \Sigma_{Y,1}^h \cup \Sigma_{N,1}^h = \{b\}$.

A seguir, deve-se fazer $\Sigma_{mdbc} = \Sigma_{mdbc} \cup (\{\Sigma_o^{'}\} \dot{\times} \Sigma_{ies}) = \{\{b, c\}, \{a, b, c\}\}$. *Contudo, como* $\{b, c\} \subseteq \{a, b, c\}$, $\Sigma_{mdbc} = \{b, c\}$ *e deve-se voltar ao passo 3 do algoritmo 3.5.*

Voltando ao passo 3, $\Sigma_o^{'} = \{b, c\}$ *e* $\Sigma_{mdbc} = \emptyset$. *A figura 3.7 mostra o diagnosticador com observação reduzida considerando-se* $\Sigma_o^{'} = \{b, c\}$.

Note que $G_d^{'}$ *possui um ciclo indeterminado observado e, portanto, a linguagem L é não diagnosticável com relação a* $P_o^{'}$ *e* Σ_f. *Dessa forma, será necessário, novamente, obter o conjunto* Σ_{ies} *para adicionar eventos a* $\Sigma_o^{'}$.

Deve-se então calcular $G_{teste}^{'}$ *(mostrado na figura 3.8) e obter todos os caminhos primos de* $G_d^{'}$ *e* $G_{teste}^{'}$ *a partir do algoritmo 3.2.*

Em seguida, o algoritmo 3.3 deve ser utilizado de forma a obter Σ_{ies}^o. *Primeiramente, deve-se formar o conjunto* $S_d^{'}$, *cujos elementos são traços formados com os eventos dos caminhos primos de* $G_d^{'}$ *associados a ciclos indeterminados observados. Então, a partir da figura 3.7, sabe-se que* $S_d^{'} = \{bcc\}$. *Utilizando a figura 3.9, deve-se identificar os traços associados a caminhos primos-Y de* $G_{teste}^{'}$ *e os traços que*

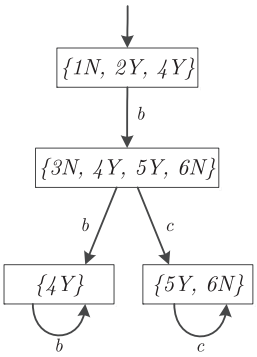


Figura 3.7: Diagnosticador com observação reduzida $G_d^{'}$ sendo $\Sigma_o^{'} = \{b, c\}$.

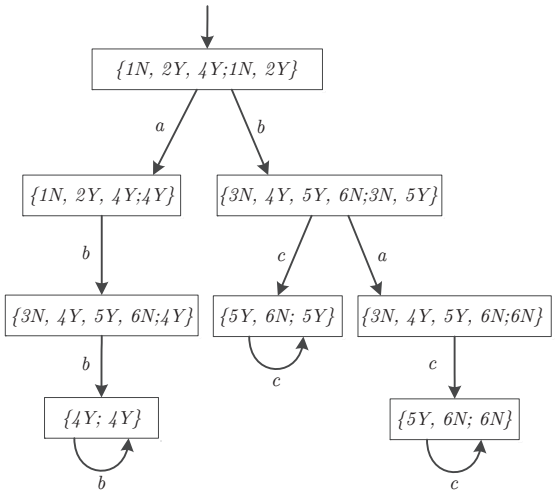


Figura 3.8: Autômato teste $G_{teste}^{'}$ sendo $\Sigma_o^{'} = \{b, c\}$.

alcançam folhas cujas primeiras componentes dos estados do único ciclo são estados incertos de um ciclo indeterminado de $G_d^{'}$ *e as segundas componentes são formadas por estados normais de* G_d *para formar os conjuntos* S_Y^o *e* S_N^o*, respectivamente. Assim,* $S_Y^o = \{bcc\}$ *e* $S_N^o = \{bacc\}$.

Conhecendo-se os conjuntos S_Y^o *e* S_N^o*, basta identificar os eventos que pertencem a* $\Sigma_o \setminus \Sigma_o^{'}$*. Então,* $\Sigma_{Y,1}^1 = \emptyset$ *e* $\Sigma_{N,1}^1 = \{a\}$*, resultando em* $\Sigma_{ies}^o = \Sigma_{ies,1}^o = \{\{a\}\}$.

Note que, uma vez que não existem ciclos indeterminados escondidos em $G_d^{'}$*, o algoritmo 3.4 retorna* $\Sigma_{ies}^h = \{\emptyset\}$.

A seguir, deve-se fazer $\Sigma_{mdbc} = \Sigma_{mdbc} \cup (\{\Sigma_o^{'}\} \dot{\times} \Sigma_{ies}) = \{\{a, b, c\}\}$*. Assim, como* $\Sigma_{mdbc} = \{\Sigma_o\}$*, então* $\Sigma_{mdb} = \{\Sigma_o\}$ *e o algoritmo chega ao fim. Dessa forma, o sistema possui uma única base para o diagnóstico de falhas,* $\{a, b, c\}$*, que também é mínima.*

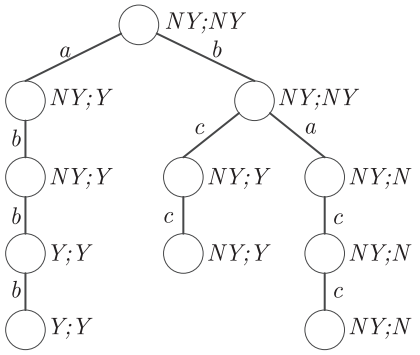


Figura 3.9: Árvore correspondete ao autômato teste G'_{teste} em que $\Sigma'_o = \{b, c\}$.

3.3.5 Complexidade computacional do algoritmo 3.5 proposto em [29]

No passo 1 do algoritmo 3.5, utilizado para calcular Σ_{edes}, é utilizado o algoritmo 3.1, que consiste em calcular G_d e criar árvores a partir de determinados estados do mesmo. O autômato diagnosticador G_d possui $|X_d|$ estados e $|X_d||\Sigma_o|$ transições. Note que $|X_d|$ é limitado superiormente por $2^{|X|}$. Particionando o espaço de estados de G_d como $X_d = X^Y_{YN} \dot\cup X_Y \dot\cup X'_d$, pode-se afirmar que o custo total na construção das árvores do algoritmo 3.1 é de $|X^Y_{YN}||\Sigma_o|^{|X_Y|}$. A cada iteração do passo 3, será necessário checar a diagnosticabilidade, o que pode ser realizado com complexidade polinomial utilizando-se autômatos verificadores (vide [26]) ou construindo-se G'_d a partir de G_d e buscando-se por ciclos indeterminados. Após isso, será necessário obter todos os caminhos primos de G'_d e G'_{teste} utilizando-se o algoritmo 3.2. Como G'_d possui pelo menos $|X_d|$ estados e $|X_d||\Sigma'_o|$ transições e G'_{teste} possui pelo menos $|X_d|^2$ estados e $|X_d|^2|\Sigma'_o|$ transições, computar os caminhos primos desses autômatos poderá resultar em árvores de complexidade, no pior caso, $|\Sigma'_o|^{|X_d|}$ e $|\Sigma_o|^{|X_d|^2}$, respectivamente, o que é limitado, considerando-se o espaço de estados do autômato G, por $|\Sigma'_o|^{2^{|X|}}$ e $|\Sigma_o|^{2^{|X|^2}}$, respectivamente. Os demais cálculos do passo 3 do algoritmo 3.5, por serem limitados por termos de ordem mais baixas que os demais, serão omitidos. Note que o passo 3 poderá ser executado em até no máximo $2^{|\Sigma_o \setminus \Sigma_{edes,min}|}$ iterações.

Dessa forma, conclui-se que a complexidade do algoritmo 3.5 é, no pior caso, $O(2^{|\Sigma_o \setminus \Sigma_{edes,min}|}|\Sigma_o|^{2^{|X|^2}})$.

Capítulo 4

Primeiro método para obtenção de bases mínimas para o diagnóstico de falhas em SEDs utilizando verificadores

No trabalho desenvolvido por BASILIO *et al.* [29], apresentado no capítulo 3, o problema de se encontrar todas as bases mínimas que façam com que a linguagem de um sistema seja diagnosticável foi formulado e abordado utilizando autômatos diagnosticadores. Essa abordagem, no entanto, exige a busca por ciclos com características que violam as condições de diagnosticabilidade em um autômato diagnosticador que, no pior caso, apresenta um crescimento exponencial na cardinalidade do espaço de estados do sistema. Para contornar esse problema, será proposto, neste capítulo, uma metodologia de obtenção de todas as bases mínimas para o diagnóstico utilizando apenas verificadores que, como apresentado no capítulo 2, possuem crescimento, no pior caso, polinomial com a cardinalidade do espaço de estados do sistema original. Assim, é esperado que se obtenha um novo algoritmo com menor complexidade computacional do que o desenvolvido em [29]. A análise da complexidade computacional do novo método é apresentada no final deste capítulo.

4.1 Bases para o diagnóstico de falhas

O algoritmo que será apresentado neste capítulo tem por objetivo seguir a mesma filosofia utilizada em [29]. Contudo, ao utilizar autômatos verificadores no lugar de diagnosticadores, algumas diferenças irão naturalmente surgir. Primeiramente, deve-se observar que a metodologia para obtenção de verificadores utilizada neste trabalho, apresentada no capítulo 2, é a proposta em [26]. Essa metodologia é, atu-

almente, a de menor complexidade computacional, pois apenas os traços normais de mesma projeção que os traços de falha são considerados na construção do verificador. Além disso, uma consequência imediata do uso de verificadores é a inexistência de ciclos escondidos. Note que no algoritmo proposto em [29], é necessário realizar separadamente as análises de ciclos indeterminados observados e escondidos. Utilizando verificadores, será visto que a análise pode ser realizada em uma única etapa, tornando o algoritmo mais objetivo.

Observe também que, conforme discutido no capítulo 2, as hipóteses **H1** e **H2** podem ser removidas ao utilizar o verificador proposto em [26]. A hipótese **H3** será mantida por simplicidade de análise. Contudo, caso desejado, a mesma pode ser removida e, nesse caso, deve-se particionar o conjunto de eventos de falha $\Sigma_f = \Sigma_{f_1} \dot{\cup} \Sigma_{f_2} \dot{\cup} \ldots \dot{\cup} \Sigma_{f_k}$, em que Σ_{f_i} denota o conjunto de eventos de falha do mesmo tipo e k representa o número de tipos de falhas existentes no sistema. Em seguida, o algoritmo que será desenvolvido neste capítulo deverá ser executado para cada uma das partições Σ_{f_i}, $i = 1, \ldots, k$, considerando os eventos de falha das demais partições como eventos não observáveis comuns. Por fim, também será mantida a hipótese **H4**, pois caso a linguagem L não seja diagnosticável considerando Σ_o e Σ_f, é evidente que a propriedade de diagnosticabilidade não será verificada para $\Sigma_o^{'} \subset \Sigma_o$ e Σ_f.

4.1.1 Conjunto de eventos elementares para o diagnóstico

No capítulo 2, o algoritmo para obtenção do verificador G_V para um sistema a eventos discretos foi apresentado e, de acordo com o teorema 2.2, a linguagem L do sistema não será diagnosticável com relação a P_o e Σ_f se, e somente se, existir um caminho cíclico em G_V, $cl := (x_V^k, \sigma_k, x_V^{k+1}, \sigma_{k+1}, \ldots, x_V^l, \sigma_l, x_V^k)$ que satisfaz a condição (2.1).

De acordo com a condição (2.1), pelo menos um evento de um caminho cíclico cl que viola as condições de diagnosticabilidade do sistema deve pertencer a Σ. Logo, um caminho cíclico em G_V formado apenas por eventos renomeados (e, consequentemente, não observáveis e pertencentes a G_N) não altera a propriedade de diagnosticabilidade de G. Assim, pode-se classificar um caminho cíclico ambíguo em G_V de duas formas distintas, cujas definições formais são mostradas a seguir.

Definição 4.1. *(Caminhos cíclicos ambíguos-F) Um caminho cíclico* $(x_V^k, \sigma_k, x_V^{k+1}, \sigma_{k+1}, \ldots, x_V^{k+n}, \sigma_{k+n}, x_V^k)$ *em um verificador* G_V *que satisfaz a condição (2.1), é dito ser um caminho cíclico ambíguo-F se* $x_V^{k+j} = (x_N, x_F^{k+j})$ *e* $\sigma_{k+j} \notin \Gamma_N(x_N)$, $\forall j \in \{0, \ldots, n\}$.

Definição 4.2. *(Caminhos cíclicos ambíguos-NF) Um caminho cíclico* $(x_V^k, \sigma_k, x_V^{k+1}, \sigma_{k+1}, \ldots, x_V^{k+n}, \sigma_{k+n}, x_V^k)$ *em um verificador* G_V *que satisfaz a*

condição (2.1), é dito ser um caminho cíclico ambíguo-NF se $x_V^{k+j} = (x_N^{k+j}, x_F^{k+j})$, $\forall j \in \{0, \ldots, n\}$,*e* $\sigma_{k+j} \in \Gamma_N(x_N^{k+j})$, *para algum* $j \in \{0, \ldots, n\}$.

O exemplo a seguir ilustra os conceitos apresentados nas definições 4.1 e 4.2.

Exemplo 4.1. *Considere o autômato G mostrado anteriormente na figura 3.1, cujo conjunto de eventos observáveis é* $\Sigma_o = \{a, b, c\}$ *e o conjunto de eventos de falhas é* $\Sigma_f = \{\sigma_f\}$ *e seja* G_V' *o verificador obtido ao considerar* $\Sigma_o' = \{c\}$, *apresentado na figura 4.1.*

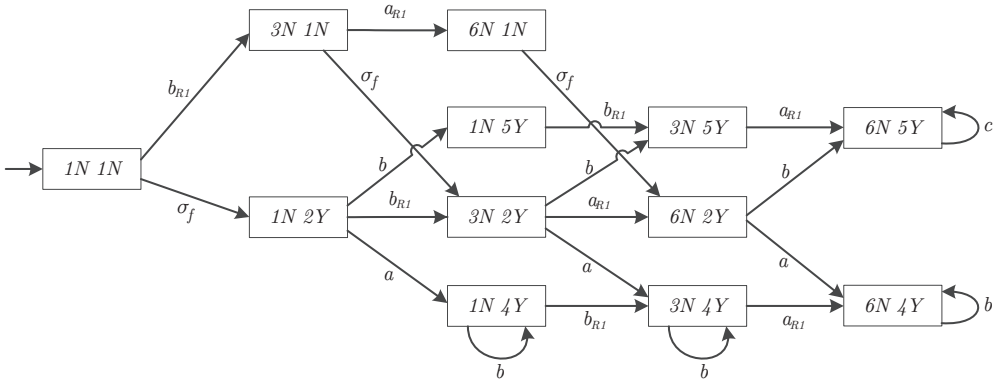


Figura 4.1: Verificador G_V' considerando o conjunto de eventos observáveis $\Sigma_o' = \{c\}$.

Observe que a sequência $s_1 = \sigma_f b_{R1} a_{R1} a$ *leva ao estado* $(6N, 4Y)$ *do verificador* G_V' *e que, a partir desse estado, a ocorrência do evento b gera o caminho cíclico ambíguo* $((6N, 4Y), b, (6N, 4Y))$. *Uma vez que* $b \notin \Gamma_N(6N)$, *pode-se afirmar que* $((6N, 4Y), b, (6N, 4Y))$ *é um caminho cíclico ambíguo-F.*

Considere agora a sequência $s_2 = \sigma_f b b_{R1} a_{R1}$ *que leva ao estado* $(6N, 5Y)$ *do verificador* G_V'. *Note que a ocorrência do evento c nesse estado gera o caminho cíclico ambíguo* $((6N, 5Y), c, (6N, 5Y))$. *Como* $c \in \Gamma_N(6N)$, *pode-se afirmar que* $((6N, 5Y), c, (6N, 5Y))$ *é um caminho cíclico ambíguo-NF.*

No capítulo 3, foi apresentada a metodologia desenvolvida em [29] para obtenção do conjunto de eventos elementares para o diagnóstico, cuja ideia era de que, para evitar a ocorrência de ciclos escondidos indeterminados em um autômato diagnosticador com observação reduzida G_d', pelo menos um evento de cada caminho que conecta um estado incerto a um estado certo de G_d deve fazer parte do conjunto de eventos elementares.

Contudo, ao utilizar autômatos verificadores, note que não existirão mais ciclos escondidos e, assim, deve-se encontrar uma correspondência nos verificadores para a ocorrência de ciclos escondidos indeterminados nos diagnosticadores, possibilitando, dessa forma, a obtenção do conjunto de eventos elementares para o diagnóstico. O teorema enunciado a seguir, estabelece essa correlação.

Teorema 4.1. *Sejam G_d e $G_d^{'}$ diagnosticadores obtidos considerando-se os conjuntos de eventos observáveis Σ_o e $\Sigma_o^{'}$, respectivamente, e seja $G_V^{'}$ o verificador obtido considerando-se o conjunto de eventos observáveis $\Sigma_o^{'}$. Então, pode-se afirmar que $G_V^{'}$ possui um caminho cíclico ambíguo-F se, e somente se, existe um ciclo escondido indeterminado em $G_d^{'}$.*

Prova. ($\Rightarrow$) Suponha que exista um ciclo escondido indeterminado em $G_d^{'}$, formado pelos estados $\{x_{d_1}, x_{d_2}, \ldots, x_{d_k}\}$ de G_d ($x_{d_1} = x_{d_k}$), associado a um traço $s = \sigma_1\sigma_2 \ldots \sigma_{k-1}$, tal que $s \in (\Sigma_o \setminus \Sigma_o^{'})^*$. Observe que sendo $G_d = Obs(G\|A_{label})$ e $G_F = CoAc(G\|A_{label})$, então, como os estados $\{x_{d_1}, x_{d_2}, \ldots, x_{d_k}\}$ de G_d estão certos sobre a ocorrência do evento de falha σ_f, pode-se afirmar que existe uma sequência s_F em G, que contém o evento de falha σ_f, tal que $s_F \in P_o^{-1}(s)$ e s_F leva a um ciclo em G. Portanto, existe um ciclo em G_F formado pelos componentes dos estados x_{d_j}, para $j = 1, 2, \ldots, k$. Note também que, uma vez que $s_F \in ((\Sigma_o \setminus \Sigma_o^{'}) \cup \Sigma_{uo})^*$, os eventos pertencentes a s_F serão exclusivos de G_F, o que significa que existirá uma sequência $s_V^{'} = s_F$ que gera um caminho cíclico em $G_V^{'}$ que satisfaz a definição de caminhos cíclicos ambíguos-F.
($\Leftarrow$) Suponha que $P_V^F = (x_V^k, \sigma_k, x_V^{k+1}, \sigma_{k+1}, \ldots, x_V^{k+n}, \sigma_{k+n}, x_V^k)$ seja um caminho cíclico ambíguo-F em $G_V^{'}$, em que $x_V^k = (x_N, x_F^k)$. Como $\sigma_{k+j} \notin \Gamma_N(x_N)$ e $\sigma_{k+j} \in \Gamma_V(x_V^{k+j})$, pode-se afirmar que $\sigma_{k+j} \notin \Sigma_o^{'}$, $\forall j \in \{0, \ldots, n\}$. Logo, existe um caminho cíclico $P_F^F = (x_F^k, \sigma_k, x_F^{k+1}, \sigma_{k+1}, \ldots, x_F^{k+n}, \sigma_{k+n}, x_F^k)$ após a ocorrência de σ_f em G_F associado ao caminho cíclico ambíguo-F P_V^F em $G_V^{'}$. Uma vez que $G_F = CoAc(G\|A_{label})$ e $G_d = Obs(G\|A_{label})$, pode-se afirmar que existe um ciclo em G_d, gerado por uma sequência s_d, associado ao caminho cíclico P_F^F em G_F, tal que $s_d = P_o(\sigma_k\sigma_{k+1} \ldots \sigma_{k+n})$. Como $s_d \in (\Sigma_o \setminus \Sigma_o^{'})^*$, existe um ciclo escondido indeterminado em $G_d^{'}$ em um estado $x_d^{'}$ tal que $x_N, x_F^k, x_F^{k+1}, \ldots, x_F^{k+n}$ são componentes dos estados de G_d que compõem $x_d^{'}$. □

Seja $x_N \in X_N$ e $x_F \in X_F$ as componentes de um estado (x_N, x_F) do verificador G_V obtido considerando-se o conjunto de eventos observáveis Σ_o. Forme os seguintes conjuntos:

$$X_{YN} = \{x_V \in X_V : x_V = (x_N, x_F) \wedge [x_F = (x, Y)]\},$$

$$X_O = \{x_F \in X_F : x_V = (x_N, x_F) \in X_{YN}\}.$$

Note que, conforme apresentado no capítulo 2, ao utilizar o algoritmo proposto em [26], apenas os estados dos autômatos G_N e G_F alcançados por sequências de mesma projeção serão exibidos no verificador G_V. Então, $X_{YN} \subset X_V$ representa todos os estados do verificador G_V que serão inevitavelmente alcançados após a ocorrência de σ_f sob a observação dos eventos de Σ_o. Assim, pode-se afirmar que, cada estado de X_O dará origem a pelo menos um caminho em G_F, $P_F = (x_F^k, \sigma_k,$

x_F^{k+1}, ..., x_F^{k+n}), que satisfaz às seguintes condições: *(i)* $x_F^{k+n} = x_F^{k+j}$, para algum $j \in \{0, 1, \ldots, n\}$, isto é, ($x_F^{k+j}$, σ_{k+j}, ..., x_F^{k+n}) forma um caminho cíclico; *(ii)* (x_F^{k+j}, σ_{k+j}, ..., x_F^{k+n}) é o único caminho cíclico em P_F. Consequentemente, de maneira semelhante ao realizado no capítulo 3, o conjunto X_O será referido como conjunto de estados-origem de caminhos de falha e o caminho P_F como caminho de falha. Os elementos de X_O são chamados estados-origem de caminhos de falha.

A definição dada a seguir, semelhante à definição 3.5, será apresentada no contexto de autômatos verificadores.

Definição 4.3. *(Evento de um caminho de falha e conjunto de eventos de um caminho de falha)*

A. *Um evento $\sigma \in \Sigma_o$ é um evento de um caminho de falha se ele pertence a qualquer caminho de falha iniciado em algum estado $x_F \in X_O$.*

B. *Um conjunto de eventos de um caminho de falha, denotado por $\Sigma_{fpes,i}$, é um conjunto formado por todos os eventos de um caminho de falha P_{F_i}.*

Considere o seguinte lema que relaciona um caminho cíclico ambíguo-F a um caminho de falha P_F, com origem em um estado de X_O.

Lema 4.1. *Seja Σ_{fpes} o conjunto de eventos do caminho de falha $P_F = (x_F^k, \sigma_k, x_F^{k+1}, \ldots, x_F^{k+n})$, em que $x_F^k \in X_O$, e $s_F = \sigma_k \sigma_{k+1} \ldots \sigma_{k+n-1}$ a sequência de eventos associada a P_F. Suponha que $\Sigma_{fpes} \subseteq \Sigma_{uo}$. Então, existe um caminho cíclico ambíguo-F cuja sequência de eventos associada é s_F.*

Prova. Seja Σ_R o conjunto de eventos do autômato $G_{N,1}$ obtido após a renomeação dos eventos não observáveis. Uma vez que $\Sigma_{fpes} \subseteq \Sigma_{uo}$, pode-se afirmar que $\Sigma_{fpes} \cap \Sigma_R = \emptyset$ e, portanto, os eventos de Σ_{fpes} são particulares do autômato G_F. Assim, pode-se ver que existirá um caminho cíclico no verificador G'_V cuja sequência de eventos é s_F. Como todos os eventos de s_F são particulares de G_F, então, o caminho cíclico formado será um caminho cíclico ambíguo-F. □

De acordo com o lema 4.1, para evitar a ocorrência de caminhos cíclicos ambíguos-F, pelo menos um evento de cada caminho de falha P_F deve fazer parte do conjunto de eventos elementares.

O teorema enunciado a seguir define as condições necessárias para que um conjunto $\Sigma'_o \subset \Sigma_o$ seja uma base para o diagnóstico de falhas.

Teorema 4.2. *Seja N_{fpes} o número de caminhos de falhas P_{F_i} obtidos a partir de G_F. Então, uma condição necessária para que $\Sigma'_o \subset \Sigma_o$ seja uma base para o diagnóstico de L com relação a P'_o e Σ_f é*

$$\Sigma'_o \cap \Sigma_{fpes,i} \neq \emptyset, i = 1, 2, \ldots, N_{fpes}.$$

Prova. Suponha que $\Sigma_o^{'}$ seja uma base para o diagnóstico de falhas de L, mas que exista i tal que $\Sigma_o^{'} \cap \Sigma_{fpes,i} = \emptyset$. Logo, existe um caminho de falha $P_{F_i} = (x_F^k, \sigma_k, x_F^{k+1}, \ldots, x_F^{k+m})$ tal que $\Sigma_{fpes,i} \subseteq ((\Sigma_o \setminus \Sigma_o^{'}) \cup \Sigma_{uo})$, o que implica, de acordo com o lema 4.1, que existe um caminho cíclico ambíguo-F em $G_V^{'}$. Portanto, de acordo com o teorema 2.2, L é não diagnosticável com relação a $P_o^{'}$ e Σ_f. □

Observação 4.1. *Note que a condição apresentada no teorema 4.2, por tratar apenas dos ciclos relacionados a estados X_{YN} que inevitavelmente serão alcançados em G_V sob a observação de todos os eventos observáveis Σ_o, é apenas necessária para que o sistema seja diagnosticável, mas não suficiente. Então, é possível que, mesmo que ela seja atendida, $\Sigma_o^{'}$ não seja uma base para o diagnóstico de falhas do sistema.*

O algoritmo apresentado a seguir fornece uma maneira sistemática de se obter todos os conjuntos de eventos elementares para o diagnóstico de falhas de um sistema.

Algoritmo 4.1. *(Algoritmo para encontrar todos os conjuntos de eventos elementares utilizando verificadores)*

Passo 1) *Construa o verificador G_V seguindo os passos do algoritmo 2.3. Obtenha o conjunto X_{YN} a partir de G_V. Defina $|X_{YN}| = N_{YN}$.*

Passo 2) *Para cada estado $x_F^i \in X_O$, $i = 1, 2, \ldots, N_{YN}$, verifique se $|\Gamma_F(x_F^i) \setminus \Gamma_V(x_N, x_F^i)| \neq \emptyset$. Em caso negativo, defina T_i como uma árvore vazia. Em caso positivo, construa uma árvore de alcançabilidade T_i a partir de G_F com raiz em x_F^i, conforme abaixo.*

Passo 2.1) *Defina $|\Gamma_F(x_F^i) \setminus \Gamma_V(x_N, x_F^i)| = \eta_{YN,i}$. Crie $\eta_{YN,i}$ descendentes de x_F^i e rotule-os como x_F^{i+1}, sendo $x_F^{i+1} = f_F(x_F^i, \sigma)$, $\sigma \in \Gamma_F(x_F^i) \setminus \Gamma_V(x_N, x_F^i)$. Rotule os ramos (x_F^i, x_F^{i+1}) com σ;*

Passo 2.2) *Um nó rotulado como x_F^{i+1}, definido na árvore, será uma folha se o estado x_F^{i+1} já tiver rotulado algum ancestral de x_F^{i+1}. Caso contrário, defina $|\Gamma_F(x_F^{i+1})| = \eta_F$. Crie η_F descendentes de x_F^{i+1} e rotule-os como $x_{F_{new}}$, sendo $x_{F_{new}} = f_F(x_F^{i+1}, \sigma)$, $\sigma \in \Gamma_F(x_F^{i+1})$. Rotule os ramos $(x_F^{i+1}, x_{F_{new}})$ como σ. Repita esse passo até que todos os nós $x_{F_{new}}$ sejam folhas.*

Passo 3) *Para cada árvore T_i não vazia, $i = 1, 2, \ldots, N_{YN}$, identifique as folhas $x_{F,i}^l$, $l = 1, \ldots, l_{T_i}$, em que l_{T_i} é o número de folhas da árvore T_i. Forme caminhos $P_{F,i}^l$, $l = 1, \ldots, l_{T_i}$, iniciando em x_F^i e terminando em $x_{F,i}^l$, $l = 1, \ldots, l_{T_i}$ (esses caminhos são os caminhos de falha que se iniciam em x_F^i).*

Passo 4) *Forme o conjunto de eventos $\Sigma_{fpes,i}$ a partir dos caminhos de falha $P^l_{F,i}$ obtidos no passo anterior utilizando produto união:*

$$\Sigma_{fpes,i} = \Sigma^1_{fpes,i} \dot{\times} \Sigma^2_{fpes,i} \dot{\times} \ldots \dot{\times} \Sigma^{l_{T_i}}_{fpes,i}.$$

Passo 5) *Calcule o conjunto Σ_{edes} formado pelos conjuntos de eventos elementares da seguinte forma:*

$$\Sigma_{edes} = \Sigma_{fpes,i_1} \dot{\times} \Sigma_{fpes,i_2} \dot{\times} \ldots \dot{\times} \Sigma_{fpes,i_k},$$

em que $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, N_{YN}\}$, tal que os índices i_1, ..., i_k denotam os índices das árvores T_i não vazias.

No passo 1 do algoritmo 4.1, o verificador G_V é obtido considerando a observação de todos os eventos do conjunto de eventos observáveis Σ_o e, em seguida, o conjunto X_{YN} é formado. O passo 2 define o critério de construção das árvores de alcançabilidade. Observe que, caso haja algum evento ativo σ no estado $(x_N, x^i_F) \in X_{YN}$, então, na construção da árvore cuja raiz é $x^i_F \in X_O$, o ramo rotulado por σ não deve ser considerado. Isso pode ser explicado pois, dada a existência de algum evento ativo σ em (x_N, x^i_F), a ocorrência de σ gera uma mudança de estado tanto em G_N quanto em G_F e, para obtenção do conjunto de eventos elementares, deve-se considerar apenas os caminhos cíclicos ambíguos-F. Nos passos 3 e 4, os caminhos das raízes até as folhas serão criados e os eventos que formam esses traços são computados. No passo 5, o produto união é utilizado para computar todos os conjuntos de eventos elementares, de forma a garantir que cada traço que compõe um caminho de falha P_F possua pelo menos um evento em cada um desses conjuntos.

Observação 4.2. *Observe que, ao final do algoritmo 4.1, é possível que Σ_{edes} possua conjuntos tais que $\widetilde{\Sigma}_{edes} \in \Sigma_{edes}$, $\widehat{\Sigma}_{edes} \in \Sigma_{edes}$ e $\widehat{\Sigma}_{edes} \subseteq \widetilde{\Sigma}_{edes}$. Assim, caso o interesse seja pelos conjuntos de eventos elementares de menor cardinalidade, deve-se acrescentar o seguinte passo ao algoritmo 4.1:*

Passo 6) *Remova de Σ_{edes} todos os conjuntos $\widetilde{\Sigma}_{edes} \in \Sigma_{edes}$ caso exista $\widehat{\Sigma}_{edes} \in \Sigma_{edes}$ tal que $\widehat{\Sigma}_{edes} \subseteq \widetilde{\Sigma}_{edes}$. Forme o conjunto Σ'_{edes} formado pelos conjuntos de eventos de menor cardinalidade com os conjuntos restantes de Σ_{edes}.*

O exemplo dado a seguir ilustra o uso do algoritmo 4.1 para computar todos os conjuntos de eventos elementares.

Exemplo 4.2. *Considere o autômato G utilizado no exemplo 3.1 e mostrado novamente na figura 4.2. Suponha que $\Sigma_o = \{a, b, c\}$ e $\Sigma_f = \{\sigma_f\}$. De acordo com o passo 1 do algoritmo 4.1, deve-se calcular o verificador utilizando o algoritmo 2.3*

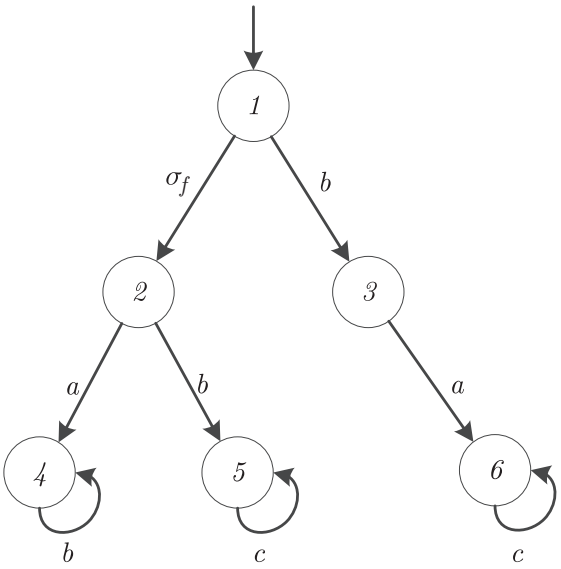


Figura 4.2: Autômato G do exemplo 4.2.

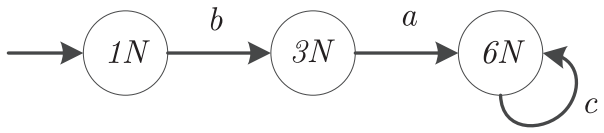
considerando-se todos os eventos de Σ_o como observáveis. As figuras 4.3(a), 4.3(b) e 4.3(c) mostram, respectivamente, os autômatos G_N, G_F e o verificador G_V. Note que G_V não possui qualquer ciclo que viole a condição 2.1 para a diagnosticabilidade de um sistema.

Em seguida, para identificar os conjuntos de eventos elementares para o diagnóstico de falhas, deve-se formar o conjunto X_{YN}, que, de acordo com a figura 4.3(c), é $X_{YN} = \{(1N\ 2Y), (3N\ 5Y)\}$.

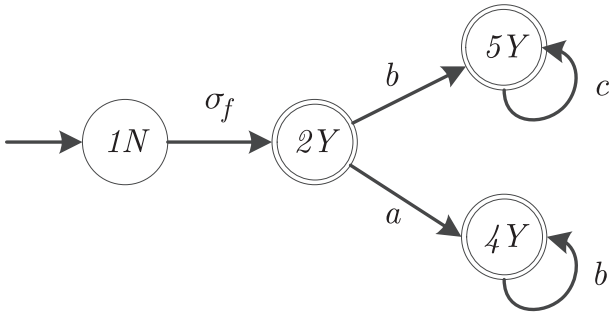
O próximo passo consiste em construir árvores de alcançabilidade a partir dos estados-origem de caminhos de falha, mostradas nas figuras 4.4(a) e 4.4(b). Observe que, apesar do evento b estar ativo no estado $2Y$ de G_F, o mesmo não aparece na árvore da figura 4.4(a). Isso ocorre pois b está ativo no estado $(1N\ 2Y)$, o que implica que sua observação não impede que tal transição ocorra, levando ao estado $(3N\ 5Y)$ em G_V. Portanto, os caminhos de falha P_F deverão ser analisados a partir da árvore de alcançabilidade que será construída posteriormente com raiz no estado $5Y$ de G_F.

A partir da figura 4.4(a), pode-se obter o caminho $P^1_{F,1} = \{2Y, a, 4Y, b, 4Y\}$, enquanto da figura 4.4(b), obtem-se o caminho $P^1_{F,2} = \{5Y, c, 5Y\}$. Nesse caso, é trivial que $\Sigma^1_{fpes,1} = \{a, b\}$ e $\Sigma^1_{fpes,2} = \{c\}$. Assim, conforme passo 5 do algoritmo 4.1, $\Sigma_{edes} = \{\{a, c\}, \{b, c\}\}$, de onde não se poderia excluir nenhum conjunto caso o passo 6 fosse executado.

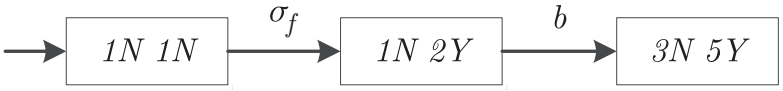
Observe que o algoritmo 4.1 garante que as condições necessárias estabelecidas no teorema 4.2 sejam atendidas. Entretanto, uma vez que essas condições são apenas necessárias, mas não suficientes, não existem garantias de que a linguagem do



(a) Autômato G_N do exemplo 4.2.



(b) Autômato G_F do exemplo 4.2.



(c) Verificador centralizado G_V.

Figura 4.3: Cálculo do autômato verificador G_V do exemplo 4.2.

(a) Árvore com raiz $x_F^1 = 2Y$. (b) Árvore com raiz $x_F^2 = 5Y$.

Figura 4.4: Árvores rotuladas criadas a partir de estados-origem de caminhos de falha.

sistema, sob observação dos conjuntos de eventos elementares, será diagnosticável. Na próxima seção, será apresentada uma condição necessária e suficiente para que o autõmato G seja diagnosticável sob observação de um subconjunto de Σ_o e um novo algoritmo, utilizando verificadores, será estabelecido.

4.2 Bases mínimas para o diagnóstico de falhas em SEDs

Uma vez que a condição imposta no teorema 4.2 é apenas necessária, é provável que um conjunto $\Sigma_o^{'} \in \Sigma_{edes}$ não seja uma base para o diagnóstico de falhas do sistema. Dessa forma, assim como em [29], para obter novos conjuntos candidatos a serem bases mínimas do sistema, será necessário adicionar eventos pertencentes a $\Sigma_o \setminus \Sigma_o^{'}$ em $\Sigma_o^{'}$ para formar um novo conjunto $\Sigma_o^{''}$. Isto é, deverá ser criteriosamente avaliado cada um dos eventos $\sigma \in \Sigma_o \setminus \Sigma_o^{'}$ de forma a criar um conjunto Σ_{new} tal que o conjunto $\Sigma_o^{''} = \Sigma_o^{'} \cup \Sigma_{new}$ seja base mínima para o diagnóstico de falhas do sistema.

Conforme enunciado em [29], uma maneira de se encontrar os eventos que devem ser adicionados a $\Sigma_o^{'}$ seria pela força bruta, que consiste em adicionar evento a evento exaustivamente no conjunto $\Sigma_o^{'}$ e verificar a diagnosticabilidade da linguagem do sistema com relação a $\Sigma_o^{''}$ e Σ_f. Entretanto, o objetivo dessa dissertação é explorar a estrutura do sistema para obter, de forma sistemática, todos os eventos essenciais para manutenção da diagnosticabilidade do sistema original.

A seguir, será apresentado um teorema que estabelece as condições necessárias e suficientes para que a linguagem de um sistema seja diagnosticável com relação a uma projeção $P_o^{'} : \Sigma^* \rightarrow \Sigma_o^{'*}$ e a um conjunto de eventos de falhas Σ_f para que, a partir desse, seja proposto um algoritmo que garanta essas condições e retorne todas as bases mínimas para o diagnóstico de falhas em SEDs. Entretanto, será necessário apresentar previamente o conceito de componentes fortemente conexos, dado abaixo.

Definição 4.4. *(Componentes fortemente conexos) Um conjunto de estados* $\{x_1, x_2, \ldots, x_n\} \subseteq X$ *forma um componente fortemente conexo (CFC) em um autômato* G *se,* $\forall(x_i,\ x_k)$, $i = 1, \ldots, n$, $k = 1, \ldots, n$, *existe uma sequência* $s \in \Sigma^*$ *tal que* $f(x_i, s) = x_k$.

Conforme definição 4.4, um componente fortemente conexo (CFC) no contexto de sistemas a eventos discretos, é um conjunto de um ou mais estados entre os quais existe alcançabilidade mútua, ou seja, a partir de cada estado do componente é possível alcançar todos os demais e retornar a ele mesmo, isto é, existe um caminho de ida e de volta que conecta cada estado do componente aos demais. A partir desse conceito, o teorema 4.3 pode ser enunciado.

Teorema 4.3. *Seja* L *diagnosticável com relação à projeção* P_o *e* $\Sigma_f = \{\sigma_f\}$ *e seja* $G_V^{'}$ *o verificador com observação reduzida, obtido a partir de* $\Sigma_o^{'} \subset \Sigma_o$ *e* $x_{YN}^{i'} \in X_{YN}^{'}$ *um estado de* $G_V^{'}$ *cuja componente oriunda de* G_F *seja igual a* $(x,\ Y)$, *em que* $x \in X$. *Então,* L *será diagnosticável com relação a projeção* $P_o^{'}$ *e* Σ_f *se, e somente se,*

o verificador $G_V^{'}$ *não possuir componentes fortemente conexos* $CFC = \{x_{YN}^{1'}, \ldots, x_{YN}^{n'}\}$ *que satisfaçam à seguinte condição:*

$\exists i, j \in \{1, 2, \ldots, n\}$ *tal que: para algum* $x_{YN}^{i'}$, *tem-se* $x_{YN}^{j'} = f(x_{YN}^{i'}, \sigma_i)$, $\sigma_i \in \Sigma$.

Prova. Seja $cl = (x_V^k, \sigma_k, x_V^{k+1}, \ldots, x_V^l, \sigma_l, x_V^k)$ um caminho cíclico que viola as condições do teorema 2.2. Uma vez que todos os estados que formam o caminho cíclico podem alcançar os demais estados que o compõe e retornar a si mesmo, então a existência de cl implica na existência de um componente fortemente conexo formado pelo estados de cl. Considere agora a existência de um componente fortemente conexo CFC que viola a condição de diagnosticabilidade. Então, dado que os estados que compõem o componente possuem alcançabilidade mútua, pode-se concluir que a existência do CFC implica na existência de pelo menos um ciclo interno ao CFC que viola a condição de diagnosticabilidade. Dessa forma, a prova realizada para o teorema 2.2 em termos de ciclos pode ser estendida em termos de componentes fortemente conexos. □

Uma vez definida uma condição necessária e suficiente para que a linguagem de um sistema seja diagnosticável com relação a $P_o^{'}$ e Σ_f, a idéia do algoritmo para obtenção de todas as bases mínimas para o diagnóstico de falhas em um sistema é verificar, após o cálculo dos conjuntos de eventos elementares, para quais sequências de $G_V^{'}$ essa condição é violada. Dessa forma, bastará analisar as sequências que violam a condição imposta pelo teorema 4.3 para definir quais eventos deverão ser incluídos em $\Sigma_o^{'}$.

Entretanto, antes da apresentação do algoritmo, será necessário apresentar a definição da função de renomeação inversa R^{-1}. Observe que, ao construir um autômato verificador conforme proposto em [26], os eventos não-observáveis, com exceção do evento de falha, são renomeados por uma função de renomeação R. Assim, para avaliar os traços que violam a condição necessária e suficiente imposta no teorema 4.3, será necessário recuperar, a partir de $G_V^{'}$, as sequências originais que ocorrem em G, o que será realizado através da função R^{-1}.

Definição 4.5. *(Função de renomeação inversa* R^{-1}*) A função de renomeação inversa* R^{-1} *é definida como:*

$$R^{-1} : \Sigma_R \rightarrow \Sigma_N$$

$$\sigma_R \mapsto \sigma,$$

sendo $\sigma_R = R(\sigma)$ *com a seguinte extensão de domínio* Σ_R^* : $R^{-1}(s_R\sigma_R) = R^{-1}(s_R)R^{-1}(\sigma_R)$, $\forall s_R \in \Sigma_R^*$ *e* $\sigma_R \in \Sigma_R$ *e* $R^{-1}(\varepsilon) = \varepsilon$.

Considere também a projeção $P_R : (\Sigma \cup \Sigma_R)^* \rightarrow \Sigma_R^*$, e $P : (\Sigma \cup \Sigma_R)^* \rightarrow \Sigma^*$. Então, o algoritmo abaixo provê uma forma direta de obtenção, a partir do verifica-

dor $G_V^{'}$, dos traços normais e de falha de mesma projeção que violam as condições de diagnosticabilidade consideradas nessa dissertação.

Algoritmo 4.2. *(Algortimo de obtenção dos traços ambíguos normal e de falha a partir de uma sequência em $G_V^{'}$)*

Passo 1) *Dado o verificador $G_V^{'}$, identifique um caminho cíclico que viole as condições impostas pelo teorema 4.3 e obtenha o traço v associado a esse caminho cíclico.*

Passo 2) *Obtenha $s_R = P_R(v)$ e $s_F = P(v)$.*

Passo 3) *Obtenha $s_N = R^{-1}(s_R)$.*

Uma vez definida a forma de se recuperar os traços normal e de falha, s_N e s_F, respectivamente, a partir de uma sequência em $G_V^{'}$, é possível apresentar o algoritmo que retorna as bases mínimas para o diagnóstico de falhas em SEDs.

Algoritmo 4.3. *(Método da árvore de caminhos ambíguos)*

Passo 1) *Dado o autômato G que descreve o sistema, utilize o algoritmo 4.1 para obter $\Sigma_{edes}^{'}$.*

Passo 2) *Para cada elemento $\Sigma_{edes,i}^{'}$ pertencente a $\Sigma_{edes}^{'}$, faça:*

Passo 2.1) $\Sigma_o^{'} = \Sigma_{edes,i}^{'}$.

Passo 2.2) *Calcule o verificador $G_V^{'}$ considerando $\Sigma_o^{'}$ como o conjunto de eventos observáveis.*

Passo 2.3) *Verifique se $G_V^{'}$ atende às condições de diagnosticabilidade apresentadas no teorema 4.3. Em caso positivo, $\Sigma_o^{'}$ é uma base mínima para diagnosticabilidade do sistema. Caso contrário, $\Sigma_o^{'}$ deve ser aumentado com eventos de $\Sigma_o \setminus \Sigma_o^{'}$.*

Passo 2.4) *Utilize o algoritmo 3.2 para obter todos os caminhos primos de $G_V^{'}$.*

Passo 2.5) *Utilize a árvore obtida no passo anterior para identificar as sequências v_j de $G_V^{'}$ que violam às condições de diagnosticabilidade estabelecidas no teorema 4.3. Para cada uma dessas sequências v_j, faça:*

Passo 2.5.1) *Utilize o algoritmo 4.2 para recuperar as sequências normal e de falha, s_N e s_F, respectivamente, a partir de v_j.*

Passo 2.5.2) *Verifique os eventos σ pertencentes a $\Sigma_o \setminus \Sigma_o^{'}$ que tornam $P_o^{''}(s_N) \neq P_o^{''}(s_F)$, em que $P_o^{''} : \Sigma^* \rightarrow \Sigma_o^{''*}$ e $\Sigma_o^{''} = \Sigma_o^{'} \cup \{\sigma\}$, e forme o conjunto Σ_{new}^{j} com esses eventos.*

Passo 2.6) *Obtenha* Σ_{new} *da seguinte forma:*

$$\Sigma_{new} = \Sigma^1_{new} \dot{\times} \Sigma^2_{new} \dot{\times} \ldots \Sigma^n_{new},$$

sendo n o número de sequências violadoras encontradas no passo 2.5.

Passo 2.7) *Remova de* Σ_{new} *todos os conjuntos* $\widetilde{\Sigma}_{new} \in \Sigma_{new}$ *caso exista* $\widehat{\Sigma}_{new} \in \Sigma_{new}$ *tais que* $\widetilde{\Sigma}_{new} \supseteq \widehat{\Sigma}_{new}$.

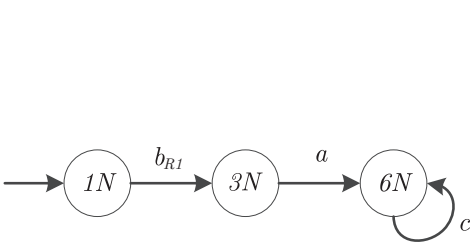
Passo 2.8) *Defina* $\Sigma^i_{min} = \Sigma^i_{edes} \cup \Sigma_{new}$

Passo 3) *Obtenha* $\Sigma_{min} = \{\Sigma^1_{min}\} \cup \{\Sigma^2_{min}\} \cup \ldots \cup \{\Sigma^k_{min}\}$, *sendo k o número de elementos de* Σ_{edes} *obtidos no passo 1.*

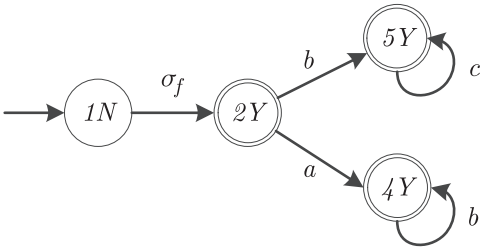
Passo 4) *Remova de* Σ_{min} *todos os conjuntos* $\widetilde{\Sigma}_{min} \in \Sigma_{min}$ *caso exista* $\widehat{\Sigma}_{min} \in \Sigma_{min}$ *tais que* $\widetilde{\Sigma}_{min} \supseteq \widehat{\Sigma}_{min}$.

Para ilustrar a aplicação do algoritmo 4.3, considere os dois exemplos dado a seguir.

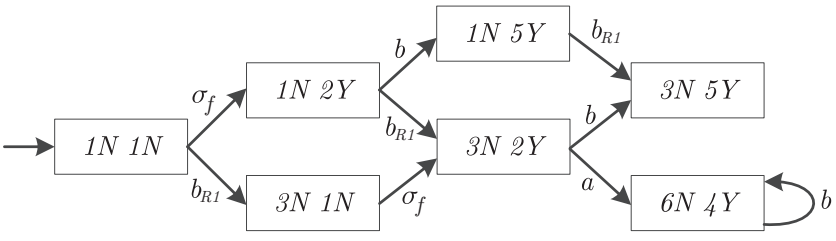
Exemplo 4.3. *Seja G o autômato mostrado na figura 4.2 e suponha que* $\Sigma_o = \{a, b, c\}$ *e* $\Sigma_f = \{\sigma_f\}$. *No primeiro passo do algoritmo 4.3, deve-se utilizar o algoritmo 4.1 para obter* Σ'_{edes}, *procedimento já realizado no exemplo 4.2, de onde se concluiu que* $\Sigma_{edes} = \Sigma'_{edes} = \{\{a, c\}, \{b, c\}\}$. *Considere, inicialmente,* $\Sigma'_o = \{a, c\}$.



(a) $G_{N,1}$ considerando $\Sigma'_o = \{a, c\}$.



(b) Autômato G_F.



(c) Verificador G'_V, considerando $\Sigma'_o = \{a, c\}$.

Figura 4.5: Cálculo do autômato verificador G'_V considerando $\Sigma'_o = \{a, c\}$.

A figura 4.5(c) mostra o verificador $G_V^{'}$ obtido após a realização da composição paralela entre os autômatos $G_{N,1}$ e G_F, mostrados nas figuras 4.5(a) e 4.5(b), respectivamente.

Em seguida, deve ser verificado se $G_V^{'}$ possui algum componente fortemente conexo que viole as condiçoes de diagnosticabilidade apresentadas no teorema 4.3. A partir da figura 4.5(c), pode-se afirmar que a existência do componente fortemente conexo $\{(6N\ 4Y)\}$ torna a linguagem do sistema não diagnosticável com relação a $P_o^{'}$ e Σ_f. Nesse caso, eventos de $\Sigma_o \setminus \Sigma_o^{'}$ deverão ser adicionados a $\Sigma_o^{'}$. Será necessário, então, construir uma árvore de alcançabilidade de $G_V^{'}$. Para isso, deve-se utilizar o algoritmo 3.2. O resultado pode ser visto na figura 4.6, de onde podem ser obtidos os traços violadores $v^1 = \sigma_f b_{R1} ab$ e $v^2 = b_{R1}\sigma_f ab$.

Uma vez encontradas as sequências que violam as condições de diagnosticabilidade, deve-se utilizar o algoritmo 4.2 para recuperar os traços normais e de falha associados, para, em seguida, compará-los de forma a descobrir o(s) evento(s) que tornam suas projeções distintas. Observe que a sequência v^1 corresponde aos traços $s_N^1 = ba$ e $s_F^1 = \sigma_f ab$. É evidente que, para que as projeções $P_o^{''}(s_N^1) \neq P_o^{''}(s_F^1)$, o evento b deve ser observado. Assim, $\Sigma_{new}^1 = \{b\}$.Uma vez que a sequência v^2 corresponde aos mesmos traços de v^1, então $\Sigma_{new}^2 = \{b\}$, resultando em $\Sigma_{new} = \{b\}$. Assim, $\Sigma_{min}^1 = \Sigma_{edes}^1 \cup \Sigma_{new} = \{a, b, c\}$.

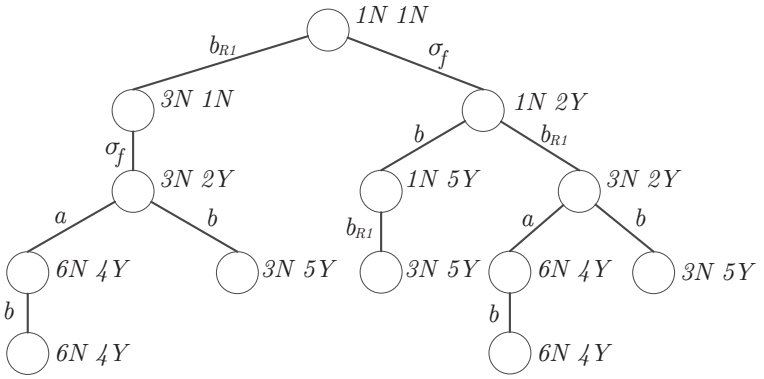
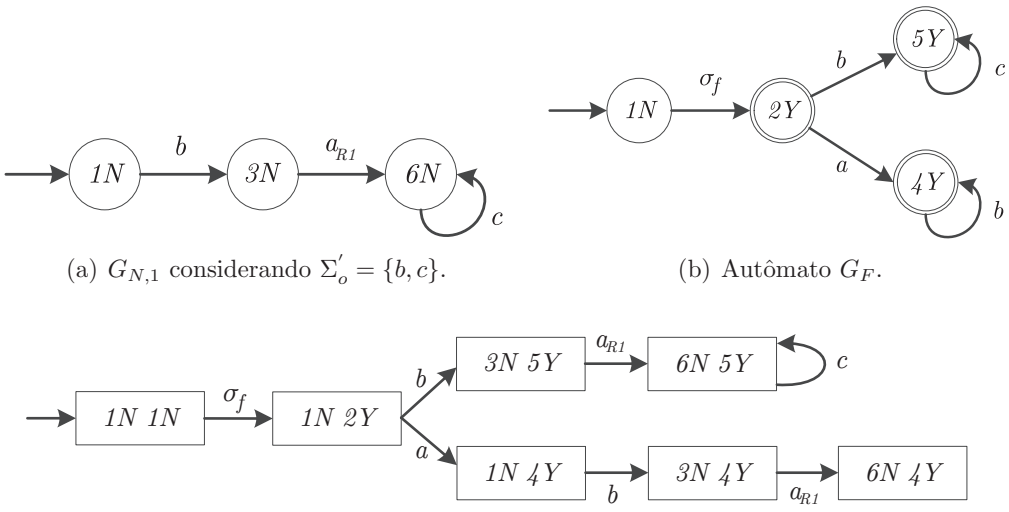


Figura 4.6: Árvore de alcançabilidade de $G_V^{'}$ considerando $\Sigma_o^{'} = \{a, c\}$.

Será iniciada, então, a análise para $\Sigma_o^{'} = \{b, c\}$. A figura 4.7(c) mostra o verificador com observação reduzida para esse conjunto de eventos observáveis. Mais uma vez, o verificador $G_V^{'}$, mostrado na figura 4.7(c), possui um componente fortemente conexo que viola as condições estabelecidas pelo teorema 4.3. Por isso, será utilizado o algoritmo 3.2 para obter todos os caminhos primos de $G_V^{'}$. Esse resultado é mostrado na figura 4.8.

Note que a sequência que viola as condições de diagnosticabilidade $v^1 = \sigma_f ba_{R1} c$ corresponde às sequências $s_N^1 = bac$ e $s_F^1 = \sigma_f bc$. Para que as projeções



(a) $G_{N,1}$ considerando $\Sigma_o^{'} = \{b, c\}$.

(b) Autômato G_F.

(c) Verificador $G_V^{'}$, considerando $\Sigma_o^{'} = \{b, c\}$.

Figura 4.7: Cálculo do autômato verificador $G_V^{'}$ sendo $\Sigma_o^{'} = \{b, c\}$.

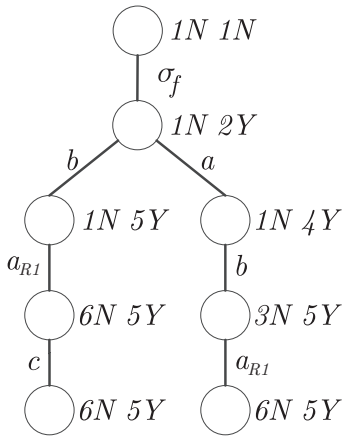


Figura 4.8: Árvore de alcançabilidade de $G_V^{'}$ considerando $\Sigma_o^{'} = \{b, c\}$.

$P_o^{''}(s_N^1) \neq P_o^{''}(s_F^1)$, o evento a deve ser observado. Assim, $\Sigma_{new} = \{a\}$ e $\Sigma_{min}^2 = \{a, b, c\}$, resultando em $\Sigma_{min} = \{\Sigma_{min}^1\} \cup \{\Sigma_{min}^2\} = \{a, b, c\}$. Dessa forma, o conjunto $\{a, b, c\}$ será a única base mínima para o diagnóstico desse sistema.

Exemplo 4.4. *Considere o sistema modelado pelo autômato G mostrado na figura 4.9 e suponha $\Sigma_o = \{a, b, c, d, e\}$ e $\Sigma_f = \{\sigma_f\}$. O passo 1 do algoritmo 4.3 requer o cálculo do verificador G_V, conforme algoritmo 2.3. Note que o verificador G_V, mostrado na figura 4.10, não possui componentes fortemente conexos que violam a diagnosticabilidade do sistema e, portanto, a linguagem L é diagnosticável considerando a observação dos eventos pertencentes a Σ_o.*

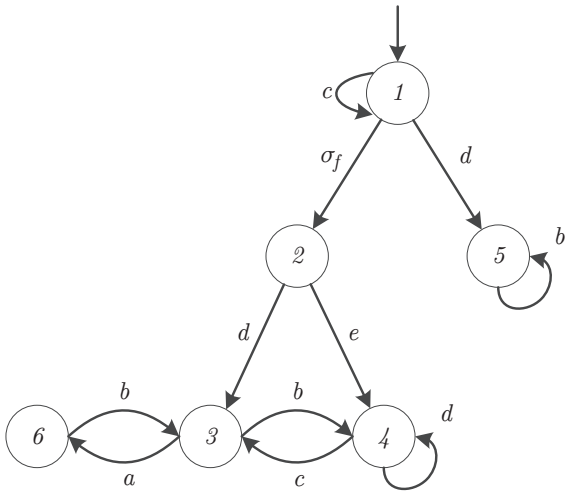


Figura 4.9: Autômato G do exemplo 4.4.

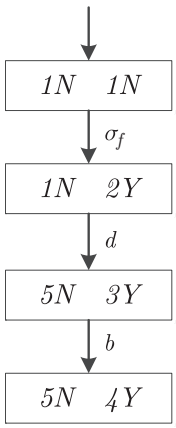


Figura 4.10: Verificador centralizado G_V.

O próximo passo consiste em identificar os estados do verificador nos quais a componente associada ao autômato G_F indica a ocorrência do evento de falha σ_f. Observando a figura 4.10, pode-se afirmar que $X_{YN} = \{(1N\ 2Y), (5N\ 3Y), (5N\ 4Y)\}$.

Em seguida, deve-se construir uma árvore de alcançabilidade a partir do autômato $G_F = G \| A_{label}$ para cada um dos estados-origem de caminhos de falha que pertencem a X_O, obtidos por intermédio da figura 4.11. Observe, na figura 4.10, que é possível alcançar o estado $(5N\ 3Y)$ a partir de $(1N\ 2Y)$, através da transição rotulada por d. Dessa forma, conforme o algoritmo 4.1, a árvore cuja raiz é $2Y$ não apresentará o ramo rotulado por d. De maneira equivalente, a árvore cuja raiz é $3Y$ não apresentará o ramo b.

Com base na figura 4.11, os seguintes caminhos podem ser forma-

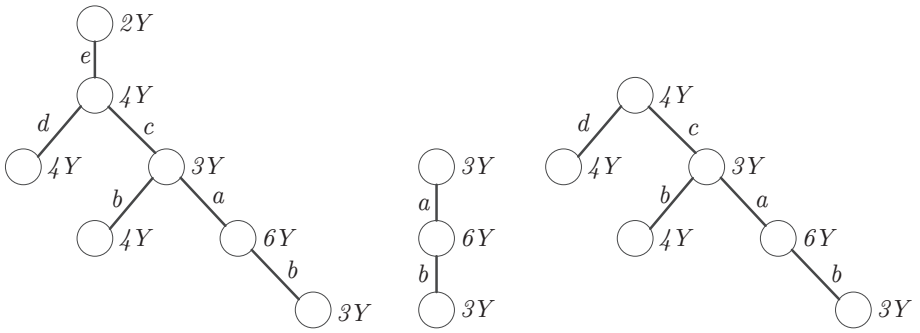


Figura 4.11: Árvores de alcançabilidade criadas a partir dos estado-origem de caminhos de falha.

dos: $P^1_{F,1} = \{2Y, e, 4Y, d, 4Y\}$, $P^2_{F,1} = \{2Y, e, 4Y, c, 3Y, b, 4Y\}$, $P^3_{F,1} = \{2Y, e, 4Y, c, 3Y, a, 6Y, b, 3Y\}$, $P^1_{F,2} = \{3Y, a, 6Y, b, 3Y\}$, $P^1_{F,3} = \{4Y, d, 4Y\}$, $P^2_{F,3} = \{4Y, c, 3Y, b, 4Y\}$ *e* $P^3_{F,3} = \{4Y, c, 3Y, a, 6Y, b, 3Y\}$. *Assim,* $\Sigma^1_{fpes,1} = \{d, e\}$, $\Sigma^2_{fpes,1} = \{b, c, e\}$, $\Sigma^3_{fpes,1} = \{a, b, c, e\}$, $\Sigma^1_{fpes,2} = \{a, b\}$, $\Sigma^1_{fpes,3} = \{d\}$, $\Sigma^2_{fpes,3} = \{b, c\}$ *e* $\Sigma^3_{fpes,3} = \{a, b, c\}$.

Utilizando o produto união, definido no capítulo 3, e reduzindo o conjunto Σ_{edes} *conforme passo 6 do algoritmo 4.1, obtém-se* $\Sigma'_{edes} = \{\{b, d\}, \{a, c, d\}\}$.

Primeiramente, considere $\Sigma'_o = \{b, d\}$. *A figura 4.12 mostra o verificador com observação reduzida obtido para esse conjunto de eventos observáveis. Note que existe um componente fortemente conexo em* G'_V *que torna o sistema não diagnosticável para* $\Sigma'_o = \{b, d\}$. *Dessa forma, será necessário incluir, de maneira ordenada, novos eventos ao conjunto* Σ'_o.

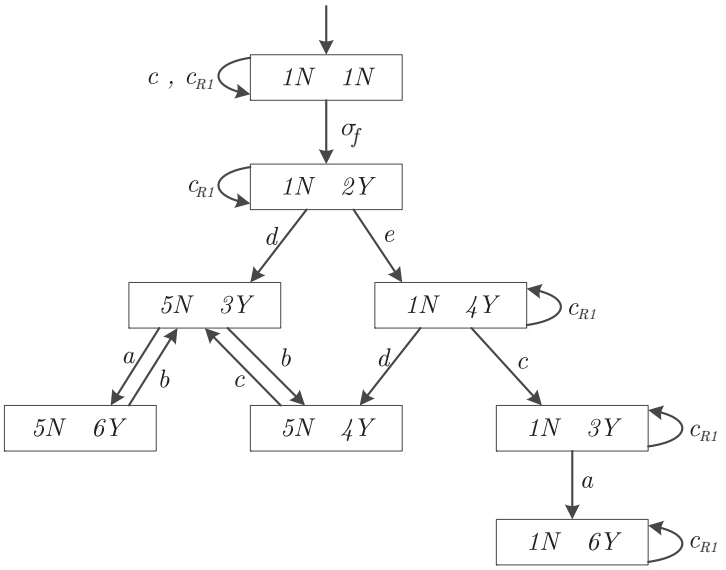


Figura 4.12: Verificador G'_V considerando $\Sigma'_o = \{b, d\}$.

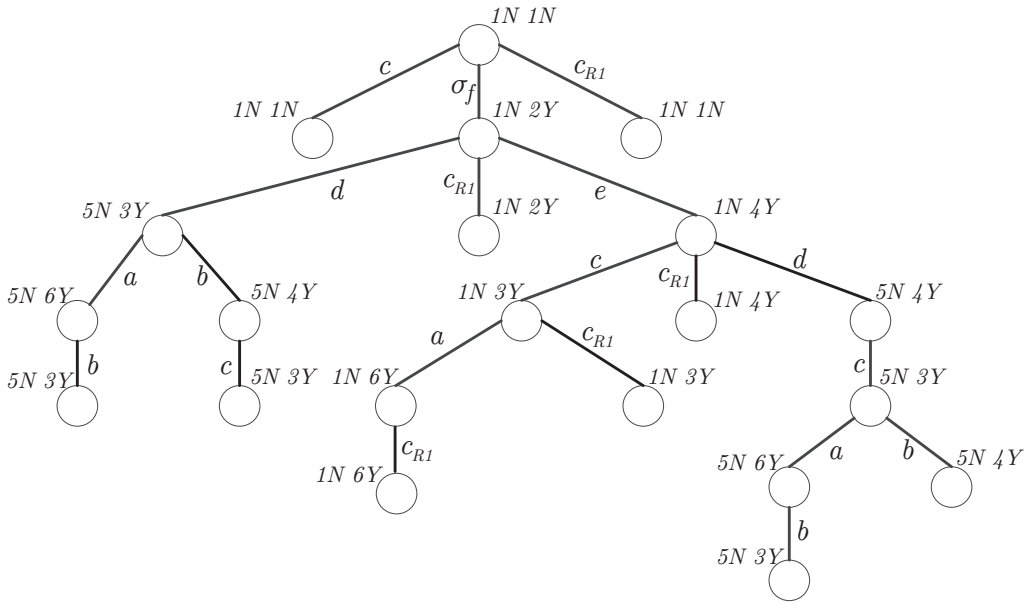


Figura 4.13: Árvore de $G_V^{'}$ considerando $\Sigma_o^{'} = \{b, d\}$.

A figura 4.13 mostra a árvore construída aplicando-se o algoritmo 3.2 ao verificador $G_V^{'}$, obtido no passo anterior. Dada a árvore de alcançabilidade de $G_V^{'}$, as sequências que violam a definição de diagnosticabilidade são:

$$\begin{cases} s_F^1 = \sigma_f dab \Rightarrow \Sigma_{new}^1 = \{a\} \\ s_N^1 = db \end{cases}$$

$$\begin{cases} s_F^2 = \sigma_f dbc \Rightarrow \Sigma_{new}^2 = \{c\} \\ s_N^2 = db \end{cases}$$

$$\begin{cases} s_F^3 = \sigma_f edcb \Rightarrow \Sigma_{new}^3 = \{c, e\} \\ s_N^3 = db \end{cases}$$

$$\begin{cases} s_F^4 = \sigma_f edcab \Rightarrow \Sigma_{new}^4 = \{a, c, e\} \\ s_N^4 = db \end{cases}$$

Em seguida, o produto união deve ser utilizado de forma a garantir que o conjunto final obtido será mínimo:

$$\Sigma_{new} = \Sigma_{new}^1 \dot{\times} \ldots \dot{\times} \Sigma_{new}^4 = \{\{a, c\}, \{a, c, e\}\}$$

Contudo, como $\{a, c, e\} \supseteq \{a, c\}$*, então* $\Sigma_{new} = \{a, c\}$*, resultando em* $\Sigma^1_{min} = \Sigma^1_{edes} \cup \Sigma_{new} = \{\{a, b, c, d\}\}$.

Os mesmos passos serão repetidos agora para $\Sigma_o^{'} = \Sigma^2_{edes} = \{a, c, d\}$.

A figura 4.14 e a figura 4.15 mostram, respectivamente, o verificador com observação reduzida $G_V^{'}$ *e sua árvore de alcançabilidade, considerando* $\Sigma_o^{'} = \{a, c, d\}$.

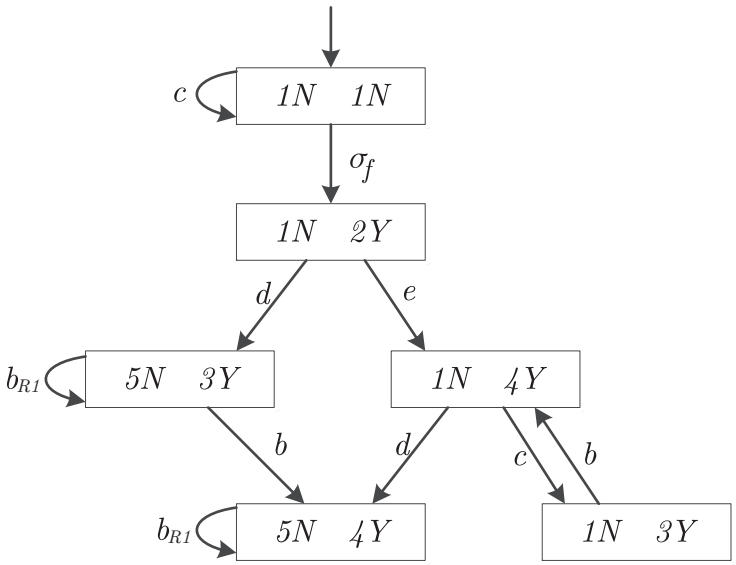


Figura 4.14: Verificador $G_V^{'}$ considerando $\Sigma_o^{'} = \{a, c, d\}$.

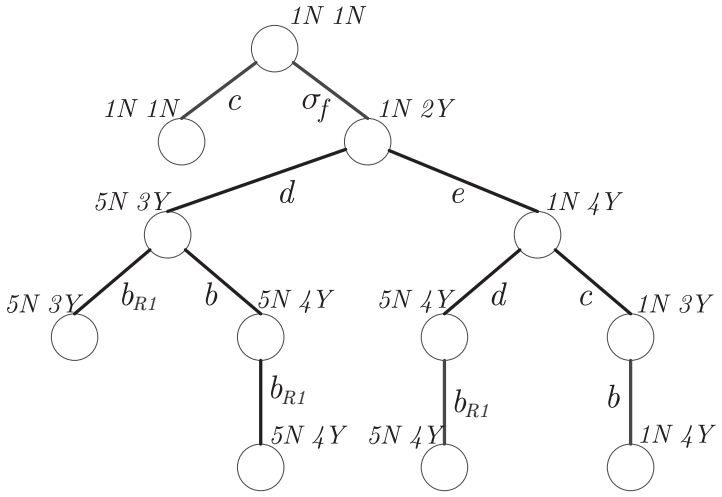


Figura 4.15: Árvore de $G_V^{'}$ considerando $\Sigma_o^{'} = \{a, c, d\}$.

A seguir, a sequência que viola as condições de diagnosticabilidade, obtida através da figura 4.15, é analisada.

$$\begin{cases} s_F^1 = \sigma_f ecb \Rightarrow \Sigma_{new}^1 = \{b, e\} \\ s_N^1 = c \end{cases}$$

Assim, $\Sigma_{new} = \{\{b\}, \{e\}\}$*, resultando em* $\Sigma_{min}^2 = \{\{a, b, c, d\}, \{a, c, d, e\}\}$.
Então, $\Sigma_{min} = \Sigma_{min}^1 \cup \Sigma_{min}^2 = \{\{a, b, c, d\}, \{a, c, d, e\}\}$.

4.3 Complexidade computacional dos algoritmos 4.1 e 4.3

O algoritmo 4.1 prevê, em seu primeiro passo, o cálculo do verificador G_V proposto no algoritmo 2.3, cuja complexidade computacional é $O(|X|^2|\Sigma|)$, sendo $|X|$ a cardinalidade do conjunto de estados de G e $|\Sigma|$ a cardinalidade do conjunto de eventos de G. Em seguida, para cada estado $x_{V_{YN}}^i \in X_{YN}$, $i = 1, 2, \ldots, N_{YN}$, deve-se criar uma árvore de alcançabilidade cuja raiz é $x_F^i \in X_O$. Observe que, o pior caso computacional é formado da seguinte maneira: ao construir o verificador G_V, o estado inicial será da forma NN, ou seja, será um estado normal. A partir do estado inicial do verificador, $|\Sigma|$ transições podem ocorrer e, uma vez que G_V é determinístico, apenas uma transição pode ser rotulada pelo evento de falha σ_f, sendo que as $|\Sigma|-1$ transições restantes levam a estados normais. Posteriormente, em cada um dos $|\Sigma|-1$ estados normais alcançados podem existir transições rotuladas pelo evento de falha σ_f, causando a ocorrência de estados do tipo $N\,Y$. Dessa forma, pode-se afirmar que a complexidade computacional do passo 2 é $O(|\Sigma|^{|X|-1}+(|\Sigma|-1)|\Sigma|^{|X|-2})$. Portanto, a complexidade computacional do algoritmo 4.1 é $O(|\Sigma|^{|X|-1})$.

Já a complexidade do algoritmo 4.3 é dada da seguinte forma: o passo 1 requer o uso do algoritmo 4.1, cuja complexidade computacional é $O(|\Sigma|^{|X|-1})$. Em seguida, deverá ser calculado um verificador com observação reduzida para cada elemento de Σ_{edes} obtido no passo 1, cuja complexidade é $O(|X|^2|\Sigma|)$. Uma vez que um ou mais verificadores com observação reduzida $G_V^{'}$ apontem que o sistema é não diagnosticável, os demais passos do algoritmo 4.3 deverão ser executados. Assim, deverá ser construída uma árvore de alcançabilidade para cada verificador não diagnosticável do passo anterior. A complexidade do algoritmo 3.2 aplicado a um verificador $G_V^{'}$ é $O(|\Sigma|^{|X_V^{'}|})$, sendo $|X_V^{'}|$ a cardinalidade do número de estados de $G_V^{'}$, que é limitada superiormente por $2|X|^2$. Pode-se afirmar, então, que o algoritmo 3.2 aplicado a $G_V^{'}$ é $O(|\Sigma|^{|X|^2})$. Os demais cálculos do algoritmo 4.3, por serem de ordem mais baixa, são dominados pelos termos de ordem superior, e, por isso, serão omitidos. Dessa forma, pode-se afirmar que a complexidade computacional do algoritmo 4.3 é $O(|\Sigma|^{|X|^2})$.

Observe que o algoritmo 4.3 proposto neste capítulo possui menor complexidade computacional que o desenvolvido em [29]. Contudo, ambos, apesar de explorarem a estrutura do sistema em questão, ainda podem ser, no pior caso, computacionalmente piores que a utilização da força bruta. A força bruta consiste em uma tentativa exaustiva de todos os subconjuntos possíveis formados a partir de Σ_o. Seja $|\Sigma_o|$ a cardinalidade do conjunto de eventos observáveis, então, existem $2^{|\Sigma_o|-2}$ subconjuntos que devem ser testados utilizando a força bruta. Se cada um desses testes for realizado utilizando autômato verificadores, o custo computacional total será $O(2^{|\Sigma_o|-2}|X|^2|\Sigma|)$, que é menor que complexidade apresentada pelos algoritmos existentes atualmente, inclusive os apresentados nos capítulos 3 e 4 desta dissertação.

Para contornar esse problema, será proposto, no capítulo 5, um novo algoritmo para obtenção das bases mínimas para diagnosticabilidade de sistemas utilizando verificadores.

Capítulo 5

Segundo método para obtenção de bases mínimas para o diagnóstico de falhas em SEDs utilizando verificadores

No capítulo 4, um algoritmo baseado em autômatos verificadores foi apresentado para o cálculo de todas as bases mínimas necessárias para o diagnóstico de falhas em um sistema a eventos discretos. Embora o método proposto utilize as informações e propriedades do sistema, sua complexidade computacional pode ser, no pior caso, superior à complexidade do método de busca exaustiva. Esse mesmo problema de complexidade computacional ocorre também com o algoritmo 3.5, que possui o mesmo propósito. O problema de se encontrar todos os subconjuntos de Σ_o com menor cardinalidade que garantem a diagnosticabilidade da linguagem do sistema é classificado na literatura como *NP-completo* (vide [8], [33], [34], dentre outros) e, por isso, já era esperado que o custo computacional do método proposto no capítulo anterior não fosse polinomial.

Um problema pertence à classe P se pode ser resolvido em tempo polinomial. Por outro lado, a classe NP é formada pelos problemas que são verificáveis em tempo polinomial. Se um problema pertence à classe NP e sua solução está computacionalmente relacionada à solução dos demais problemas NP, isto é, caso uma solução polinomial seja obtida para um deles, todos os demais podem ser resolvidos em tempo polinomial, então o problema é dito *NP-completo* [37]. Problemas *NP-completos* representam, atualmente, um grande desafio na área de computação, uma vez que nenhum algoritmo em tempo polinomial foi descoberto para resolvê-los.

Neste capítulo, será apresentado um novo algoritmo para a obtenção de todas as bases mínimas que garantem a diagnosticabilidade da linguagem do sistema. Con-

forme será visto ao fim do capítulo, esse algoritmo, baseado em autômatos verificadores, possui complexidade computacional menor que todos os demais apresentados neste trabalho.

5.1 Método da árvore de eventos

Assim como o algoritmo proposto no capítulo 4, o algoritmo que será desenvolvido nesta seção é baseado em autômatos verificadores. Verificadores possuem a vantagem de, no pior caso, terem crescimento de estados polinomial com relação ao número de estados do autômato que modela o sistema. Além disso, ao utilizar verificadores em detrimento dos diagnosticadores, evita-se a ocorrência de ciclos escondidos, o que possibilita uma análise direta das sequências que violam a diagnosticabilidade. Note que o algoritmo 4.3 foi desenvolvido traçando-se um paralelo com o algoritmo 3.5 e adaptando-se suas ideias para o uso de verificadores. Portanto, uma vez que o algoritmo 3.5 possui complexidade computacional maior que o método de busca exaustiva, era esperado que o mesmo resultado ocorresse com o algoritmo 4.3. Entretanto, a maior vantagem do algoritmo 4.3, se comparado ao 3.5, é a simplicidade de uso e compreensão que o mesmo apresenta. O objetivo do algoritmo que será desenvolvido a seguir é, primeiramente, obter um método computacionalmente melhor que os demais apresentados neste trabalho, podendo ter, inclusive, desempenho superior ao uso do método de busca exaustiva e, por fim, inovar a maneira como as bases mínimas de um determinado sistema podem ser obtidas.

Como será visto neste capítulo, o algoritmo que será proposto requer o cálculo de autômatos verificadores repetidas vezes. Assim, o conhecimento do mecanismo de cálculo de um verificador $G_V^{'}$ com observação reduzida e também das mudanças que ocorrem no mesmo caso um evento passe a ser considerado observável são de grande relevância. Para tal, o algoritmo a seguir apresenta uma forma alternativa de se obter um verificador $G_V^{''}$ com o conjunto de eventos observáveis $\Sigma_o^{''}$, a partir de outro verificador $G_V^{'}$, cujo conjunto de eventos observáveis é $\Sigma_o^{'}$, sendo $\Sigma_o^{''} = \Sigma_o^{'} \cup \{\sigma\}$, em que σ é um evento qualquer que se deseja observar. Apesar das complexidades desse novo algoritmo e do algoritmo 2.3 serem idênticas, o algoritmo 5.1 permite compreender como componentes fortemente conexos que violam a diagnosticabilidade da linguagem do sistema podem ser eliminados com a observação de eventos.

Algoritmo 5.1. *(Algoritmo para obtenção de um verificador a partir de outro verificador com observação reduzida)*

Seja $G_V^{'}$ o verificador de G obtido considerando-se o conjunto de eventos observáveis $\Sigma_o^{'}$, tal que o evento $\sigma \notin \Sigma_o^{'}$, e seja $x_{0,V}^{'}$, o estado inicial de $G_V^{'}$. Então, os seguintes passos devem ser executados para a obtenção de um verificador $G_V^{''}$ cujo conjunto de eventos observáveis é $\Sigma_o^{''} = \Sigma_o^{'} \cup \{\sigma\}$:

Passo 1) *Faça* $Q \leftarrow x_{0V}^{'}$.

Passo 2) *Enquanto* $Q \neq \emptyset$, *faça:*

Passo 2.1) $u \leftarrow Q_1$, *sendo* Q_1 *o primeiro elemento da fila* Q. *Retire* Q_1 *de* Q.

Passo 2.2) *Verifique se* $\sigma \in \Gamma(u)$. *Em caso positivo:*

Passo 2.2.1) *Verifique a existência de um estado* v *tal que* $f_V^{'}(v, \sigma_R) = u$. *Em caso positivo:*

→ *Conecte o estado* v *ao estado* $f_V^{'}(u, \sigma)$ *com* σ, *caso essa transição não exista.*

→ *Faça* $Q \leftarrow f_V^{'}(u, \sigma)$, *caso esse estado não tenha sido visitado anteriormente.*

→ *Apague* σ_R *de* $\Gamma(v)$.

→ *Apague* σ *de* $\Gamma(u)$.

Passo 2.2.2) *Verifique a existência de um estado* $w = f_V^{'}(u, \sigma)$, *tal que* $\sigma_R \in \Gamma(w)$. *Em caso positivo:*

→ *Apague* σ *de* $\Gamma(u)$.

→ *Conecte o estado* u *ao estado* $f_V^{'}(w, \sigma_R)$ *com* σ, *caso essa transição não exista.*

→ *Faça* $Q \leftarrow f_V^{'}(w, \sigma_R)$, *caso esse estado não tenha sido visitado anteriormente.*

→ *Apague* σ_R *de* $\Gamma(w)$.

Passo 2.2.3) *Caso os passos 2.2.1 e 2.2.2 não tenham sido realizados, apague* σ *de* $\Gamma(u)$.

Passo 2.3) *Para cada* $y = f_V^{'}(u, \sigma_l)$, $\sigma_l \in \Gamma(u)$, *tal que* $y \notin Q$, *faça* $Q \leftarrow y$.

Passo 3) *Para cada estado* $x_V^{'}$ *do autômato resultante dos passos anteriores, verifique se* $\sigma_R \in \Gamma(x_V^{'})$. *Em caso positivo, apague* σ_R *de* $\Gamma(x_V^{'})$, *obtendo o autômato* $G_{V,mod}^{'}$.

Passo 4) *Faça* $G_V^{''} = Ac(G_{V,mod}^{'})$.

O exemplo a seguir ilustra o uso do algoritmo 5.1.

Exemplo 5.1. *Seja* $G_V^{'}$ *o verificador obtido no exemplo 4.3 considerando-se* $\Sigma_o^{'} = \{a, c\}$ *e mostrado novamente na figura 5.1. O algoritmo 5.1 será utilizado para se obter* $G_V^{''}$ *a partir do verificador* $G_V^{'}$, *sendo* $\Sigma_o^{''} = \Sigma_o^{'} \cup \{b\}$.

O algoritmo 5.1 é iniciado a partir do estado $(1N\ 1N)$ *do verificador com observação reduzida, que é incluído na fila* Q. *Uma vez que o evento* b *não está ativo*

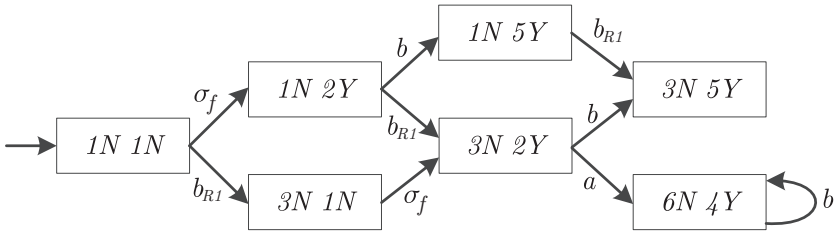


Figura 5.1: Verificador G'_V, considerando $\Sigma'_o = \{a, c\}$.

no mesmo, os estados $(1N\ 2Y)$ *e* $(3N\ 1N)$ *são inseridos em* Q*. Em seguida, note que o evento* b *está ativo no estado* $(1N\ 2Y)$*, e, portanto, deve-se verificar se* $(1N\ 2Y)$ *é alcançado por algum estado por meio de uma transição rotulada por* b_{R1} *ou se algum estado alcançado a partir de* $(1N\ 2Y)$ *com* b *possui o evento* b_{R1} *ativo. A partir da figura 5.1, pode-se ver que* $b_{R1} \in \Gamma(1N\ 5Y)$*. Então, deve-se apagar* b *de* $\Gamma(1N\ 2Y)$*, conectar o estado* $(1N\ 2Y)$ *a* $(3N\ 5Y)$ *com* b*, incluir* $(3N\ 5Y)$ *na fila* Q *e apagar* b_{R1} *de* $(1N\ 5Y)$*. Em seguida, no passo 2.3, o estado* $(3N\ 2Y)$ *é inserido na fila* Q*.*

Como o estado $(3N\ 1N)$ *não possui o evento* b *ativo e* $(3N\ 2Y)$ *já se encontra na fila, nada ocorre nessa etapa. O mesmo ocorre com* $(3N\ 5Y)$*, uma vez que o mesmo não possui qualquer transição para outro estado. Em seguida, o estado* $(3N\ 2Y)$ *deve ser analisado. Note que o mesmo possui* b *ativo e, dessa forma, deve-se investigar os estados que o alcançam e também os estados alcançados pelo mesmo com* b*. Como* $(1N\ 2Y)$ *alcança* $(3N\ 2Y)$ *por intermédio de* b_{R1}*, deve-se executar o passo 2.2.1. Assim, deve-se conectar o estado* $(1N\ 2Y)$ *a* $(3N\ 5Y)$ *com* b*. Entretanto, essa transição já foi criada anteriormente, não sendo necessário realizá-la novamente. Em seguida, o evento* b_{R1} *deve ser apagado de* $(1N\ 2Y)$*,* b *deve ser apagado de* $(3N\ 2Y)$ *e o estado* $(6N\ 4Y)$ *é inserido na fila* Q*.*

Por fim, note que o estado $(6N\ 4Y)$ *possui o evento* b *ativo, mas não existe qualquer estado que o alcance com* b_{R1}*. Uma vez que esse estado pode alcançar apenas ele mesmo e não possui o evento* b_{R1} *ativo, então o passo 2.3 é executado e o evento* b *é apagado do conjunto de eventos ativos de* $(6N\ 4Y)$*. A figura 5.2 resume os passos descritos até esse ponto. Note que a transição criada durante o algoritmo foi representada por uma seta tracejada.*

Para finalizar o uso do algoritmo 5.1, deve-se realizar os passos 3 e 4, que consistem em apagar as transições b_{R1} *remanescentes e tomar a parte acessível do verificador* $G'_{V,mod}$*, respectivamente. A figura 5.3 mostra o resultado dos passos 3 e 4 do algoritmo 5.1, enquanto a figura 5.4 apresenta o resultado final obtido.*

Observe que, conforme esperado, o verificador mostrado na figura 5.4 é idêntico ao mostrado na figura 4.3 (c). Note também que, uma vez assimilada a ideia do

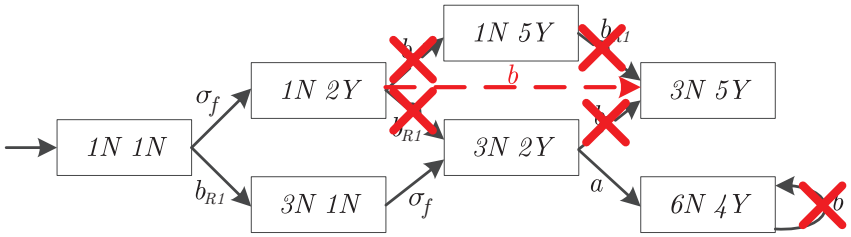


Figura 5.2: Verificador $G_V^{'}$ após aplicar os passos 1 e 2 do algoritmo 5.1.

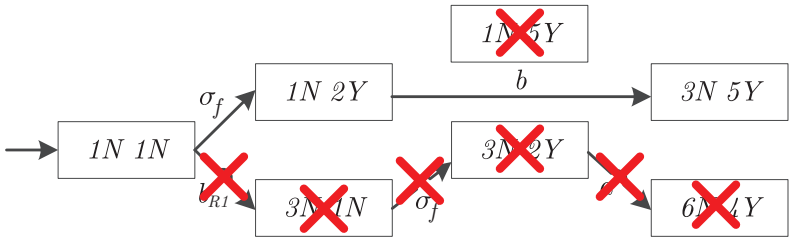


Figura 5.3: Verificador $G_V^{'}$ após aplicar o passo 3 do algoritmo 5.1.

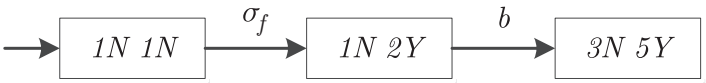


Figura 5.4: Verificador obtido após aplicar o algoritmo 5.1.

algoritmo 5.1, é possível perceber, a partir das figuras 5.1 e 5.4, que a observação do evento b desfaz o componente fortemente conexo formado pelo estado $(6N\ 4Y)$ que viola as condições impostas no teorema 4.3.

Observação 5.1. *(Complexidade computacional do algoritmo 5.1)*

Observe que o algoritmo 5.1 foi fundamentado no algoritmo de busca em largura, apresentado no capítulo 2, cuja complexidade é $O(V + E)$, sendo E o número de arcos, que equivale, no contexto de SEDs, ao número de transições e V o número de vértices, que equivale ao número de estados. Além de percorrer todos os estados do verificador, uma vez encontrado um estado tal que o evento que se deseja observar esteja ativo, será necessário acessar a lista de adjacências do autômato para verificar se algum estado anterior ao mesmo o alcança com o evento renomeado e também verificar se algum estado alcançado pelo mesmo com o evento que se deseja observar possui o evento renomeado ativo. Dessa forma, para cada estado, pode ser necessário acessar a lista de adjacências de $G_V^{'}$ por até duas vezes. Por fim, ao executar os passos 3 e 4, será necessário acessar a lista de adjacências para cada um dos estados do verificador resultante uma única vez. Assim, ao executar o algoritmo

5.1, a lista de adjacências será percorrida por, no máximo, três vezes, resultando em uma complexidade de $O(V + E)$, que equivale, em termos de estados e eventos do autômato G a $O(|X|^2|\Sigma|)$.

O algoritmo 5.1 apresentou uma nova forma de se obter um verificador a partir de outro já conhecido, caso o mesmo conjunto de eventos observáveis seja considerado a menos da inclusão de um evento σ. Apesar de computacionalmente o algoritmo 5.1 não apresentar vantagem se comparado ao algoritmo 2.3 para obtenção de um novo verificador, o mesmo introduz a ideia de como um verificador é afetado pela observação de um evento que antes era não observável. Essa ideia pode ser de grande utilidade para a utilização do algoritmo de obtenção das bases mínimas para o diagnóstico, que será apresentado posteriormente.

Seja $G_{V,\emptyset}$ o verificador obtido considerando o conjunto de eventos observáveis vazio. Observe que em $G_{V,\emptyset}$ serão mostradas todas as sequências que tornam a linguagem L não diagnosticável. Então, o teorema abaixo estabelece as condições necessárias para que a linguagem L seja diagnosticável com relação a $P_o^{'}$ e Σ_f.

Teorema 5.1. *Seja L diagnosticável com relação a $P_o : \Sigma^* \rightarrow \Sigma_o^*$ e Σ_f e $G_{V,\emptyset}$ o verificador obtido considerando o conjunto de eventos observáveis vazio. Seja N_v o número de sequências v^i associadas a caminhos cíclicos ambíguos em $G_{V,\emptyset}$ e Σ_v^i o conjunto formado pelos eventos que compõem v^i. Então, uma condição necessária para que a linguagem L seja diagnosticável com relação a $P_o^{'} : \Sigma^* \rightarrow \Sigma_o^{'*}$, em que $\Sigma_o^{'} \subset \Sigma_o$, e Σ_f é*

$$\forall i, i = 1, \ldots, N_v, \Sigma_o^{'} \cap \Sigma_v^i \neq \emptyset.$$

Prova. Suponha que exista uma sequência s_v associada a um caminho cíclico ambíguo em $G_{V,\emptyset}$ e um conjunto de eventos $\Sigma_o^{'}$ tal que $\Sigma_o^{'} \cap \Sigma_{s_v} = \emptyset$. Como os eventos de Σ_{s_v} são exclusivos do autômato que modela a parte de falha, então, ao obter o verificador $G_V^{'}$ considerando o conjunto de eventos observáveis $\Sigma_o^{'}$, existirá um caminho cíclico ambíguo em $G_V^{'}$ e, portanto, L será não diagnosticável com relação a $P_o^{'}$ e Σ_f. □

Observe que a condição dada no teorema 5.1 é apenas necessária, mas não suficiente. Dessa forma, é possível que, mesmo que a condição dada no teorema 5.1 seja atendida, L seja não diagnosticável com relação a $P_o^{'}$ e Σ_f. A condição necessária e suficiente para que a diagnosticabilidade da linguagem do sistema seja verificada foi dada nos teoremas 2.2 e 4.3.

Observe que, para satisfazer as condições dadas no teorema 5.1, bastaria calcular $G_{V,\emptyset}$, obter todas as suas sequências associadas a caminhos cíclicos ambíguos e verificar os eventos que as eliminam. Contudo, apesar de obter as bases mínimas para o diagnóstico de falhas, a complexidade computacional que seria obtida para esse

algoritmo seria idêntica à alcançada pelo algoritmo 4.3, apresentado no capítulo 4, uma vez que o número de caminhos cíclicos pode crescer pior que exponencialmente em um verificador. Note também que, apesar da possibilidade do número de caminhos cíclicos crescer pior que exponencialmente com relação ao número de estados do verificador, é provável que a observação de um único evento elimine mais de uma sequência associada a um caminho cíclico em $G_{V,\emptyset}$. Essa é a ideia utilizada no algoritmo 5.2, que será formalizado a seguir. No lugar de analisar todas as sequências que violam as condições de diagnosticabilidade, apenas uma será escolhida e analisada. Após verificar o que ocorre com a observação individual de cada um dos eventos que eliminam a sequência escolhida, outra sequência é escolhida e analisada, caso seja necessário. Dessa forma, evita-se a obtenção de todos os caminhos cíclicos e, assim, espera-se obter um algoritmo de menor complexidade que os demais existentes.

Algoritmo 5.2. *(Método da árvore de eventos)*

Seja G o autômato que modela o sistema, Σ_o seu conjunto de eventos observáveis e $\Sigma_f = \{\sigma_f\}$ o conjunto de eventos de falha.

Passo 1) *Faça $N = 0$, $\Sigma_{min} = \emptyset$ e $\Sigma_t = \emptyset$.*

Passo 2) *Para cada evento $\sigma_o \in \Sigma_o$, construa uma árvore seguindo os seguintes passos:*

Passo 2.1) *Crie a raiz da árvore e rotule-a com $\{\sigma_o\}$.*

Passo 2.2) *ANALISE($\{\sigma_o\}$) (algoritmo 5.3).*

Passo 2.3) $\Sigma_t \leftarrow \Sigma_t \cup \{\sigma_o\}$.

Passo 3) *Se $N = 0$, faça $\Sigma_{min} = \Sigma_o$. Caso contrário, forme o conjunto $\Sigma_{min} = \{\Sigma^1_{min}\} \cup \{\Sigma^2_{min}\} \cup \ldots \cup \{\Sigma^N_{min}\}$.*

Passo 4) *Remova de Σ_{min} todos os conjuntos $\widetilde{\Sigma}_{min} \in \Sigma_{min}$ caso exista $\widehat{\Sigma}_{min} \in \Sigma_{min}$ tais que $\widetilde{\Sigma}_{min} \supseteq \widehat{\Sigma}_{min}$.*

Algoritmo 5.3. *ANALISE($\widehat{\Sigma}$)*

Passo 1) $\widetilde{\Sigma}_o \leftarrow \widehat{\Sigma}$.

Passo 2) *Obtenha o verificador $G_{V,\widetilde{\Sigma}_o}$, em que $G_{V,\widetilde{\Sigma}_o}$ denota o verificador obtido considerando $\widetilde{\Sigma}_o$ como conjunto de eventos observáveis.*

Passo 3) *Verifique se existe um caminho cíclico em $G_{V,\widetilde{\Sigma}_o}$ que viole as condições estabelecidas no teorema 2.2. Em caso negativo, faça:*

Passo 3.1) $N = N + 1$.

Passo 3.2) *Forme o conjunto* $\Sigma_{min}^{N} = \widetilde{\Sigma}_o$.

Em caso positivo, faça:

Passo 3.1) *Escolha uma sequência v que viola as condições de diagnosticabilidade da linguagem do sistema.*

Passo 3.2) *Utilize o algoritmo 4.2 para obter as sequências normal e de falha,* s_N *e* s_F*, respectivamente, associadas à sequência v escolhida no passo anterior.*

Passo 3.3) *Para cada evento* $\sigma_o^i \in \Sigma_o \setminus \widetilde{\Sigma}_o$*, tal que* $\widetilde{P}_o(s_N) \neq \widetilde{P}_o(s_F)$*, em que* $\widetilde{P}_o : \Sigma^* \rightarrow (\widetilde{\Sigma}_o \cup \{\sigma_o^i\})^*$*, faça:*

Passo 3.3.1) *Verifique se o verificador* $G_{V,\widetilde{\Sigma}_o \cup \{\sigma_o^i\}}$ *já foi analisado anteriormente.*

Passo 3.3.2) *Se* $N > 0$*, verifique se, para algum* $j \in \{1, \ldots, N\}$*,* $\widetilde{\Sigma}_o \cup \{\sigma_o^i\} \supset \Sigma_{min}^{j}$.

Passo 3.3.3) *Verifique se* $(\widetilde{\Sigma}_o \cup \{\sigma_o^i\}) \cap \Sigma_t \neq \emptyset$.

Passo 3.3.4) *Verifique se* $\widetilde{\Sigma}_o \cup \{\sigma_o^i\} = \Sigma_o$.

Caso os testes dos passos 3.3.1 a 3.3.4 sejam negativos, faça:

Passo 3.3.5) *Crie um descendente do nó rotulado por* $\widetilde{\Sigma}_o$ *na árvore de eventos.*

Passo 3.3.6) *Rotule o nó criado no passo 3.3.1 com* $(\widetilde{\Sigma}_o \cup \{\sigma_o^i\})$.

Passo 3.4) *Para cada nó criado no passo anterior, faça ANALISE*$(\widetilde{\Sigma}_o \cup \{\sigma_o^i\})$.

Observe que o algoritmo 5.2 possui uma filosofia completamente distinta dos demais algoritmos existentes para obtenção de bases mínimas de um SED, considerando, inclusive, o algoritmo 4.3 desenvolvido nesta dissertação. A principal característica que distingue o algoritmo 5.2 é que em nenhum de seus passos, ao contrário dos demais existentes na literatura, existe uma busca por todos os ciclos que violam as condições de diagnosticabilidade da linguagem do sistema. Isso representa uma grande vantagem, uma vez que o número de ciclos elementares pode apresentar um crescimento pior que exponencial com relação à cardinalidade do conjunto de estados do autômato no qual os ciclos estão sendo procurados ([33],[35]).

A seguir, será realizada a prova da veracidade do algoritmo 5.2 ao retornar as bases mínimas de um determinado sistema.

Teorema 5.2. *Seja G o autômato que modela o sistema. O conjunto* Σ_{min} *obtido a partir do algoritmo 5.2 é formado por todas as bases mínimas para o diagnóstico de falhas da linguagem gerada por G.*

Prova. A prova da veracidade do algoritmo 5.2 será realizada em três etapas: na primeira etapa, é provado que os conjuntos pertencentes a Σ_{min} são bases para diagnóstico do sistema. Na segunda etapa, é provado que, se existe uma base mínima, então ela rotula pelo menos uma das folhas de uma das árvores de eventos criadas no algoritmo 5.2. Finalmente, na terceira parte, é mostrado que todos os conjuntos de Σ_{min}, obtidos por intermédio do algoritmo 5.2, são bases mínimas.

Parte 1 - Os conjuntos soluções obtidos são bases para diagnóstico de falha?

Note que o algoritmo 5.2 é baseado na construção de verificadores para o diagnóstico de falhas e que um conjunto de eventos $\widetilde{\Sigma}_o$ é adicionado a Σ_{min} somente quando o verificador $G_{V,\widetilde{\Sigma}_o}$ não possui caminhos cíclicos ambíguos. Portanto, todos os conjuntos que pertencem a Σ_{min} são bases para o diagnóstico de falhas da linguagem de G.

Parte 2 - Se uma base é mínima, então ela rotula pelo menos uma das folhas de uma das árvores.

Seja Σ_o o conjunto de eventos observáveis de G. Então, de acordo com o algoritmo 5.2, $|\Sigma_o|$ árvores serão construídas, cada uma iniciando por um evento distinto de Σ_o.

Seja $\Sigma^j_{min} = \{\sigma_1, \sigma_2, \ldots, \sigma_k\} \subset \Sigma_o$ uma base mínima para o diagnóstico da linguagem de G. Considere, sem perda de generalidade, a árvore cuja raiz é σ_1. Como Σ^j_{min} é uma base mínima, então existe pelo menos uma sequência $v \in \mathcal{L}(G_{V,\{\sigma_1\}})$ que viola a diagnosticabilidade da linguagem de G.

De acordo com o passo 4.1 do algoritmo 5.3, uma sequência v associada a um caminho cíclico ambíguo em $G_{V,\{\sigma_1\}}$ é escolhida para obter os descendentes de σ_1 em sua respectiva árvore. Note que, como Σ^j_{min} é uma base mínima, então a observação dos eventos $\{\sigma_1\sigma_2 \ldots \sigma_k\}$ elimina todos os caminhos cíclicos ambíguos existentes em $G_{V,\{\sigma_1\}}$, inclusive aquele associado a v e, portanto, pelo menos um evento de $\Sigma^j_{min} \setminus \{\sigma_1\}$, junto com σ_1, é um descendente de σ_1 na árvore.

Seguindo o mesmo raciocínio, pode-se notar que pelo menos uma folha da árvore cuja raiz é σ_1 será rotulada pelo eventos pertencentes a Σ^j_{min}.

Parte 3 - Todas as bases de Σ_{min} são mínimas.

Observe que, no passo 4 do algoritmo 5.2, todos os ramos criados com eventos não essenciais são apagados, garantindo, assim, que apenas as bases mínimas se mantenham em Σ_{min}.

□

Uma vez que o algoritmo 5.2 é executado até que não haja mais sequências que violam as condições do teorema 2.2, pode-se concluir que o algoritmo 5.2 satisfaz as condições impostas no teorema 5.1. Note ainda que, caso uma árvore cuja raiz é rotulada por $\{\sigma_1\}$ tenha sido concluída, ao iniciar a construção de outra árvore com raiz $\{\sigma_2\}$, não será necessário analisar nós rotulados por conjuntos de eventos que contenham σ_1. Isso ocorre pois todas as bases mínimas nas quais o evento σ_1 faz parte já foram obtidas na construção na árvore com raiz $\{\sigma_1\}$. Os exemplos a seguir ilustram o uso do algoritmo 5.2 desenvolvido acima.

Exemplo 5.2. *Considere um sistema modelado pelo autômato G utilizado nos exemplos 4.2 e 4.3 e mostrado na figura 4.2. Seja $\Sigma_o = \{a, b, c\}$ o conjuntos de eventos observáveis de G e $\Sigma_f = \{\sigma_f\}$ o conjunto de eventos de falha. Será aplicado o algoritmo 5.2 para obter todas as bases mínimas que mantenham a linguagem do sistema diagnosticável.*

O algoritmo 5.2 prevê a construção de uma árvore de eventos para cada um dos eventos $\sigma_o \in \Sigma_o$. Será escolhido, inicialmente, o evento a como raiz da primeira árvore de eventos. A função ANALISE será iniciada a partir da raiz rotulada por $\{a\}$. A figura 5.5 mostra o verificador $G_{V,\{a\}}$ calculado considerando o conjunto $\{a\}$ como conjunto de eventos observáveis.

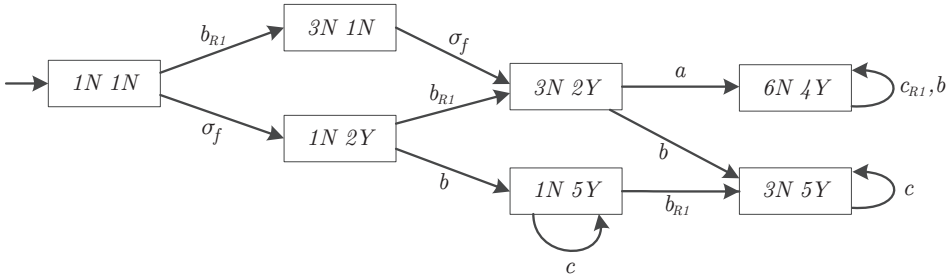


Figura 5.5: Verificador $G_{V,\{a\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a\}$.

Observando a figura 5.5, pode-se concluir que a linguagem do sistema é não diagnosticável considerando a observação apenas do evento a. Dessa forma, uma única sequência v que viola o teorema 2.2 deve ser selecionada e analisada. Seja $v = \sigma_f b b_{R1} c$ a sequência escolhida, obtenha os traços normal $s_N = b$ e de falha $s_F = \sigma_f bc$ utilizando-se o algoritmo 4.2. Assim, de forma a tornar $\widetilde{P}_o(s_N) \neq \widetilde{P}_o(s_F)$, em que $\widetilde{P}_o : \Sigma^* \to \widetilde{\Sigma}_o^*$ e $\widetilde{\Sigma}_o = \{a\}$, o evento c deve ser observado e, portanto, será adicionado à árvore de eventos como descendente de $\{a\}$ e a função ANALISE($\{a, c\}$) deve ser executada. Em seguida, o verificador $G_{V,\{a,c\}}$ deve ser calculado utiizando o algoritmo 2.3. A figura 5.6 mostra o verificador $G_{V,\{a,c\}}$ obtido ao se considerar a observação dos eventos a e c simultaneamente.

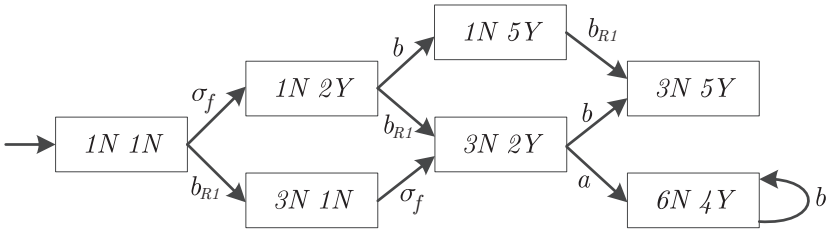


Figura 5.6: Verificador $G_{V,\{a,c\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, c\}$.

*Note que o verificador $G_{V,\{a,c\}}$, mostrado na figura 5.6, ainda possui ciclos que violam as condições de diagnosticabilidade apresentadas no teorema 2.2. Assim, será escolhida a sequência $v = \sigma_f b_{R1} ab$, que corresponde aos traços $s_N = ba$ e $s_F = \sigma_f ab$. Então, para que $\widetilde{P}_o(s_N) \neq \widetilde{P}_o(s_F)$, o evento b deve ser observado. Contudo, como $\{a, b, c\} = \Sigma_o$, nenhum novo nó será criado na árvore cuja raiz é $\{a\}$. Isso ocorre pois, de acordo com a premissa **H4**, o sistema original é diagnosticável com relação a P_o e Σ_f, não sendo necessário realizar testes com relação à diagnosticabilidade da linguagem do sistema sob observação de $\widetilde{\Sigma}_o = \Sigma_o$ nesse caso. Dessa forma, a figura 5.7 mostra a árvore de eventos construída a partir de a utilizando o algoritmo 5.2.*

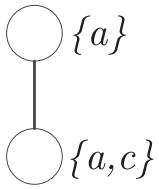


Figura 5.7: Árvore de eventos criada a partir de a conforme regras definidas no algoritmo 5.2.

A partir da figura 5.7, é possível concluir que, uma vez que o evento a é observado, apenas a observação simultânea de b e c mantêm a propriedade de diagnosticabilidade da linguagem do sistema inalterada. Os mesmos passos realizados para a construção da árvore de eventos a partir de a serão, agora, repetidos considerando, separadamente, os eventos b e c como raízes das novas árvores.

Seja $G_{V,\{b\}}$ o verificador obtido com o algoritmo 2.3 considerando apenas a observação do evento b, mostrado na figura 5.8.

Observe que $v = \sigma_f bc$ é uma sequência que viola as condições do teorema 2.2 é será o traço escolhido para ser analisado. É possível obter, a partir do algoritmo 4.2, os traços $s_N = b$ e $s_F = \sigma_f bc$. Assim, de forma que a projeção de s_N seja distinta da projeção de s_F considerando a observação de $\widetilde{\Sigma}_o$, o evento c deve ser observado e será, portanto, adicionado à árvore de eventos como descendente da raiz $\{b\}$. Em

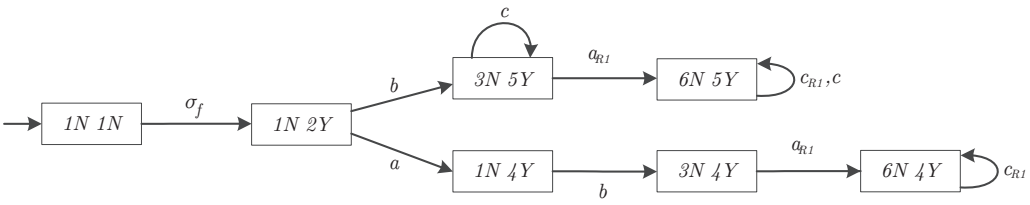


Figura 5.8: Verificador $G_{V,\{b\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{b\}$.

seguida a função ANALISE$(\{b, c\})$ *é chamada.*

Uma vez que o verificador $G_{V,\{b,c\}}$ *ainda não foi analisado anteriormente, o mesmo deve ser obtido a partir do algoritmo 2.3.* $G_{V,\{b,c\}}$ *é mostrado na figura 5.9. Note que existe em* $G_{V,\{b,c\}}$ *uma única sequência que viola a definição de diagnosticabilidade utilizada nesta disssertação,* $v = \sigma_f b a_{R1} c$*, que corresponde aos traços* $s_N = bac$ *e* $s_F = \sigma_f bc$*. Então, o evento a deve ser observado e, novamente, como* $\{a, b, c\} = \Sigma_o$*, não é necessário testar a diagnosticabilidade da linguagem do sistema sob observação de* $\widetilde{\Sigma}_o$*, nenhum nó é criado na árvore cuja raiz é rotulada por* $\{b\}$ *e sua construção chega ao fim.*

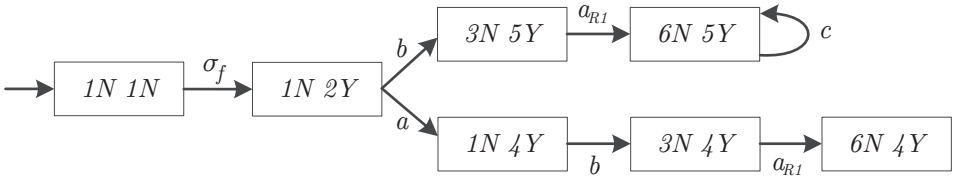


Figura 5.9: Verificador $G_{V,\{b,c\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{b, c\}$.

A árvore de eventos construída a partir de b é apresentada na figura 5.10.

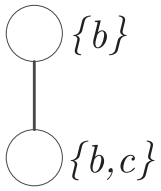


Figura 5.10: Árvore de eventos criada a partir de b conforme regras definidas no algoritmo 5.2.

Para finalizar o algoritmo 5.2, deve ser construída a árvore de eventos a partir de c. A figura 5.11 mostra o verificador $G_{V,\{c\}}$*.*

Escolhendo-se a sequência $v = \sigma_f ab$*, os traços* $s_N = \varepsilon$ *e* $s_F = \sigma_f ab$ *podem ser recuperados utilizando o algoritmo 4.2. É evidente que tanto a observação do*

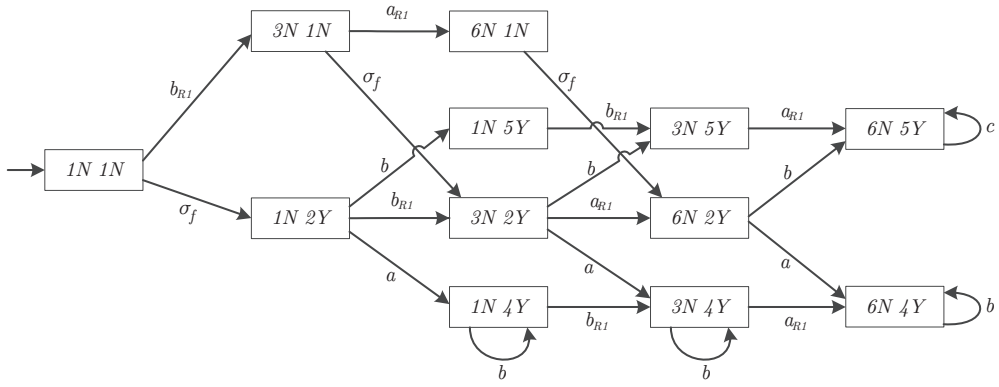


Figura 5.11: Verificador $G_{V,\{c\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{c\}$.

evento a quanto do evento b tornam a projeção de s_N diferente da projeção de s_F. Contudo, de acordo com os testes contidos no passo 3.3 da função ANALISE, como os conjuntos $\{a, c\}$ e $\{b, c\}$ já tiveram seus respectivos verificadores $G_{V,\{a,c\}}$ e $G_{V,\{b,c\}}$ construídos, conforme mostrado na figura 5.6 e na figura 5.9, e as árvores cuja raiz é $\{a\}$ e $\{b\}$ já foram construídas, não será necessário realizar novamente essa análise para esses conjuntos. A figura 5.12 mostra a árvore de eventos construída a partir de c.

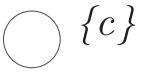


Figura 5.12: Árvore de eventos criada a partir de c conforme regras definidas no algoritmo 5.2.

Como todas as árvores de eventos já foram construídas, deve-se avaliar as soluções Σ^i_{min} obtidas. Uma vez que nenhum conjunto foi adicionado a Σ_{min}, então $N = 0$. Assim, de acordo com o passo 3 do algoritmo 5.2, é possível afirmar que, $\Sigma_{min} = \{a, b, c\}$.

Conforme esperado, o resultado obtido no exemplo 5.2 utilizando-se o algoritmo 5.2 foi idêntico aos dos exemplos 3.1 e 4.3. Observe que, como o conjunto de menor cardinalidade que mantém a linguagem do sistema diagnosticável é igual a Σ_o, o número de verificadores necessários ao executar o algoritmo 5.2 tende a se aproximar do número de verificadores necessários ao utilizar a força bruta. Ainda assim, note que no exemplo 5.2 não foi necessária a construção do verificador $G_{V,\{a,b\}}$ para concluir que $\Sigma_{min} = \Sigma_o$. A seguir, será dado mais um exemplo para ilustrar o uso do algoritmo 5.2.

Exemplo 5.3. *Considere o sistema modelado pelo autômato G mostrado na figura 4.9. Suponha* $\Sigma_o = \{a, b, c, d, e\}$ *e* $\Sigma_f = \{\sigma_f\}$*. Será aplicado então o algoritmo 5.2 desenvolvido neste capítulo.*

Serão construídas cinco árvores de eventos, cada uma iniciando por um evento de Σ_o*. Primeiramente, a árvore a partir do evento a será analisada. Para tal, será utilizada a função ANALISE*$(\{a\})$*, cujo passo 3 prevê a construção do verificador* $G_{V,\{a\}}$*, mostrado na figura 5.13.*

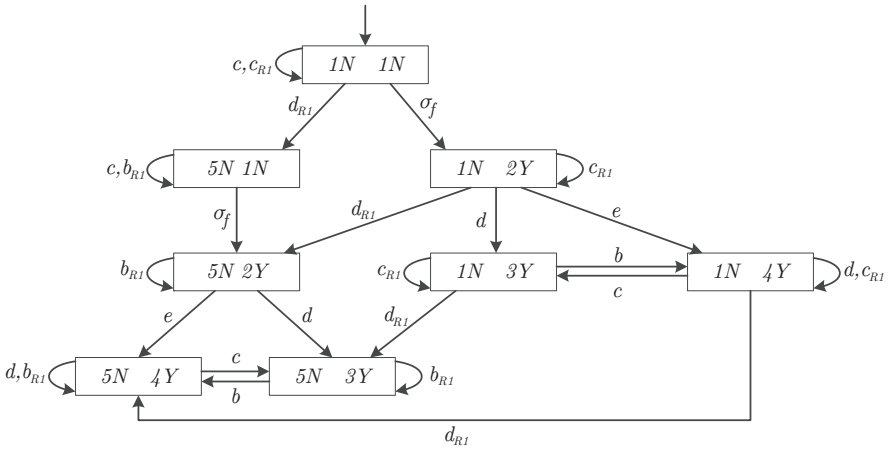


Figura 5.13: Verificador $G_{V,\{a\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a\}$.

A partir da figura 5.13, pode-se constatar que, ao observar somente o evento a, existem diversos ciclos que violam as condições dadas no teorema 2.2. Note que, ao escolher a sequência $v = \sigma_f d_{R1} dbc$*, que corresponde aos traços* $s_N = d$ *e* $s_F = \sigma_f dbc$*, os eventos b e c serão inseridos na árvore de eventos como descendentes de* $\{a\}$*, conforme passos 3.3.5 e 3.3.6 do algoritmo 5.3.*

Considerando o nó $\{a, b\}$ *da árvore e eventos cuja raiz é* $\{a\}$*, deve-se obter o verificador* $G_{V,\{a,b\}}$*, cujo diagrama é mostrado na figura 5.14.*

Como ainda existem sequências que violam a definição de diagnosticabilidade em $G_{V,\{a,b\}}$*, deve ser escolhido outro traço e, a partir do mesmo, mais eventos devem ser inseridos na árvore. Escolhendo* $v = \sigma_f ed$ *(*$s_N = \varepsilon$ *e* $s_F = \sigma_f ed$*), conclui-se que os eventos d e e deverão ser inseridos na árvore como descendentes do nó rotulado por* $\{a, b\}$ *e, para cada um dos nós criados, será executada a função ANALISE.*

Seja $G_{V,\{a,b,d\}}$ *o verificador calculado considerando* $\widetilde{\Sigma}_o = \{a, b, d\}$*, mostrado na figura 5.15. Uma vez que as condições no teorema 2.2 não foram atendidas, deve-se escolher uma sequência v para dar continuidade ao processo de obtenção das bases mínimas para o dioagnóstico.*

Observe, a partir da figura 5.15 que, utilizando $v = \sigma_f dbc$ *, que equivale aos traços* $s_N = db$ *e* $s_F = \sigma_f dbc$*, pode-se deduzir que o evento c deve ser incluído na*

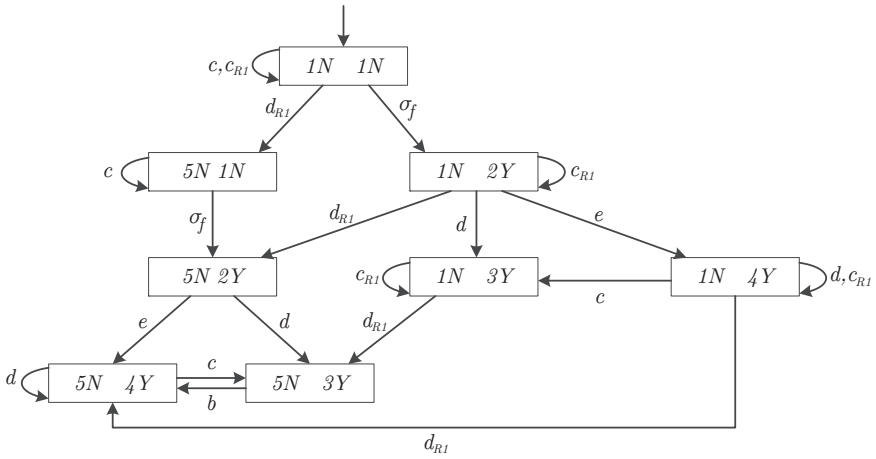


Figura 5.14: Verificador $G_{V,\{a,b\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, b\}$.

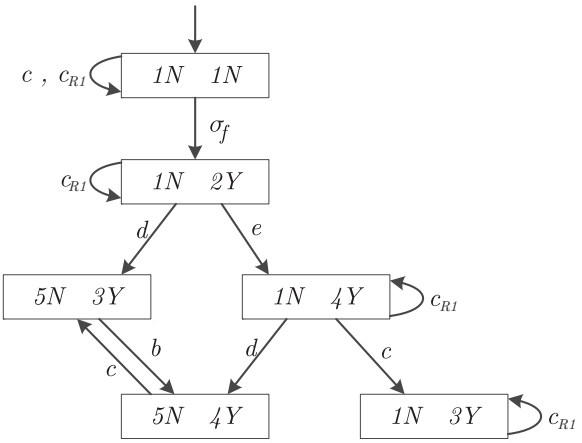


Figura 5.15: Verificador $G_{V,\{a,b,d\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, b, d\}$.

árvore de eventos como descendente de $\{a, b, d\}$*. A figura 5.16 apresenta o verificador obtido para* $\widetilde{\Sigma}_o = \{a, b, c, d\}$.

Uma vez que não existem ciclos $G_{V,\{a,b,c,d\}}$ *que violem às condições de diagnosticabilidade estabelecidas nesta dissertação, pode-se afirmar que* $\{a, b, c, d\}$ *é um conjunto solução. Assim,* $\Sigma^1_{min} = \{a, b, c, d\}$.

Em seguida, assim como ocorre no algoritmo de busca em profundidade, o algoritmo retrocede aos nós da árvore cuja raiz é $\{a\}$ *que ainda não foram explorados. O primeiro deles é o nó* $\{a, b, e\}$*. Considere* $G_{V,\{a,b,e\}}$ *o verificador construído considerando o conjunto de eventos observáveis* $\widetilde{\Sigma}_o = \{a, b, e\}$*, cujo grafo é mostrado na figura 5.17.*

Seja $v = \sigma_f dd_{R1}bc$ *a sequência escolhida em* $G_{V,\{a,b,e\}}$*, então o evento c deve ser acrescentado à árvore como descendente de* $\{a, b, e\}$ *de forma que as projeções dos*

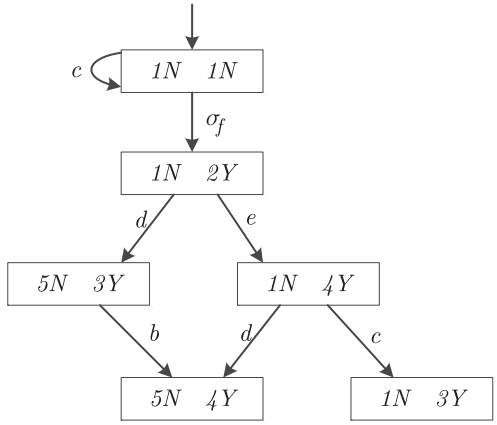


Figura 5.16: Verificador $G_{V,\{a,b,c,d\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, b, c, d\}$.

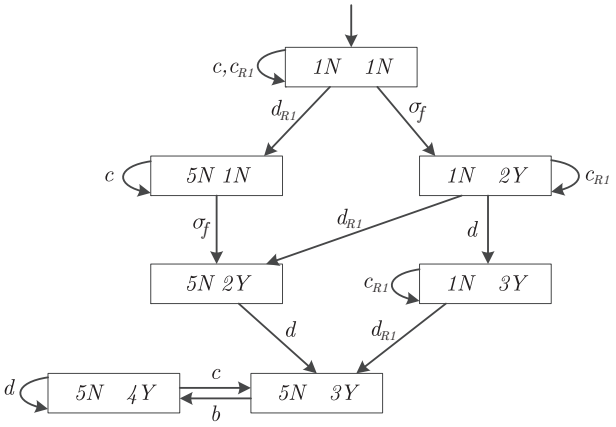


Figura 5.17: Verificador $G_{V,\{a,b,e\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, b, e\}$.

traços $s_N = db$ e $s_F = \sigma_f dbc$ sejam distintas. A figura 5.18 mostra o verificador $G_{V,\{a,b,c,e\}}$ obtido.

Como o verificador $G_{V,\{a,b,c,e\}}$ aponta que a linguagem do sistema é não diagnosticável para seu respectivo conjunto de eventos observáveis $\widetilde{\Sigma}_o = \{a, b, c, e\}$, a sequência $v = \sigma_f dd_{R1}bd$ foi selecionada para avaliação. De forma que $s_N = db$ e $s_F = \sigma_f dbd$ não possuam a mesma projeção considerando $\widetilde{\Sigma}_o$, deve-se considerar a observação de d, conforme já esperado, uma vez que o sistema sob observação de Σ_o é diagnosticável. Então, como $\{a, b, c, d, e\} = \Sigma_o$, não será necessária a criação do nó rotulado por $\{a, b, c, d, e\}$.

Para concluir o estudo da árvore a partir de a, considere o único nó não explorado, rotulado por $\{a, c\}$, cujo verificador $G_{V,\{a,c\}}$ é mostrado na figura 5.19.

Note que a sequência $v = \sigma_f dbc$ em $G_{V,\{a,c\}}$ viola as condições de diagnosticabilidade da linguagem do sistema. Assim, para que as projeções das sequências $s_N = c$ e $s_F = \sigma_f dbc$ sejam distintas, deve-se acrescentar b e d a $\widetilde{\Sigma}_o$.

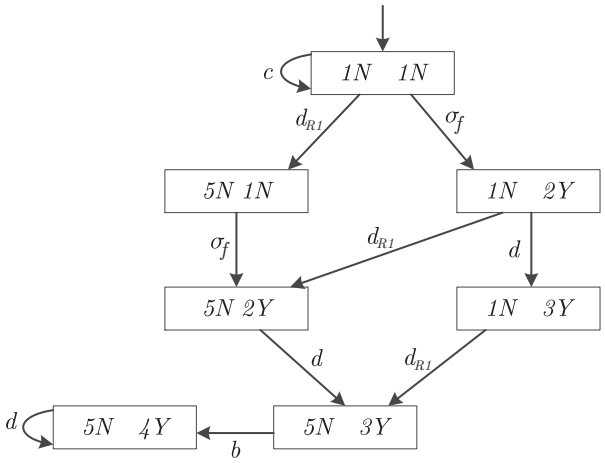


Figura 5.18: Verificador $G_{V,\{a,b,c,e\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, b, c, e\}$.

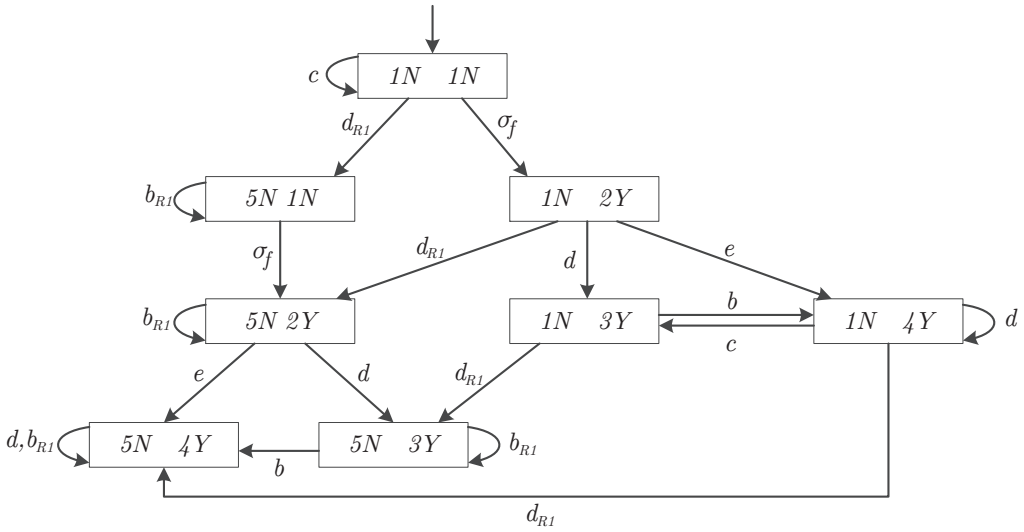


Figura 5.19: Verificador $G_{V,\{a,c\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, c\}$.

As figuras 5.20 e 5.21 mostram, respectivamente, os verificadores $G_{V,\{a,b,c\}}$ *e* $G_{V,\{a,c,d\}}$.

Na figura 5.20, considere a sequência $v = \sigma_f ed$ *em* $G_{V,\{a,b,c\}}$. *Para tornar as projeções de* $s_N = \varepsilon$ *e* $s_F = \sigma_f ed$ *diferentes, basta observar o evento* e *ou o evento* d. *Uma vez que tanto o conjunto* $\{a, b, c, d\}$ *quanto* $\{a, b, c, e\}$ *já foram testados anteriormente, suas soluções já são conhecidas. Então, de acordo com os testes contidos no passo 3.3 da função ANALISE, esses nós não serão criados na árvore cuja raiz é* $\{a\}$. *Note que a figura 5.16 mostrou que o conjunto* $\{a, b, c, d\}$ *é solução, enquanto a figura 5.18 apontou a necessidade de se observar o evento* d.

A partir da figura 5.21, deve-se concluir que a linguagem do sistema é ainda não

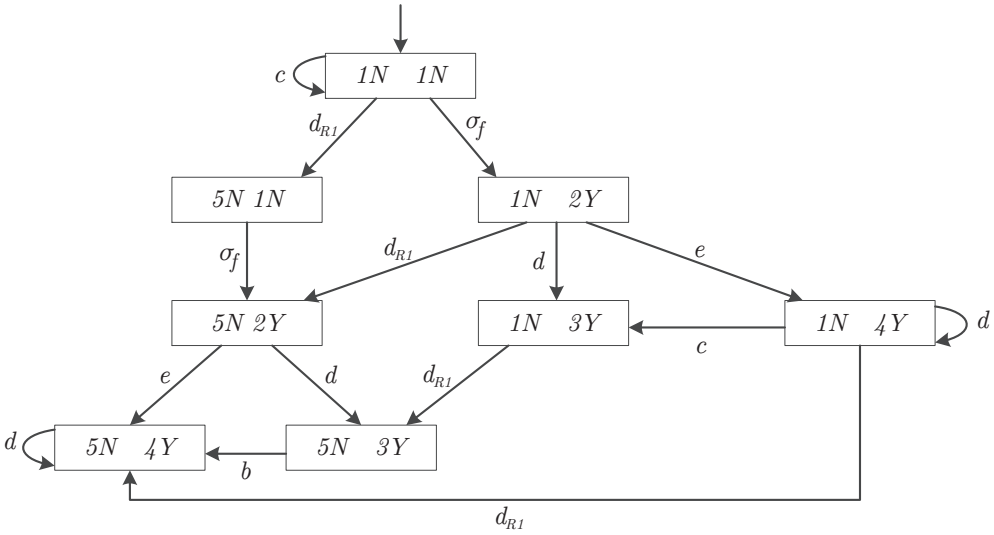


Figura 5.20: Verificador $G_{V,\{a,b,c\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, b, c\}$.

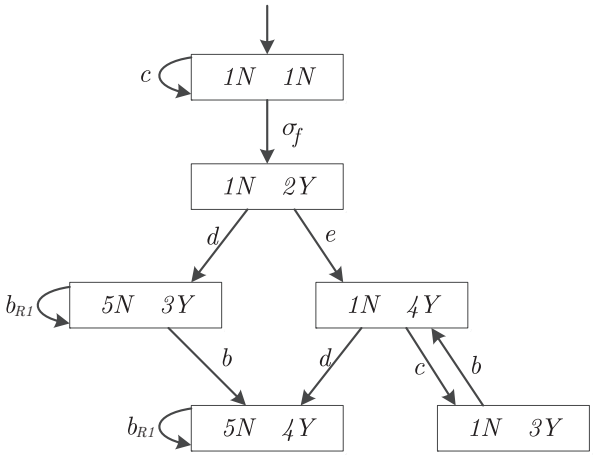


Figura 5.21: Verificador $G_{V,\{a,c,d\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, c, d\}$.

diagnosticável, havendo a necessidade de se escolher uma sequência para análise. Considere $v = \sigma_f ecb$*, que corresponde a* $s_N = c$ *e* $s_F = \sigma_f ecb$*. Assim, para que as projeções das sequências normal e de falha sejam distintas, deve-se observar o evento* b *ou o evento* e*. Uma vez que o conjunto* $\{a, b, c, d\}$ *é solução, basta averiguar o conjunto* $\{a, c, d, e\}$*, cujo verificador é apresentado na figura 5.22.*

Utilizando a figura 5.22, é possível verificar que L *é diagnosticável considerando o conjunto* $\{a, c, d, e\}$*. Então,* $\Sigma^2_{min} = \{a, c, d, e\}$*. Dessa forma, a árvore de eventos a partir de* a *foi concluída, e seu resultado é mostrado na figura 5.23.*

Durante a construção da árvore cuja raiz é $\{a\}$*, as seguintes soluções foram obtidas:* $\Sigma^1_{min} = \{a, b, c, d\}$*,* $\Sigma^2_{min} = \{a, c, d, e\}$*.*

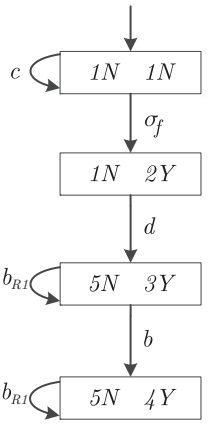


Figura 5.22: Verificador $G_{V,\{a,c,d,e\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{a, c, d, e\}$.

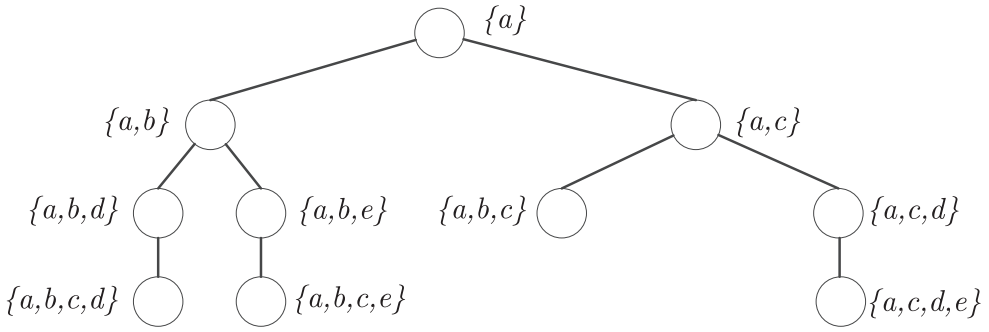


Figura 5.23: Árvore de eventos criada a partir de a conforme regras definidas no algoritmo 5.2.

Uma vez finalizada a construção da árvore de eventos a partir de a, será iniciada a análise considerando-se $\{b\}$ *como raiz da árvore de eventos.*

Considere o autômato $G_{V,\{b\}}$*, mostrado na figura 5.24. Note que a observação apenas do evento b não é suficiente para que a linguagem do sistema mantenha sua diagnosticabilidade. Então, deverá ser selecionada uma sequência que viola o teorema 2.2 para análise. Seja* $v = \sigma_f d_{R1} dab$ *e utilizando-se o algoritmo 4.2, pode-se recuperar os respectivos traços normal* $s_N = db$ *e de falha* $s_F = \sigma_f dab$*. Assim, somente com a observação do evento a as projeções de* s_N *e* s_F *serão distintas.*

Note que, ao executar a função ANALISE$(\{b\})$ *e escolhendo-se a sequência* $v = \sigma_f d_{R1} dab$*, nenhum nó será criado, uma vez que o autômato* $G_{V,\{a,b\}}$ *já foi construído e analisado durante o processo de construção da árvore de eventos com raiz a, conforme figura 5.14. Além disso, como a árvore com raiz* $\{a\}$ *já foi finalizada, todas as bases mínimas nas quais o evento a faz parte já foram obtidas. Dessa forma, não há necessidade de realizar novamente essa análise, pois as mesmas*

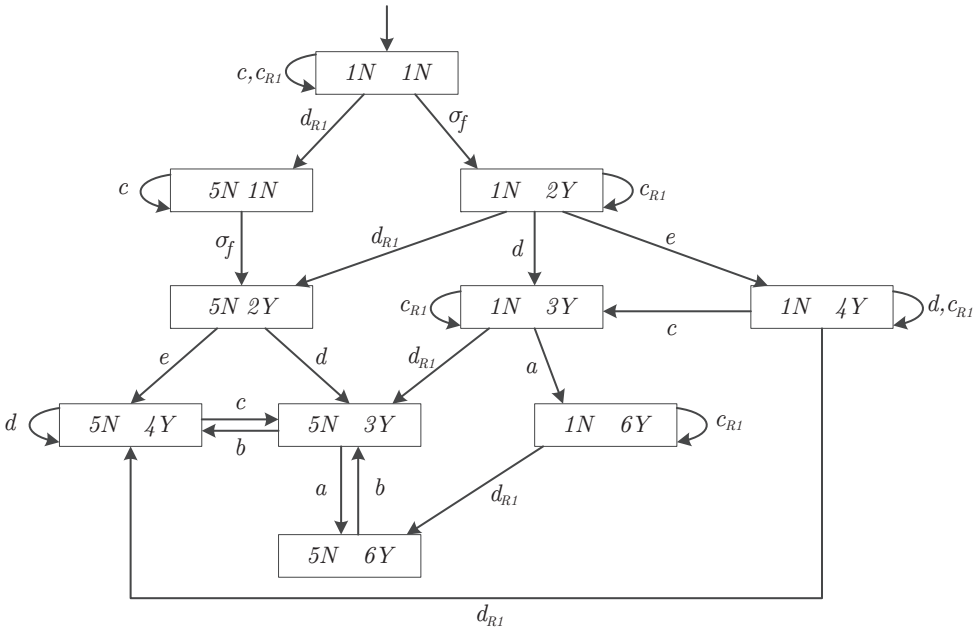


Figura 5.24: Verificador $G_{V,\{b\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{b\}$.

soluções seriam alcançadas. Com essa consideração, pode-se construir a árvore de eventos a partir do evento b, mostrada na figura 5.25.

Figura 5.25: Árvore de eventos criada a partir de b conforme regras definidas no algoritmo 5.2.

Em seguida, deve ser construída a árvore de eventos cuja raiz é rotulada por $\{c\}$*. A figura 5.26 apresenta o verificador* $G_{V,\{c\}}$*, cuja premissa é observar apenas o evento c dentre os eventos de* Σ_o*.*

Considere a sequência $v = \sigma_f dd_{R1}ab$ *em* $G_{V,\{c\}}$ *que viola as condições de diagnosticabilidade da linguagem gerada por G. Utilizando o algoritmo 4.2, é possível recuperar os traços* $s_N = d$ *e* $s_F = \sigma_f dab$*. Assim, para que suas projeções sejam distintas, será necessário observar o evento a ou b. Uma vez que as árvores cujas raízes são* $\{a\}$ *e* $\{b\}$ *já foram construídas, todas as bases mínimas nas quais esses eventos pertencem já foram obtidas e, portanto, nenhum novo nó será criado na árvore cuja raiz é* $\{c\}$*. A figura 5.27, dada a seguir, mostra a árvore de eventos construída a partir da raiz* $\{c\}$*.*

Observe que nenhuma nova solução foi obtida na árvore mostrada na figura 5.27.

Será iniciada, então, a construção da árvore de eventos com raiz $\{d\}$*. Consi-*

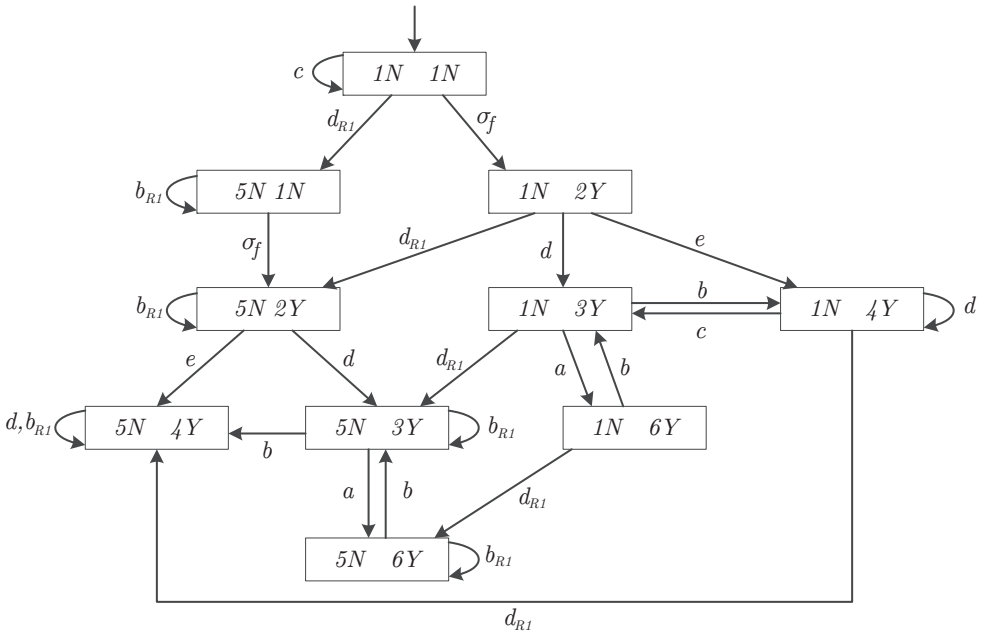


Figura 5.26: Verificador $G_{V,\{c\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{c\}$.

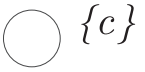


Figura 5.27: Árvore de eventos criada a partir de c conforme regras definidas no algoritmo 5.2.

dere $G_{V,\{d\}}$ o verificador obtido considerando apenas que o evento d é observável, mostrado na figura 5.28.

Seja $v = \sigma_f dbc$ a sequência de $G_{V,\{d\}}$ selecionada e $s_N = d$ e $s_F = \sigma_f dbc$ os traços obtidos utilizando o algoritmo 4.2. Então, de forma a tornar as projeções das sequências normal e de falha distintas, deve-se observar o evento b ou c. Uma vez que as árvores cujas raízes são rotuladas por $\{b\}$ e $\{c\}$ já foram desenvolvidas, os nós $\{b, d\}$ e $\{c, d\}$ não serão criados como descendentes de $\{d\}$, conforme testes descritos no passo 3.3 do algoritmo 5.3. A figura 5.29 mostra a árvore com raiz rotulada por $\{d\}$ após realizados os passos para sua construção.

Por fim, será iniciada a construção da árvore de eventos a partir do evento e.

A figura 5.30 mostra o verificador $G_{V,\{e\}}$, cuja construção é realizada considerando-se apenas a observação do evento e.

Considere a sequência $v = d_{R1}\sigma_f dbc$ que viola a condição proposta no teorema 2.2. Então, pode-se obter, por intermédio do algoritmo 4.2, os traços $s_N = d$ e $s_F = \sigma_f dbc$. Portanto, para que as projeções de s_N e s_F sejam distintas, é necessária a observação dos eventos b ou c. Assim como realizado na construção da árvore cuja

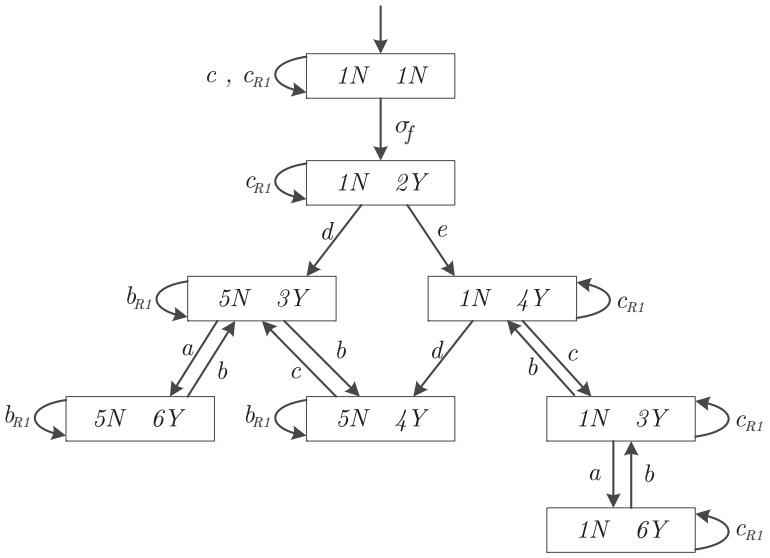


Figura 5.28: Verificador $G_{V,\{d\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{d\}$.

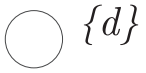


Figura 5.29: Árvore de eventos criada a partir de d conforme regras definidas no algoritmo 5.2.

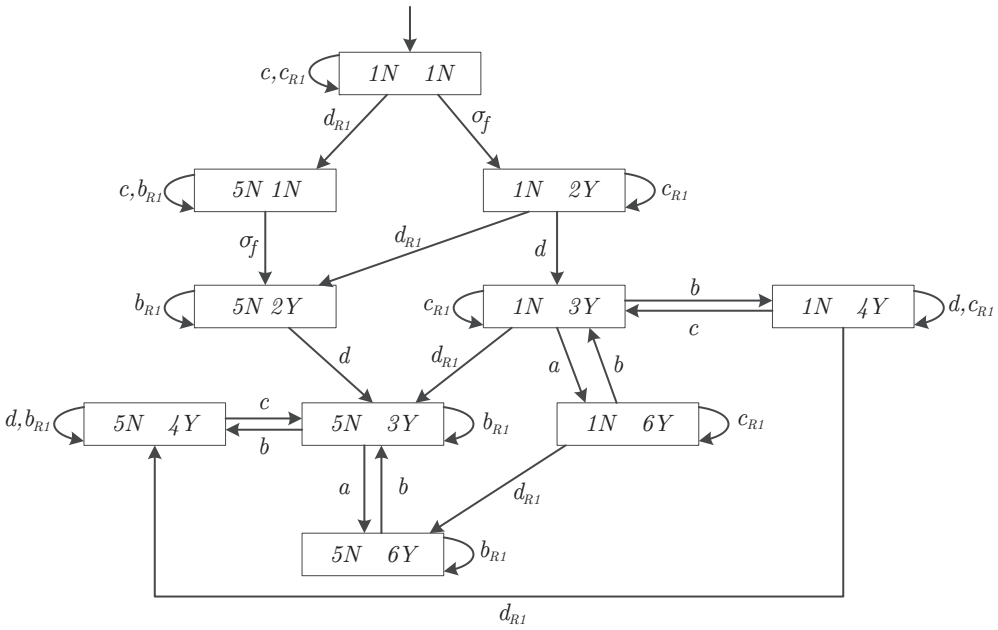


Figura 5.30: Verificador $G_{V,\{e\}}$ obtido ao considerar $\widetilde{\Sigma}_o = \{e\}$.

raiz é $\{d\}$, os nós $\{b, e\}$ e $\{c, e\}$ não serão criados como descendentes de $\{e\}$, uma vez que as árvores a partir do nós $\{b\}$ e $\{c\}$ já foram concluídas. Dessa forma, a construção da árvore com raiz $\{e\}$ é finalizada e apresentada na figura 5.31.

Figura 5.31: Árvore de eventos criada a partir de e conforme regras definidas no algoritmo 5.2.

Após construir todas as árvores de eventos, cada uma iniciando por um evento de Σ_o, foram obtidos os seguintes conjuntos candidatos a bases mínimas de G: $\Sigma^1_{min} = \{a, b, c, d\}$ e $\Sigma^2_{min} = \{a, c, d, e\}$. Então, ao realizar o passo 3 do algoritmo 5.2, o seguinte conjunto é formado: $\Sigma_{min} = \{\{a, b, c, d\}, \{a, c, d, e\}\}$. Observe que nenhum dos conjuntos obtidos será excluído de Σ_{min} no passo 4 do algoritmo 5.2.

Observe que, no exemplo 5.3, foi necessário a construção de apenas 14 verificadores até que a solução fosse obtida utilizando o algoritmo 5.2. Caso a força bruta fosse o método utilizado, seriam necessários 30 verificadores. A partir dos exemplos 5.2 e 5.3, é possível observar que quanto mais próxima a cardinalidade dos conjuntos pertencentes a Σ_{min} está da cardinalidade do conjunto de eventos observáveis Σ_o, mais verificadores serão necessários para que a solução seja obtida. No exemplo 5.3, a construção das ávores de eventos foi iniciada pela árvore cuja raiz é rotulada pelo evento a, que pertence as duas bases mínimas para o diagnóstico da linguagem de G. O exemplo 5.4, dado a seguir, mostra o resultado obtido ao iniciar a construção das árvores de eventos por uma árvore cuja raiz é rotulada pelo evento b.

Exemplo 5.4. *Considere novamente o sistema modelado pelo autômato G, mostrado na figura 4.9. Suponha $\Sigma_o = \{a, b, c, d, e\}$ e $\Sigma_f = \{\sigma_f\}$.*

Ao aplicar o algoritmo 5.2, será necessária a construção de cinco árvores de eventos, cada uma iniciando por um evento de Σ_o. Contudo, mesmo o algoritmo sendo iniciado pela árvore cuja raiz é rotulada por $\{b\}$, e escolhendo-se aleatoriamente as sequências que violam as condições de diagnosticabilidade, é possível obter as bases mínimas $\Sigma_{min} = \{\{a, b, c, d\}, \{a, c, d, e\}\}$ calculando apenas 14 verificadores.

Note que, de acordo com o exemplo 5.4, mesmo iniciando o algoritmo 5.2 por uma árvore cuja raiz é rotulada por um evento que não pertence às duas bases mínimas para o diagnóstico existente para o respectivo sistema, o número de verificadores necessários não foi alterado se comparado ao resultado obtido no exemplo 5.3. Considere agora o exemplo 5.5, dado a seguir.

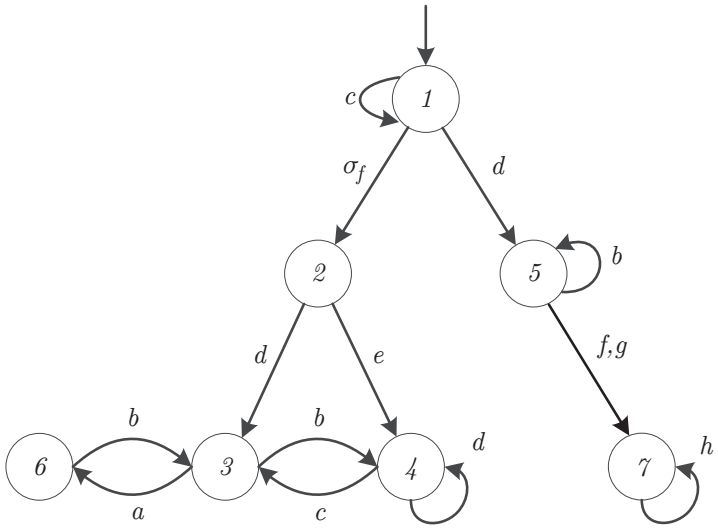


Figura 5.32: Autômato G do exemplo 5.5.

Exemplo 5.5. *Considere um sistema modelado pelo autômato G mostrado na figura 5.32. Seja $\Sigma_o = \{a, b, c, d, e, f, g, h\}$ o conjuntos de eventos observáveis de G e $\Sigma_f = \{\sigma_f\}$ o conjunto de eventos de falha.*

O sistema mostrado na figura 5.32 é o mesmo utilizado no exemplo 5.3 a menos das alterações realizadas na parte normal de G com a inclusão do estado 7 e dos eventos f, g e h. Então, uma vez que não houve alterações em G_F, é possível, ao utilizar o algoritmo 5.2, selecionar as mesmas sequências que violam a diagnosticabilidade do sistema escolhidas no exemplo 5.3. Dessa forma, serão necessários dezessete verificadores para concluir que $\Sigma_{min} = \{\{a, b, c, d\}, \{a, c, d, e\}\}$ (os mesmos catorze do exemplo 5.3 mais três verificadores devido aos três novos eventos de Σ_o).

Observe, a partir do exemplo 5.5, que, utilizando o algoritmo 5.2 foram necessários dezessete verificadores para obter Σ_{min}. Para esse exemplo, o método de busca exaustiva calcularia 254 verificadores.

Observação 5.2. *No exemplo 5.3 foram encontradas duas bases mínimas para o diagnóstico de falhas do sistema. Considere a árvore de eventos iniciada em a, mostrada na figura 5.23. Note que, uma vez que o evento a pertence aos dois conjuntos solução do problema, ambos podem ser obtidos a partir da árvore de eventos cuja raiz é a. Isso significa que, ao construir somente uma árvore de eventos conforme o algoritmo 5.2 a partir de um evento qualquer $\sigma_o \in \Sigma_o$, todas as bases mínimas para o diagnóstico do sistema nas quais o evento σ_o faz parte serão obtidas. Assim, caso seja de conhecimento prévio que um determinado evento pertence às bases mínimas de diagnóstico de falhas de um sistema, apenas a árvore de eventos cuja raiz é esse*

mesmo evento poderia ser construída para alcançar o mesmo resultado que se obtem ao construir todas as árvores de eventos.

Observação 5.3. *No exemplo 5.4 foram necessários apenas dezessete verificadores para obter Σ_{min}. Entretanto, caso a construção das árvores fosse iniciada pelas árvores cujas raízes são rotuladas por $\{f\}$, $\{g\}$ e $\{h\}$, inevitavelmente seria necessário o cálculo de mais verificadores. Uma alternativa para tentar evitar iniciar a construção das árvores por eventos que não fazem parte de nenhuma das bases mínimas para o diagnóstico é obter $G_{V,\emptyset}$ e, a partir do mesmo, obter uma sequência v que viole as condições de diagnosticabilidade da linguagem do sistema. Em seguida, basta obter as sequências s_N e s_F utilizando o algoritmo 4.2 e iniciar a construção das árvores pelos eventos de s_N e s_F que tornam suas projeções distintas.*

O conceito exposto na observação 5.2 nos leva a uma outra forma de se utilizar as ideias contidas no algoritmo 5.2. Suponha que se deseja obter todas as bases mínimas que contenham um determinado evento σ_o. Para alcançar essa resposta, bastaria construir a árvore de eventos cuja raiz é rotulada por $\{\sigma_o\}$. Caso o interesse seja em obter todas as bases mínimas de diagnóstico, basta construir $|\Sigma_o|$ árvore de eventos, cada uma iniciando a partir de um evento σ_o distinto, sendo $|\Sigma_o|$ a cardinalidade do conjunto de eventos observáveis.

Por fim, após ilustrar o uso do algoritmo 5.2 com quatro exemplos, será analisado o custo computacional que o mesmo apresenta.

5.1.1 Complexidade computacional do algoritmo 5.2

Ao utilizar o algoritmo 5.2, $|\Sigma_o|$ árvores de eventos serão construídas para obter as bases mínimas de diagnóstico da linguagem do sistema, cada uma iniciando a partir de um evento distinto de Σ_o, sendo $|\Sigma_o|$ a cardinalidade do conjunto de eventos observáveis de G.

Observe que a função ANALISE foi desenvolvida com base no algoritmo de busca em profundidade, apresentado no capítulo 2, e será chamada a partir da raiz da árvore de eventos. A partir de então, será iniciado um processo de verificação da diagnosticabilidade considerando a observação do conjunto que rotula o nó atual da árvore de eventos. Caso a diagnosticabilidade não seja verificada, será escolhida uma sequência v que viola as condições estabelecidas no teorema 2.2 e, a partir da mesma, devem ser recuperadas as sequências normal e de falha associadas a v. Em seguida, devem ser avaliados os eventos que tornam as projeções dessas sequências distintas. Com base nesses eventos, são criados nós na árvore de estados, como descendentes do nó atual. Em seguida, a função ANALISE é chamada recursivamente até que o conjunto que rotule um nó seja uma base para o diagnóstico ou alcance uma condição na qual o nó não necessite mais de qualquer análise.

Note ainda que, como o verificador obtido para o nó $\{a, b\}$ proveniente da árvore cuja raiz é $\{a\}$ é idêntico ao verificador para o nó $\{a, b\}$ oriundo da árvore cuja raiz é $\{b\}$, por exemplo, não existe a necessidade de se realizar essa análise novamente. O passo 3.3.1 do algoritmo 5.3 impede que verificadores já obtidos sejam calculados novamente, assim como os passos 3.3.4 e 3.3.2 impedem o cálculo do verificador sob observação de Σ_o ou sob a observação de um conjunto $\widetilde{\Sigma}_o$ que contenha uma solução obtida anteriormente, respectivamente. Além disso, o passo 3.3.3 impede a criação de um nó cujo rótulo seja composto por algum evento que rotula a raiz de alguma árvore já construída. Dessa forma, pode-se observar que o algoritmo 5.2 busca não realizar a verificação de todos os $2^{|\Sigma_o|}$ conjuntos possíveis. Contudo, em casos particulares e improváveis de ocorrerem em um sistema real, serão calculados, no pior caso, todas as combinações possíveis dos eventos de Σ_o a menos do conjunto vazio e de Σ_o, o que corresponde a $2^{|\Sigma_o|-2}$ combinações distintas. Como o custo computacional para se obter um verificador utilizando o algoritmo 2.3 é $O(|X|^2|\Sigma|)$, a complexidade computacional do algoritmo 5.2, considerando que os termos de ordem mais baixa são dominados pelos termos de ordem superior, será $O(2^{|\Sigma_o|-2}(|X|^2|\Sigma|))$.

Ao utilizar a força bruta, o custo computacional é, conforme apresentado no capítulo 4, $O(2^{|\Sigma_o|-2}(|X|^2|\Sigma|))$, que consiste no cálculo de um verificador para cada subconjunto de Σ_o a menos do conjunto vazio e de Σ_o.

A análise da complexidade computacional realizada nesta seção mostrou que o algoritmo 5.2 desenvolvido nesta dissertação, possui menor complexidade que os demais algoritmos existentes na literatura para obtenção das bases mínimas para o diagnóstico de falhas em um sistema a eventos discretos, sendo, no pior caso, de mesma complexidade que a utilização da força bruta. Contudo, observe que, a complexidade computacional do algoritmo 5.2 é obtida para um pior cenário possível, que dificilmente ocorre e, portanto, em geral, não será necessário obter os $2^{|\Sigma_o|-2}$ verificadores para executá-lo, conforme visto nos exemplo 5.2, 5.3, 5.4 e 5.5. Além disso, o algoritmo 5.2 utiliza as propriedades do sistema e, ao utilizá-lo, tem-se a garantia de que somente em um pior cenário possível o mesmo será equivalente à utilização do método de busca exaustiva, resultado até então não obtido.

Capítulo 6

Conclusão e trabalhos futuros

Neste trabalho, foram propostos dois novos algoritmos para obtenção de todas as bases mínimas para o diagnóstico de falhas em um sistema a eventos discretos, o método da árvore de caminhos ambíguos e o método da árvore de eventos, indicando, dessa forma, os eventos essenciais e os redundantes do sistema analisado. Uma vez que o diagnóstico de falhas em um SED necessita apenas dos eventos essenciais, a obtenção das bases mínimas permitem que o diagnóstico de falhas seja realizada com um número mínimo de sensores, o que pode representar uma vantajosa economia de recursos em um sistema real.

O método da árvore de caminhos ambíguos, primeiro algoritmo proposto, foi desenvolvido traçando-se um paralelo entre as definições e ideias utilizadas em [29], que utiliza autômatos diagnosticadores. Dessa forma, apesar do algoritmo proposto apresentar melhor desempenho computacional que o apresentado em [29], o mesmo pode, ainda, apresentar menor eficiência que a utilização da busca exaustiva pelas soluções, que consiste em verificar, para cada subconjunto de Σ_o, se o sistema é diagnosticável.

O segundo algoritmo apresentado neste trabalho buscou resolver o incoveniente da complexidade computacional existente no primeiro. De forma a alcançar menor complexidade computacional que a despendida utilizando a força bruta, foi utilizada uma metodologia inédita para construir os passos do método da árvore de eventos (algoritmo 5.2). Essa metodologia, que utiliza novamente autômatos verificadores, inova no sentido de não mais buscar por todos os ciclos que violam as condições de diagnosticabilidade de um sistema. A busca, passa a ser realizada apenas por um ciclo para cada verificador construído. Além disso, as árvores que são construídas não refletem mais os estados de um autômato, mas sim os conjuntos de eventos que estão sendo observados a cada passo do algoritmo. Esse conjunto de fatores permitiu o desenvolvimento do algoritmo com menor complexidade computacional para obtenção de todas as bases mínimas para diagnóstico de falhas em um SED dentre os existentes atualmente.

Deve ser destacado ainda que todos os verificadores utilizados nesta dissertação foram calculados conforme trabalho desenvolvido em [26], que possui atualmente a menor complexidade entre os verificadores existentes na literatura.

Como trabalhos futuros, é desejado que o estudo realizado nesta dissertação seja estendido para sistemas sob observação dinâmica de eventos. Um outro trabalho futuro possível seria a extensão para o caso de diagnóstico de falhas descentralizado.

Referências Bibliográficas

[1] M. SAMPATH, R. SENGUPTA, S. L. K. S., TENEKETZIS, D. "Failure diagnosis using discrete-event models", *IEEE Trans. on Control Systems Technology*, v. 4, n. 2, pp. 105–124, 1996.

[2] CASSANDRAS, C., LAFORTUNE, S. *Introduction to Discrete Event System*. Secaucus, NJ, Springer-Verlag New York, Inc., 2008.

[3] M. SAMPATH, R. SENGUPTA, S. L. K. S., TENEKETZIS, D. "Diagnosability of discrete-event systems", *IEEE Trans. on Automatic Control*, v. 40, n. 9, pp. 1555–1575, 1995.

[4] SAMPATH, M. "A Hybrid Aproach to Failure Diagnosis of Industrial Systems". In: *Proc 2001 American Control Conference*, pp. 2077–2082, Arlington, VA, 2001.

[5] M. SAMPATH, S. L., TENEKETZIS, D. "Active diagnosis of discrete-event systems", *IEEE Transactions on Automatic Control*, v. 43, pp. 908–929, 1998.

[6] 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, 2000.

[7] CONTANT, O., LAFORTUNE, S., TENEKETZIS, D. "Diagnosability of discrete event systems with modular structure", *Discrete Event Dynamic Systems: Theory And Applications*, v. 16, n. 1, pp. 9–37, 2006.

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

[9] LUNZE, J., SCHRODER, J. "State observation and diagnosis of discrete event systems described by stochastic automata", *Discrete Event Dynamic Systems-Theory And Applications*, v. 11, n. 4, pp. 319–369, 2001.

[10] THORSLEY, D., TENEKETZIS, D. "Diagnosability of stochastic discrete-event systems", *IEEE Trans. on Automatic Control*, v. 50, pp. 476–492, 2005.

[11] 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, 2006.

[12] JIANG, S., KUMAR, R., GARCIA, H. "Diagnosis of repeated/intermittent failures in discrete event systems", *IEEE Transactions on Robotics and Automation*, v. 19, n. 2, pp. 310–323, 2003.

[13] KILIC, E. "Diagnosability of fuzzy discrete event systems", *Information Sciences*, v. 178, n. 3, pp. 858–870, 2008.

[14] ZAD, S., KWONG, R., WONHAM, W. "Fault diagnosis in discrete-event systems: incorporating timing information", *IEEE Transactions on Automatic Control*, v. 50, n. 7, pp. 1010–1015, 2005.

[15] PETERSON, J. *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ, Prentice Hall, 1981.

[16] MURATA, T. "Petri nets - properties, analysis and applications", *Proceedings of the IEEE*, v. 77, n. 4, 1989.

[17] DAVID, R., ALLA, H. *Discrete, Continuous, and Hybrid Petri Nets*. New York, NY, Springer, 2005.

[18] GENC, S., LAFORTUNE, S. "Distributed diagnosis of place-bordered Petri nets", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 2, pp. 206–219, 2007.

[19] CABASINO, M. P., GIUA, A., SEATZU, C. "Diagnosability of bounded Petri nets". In: *28th Chinese Control Conference*, pp. 1254–1260, Shangai, China, 2009.

[20] CABASINO, M. P., GIUA, A., LAFORTUNE, S., et al. "Diagnosability Analysis of Unbounded Petri nets". In: *28th Chinese Control Conference*, pp. 1267–1272, Shangai, China, 2009.

[21] GAUBERT, S., GIUA, A. "Petri Net Languages and Infinite Subsets of N m". 1999.

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

[23] S. JIANG, Z. HUANG, V. C., KUMAR, R. "A polynomial algorithm for testing diagnosability of discrete-event systems", *IEEE Trans. on Automatic Control*, v. 46, n. 8, 2001.

[24] 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, 2002.

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

[26] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C. "Polynomial time verification of decentralized diagnosability of discrete event systems", *IEEE Transactions on Automatic Control*, pp. 1679–1684, 2011.

[27] BASILIO, J. C., LAFORTUNE, S. "Robust codiagnosability of discrete event systems". In: *Proc 2009 American Control Conference*, pp. 2202–2209, St. Louis, MO, 2009.

[28] TAKAI, S. "Robust failure diagnosis of partially observed discrete event systems". In: *Proceedings of 10th International Workshop on Discrete Event Systems*, Berlim, 2010.

[29] BASILIO, J. C., LIMA, S. T. S., LAFORTUNE, S., et al. "Computation of minimal event bases that ensure diagnosability", *Discrete Event Dynamic Systems: Theory and Applications*, v. 22, pp. 249–292, 2012.

[30] DEBOUK, R., LAFORTUNE, S., TENEKETZIS, D. "On an optimization problem in sensor selection", *Discrete Event Dynamic Systems: Theory and Applications*, v. 12, pp. 417–445, 2002.

[31] TRAVÉ-MASSUYÈS, L., E. T. O. X. "Diagnosability analysis based on component-supported analytical redundancy relations", *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, v. 36, pp. 1146–1160, 2006.

[32] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C. "Generalized robust diagnosability of discrete event systems". In: *18th IFAC World Congress*, pp. 8737–8742, Milano, Italy, 2011.

[33] M. SAFAR, K. A., ALBEHAIRY, S. "Counting Cycles in an Undirected Graph using DFS-XOR Algorithm", *IEEE Transactions on Automatic Systems*, 2009.

[34] T.-S, Y., LAFORTUNE, S. "On the computacional complexity of some problems arising in partially-observed discrete-event systems". In: *Proc 2001 American Control Conference*, pp. 307–312, Arlington, VA, 2001.

[35] JOHNSON, D. B. "Finding all the elementary circuits of a directed graph". In: *SIAM J. Comput.*, pp. 77–84, 1975.

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

[37] T. H. CORMEN, C. E. LEISERSON, R. L. R., STEIN, C. *Introduction to algorithms*. Massachusetts, MIT Press, 2007.