VERIFICAÇÃO DA DIAGNOSTICABILIDADE E DIAGNÓSTICO ONLINE DE FALHAS EM SISTEMAS HÍBRIDOS

Victor Ruas Alvarez

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.

Orientadores: Marcos Vicente de Brito Moreira
Oumar Diene

Rio de Janeiro
Março de 2015

VERIFICAÇÃO DA DIAGNOSTICABILIDADE E DIAGNÓSTICO ONLINE DE FALHAS EM SISTEMAS HÍBRIDOS

Victor Ruas Alvarez

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. Eduardo Vieira Leão Nunes, D.Sc.

Prof. Patrícia Nascimento Pena, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2015

Alvarez, Victor Ruas

Verificação da Diagnosticabilidade e Diagnóstico Online de Falhas em Sistemas Híbridos/Victor Ruas Alvarez. – Rio de Janeiro: UFRJ/COPPE, 2015.

XII, 76 p.: il.; 29, 7cm.

Orientadores: Marcos Vicente de Brito Moreira
Oumar Diene

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2015.

Referências Bibliográficas: p. 73 – 76.

1. Sistemas a Eventos Discretos. 2. Sistemas Híbridos. 3. Autômatos. 4. Redes de Petri. 5. Diagnóstico de Falhas. I. Moreira, Marcos Vicente de Brito *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Aos meus pais Sheila e Rogério,
ao meu irmão Vinicius e a
minha noiva Giselle.

Agradecimentos

Aos meus pais, Sheila Miranda Ruas Alvarez e Rogério Guimarães Alvarez, que desde sempre cuidaram de mim de forma exemplar, não deixando que as dificuldades impostas pela vida afetassem o investimento em minha educação. Sempre presentes, me ensinaram que conhecimento é sempre o melhor investimento que pode ser feito e que família é a base de tudo.

Ao meu irmão Vinicius Ruas Alvarez, à quem tenho a responsabilidade de servir de exemplo, fazendo com que todas as minhas ações sejam muito bem pensadas antes de serem realizadas. Apesar das pequenas brigas normais entre irmãos, o amo e me preocupo com ele desde o primeiro segundo de vida.

À minha noiva Giselle Roza Accampora, o anjo que me escolheu como companheiro para passarmos juntos todos os dias de nossas vidas e construirmos uma família. Me dando apoio e incentivo sempre que preciso, cercando minha vida de amor e carinho, e sempre apoiando minhas decisões, me fazendo sentir a pessoa mais feliz e realizada do mundo. Ela certamente merece mais do que estas poucas linhas escritas.

Aos meus orientadores Marcos Vicente de Brito Moreira e Oumar Diene, que, apesar das adversidades encontradas, confiaram em mim e no meu trabalho. Me deram todo o suporte e orientação necessários para a conclusão desta etapa em minha vida, além dos ensinamentos passados dentro e fora da sala de aula.

Aos amigos feitos durante as disciplinas do mestrado, em especial a Gustavo Viana e Rafael Pereira que compartilharam o aprendizado e sempre me apoiaram a continuar. A motivação e o apoio de vocês foi fundamental para superar as dificuldades.

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

VERIFICAÇÃO DA DIAGNOSTICABILIDADE E DIAGNÓSTICO ONLINE DE FALHAS EM SISTEMAS HÍBRIDOS

Victor Ruas Alvarez

Março/2015

Orientadores: Marcos Vicente de Brito Moreira
Oumar Diene

Programa: Engenharia Elétrica

O diagnóstico de falhas é um problema muitas vezes tratado construindo modelos do sistema em que a ocorrência de falhas deve ser diagnosticada. A partir desses modelos, deve-se inferir sobre a ocorrência de falhas considerando eventos que foram observados. Quando a linguagem do sistema não é diagnosticável, ou seja, quando não é possível afirmar que uma falha ocorreu, ou quando o tempo de atraso entre a ocorrência da falha e sua detecção não é aceitável, mais informações precisam ser integradas ao modelo do sistema. Em estudos recentes, a verificação da diagnosticabilidade e o diagnóstico online do sistema são realizados inserindo-se informações das dinâmicas contínuas do sistema no modelo, levando a modelos híbridos, e construindo-se autômatos diagnosticadores para sistemas híbridos, que possuem complexidade computacional exponencial para sua construção. Visando reduzir o tempo computacional para a verificação da diagnosticabilidade, este trabalho propõe a construção de um autômato verificador, que possui complexidade computacional linear com o número de estados e eventos do sistema. Uma vez que os autômatos verificadores não são capazes de prover o diagnóstico online do sistema, é proposto um método para que, após a verificação da diagnosticabilidade por meio do autômato verificador, o diagnóstico online seja realizado. Esse método é baseado na construção de uma rede de Petri diagnosticadora, que indicará a ocorrência de uma falha.

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

DIAGNOSABILITY VERIFICATION AND ONLINE FAULT DIAGNOSIS OF HYBRID SYSTEMS

Victor Ruas Alvarez

March/2015

Advisors: Marcos Vicente de Brito Moreira
Oumar Diene

Department: Electrical Engineering

The fault diagnosis is a problem usually treated using system models in which the occurrence of fault events must be diagnosed. When the language of the system is not diagnosable, i.e., when it is not possible to state that a fault event has occurred, or when the time between the occurrence of the fault and its detection is not acceptable, more information needs to be integrated into the system model. In recently studies, the verification of diagnosability and online diagnosis have been performed by adding the information of the continuous dynamics of the system into the model, leading to a hybrid model, and by constructing a diagnoser automaton, which is computed with exponential complexity. In order to reduce the computational time for the verification of diagnosability, this work proposes the construction of a verifier automaton, which has linear computational complexity with the number of states and system events. Since the verifier automaton cannot be used for online diagnosis, we proprose in this work a new method for online diagnosis. This method is based on the construction of a diagnoser Petri net, that indicates the occurrence of a fault event.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O diagnóstico de falhas em sistemas a eventos discretos (SEDs) é um assunto que vem sendo amplamente discutido no últimos anos [1–18], e possui como objetivos detectar e informar a ocorrência de falhas em um determinado sistema. Um SED é um sistema no qual seu conjunto de estados é discreto e sua evolução é regida por eventos. As principais modelagens utilizadas para esse tipo de sistema são por meio da construção de autômatos [19, 20] e redes de Petri [21–24].

Todos os sistemas são passíveis de falhas, seja por algum defeito em um sensor, mau funcionamento de um atuador ou até por perda de conexão durante uma transmissão de dados. Essas falhas precisam ser tratadas, para eliminar prejuízos e riscos, e para que isso aconteça, as falhas devem ser identificadas.

O diagnóstico de falhas em SEDs tem como base duas condicionantes: (i) a falha que se deseja diagnosticar é caracterizada como sendo um evento não-observável, ou seja, nenhum dos sensores é capaz de detectar sua ocorrência; (ii) o evento de falha, depois de ocorrido, muda o comportamento do sistema, não acarretando, necessariamente, em uma parada (bloqueio) do mesmo.

Em [2], condições necessárias e suficientes para a diagnosticabilidade de falhas em sistemas a eventos discretos foi apresentada, juntamente com a proposição de construção de um autômato diagnosticador, que permite inferir a capacidade do sistema de diagnosticar falhas existentes e, também, para realizar o diagnóstico online de falhas. O diagnosticador é um autômato cujos estados representam uma estimativa do estado atual do sistema, rotulados para indicar se o comportamento está normal ou se alguma falha ocorreu.

Para que a diagnosticabilidade de falhas, por intermédio do autômato diagnosticador, seja verificada, é preciso que seja feita uma busca por ciclos indeterminados. Apesar da construção do diagnosticador ser simples e possuir aplicação tanto para verificação da diagnosticabilidade quanto para detecção online de falhas, a sua construção possui complexidade exponencial [25, 26].

Quando a linguagem de um SED não é diagnosticável ou quando o atraso en-

tre a ocorrência do evento de falha e a sua detecção não é tolerável, ou seja, o sistema evolui por um tempo excessivamente longo até que a falha seja detectada, o sistema e a sua modelagem, precisam ser revistos. Levando-se em conta apenas a parte dirigida por eventos discretos do sistema, é possível que, após alterações feitas no sistema, como, por exemplo, a inclusão de novos sensores para detectar a ocorrência de eventos, o sistema continue sendo não-diagnosticável em relação a uma ou mais falhas. Quando a parte a eventos discretos do sistema não é suficiente para que a linguagem seja diagnosticável, é possível utilizar informações da dinâmica de variáveis contínuas do sistema, obtendo, assim, um sistema híbrido (SH), ou seja, que possui variáveis de estado discretas que evoluem com a ocorrência de eventos e variáveis de estado contínuas que evoluem com o tempo. Diversos trabalhos têm sido recentemente publicados sobre o diagnóstico de falhas em sistemas híbridos [27–31].

A modelagem de sistemas híbridos é feita levando-se em conta duas partes: uma parte de sistemas a eventos discretos extraída do sistema e uma que evolui no tempo segundo um equação diferencial ou a diferenças finitas. Essa separação faz com que as teorias propostas para cada tipo de sistema possam ser utilizadas separadamente e integradas para se gerar um modelo com maior quantidade de informações.

Ao considerar a diagnosticabilidade de falhas em SHs, os trabalhos propostos em [27–29, 32] estendem a linguagem do autômato que modela o comportamento do sistema, inserindo novos eventos e/ou estados, tendo como base a dinâmica contínua do sistema, e constrõem um autômato diagnosticador para verificar a diagnosticabilidade de falhas do sistema.

Em [27], é utilizado o conceito de assinatura de falhas. Cada falha tem sua ocorrência indicada por uma assinatura, que é uma alteração causada na medida do estado em que o sistema se encontrava quando a falha ocorreu. Em [28] é introduzido o conceito de assinatura dos modos, acrescentando ao modelo a eventos discretos do sistema eventos associados a essas assinaturas. Entretanto, esses métodos de diagnóstico de falhas precisam realizar a análise de consistência de todos os estados discretos do sistema a partir da medição das variáveis contínuas, o que acarreta em um elevado esforço computacional.

Visando aumentar a eficiência do diagnóstico de falhas, em [29] é proposta a análise da consistência de estados estimados do sistema após a ocorrência de uma sequência de eventos observáveis. A estimativa de estados reduz o conjunto de estados em que a análise de consistência deve ser realizada.

Contudo, é sabido que a construção de um autômato diagnosticador requer um tempo computacional exponencial, o que, na maioria das vezes, pode ser um problema. Visando reduzir o tempo computacional, em [13] é proposto um algoritmo que realiza a construção de um autômato verificador com complexidade polinomial.

Este trabalho possui como objetivo propor uma metodologia para a verificação

da diagnosticabilidade de falhas em sistemas híbridos, assim como em [13], por meio da construção de autômatos verificadores. Verificadores podem ser construídos em tempo polinomial e a verificação da diagnosticabilidade é feita buscando-se componentes fortemente conexas [33], no lugar de caminhos cíclicos.

O autômato diagnosticador, além de verificar a diagnosticabilidade de falhas, pode ser utilizado para a realização do diagnóstico online do sistema. Uma vez que o autômato verificador não permite acompanhamento online do sistema, este trabalho propõe, também, a construção de uma rede de Petri diagnosticadora para sistemas híbridos, baseada no método de diagnóstico online de falhas em SEDs apresentado em [34].

Para apresentar as soluções propostas, este trabalho está estruturado da seguinte forma: no capítulo 2 são revistos os principais conceitos necessários ao entendimento dos resultados sobre diagnóstico de falhas. No capítulo 3, a teoria acerca do diagnóstico de falhas em sistemas a eventos discretos modelados por autômatos é apresentada juntamente com a construção do autômato diagnosticador e do autômato verificador. No capítulo 4, o diagnóstico de falhas é estendido para sistemas híbridos. Ainda no capítulo 4, a construção de um autômato diagnosticador para sistemas híbridos é apresentada e é proposto um algoritmo para a construção de um novo autômato verificador para verificação da diagnosticabilidade de falhas em SH. A rede de Petri diagnosticadora para acompanhamento online de sistemas híbridos é apresentada no capítulo 5, em conjunto com o seu algoritmo de construção. Por fim, no capítulo 6, propostas de trabalhos futuros e as conclusões acerca deste trabalho estão descritas.

Capítulo 2

Sistemas a Eventos Discretos

Neste capítulo são apresentados os fundamentos teóricos necessários para a composição desta dissertação e seu entendimento. Para tanto, este capítulo está organizado da seguinte forma: Na seção 2.1 é apresentada a definição de sistemas a eventos discretos (SEDs). Na seção 2.2 é apresentado o conceito de linguagens e projeções e na seção 2.3 a teoria de autômatos é descrita. Na seção 2.4 é apresentado o conceito de autômatos com observação reduzida e demonstrada a construção do autômato observador de estados. Por fim, na seção 2.5 está apresentada a teoria sobre Redes de Petri.

2.1 Definição de Sistemas a Eventos Discretos

Sistemas a eventos discretos [19] são sistemas que possuem espaço de estados discretos e que evoluem conforme a ocorrência de eventos, que podem ser percebidos através de sensores ou não, dependendo do sistema e de suas características. Esses eventos representam, basicamente, as ocorrências de determinadas ações ou de acontecimentos naturais, como por exemplo, o ato de ligar ou desligar algum equipamento, a detecção de presença, o término de uma contagem, o apertar de um botão, a indicação de nível alto/baixo em um tanque, entre outros. Uma vez ocorrido um dos eventos, o sistema evolui instantaneamente, mudando para um novo estado em que novos eventos são possíveis de ocorrer, ou então permanecendo no mesmo estado. Note que essas mudanças não dependem do tempo, os estados do sistema se modificam exclusivamente e imediatamente após a ocorrência de um evento, gerando imprevisibilidade temporal. Pode-se saber quais são os possíveis eventos de um determinado estado, mas não será possível determinar quando eles ocorrerão. Assim sendo, pode-se definir um SED como um sistema dinâmico de estados discretos cuja evolução se dá pela ocorrência, em geral assíncrona, de eventos ao longo do tempo.

Diferentemente de sistemas dinâmicos de variáveis contínuas, SEDs não podem ser modelados por equações diferenciais. Desta forma, estados e eventos são os

principais componentes do modelo de um SED, mais especificamente a sequência de estados alcançados e os eventos que geraram as correspondentes transições dos estados.

O conjunto de todas as sequências de eventos possíveis e de comprimento finito de um sistema é chamado de linguagem, e possui papel fundamental no acompanhamento da evolução de um SED.

2.2 Linguagens

O conjunto de eventos associado a um SED pode ser interpretado como o alfabeto de uma linguagem e uma sequência de eventos, uma palavra dessa mesma linguagem. Assim sendo, define-se uma linguagem como o conjunto de todas as sequências de eventos (palavras) de comprimento finito de um sistema.

Uma sequência que não contém eventos é chamada de sequência vazia e é denotada por ε. O comprimento de uma sequência é o número de eventos nela contidos, incluindo múltiplas ocorrências do mesmo evento. Se s é uma sequência, o comprimento de s é denotado por $|s|$. Por convenção, o comprimento da sequência vazia ε é zero.

A operação chave para a construção de sequências, e portanto linguagens, a partir de um conjunto de eventos é a concatenação. A concatenação de duas sequências de eventos s_1 e s_2 tem como resultado uma sequência de eventos $s = s_1 s_2$, cujo comprimento é $|s| = |s_1| + |s_2|$, em que os $|s_1|$ primeiros eventos são provenientes da primeira sequência e estes serão imediatamente seguidos dos $|s_2|$ eventos da segunda.

Seja Σ um conjunto de eventos. Então, o fecho de Kleene de Σ, denotado por Σ^*, é o conjunto formado por todas as sequências de comprimento finito formadas com os eventos de Σ, incluindo a sequência vazia ε.

Exemplo 2.1 *Suponha um conjunto de eventos* $\Sigma = \{a, b\}$. *O fecho de Kleene de* Σ *pode ser visto a seguir.*

$$\Sigma^* = \{\varepsilon, a, b, ab, ba, aa, bb, aab, abb, baa, bba, aabb, abab, ...\}.$$

Observe, portanto, que o fecho de Kleene de um conjunto é um conjunto infinito, porém enumerável, uma vez que contém sequências de comprimento finito.

Da mesma forma que as operações de concatenação e fecho de Kleene foram definidas para eventos e suas sequências, elas também podem ser definidas para linguagens. A concatenação de duas linguagens resulta em sequências de eventos formadas pela concatenação de sequências da primeira linguagem com a segunda.

Deste modo, sejam $L_a, L_b \subseteq \Sigma^*$, a concatenação de L_a e L_b é definida como:

$$L_a L_b := \{s \in \Sigma^* : (s = s_a s_b) \wedge (s_a \in L_a) \wedge (s_b \in L_b)\}. \tag{2.1}$$

O fecho de Kleene de uma linguagem, por sua vez, é o conjunto de todas as sequências possíveis de serem geradas pela concatenação de todas as sequências dessa linguagem. Seja $L \subseteq \Sigma^*$, o fecho de Kleene de L é representado por:

$$L^* := \{\varepsilon\} \cup L \cup LL \cup LLL \cup ... \tag{2.2}$$

Outra operação importante é o fecho de prefixo de uma linguagem, que consiste de todos os prefixos de todas as sequências da linguagem. Se $s = tu$ e $t, u \in \Sigma^*$, então t é dito ser um prefixo de s. Observe que ε e, até mesmo, s são prefixos de s. Formalizando o fecho de prefixo, seja $L \subseteq \Sigma^*$:

$$\overline{L} := \{s \in \Sigma^* : (\exists t \in \Sigma^*)[st \in L]\}. \tag{2.3}$$

Caso uma linguagem L seja igual ao seu fecho de prefixo, isto é, $L = \overline{L}$, L é dita ser prefixo-fechada.

Um outro tipo de operação frequentemente utilizada em sequências e linguagens é a operação de projeção. Essa operação é aplicada de um conjunto de eventos para um subconjunto dele, ou seja, a projeção $P : \Sigma_l^* \to \Sigma_s^*$, em que $\Sigma_s \subseteq \Sigma_l$. Define-se a operação de projeção da seguinte forma [35]:

$$P(\varepsilon) := \varepsilon, \tag{2.4}$$

$$P(e) := \begin{cases} e, & \text{se } e \in \Sigma_s \\ \varepsilon, & \text{se } e \in \Sigma_l \setminus \Sigma_s \end{cases}, \tag{2.5}$$

$$P(se) := P(s)P(e) \text{ para } s \in \Sigma_l^*, e \in \Sigma_l. \tag{2.6}$$

Dado que "$\setminus$" denota a diferença de conjuntos, note que a operação de projeção remove os eventos da sequência que não pertencem ao conjunto menor Σ_s.

A projeção inversa, $P^{-1} : \Sigma_s^* \to 2^{\Sigma_l^*}$, é definida da seguinte forma:

$$P^{-1}(t) := \{s \in \Sigma_l^* : P(s) = t\}. \tag{2.7}$$

Dada uma sequência de eventos t do conjunto menor (Σ_s^*), a projeção inversa $P^{-1}(t)$ retorna o conjunto de todas as sequências s do conjunto maior (Σ_l^*) que, ao realizar a projeção P, obtém-se a sequência do conjunto menor ($P(s) = t$).

Exemplo 2.2 *Suponha dois conjuntos de eventos $\Sigma_s = \{a\}$ e $\Sigma_l = \{a, b\}$. A projeção P para uma sequência de eventos ababbaa e a projeção inversa P^{-1} de*

uma sequência de eventos aa podem ser vistas a seguir.

$$P(ababbaa) = \{aaaa\}.$$

$$P^{-1}(aa) = \{b^i ab^j ab^k : i, j, k \in \mathbb{N}\}.$$

A linguagem pode ser vista como uma maneira formal de descrever o comportamento de um SED. Ela especifica as sequências de eventos admissíveis que o SED é capaz de processar ou gerar, enquanto dispensa alguma estrutura adicional. O problema é que representações de linguagens não são sempre fáceis de se trabalhar, se fazendo necessário a utilização de uma estrutura compacta que defina linguagens e que possa ser manipulada por meio de operações bem definidas para sua construção e, posteriormente, análise. Dois formalismos são normalmente utilizados para este fim: autômatos e redes de Petri. Nas próximas seções será apresentado um breve resumo sobre esses formalismos.

2.3 Autômatos

2.3.1 Definição

Autômatos são usados para representar linguagens de um SED de acordo com regras bem definidas. Sua representação gráfica é construída, basicamente, por meio dos estados e eventos associados ao SED.

Definição 2.1 *Um autômato determinístico é uma sêxtupla:*

$$G = (Q, \Sigma, \phi, \Gamma, q_0, Q_m), \tag{2.8}$$

em que:
Q *é o conjunto de estados;*
Σ *é o conjunto finito de eventos;*
$\phi : Q \times \Sigma^* \to Q$ *é a função de transição;*
$\Gamma : Q \to 2^{\Sigma}$ *é a função de eventos ativos;*
q_0 *é o estado inicial;*
$Q_m \subseteq Q$ *é o conjunto de estados marcados.*

A função de transição $\phi(q, e) = y$ indica que existe uma transição rotulada pelo evento e do estado q para o estado y e a função de eventos ativos $\Gamma(q)$ é o conjunto de todos os eventos e em que $\phi(q, e)$ é definido.

O autômato G opera da seguinte forma: o começo é pelo estado inicial q_0 e, com a ocorrência de um evento $e \in \Gamma(q_0) \subseteq \Sigma$, uma transição para o estado $\phi(q_0, e) \in Q$ é criada. Esse processo continua com base nas transições em que ϕ é definida.

O modo mais simples de representar um autômato é por meio do diagrama de transição de estados, onde os estados são representados por círculos, os estados marcados por dois círculos concêntricos e as transições por arcos orientados. Estados são marcados quando é necessário atribuir a eles um significado especial, como por exemplo a finalização de uma tarefa. O diagrama de transição de estados será ilustrado a seguir, no exemplo retirado de [19].

Exemplo 2.3 *A figura 2.1 representa um autômato* $G = (Q, \Sigma, \phi, \Gamma, q_0, Q_m)$*, em que o conjunto de eventos é* $\Sigma = \{a, b, g\}$ *e o conjunto de estados* $Q = \{x, y, z\}$*. O estado inicial é* $q_0 = x$ *e o conjunto de estados marcados é* $Q_m = \{x, z\}$*. A função de estados viáveis* Γ *e a função de transição de estados* ϕ *são as seguintes:* $\Gamma(x) = \{a, g\}$, $\Gamma(y) = \{a, b\}$, $\Gamma(z) = \{a, b, g\}$, $\phi(x, a) = x$, $\phi(x, g) = z$, $\phi(y, a) = x$, $\phi(y, b) = y$, $\phi(z, a) = y$, $\phi(z, b) = z$ *e* $\phi(z, g) = y$.

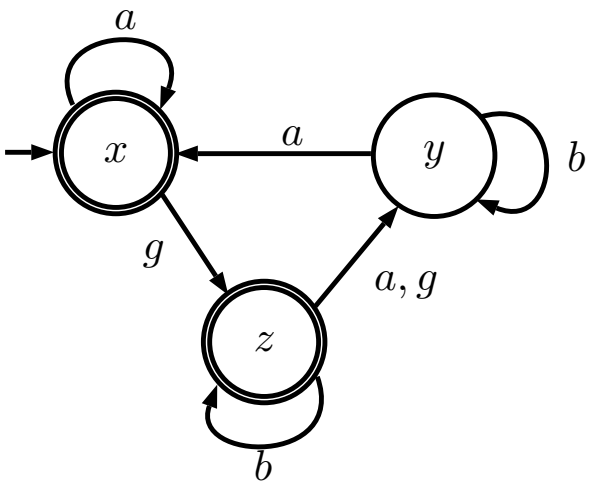


Figura 2.1: Exemplo de Autômato [19]

2.3.2 Linguagens Representadas por um Autômato

A conexão entre linguagens e o autômato que representa essas linguagens, pode ser feita por inspeção no diagrama de transição de estados do autômato. Ao considerar-se todos os caminhos que podem ser seguidos no diagrama de transição de estados de um autômato G, a partir do estado inicial, e, dentro deste conjunto, sequências de eventos que terminem em um estado marcado, têm-se, respectivamente, as linguagens gerada, $\mathcal{L}(G)$, e marcada, $\mathcal{L}_m(G)$, por G. As definições destas linguagens estão descritas a seguir:

$$\mathcal{L}(G) := \{s \in \Sigma^* : \phi(q_0, s) \text{ é definida}\}, \tag{2.9}$$

$$\mathcal{L}_m(G) := \{s \in \mathcal{L}(G) : \phi(q_0, s) \in Q_m\}. \tag{2.10}$$

A linguagem $\mathcal{L}(G)$ representa todas as sequências de eventos que podem ser geradas no diagrama de transição de estados do autômato, a partir do estado inicial. Uma sequência s estará em $\mathcal{L}(G)$ se e somente se ela corresponder a um caminho admissível no diagrama de transição de estados, equivalentemente, se e somente se ϕ for definida em (q_0, s). $\mathcal{L}(G)$ é prefixo-fechada por definição, uma vez que um caminho só é possível se todos os prefixos também o forem, ou seja, $\overline{\mathcal{L}(G)} = \mathcal{L}(G)$.

A segunda linguagem representada por G, $\mathcal{L}_m(G)$, é um subconjunto de $\mathcal{L}(G)$ consistindo apenas de sequências s em que $\phi(q_0, s) \in Q_m$. Note que a linguagem marcada por G, $\mathcal{L}_m(G)$, só será prefixo-fechada se todos os estados de Q forem marcados. No caso de G ser um autômato vazio, as linguagens gerada e marcada por ele também serão iguais ao conjunto vazio $\emptyset$.

Para perfeita compreensão deste trabalho, é necessário que sejam definidos os conceitos de *caminho* e *caminhos cíclicos*.

Definição 2.2 *Um caminho, em um autômato, é representado por uma sequência alternada de estados e eventos* $(q_k, \sigma_1, q_{k+1}, \sigma_2, q_{k+2}, ..., \sigma_l, q_{k+l}), l > 0$ *tais que* $q_{k+i} = \phi(q_{k+i-1}, \sigma_i), \forall i \in \{1, 2, 3, ..., l\}$.

Definição 2.3 *Um caminho* $(q_k, \sigma_1, q_{k+1}, \sigma_2, q_{k+2}, ..., \sigma_l, q_{k+l})$*, de um autômato, é dito ser cíclico caso o último estado do caminho seja igual ao primeiro, ou seja,* $q_{k+l} = q_k$.

Caminhos estão relacionados tanto a estados quanto a eventos. Quando é preciso fazer referência somente aos estados presentes em um caminho cíclico, utiliza-se o nome *ciclo de estados*, ou somente *ciclos*. A definição formal está representada a seguir.

Definição 2.4 *Em um caminho cíclico* $(q_k, \sigma_1, q_{k+1}, \sigma_2, q_{k+2}, ..., \sigma_l, q_k)$*, um ciclo é representado pela sequência de estados* $(q_k, q_{k+1}, q_{k+2}, ..., q_k)$*, desconsiderando-se os eventos.*

2.3.3 Operações com Autômatos

De modo a analisar SEDs modelados por autômatos, é preciso que um conjunto de operações para um único autômato seja definido, visando modificar apropriadamente o diagrama de transição de estados de acordo com alguma operação que se queira realizar em sua linguagem. Além dessas operações, chamadas de unárias, também são necessárias operações que envolvam dois ou mais autômatos, com o objetivo de compor, ou combiná-los, podendo-se modelar sistemas complexos a partir de modelos individuais de seus componentes.

Parte Acessível

A partir da definição de $\mathcal{L}(G)$ e $\mathcal{L}_m(G)$, pode ser visto que é possível remover de G todos os estados que não são acessíveis ou alcançáveis a partir de q_0 (seu estado inicial) por alguma sequência de $\mathcal{L}(G)$, sem afetar as linguagens gerada e marcada por G. Quando um estado é removido, as transições associadas a ele também são removidas. A operação de obtenção da parte acessível de um autômato é denotada por $Ac(G)$ e é formalizada do seguinte modo:

$$Ac(G) := (Q_{ac}, \Sigma, \phi_{ac}, \Gamma_{ac}, q_0, Q_{ac,m}), \tag{2.11}$$

em que $Q_{ac} = \{q \in \Sigma : (\exists s \in \Sigma^*)[\phi(q_0, s) = q]\}, Q_{ac,m} = Q_m \cap Q_{ac}$ e $\phi_{ac} = \phi \mid_{Q_{ac} \times \Sigma \to Q_{ac}}$. A função ϕ_{ac} representa uma restrição aplicada em ϕ. A função $Ac(G)$ remove os estados de G que não são alcançáveis a partir do estado inicial, e as transições associadas aos estados removidos. Note que o conjunto de eventos de $Ac(G)$ permanece igual ao de G, mesmo que algum evento do conjunto não esteja presente no diagrama de transição de estados de $Ac(G)$.

Parte Coacessível

Tomar a parte coacessível de um autômato G consiste na remoção de todos os estados de G, e suas respectivas transições, a partir das quais não é possível se alcançar um estado marcado. Note que esta operação altera $\mathcal{L}(G)$, a linguagem gerada pelo autômato. A linguagem marcada permanece inalterada, uma vez que os estados removidos não fazem parte de nenhuma caminho de q_0 a um estado de Q_m. Esta operação é denotada por $CoAc(G)$ e é definida da seguinte forma:

$$CoAc(G) := (Q_{coac}, \Sigma, \phi_{coac}, \Gamma_{coac}, q_{0,coac}, Q_m), \tag{2.12}$$

em que $Q_{coac} = \{q \in Q : (\exists s \in \Sigma^*)[\phi(q, s) \in Q_m]\}$, $q_{0,coac} = q_0$ se $q_0 \in Q_{coac}$ e indefinido caso contrário, e $\phi_{coac} = \phi \mid_{Q_{coac} \times \Sigma^* \to Q_{coac}}$. Novamente pode-se perceber que a função ϕ foi restringida, dando origem à ϕ_{coac}, e que o conjunto de eventos permanece igual ao de G.

Operação Trim

Um autômato é dito ser *trim* se ele for tanto acessível quanto coacessível. A operação *trim* é definida a seguir:

$$Trim(G) := CoAc[Ac(G)] = Ac[CoAc(G)]. \tag{2.13}$$

Produto

A operação produto, denotada por $\times$, é uma composição de dois autômatos que permite a ocorrência dos eventos que são comuns a ambos. Uma transição só ocorrerá se o evento puder ser ativado nos dois autômatos da composição. Para melhor entendimento, a seguir será apresentado o resultado do produto de dois autômatos $G_1 = (Q_1, \Sigma_1, \phi_1, \Gamma_1, q_{0_1}, Q_{m_1})$ e $G_2 = (Q_2, \Sigma_2, \phi_2, \Gamma_2, q_{0_2}, Q_{m_2})$:

$$G_1 \times G_2 := Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \phi_{1\times 2}, \Gamma_{1\times 2}, (q_{0,1}, q_{0,2}), Q_{m_1} \times Q_{m_2}), \tag{2.14}$$

em que $\Gamma_{1\times 2} = \Gamma_1(q_1) \cap \Gamma_2(q_2)$ e $\phi_{1\times 2}$ possui a seguinte definição:

$$\phi_{1\times 2}((q_1, q_2), e) := \begin{cases} (\phi_1(q_1, e), \phi_2(q_2, e)), & \text{se } e \in \Gamma_1(q_1) \cap \Gamma_2(q_2) \\ \text{Não definida}, & \text{caso contrário.} \end{cases} \tag{2.15}$$

Os estados de $G_1 \times G_2$ são denotados por pares, onde o primeiro componente é um estado de G_1 e o segundo, um estado de G_2. Pode ser verificado que $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$ e $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$.

Exemplo 2.4 *Considere os autômatos G e G_2 das figuras 2.1 e 2.2, respectivamente. Então o produto $G \times G_2$ é representado pelo diagrama da figura 2.3.*

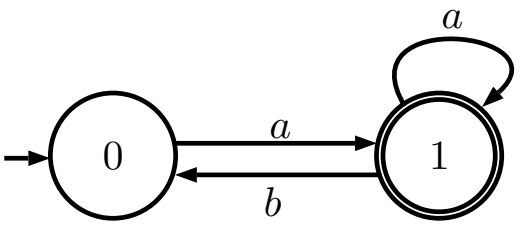


Figura 2.2: Autômato G_2

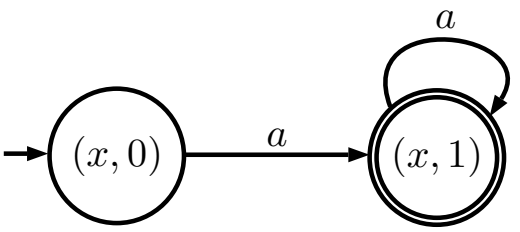


Figura 2.3: Produto entre G da figura 2.1 e G_2 da figura 2.2

Composição Paralela

A composição por produto é restritiva e somente permite transições com eventos comuns. De modo geral, quando sistemas são modelados por meio da composição

de seus componentes, o conjunto de eventos de cada componente possui eventos privados que pertencem ao seu comportamento interno e eventos comuns que são compartilhados com outros autômatos e sincronizam os diversos componentes do sistema. A maneira padrão de se construir modelos de sistemas inteiros a partir dos modelos individuais de seus componentes é através da composição paralela. Esta operação é denotada por $\|$ e, por permitir que eventos exclusivos de cada autômato da composição ocorram, está relacionada a união dos conjuntos de eventos. O resultado da composição paralela de dois autômatos G_1 e G_2 é definido como:

$$G_1 \parallel G_2 := Ac(Q_1 \parallel Q_2, \Sigma_1 \cup \Sigma_2, \phi_{1\|2}, \Gamma_{1\|2}, (q_{0,1}, q_{q,2}), Q_{m_1} \parallel Q_{m_2}), \tag{2.16}$$

em que $\Gamma_{1\|2} = [\Gamma_1(q_1) \cap \Gamma_2(q_2)] \cup [\Gamma_1(q_1) \setminus \Sigma_2] \cup [\Gamma_2(q_2) \setminus \Sigma_1]$ e $\phi_{1\|2}$ possui a seguinte definição:

$$\phi_{1\|2}((q_1, q_2), e) := \begin{cases} (\phi_1(q_1, e), \phi_2(q_2, e)), & \text{se } e \in \Gamma_1(q_1) \cap \Gamma_2(q_2) \\ (\phi_1(q_1, e), q_2), & \text{se } e \in \Gamma_1(q_1) \setminus \Sigma_2 \\ (q_1, \phi_2(q_2, e)), & \text{se } e \in \Gamma_2(q_2) \setminus \Sigma_1 \\ \text{Não definida}, & \text{caso contrário.} \end{cases} \tag{2.17}$$

Na composição paralela, eventos comuns, isto é, eventos em $\Sigma_1 \cap \Sigma_2$, só podem ser executados se os dois autômatos, simultaneamente, puderem. Por outro lado, os eventos privados não necessitam de sincronização entre os autômatos, podendo ser executados sempre que possível.

Exemplo 2.5 *Na figura 2.4 pode ser observada a composição paralela dos autômatos G e G_2 das figuras 2.1 e 2.2, respectivamente.*

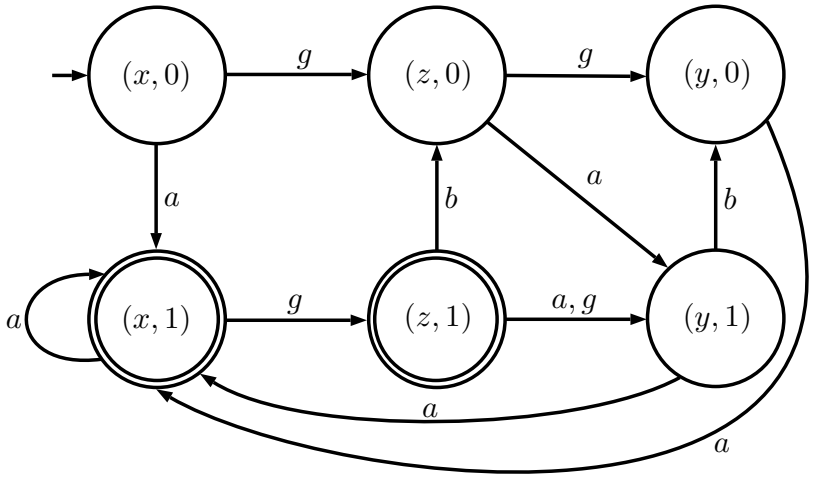


Figura 2.4: Composição Paralela entre G da figura 2.1 e G_2 da figura 2.2

2.4 Observadores de Estados

Os eventos de Σ considerados até então podem ser classificados como observáveis, isto é, sua ocorrência é sabida e pode ser detectada pelos sensores do sistema. Entretanto, nem todos os eventos têm sua ocorrência visualizada, seja por ausência de sensor, falta de comunicação entre partes do sistema, ou até mesmo por ocorrer uma falha.

Sistemas a eventos discretos que possuem eventos observáveis e não-observáveis são denominados de SEDs com observação parcial. Esses sistemas têm o conjunto de eventos Σ particionado da seguinte forma: $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, isto é, $\Sigma = \Sigma_o \cup \Sigma_{uo}$ e $\Sigma_o \cap \Sigma_{uo} = \emptyset$, em que Σ_o e Σ_{uo} denotam os conjuntos de eventos observáveis e não-observáveis, respectivamente.

Visando representar um autômato com observação parcial, identificando os possíveis estados do sistema após a observação de uma sequência de eventos, é necessário que seja definido o conceito de observador. Contudo, para que o algoritmo de construção do observador seja apresentado, é necessário, primeiro, apresentar o conceito de alcance não-observável, denotado por $UR(q)$.

Definição 2.5 *O alcance não-observável de um estado $q \in Q$, denotado por $UR(q)$, é definido como:*

$$UR(q) = \{y \in Q : (\forall s \in \Sigma_{uo}^*)[\phi(q, s) = y]\}. \tag{2.18}$$

Para um conjunto de estados $B \in 2^Q$, o alcance não-observável pode ser definido, também, da seguinte forma:

$$UR(B) = \bigcup_{q \in B} UR(q). \tag{2.19}$$

Note que o alcance não-observável de um estado q é o conjunto formado por todos os estados alcançáveis a partir de q através de transições rotuladas por eventos não observáveis, incluindo o próprio estado q.

Definição 2.6 *O observador de uma autômato G em relação a um conjunto de eventos observáveis Σ_o, denotado por $Obs(G)$, é dado por:*

$$Obs(G) = (Q_{obs}, \Sigma_o, \phi_{obs}, \Gamma_{obs}, q_{0,obs}, Q_{m,obs}), \tag{2.20}$$

em que $Q_{obs} \subset 2^Q$, $Q_{m,obs} = \{B \in Q_{obs} : B \cap Q_m \neq \emptyset\}$, ϕ_{obs}, Γ_{obs} e $q_{0,obs}$ são obtidos de acordo com o algoritmo 2.1.

Esta operação, denotada por *Obs*, tem como finalidade calcular um autômato determinístico a partir do qual se pode estimar, após a ocorrência de eventos ob-

serváveis, os estados em que o sistema pode se encontrar. A construção do autômato observador é feita de acordo com o seguinte algoritmo [36].

Algoritmo 2.1 *Construção do Observador*

- *Passo 1: Defina* $q_{0_{obs}} = UR(q_0)$ *e faça* $Q_{obs} = \{q_{0,obs}\}$ *e* $\tilde{Q}_{obs} = Q_{obs}$
- *Passo 2: Faça* $\hat{Q}_{obs} \leftarrow \tilde{Q}_{obs}$ *e* $\tilde{Q}_{obs} \leftarrow \emptyset$
- *Passo 3: Para cada* $B \in \hat{Q}_{obs}$
  - *Passo 3.1:* $\Gamma_{obs}(B) = (\bigcup_{q \in B} \Gamma(q)) \cap \Sigma_o$;
  - *Passo 3.2: Para cada* $e \in \Gamma_{obs}(B)$*, faça:*
    - *Passo 3.2.1:* $\phi_{obs}(B, e) = UR(\{q \in Q : (\forall y \in B)[q = \phi(y, e)]\})$
    - *Passo 3.2.2:* $\tilde{Q}_{obs} \leftarrow \tilde{Q}_{obs} \cup \phi_{obs}(B, e)$
- *Passo 4:* $Q_{obs} \leftarrow Q_{obs} \cup \tilde{Q}_{obs}$
- *Passo 5: Repita os passos 2 a 4 até que toda parte acessível de* $Obs(G)$ *tenha sido construída.*
- *Passo 6:* $Q_{m_{obs}} = \{B \in Q_{obs} : B \cap Q_m \neq \emptyset\}$

O algoritmo 2.1 calcula o alcance não-observável para cada estado de G, começando por $q_{0,obs}$, enquanto busca as transições possíveis desses estados com os eventos de Σ_o, construindo ϕ_{obs} e Γ_{obs}. Por fim, o conjunto de estados marcados é definido.

Pode-se concluir, a partir da definição do observador e de seu algoritmo de construção, que a linguagem gerada pelo observador, $\mathcal{L}(Obs(G))$, é a projeção da linguagem gerada por G sobre o conjunto de eventos observáveis, ou seja, $\mathcal{L}(Obs(G)) = P_o[\mathcal{L}(G)]$, em que $P_o : \Sigma^* \rightarrow \Sigma_o^*$.

Exemplo 2.6 *Considere o autômato* G *da figura 2.5 e os seguintes conjuntos de eventos:* $\Sigma = \{a, b, e\}, \Sigma_o = \{a, b\}$ *e* $\Sigma_{uo} = \{e\}$*. O observador de* G*,* $Obs(G)$*, pode ser visto na figura 2.6.*

Como não há eventos não-observáveis viáveis no estado inicial de G*, este estado será o mesmo estado inicial do observador. O único evento possível no estado* 0 *é "*a*", fazendo o sistema ir para o estado* 1*. Ao se realizar o alcance não-observável do estado* 1*, o estado* 2 *é encontrado, pois o evento não-observável "*e*" interliga os estados* 1 *e* 2*. Portanto, o segundo estado encontrado no observador é* $\{1, 2\}$*. Neste estado é preciso analisar a ocorrência de eventos observáveis nos estados* 1 *e* 2*. A ocorrência do evento "*b*" é permitida nos estados* 1 *e* 2*: em* 1*, o evento*

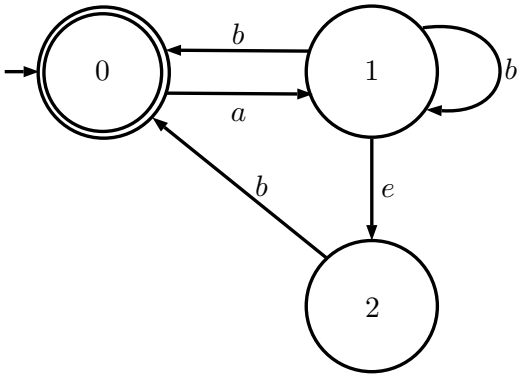


Figura 2.5: Autômato G para construção do observador

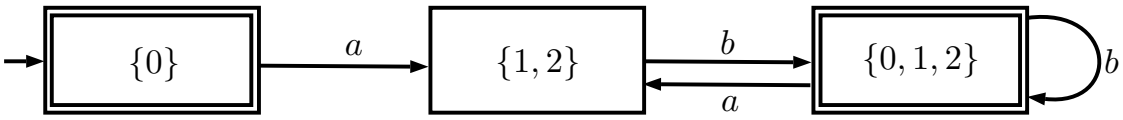


Figura 2.6: Observador de G

"b" faz o sistema permanecer neste mesmo estado, que somado ao alcance não-observável, resultam nos estados $\{1, 2\}$*; em* 2*, o evento "b" faz o sistema retornar ao estado inicial, que não tem nenhum evento não-observável como viável. Com essas informações, pode-se concluir que o terceiro estado do observador é* $\{0, 1, 2\}$*. Por fim, neste estado, a ocorrência do evento "a" retorna o sistema ao estado anterior do observador e a ocorrência do evento "b" faz com que o sistema possa estar em qualquer um dos três estados de* G*, e, portanto, um autolaço é adicionado ao estado* $\{0, 1, 2\}$ *rotulado por "b".*

2.5 Redes de Petri

Uma alternativa para modelagem de SEDs é provida pelas redes de Petri [21–23]. Assim como autômatos, as redes de Petri representam linguagens de acordo com regras bem definidas. Uma de suas características é que ela inclui condições explícitas sobre quando eventos podem ser habilitados.

2.5.1 Definição

O processo de definição de uma rede de Petri envolve dois passos. Primeiro, é preciso definir o grafo da rede de Petri, também chamado de estrutura da rede, que é análogo ao diagrama de transição de estados de um autômato. Em seguida, são inseridos alguns outros itens necessários: estado inicial, conjunto de estados marcados e a função de rotulação das transições. Com esses dois passos, tem-se um modelo completo da rede de Petri, suas dinâmicas associadas e as linguagens gerada

e marcada por ela.

Em redes de Petri, eventos são associados às *transições*. Para que uma transição ocorra, algumas condições precisam ser satisfeitas. As condições para que essas transições disparem estão contidas nos *lugares* da rede. Esses dois elementos são os principais dentro da estrutura de uma rede de Petri, e podem ser chamados de nós do grafo. Nós do mesmo tipo não podem ser conectados, ou seja, lugares são ligados a transições e transições ligadas a lugares, caracterizando um grafo bipartido. As interligações de transições e lugares são feitas por arcos.

O formalismo da estrutura de uma rede de Petri pode ser visto a seguir [23].

Definição 2.7 *A estrutura de uma rede de Petri é um grafo bipartido ponderado*

$$(P, T, Pre, Post), \tag{2.21}$$

em que
P *é o conjunto finito de lugares;*
T *é o conjunto finito de transições;*
$Pre : (P \times T) \to \mathbb{N} = 0, 1, 2, ...$ *é a função de arcos que ligam lugares a transições;*
$Post : (T \times P) \to \mathbb{N} = 0, 1, 2, ...$ *é a função de arcos que ligam transições a lugares;*

Na definição é suposto que $(P, T, Pre, Post)$ não possui lugares ou transições isolados.

É conveniente utilizar $I(t_j)$ para representar o conjunto de lugares de entrada de uma transição t_j. De modo similar, $O(t_j)$ representa o conjunto de lugares de saída de uma transição t_j. Então, tem-se:

$$I(t_j) = \{p_i \in P : Pre(p_i, t_j) > 0\} \ , \ O(t_j) = \{p_i \in P : Post(t_j, p_i) > 0\}. \tag{2.22}$$

De modo similar, pode-se descrever os conjuntos das transições de entrada e saída de um dado lugar p_i:

$$I(p_i) = \{t_j \in T : Post(t_j, p_i) > 0\} \ , \ O(p_i) = \{t_j \in T : Pre(p_i, t_j) > 0\}. \tag{2.23}$$

Exemplo 2.7 *Considere uma rede de Petri definida por*

$$P = \{p_1, p_2\},\ T = \{t_1\},\ Pre(p_1, t_1) = 3 \text{ e } Post(t_1, p_2) = 1.$$

Neste caso, $I(t_1) = \{p_1\}$ *e* $O(t_1) = \{p_2\}$. *O grafo correspondente da rede de Petri pode ser visto na figura 2.7. O fato de* $Pre(p_1, t_1)$ *possuir o valor* 3 *indica a existência de dois arcos do lugar* p_1 *para a transição* t_1, *que pode ter sua representação simplificada para um arco com peso* 3.

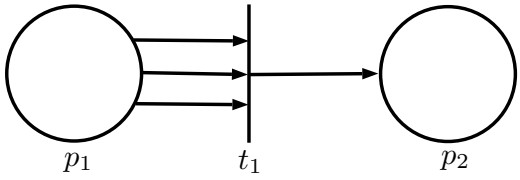


Figura 2.7: Grafo da rede de Petri do exemplo 2.7

2.5.2 Redes de Petri com Marcação e Espaço de Estados

Utilizando a idéia de que as transições em um grafo da rede de Petri representam os eventos de um SED, e que lugares descrevem as condições para que esses eventos ocorram, é preciso um mecanismo que indique quando essas condições foram alcançadas ou não. Esse mecanismo é obtido atribuindo-se *tokens* (fichas) aos lugares. Uma ficha é algo colocado nos lugares essencialmente para indicar que a condição descrita pelo lugar é satisfeita. A forma com que as fichas são atribuídas aos lugares do grafo da rede de Petri define a marcação. Formalmente, uma marcação m de um grafo da rede de Petri $(P, T, Pre, Post)$ é a função $m : P \to \mathbb{N} = \{0, 1, 2, ...\}$. Assim, uma marcação m define o vetor $\mathbf{m} = [m(p_1), m(p_2), ..., m(p_n)]$, em que n é o número de lugares da rede de Petri. A i-ésima entrada de $\mathbf{m}$ indica o número (inteiro não-negativo) de fichas no lugar p_i, $m(p_i) \in \mathbb{N}$. No grafo da rede de Petri, uma ficha é representada por um ponto preto posicionado no lugar apropriado.

Definição 2.8 *Uma rede de Petri é uma quíntupla*

$$(P, T, Pre, Post, m_0) \tag{2.24}$$

em que $(P, T, Pre, Post)$ é o grafo da rede e m_0 é a marcação inicial do conjunto de lugares P.

Exemplo 2.8 *Considere a rede de Petri da figura 2.7. Na figura 2.8, estão indicadas duas possibilidades de marcação, nomeadas pelos vetores $\mathbf{m}_1 = [1, 0]$ e $\mathbf{m}_2 = [3, 1]$.*

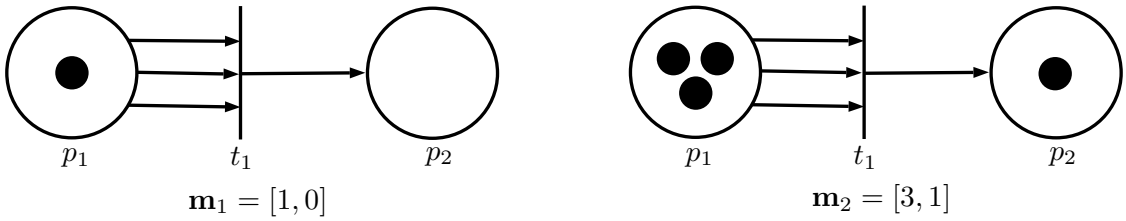


Figura 2.8: Duas marcações, $\mathbf{m}_1$ e $\mathbf{m}_2$, para o grafo da rede de Petri da figura 2.7

O estado de uma rede de Petri é definido pelo vetor de marcação $\mathbf{m} = [m(p_1), m(p_2), ..., m(p_n)]$. Note que o número de fichas atribuídas a um lugar é um valor inteiro não-negativo arbitrário, não necessariamente limitado. Seguindo este raciocínio, o número de estados possíveis é, de um modo geral, infinito. Assim, o espaço de estados de uma rede de Petri com n lugares é definido por vetores n-dimensionais com valores não-negativos.

Antes de apresentar o mecanismo de transição de estados de uma rede de Petri, é preciso que seja definido o conceito de *transições habilitadas*.

Uma transição $t_j \in T$ de uma rede de Petri é dita habilitada se:

$$m(p_i) \geq Pre(p_i, t_j), \text{ para todo } p_i \in I(t_j), \tag{2.25}$$

ou seja, a transição t_j da rede de Petri está habilitada quando o número de fichas em p_i é maior ou igual ao peso do arco que conecta p_i a t_j, para todos os lugares p_i que são entradas para a transição t_j. Na figura 2.8 com estado $\mathbf{m}_1$, como $m(p_1) = 1 < Pre(p_1, t_1) = 3$, t_1 não está habilitada.

2.5.3 Dinâmica de Redes de Petri

Em autômatos, o mecanismo da transição de estados é diretamente vinculado aos arcos que conectam os nós (estados) no diagrama de transição de estados. Já nas redes de Petri, a transição dos estados ocorre com a movimentação das fichas através da rede. Quando uma transição está habilitada, é dito que ela pode ser disparada.

Caso ocorra, pelo menos, um disparo de transição, a nova marcação $\mathbf{m}'$ da rede de Petri pode ser obtida da seguinte forma:

$$m'(p_i) = m(p_i) - Pre(p_i, t_j) + Post(t_j, p_i), \ i = 1, ..., n \tag{2.26}$$

Note que o próximo estado, definido pela equação (2.26), depende exclusivamente dos lugares de entrada e saída da transição t_j e dos pesos dos arcos que conectam esses lugares a transição t_j.

É importante observar que o número de fichas em uma rede de Petri não precisa ser conservado mediante o disparo de transições.

Exemplo 2.9 *Para ilustrar o processo de disparo de transições e mudança de estados de uma rede, considere a rede de Petri da figura 2.9(a), em que o estado inicial é* $\mathbf{m}_0 = [1, 0, 0, 0]$. *Neste caso, a única transição habilitada é* t_1, *uma vez que ela necessita de uma ficha no lugar* p_1 *e ele contém* $m_0(p_1) = 1$. *Em outras palavras,* $m_0(p_1) \geq Pre(p_1, t_1)$, *e a condição (2.25) é satisfeita para a transição* t_1. *Quando* t_1 *dispara, uma ficha é removida de* p_1 *e uma ficha é inserida nos lugares* p_2 *e* p_3. *É possível aplicar (2.26) para obter o novo estado* $\mathbf{m}_1 = [0, 1, 1, 0]$, *indicado na figura*

2.9(b). Neste estado, das transições t_1, t_2 e t_3, t_2 é a única habilitada. Após o disparo da transição t_2, uma ficha é removida do lugar de entrada p_2 e inserida no lugar de saída p_4. O estado alcançado é $\mathbf{m}_2 = [0, 0, 1, 1]$, ilustrado na figura 2.9(c). Por fim, nesse estado, a única transição habilitada é t_3 e, após seu disparo, as fichas dos lugares p_3 e p_4 são removidas e uma ficha é inserida no lugar p_1, retornando ao estado inicial $\mathbf{m}_0 = [1, 0, 0, 0]$ representado na figura 2.9(a).

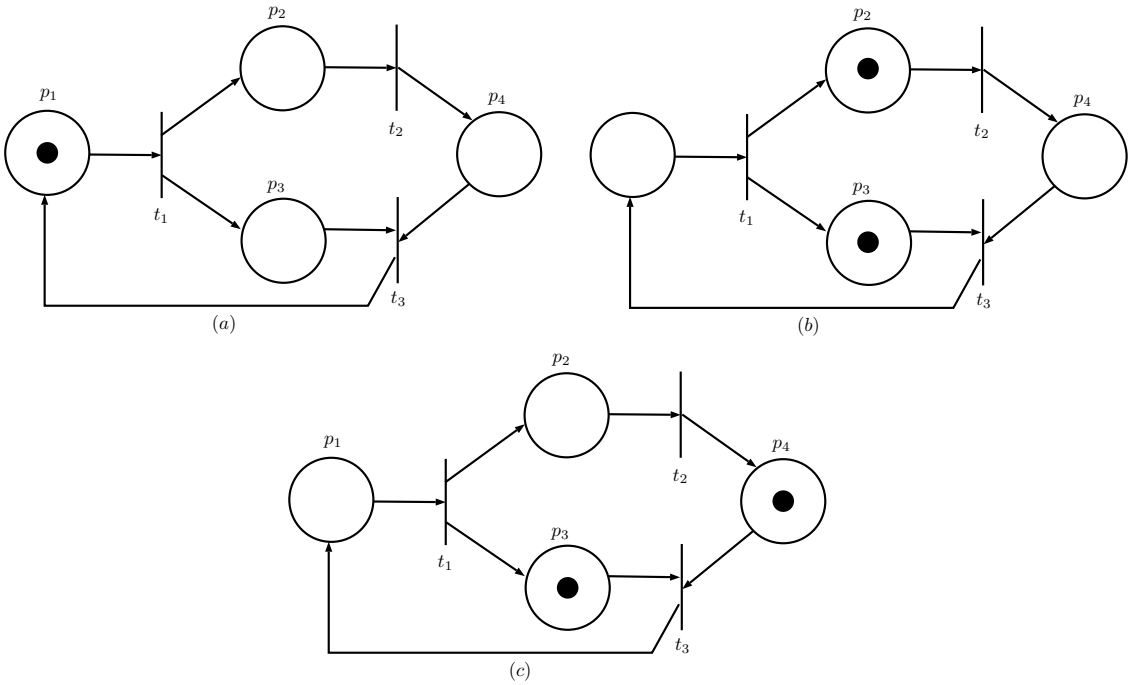


Figura 2.9: Sequência de disparo de transições do exemplo 2.9

Note que a quantidade de fichas em cada estado da rede de Petri é diferente. Em $\mathbf{m}_0$ tem-se 1 ficha e em $\mathbf{m}_1$ e $\mathbf{m}_2$ o quantitativo é de 2 fichas.

Uma observação importante sobre o comportamento da dinâmica de redes de Petri é que nem todos os estados em $\mathbb{N}^n$ são necessariamente alcançáveis a partir de um estado inicial. Considere o exemplo da figura 2.8 com o estado inicial $\mathbf{m}_2 = [3, 1]$, o único estado alcançável a partir de $\mathbf{m}_2$ é $[0, 2]$.

2.5.4 Linguagens de Redes de Petri

Para que a representação de linguagens por intermédio de redes de Petri seja possível, é preciso que seja especificado qual evento corresponde a cada transição. Para tanto, é necessário introduzir a definição de rede de Petri rotulada [23].

Definição 2.9 *Uma rede de Petri rotulada é uma séptupla:*

$$N = (P, T, Pre, Post, \Sigma, \ell, \mathbf{m}_0) \tag{2.27}$$

em que
$(P, T, Pre, Post)$ *é o grafo da rede de Petri;*
Σ *é o conjunto de eventos para rotulação das transições;*
$\ell : T \to \Sigma$ *é a função de rotulação das transições;*
$\mathbf{m}_0 \in \mathbb{N}^n$ *é o estado inicial da rede.*

No grafo da rede de Petri, o rótulo de uma transição é indicado próximo a ela.
A linguagem gerada por uma rede de Petri rotulada é definida como:

$$\mathcal{L}(N) := \{\ell(s) \in \Sigma^* : s \in T^* \text{ e } \varphi(\mathbf{m}_0, s) \text{ é definida}\} \tag{2.28}$$

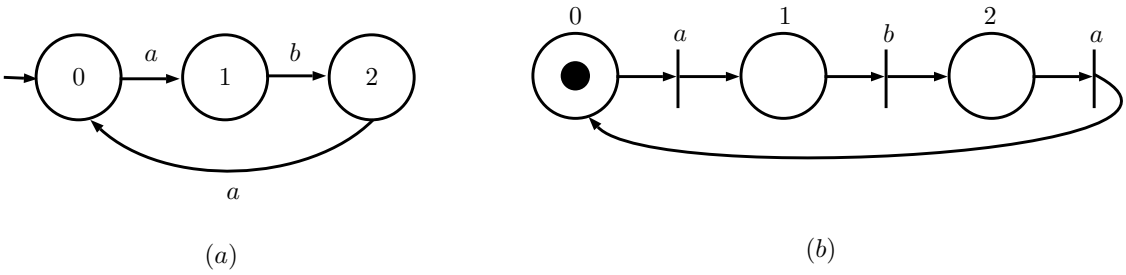
A linguagem $\mathcal{L}(N)$ representa todas as sequências de transições rotuladas que são obtidas por todas os possíveis sequências de disparo de transições em N, a partir do estado inicial $\mathbf{m}_0$ de N.

2.5.5 Redes de Petri Especiais

Redes de Petri Máquina de Estados

Uma rede de Petri é dita ser máquina de estados se cada uma das transições da rede estiver ligada somente a um lugar de entrada e a um lugar de saída. Além disso, caso esta rede possua apenas uma ficha durante toda a dinâmica, o seu comportamento é idêntico ao de um autômato, sendo cada lugar da rede de Petri máquina de estados referente a um estado do autômato correspondente.

Exemplo 2.10 *Considere o autômato G, representado na figura 2.10(a). A rede de Petri máquina de estados equivalente a este autômato G pode ser observada na figura 2.10(b). Note que para transformar um autômato em uma rede de Petri máquina de estados, é necessário substituir os estados e arcos do autômato por lugares e transições da rede, respectivamente. É de suma importância que seja respeitada a equivalência entre os lugares de entrada e saída durante o processo.*



(a) (b)

Figura 2.10: Autômato G do exemplo 2.10(a) e sua Rede de Petri Máquina de Estado equivalente (b).

Redes de Petri Binárias

Uma rede de Petri binária é aquela em que o número máximo de fichas em cada lugar é igual a um. Deste modo, caso ocorra o disparo de uma transição e o lugar que irá receber uma ficha já tiver uma, o quantitativo de fichas deste lugar permanecerá igual a um, obrigatoriamente.

O que define uma rede de Petri como sendo binária é a diferença na regra de evolução para a marcação dos lugares alcançados após o disparo de transições. Em uma rede de Petri binária, após o disparo de uma transição t_j, a marcação dos lugares segue o seguinte princípio:

$$\mathbf{m}(p_i) := \begin{cases} 0, & \text{se } m(p_i) - Pre(p_i, t_j) + Post(t_j, p_i) = 0 \\ 1, & \text{se } m(p_i) - Pre(p_i, t_j) + Post(t_j, p_i) > 0, \end{cases} \tag{2.29}$$

para $i = 1, ..., n$

Redes de Petri Estendidas

Visando aumentar o poder de modelagem de redes de Petri, é possível definir um tipo especial de arco, denominado arco inibidor. A função de arcos inibidores pode ser descrita da seguinte forma:

$$In : (P \times T) \to \mathbb{N} \tag{2.30}$$

Arcos inibidores podem ser utilizados para eliminar conflitos. Caso haja um arco inibidor ligando um lugar p_i à transição t_j, esta transição t_j será desabilitada se o número de fichas em p_i for maior ou igual ao peso do arco que conecta p_i à t_j, i.e., $m(p_i) \geq In(p_i, t_j)$. Quando uma transição ligada a um arco inibidor está habilitada e é disparada, as fichas presentes nos lugares ligados ao arco inibidor não são removidas. Arcos inibidores são graficamente representados por uma linha terminando em um pequeno círculo.

O formalismo de uma rede de Petri estendida pode ser visto a seguir [23, 24].

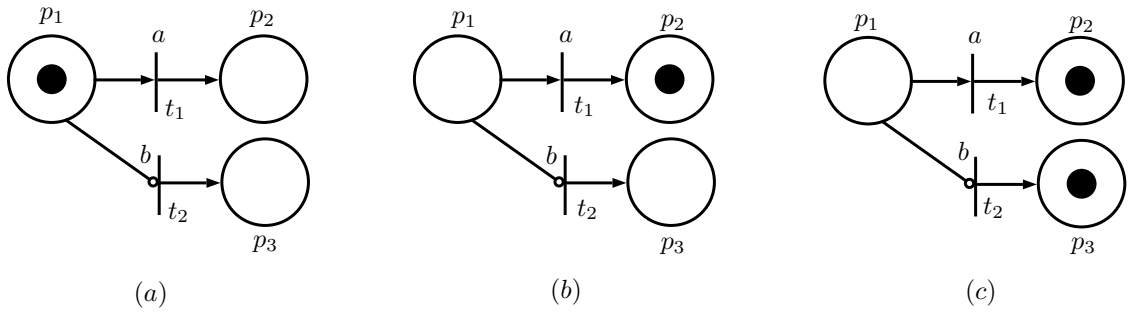
Definição 2.10 *Uma rede de Petri estendida é uma sêxtupla*

$$(P, T, Pre, Post, m, In) \tag{2.31}$$

em que $(P, T, Pre, Post, m)$ é uma rede de Petri com marcação e In a função de arcos inibidores.

Exemplo 2.11 *Considere a rede de Petri estendida da figura 2.11. A única transição habilitada é "t_1", rotulada pelo evento "a", pois o lugar p_1 inibe a habilitação da transição t_2, rotulada por "b". A figura 2.11(b) representa a mesma*

rede de Petri após o disparo da transição t_1. Note que após a saída da ficha do lugar ligado ao arco inibidor, a transição t_2 está habilitada. Após o disparo da transição t_2, a disposição de fichas na rede pode ser observado na figura 2.11(c).



(a) (b) (c)

Figura 2.11: Rede de Petri estendida do exemplo 2.11(a), rede após o disparo da transição "a"(b) e rede após o disparo da transição "b"(c)

Capítulo 3

Diagnóstico de Falhas em Sistemas a Eventos Discretos Modelados por Autômatos

O diagnóstico de falhas é uma área de estudo que tem sido amplamente discutida nos últimos anos [1–18]. Esta área surgiu aliada à crescente automatização dos sistemas e ao aumento da necessidade de se ter controle acerca de falhas, que são inerentes, e podem ocorrer. As aplicações para as teorias propostas acerca deste assunto são inúmeras, não se restringindo somente a sistemas modelados por eventos discretos, que serão o foco deste capítulo. Sistemas dinâmicos de variáveis contínuas e sistemas híbridos também fazem parte do conjunto de aplicações.

Esse capítulo está organizado do seguinte modo: na seção 3.1 o problema de diagnóstico de falhas é introduzido. Na seção 3.2 é apresentada a diagnose de falhas com base no autômato diagnosticador G_d e na seção 3.3 é apresentada uma abordagem para a verificação da diagnosticabilidade de um SED baseada em um autômato verificador G_V.

3.1 Diagnosticabilidade de Falhas

A diagnosticabilidade é uma propriedade que está ligada à possibilidade de detecção de uma falha, a partir do modelo de um sistema, após a ocorrência de uma sequência de eventos de comprimento limitado. Um sistema é dito ser diagnosticável se, após a ocorrência de um evento de falha seguido de uma sequência limitada de eventos, é possível afirmar que a falha ocorreu. Vale ressaltar que as falhas a serem diagnosticadas são eventos que não podem ser observados e que, a partir da ocorrência de uma falha, o sistema tem seu comportamento modificado. Essa mudança de comportamento do sistema não significa que a evolução dos estados acarrete em um

bloqueio, podendo os sistemas continuarem operantes mas ocasionando danos aos seus componentes e/ou gerando perdas de produção.

Para solucionar este problema, sistemas de diagnose de falhas são construídos. Utilizando apenas o conjunto de eventos observáveis, estes sistemas são capazes de alertar quanto à ocorrência de falhas. Para tal, é necessário que, primeiramente, se construa um modelo a eventos discretos do sistema, englobando tanto o comportamento normal quanto o de falha. Após a devida modelagem, é necessário utilizar a teoria de diagnose de falhas de SEDs, amplamente discutida nas últimas décadas, que possui um conjunto de regras que, ao serem seguidas, levam à identificação e ao diagnóstico das falhas do sistema em questão.

Assim como no Observador (item 2.4), o conjunto de eventos Σ deve ser particionado: $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. Um evento de falha contido no conjunto Σ_f será diagnosticado caso seja possível, por meio de observações de eventos de Σ_o, identificá-lo.
Algumas hipóteses são usualmente feitas nos trabalhos que envolvem diagnose de falhas [15, 16]:

1. $\mathcal{L}(G)$ é "viva", ou seja, $\Gamma(q_i) \neq 0$, $\forall q_i \in Q$.

2. O autômato G não possui nenhum caminho cíclico que contenha somente eventos não-observáveis.

3. O conjunto de eventos de falha é composto por um único evento, isto é, $\Sigma_f = \{\sigma_f\}$.

A primeira hipótese exclui a possibilidade de bloqueio, indicando que o sistema está sempre operando. A segunda visa eliminar a possibilidade do sistema se manter indefinidamente em um ciclo que impeça sua diagnosticabilidade, por não conter eventos observáveis neste ciclo. Por fim, a última hipótese é feita puramente por simplicidade, visando evitar aumento no número de rotulações.

Seja $\mathcal{L}(G) = L$ a linguagem gerada pelo autômato finito G. Todas as sequências de L que não possuem nenhum evento de falha do conjunto Σ_f são representadas pela linguagem prefixo-fechada L_N, isto é, L_N contém todas as sequências normais do sistema. Assim sendo, o comportamento normal de G, em relação à Σ_f, pode ser modelado pelo subautômato de G que gera a linguagem L_N, G_N.

A diagnosticabilidade de uma linguagem é definida da seguinte forma [3]:

Definição 3.1 *Seja L a linguagem prefixo-fechada do sistema e L_N a linguagem prefixo-fechada normal de G, em que G denota um autômato finito construído a partir da modelagem a eventos discretos do sistema. Então L é diagnosticável com relação a $P_o : \Sigma^* \rightarrow \Sigma_o^*$ e Σ_f, se*

$$(\exists n \in \mathbb{N})(\forall s \in L \backslash L_N)(\forall st \in L \backslash L_N, \mid t \mid \geq n) \Rightarrow (\forall w \in P_o^{-1}(P_o(st)) \cap L, w \in L \backslash L_N). \tag{3.1}$$

Portanto, de acordo com a definição 3.1, a linguagem L é dianosticável com relação a P_o e Σ_f se, e somente se, para todas as sequências de comprimento finito contendo o evento de falha, não exista nenhuma outra sequência pertencente à L_N em que as projeções sejam iguais.

3.2 Autômato Diagnosticador

Uma forma de verificar se a ocorrência de uma falha pode, ou não, ser detectada é por meio da construção de um autômato diagnosticador G_d. Este autômato é capaz, quando possível, de inferir sobre a ocorrência de uma determinada falha.

O autômato diagnosticador $G_d = (Q_d, \Sigma_o, \phi_d, \Gamma_d, q_{0,d})$ pode ser construído seguindo-se dois passos:

(i) Construa $G_l = G \parallel A_l$, sendo o autômato A_l indicado na figura 3.1, onde σ_f é o evento de falha, N representa uma rotulação que indica estado com comportamento normal e F uma rotulação que indica comportamento de falha.

(ii) Calcule $G_d = Obs(G_l)$.

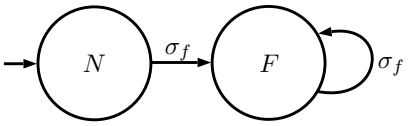


Figura 3.1: Autômato A_l.

Note que a composição paralela realizada no passo (i) apenas insere um rótulo aos estados do autômato G, não afetando sua linguagem. Os estados do autômato G_l são do formato (q, F) ou (q, N), dependendo apenas da ocorrência de σ_f na sequência de eventos que levam do estado inicial até q.

Observação 3.1 *Um estado do diagnosticador é, na verdade, um conjunto de estados formado por G e A_l. Por simplicidade, esse conjunto de estados, que dá origem a um estado q_d qualquer do diagnosticador será dito ser estado de G_d.*

Para entender como a verificação da diagnosticabilidade de falhas, utilizando o autômato diagnosticador, pode ser feita, é preciso que alguns conceitos sejam definidos.

Definição 3.2 *Um estado $q_d \in Q_d$ é denominado estado certo, se, para todo $(q, l) \in q_d$, $l = F$. Caso l seja igual a N para todo $(q, l) \in q_d$, o estado q_d é chamado de normal. Todavia, se existir $(q_1, l), (q_2, \tilde{l}) \in q_d$, em que q_1 não seja necessariamente distinto de q_2, tal que $l = F$ e $\tilde{l} = N$, q_d será denominado incerto de G_d.*

Suponha que q_d seja um estado incerto de G_d. Logo, existem duas sequências $s_1 \in L \setminus \text{Ł}_N$ e $s_2 \in L_N$, tais que $P_o(s_1) = P_o(s_2) = v$ e $\phi_d(q_{0,d}, v) = q_d$. Seja, agora, $q_d = \phi_d(q_{0,d}, v)$ um estado certo de G_d. Então, $\forall w \in (P_o^{-1}(v) \cap L), w \in L \setminus L_N$.

Uma consequência direta das constatações feitas acima é que a linguagem gerada por G será diagnosticável em relação a P_o e Σ_f se, e somente se, após um número limitado de ocorrências de eventos, posteriores à falha, o diagnosticador G_d sempre alcançar um estado certo.

Definição 3.3 *Um conjunto de estados incertos $Q'_d = \{q_{d1}, q_{d2}, ..., q_{dp}\} \subseteq Q_d$ representa um ciclo indeterminado se as condições a seguir forem satisfeitas:*

1. *$\{q_{d1}, q_{d2}, ..., q_{dp}\}$ forma um ciclo em G_d;*
2. *$\exists (q_l^{k_l}, F), (\tilde{q}_l^{r_l}, N) \in q_{d_l}$, sendo $q_l^{k_l}$ não necessariamente distinto de $\tilde{q}_l^{r_l}$, $l = 1, 2, ..., p$, $k_l = 1, 2, ..., m_l$, e $r_l = 1, 2, ..., \tilde{m}_l$ de tal modo que os estados $\{q_l^{k_l}\}$, $l = 1, 2, ..., p$, $k_l = 1, 2, ..., m_l$ e $\{\tilde{q}_l^{r_l}\}$, $l = 1, 2, ..., p$, $r_l = 1, 2, ..., \tilde{m}_l$ podem ser rearranjados para formar ciclos em G.*

Dadas as definições de ciclos indeterminados e de diagnosticabilidade de uma linguagem, é possível afirmar a seguinte condição necessária e suficiente para o diagnóstico de falhas de uma linguagem [2].

Teorema 3.1 *Uma linguagem viva L, gerada por um autômato finito G, será diagnosticável com relação a P_o e Σ_f se, e somente se, o seu diagnosticador G_d não possuir ciclos indeterminados.*

Assim sendo, após a construção do autômato G_d, é necessário que seja realizada uma busca por ciclos indeterminados, visando inferir a diagnosticabilidade da linguagem gerada por G. Em caso de existência de algum ciclo indeterminado, L será não diagnosticável com relação a P_o e Σ_f. Caso contrário, L será diagnosticável.

Exemplo 3.1 *Considere o autômato G da figura 3.2 e suponha que os conjuntos de eventos observáveis e não-observáveis sejam $\Sigma_o = \{a, b\}$ e $\Sigma_{uo} = \{\sigma_u, \sigma_f\}$, respectivamente. Após a realização do passo (i), $G_l = G \parallel A_l$ é obtido como pode ser observado na figura 3.3. O autômato $Obs(G_l)$ é obtido no passo (ii) seguindo o algoritmo 2.1, como pode ser visto na figura 3.4.*

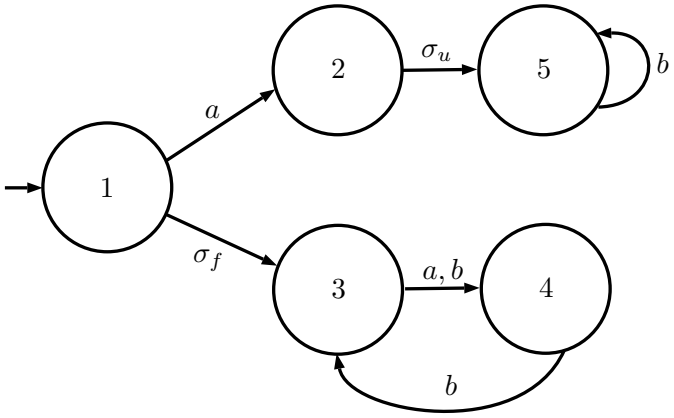


Figura 3.2: Autômato G do Exemplo 3.1.

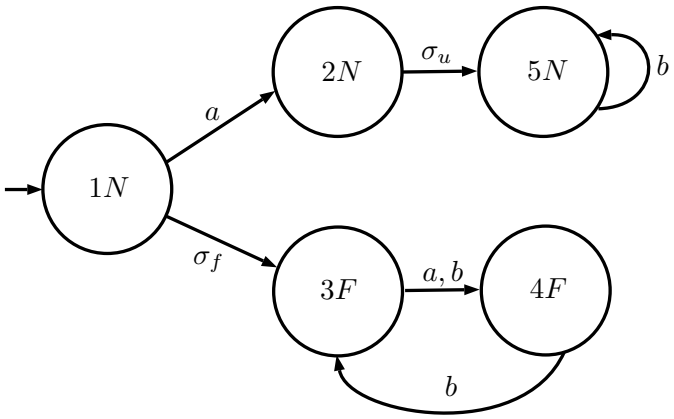


Figura 3.3: Autômato G_l do Exemplo 3.1.

Observação 3.2 *Por simplicidade, estados que tenham o formato (q, N) e (q, F) serão nomeados como qN e qF, respectivamente.*

Note que a sequência ab^n faz com que sempre haja uma incerteza sobre a ocorrência da falha. Esta incerteza só é desfeita quando o evento "a" ocorre. Os estados $\{3F, 5N\}$ e $\{4F, 5N\}$ formam um ciclo indeterminado no diagnosticador. Devido à existência deste ciclo, que possui estados rotulados por N e F, pode-se dizer que o autômato G não é diagnosticável em relação a Σ_f e P_o.

A diagnosticabilidade de falhas utilizando o autômato diagnosticador G_d requer um tempo computacional muito elevado. A construção do diagnosticador possui crescimento exponencial com o número de estados da planta que, somado ao tempo necessário para a realização da busca por ciclos inderterminados, acarreta em uma complexidade pior do que exponencial [25, 26].

Visando melhorar a eficiência computacional, em [13] foi sugerida a construção de um autômato verificador, capaz de verificar a diagnosticabilidade de falhas em tempo polinomial. A construção deste autômato será vista na próxima seção.

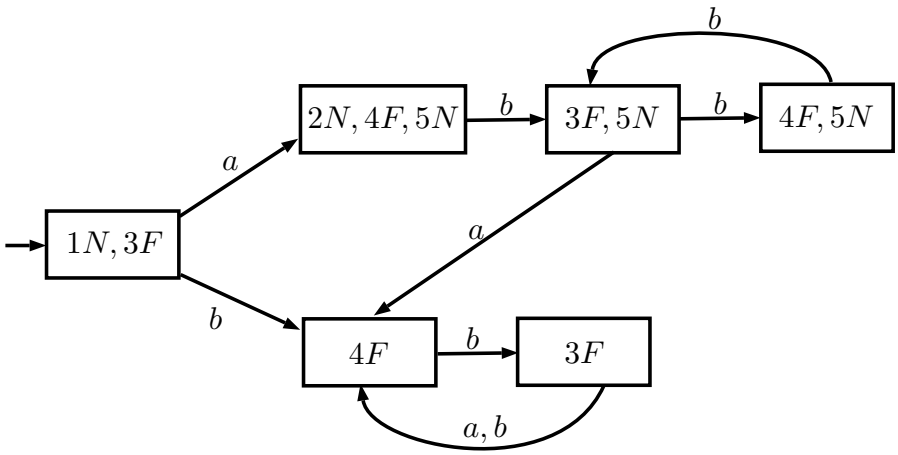


Figura 3.4: Autômato G_d do Exemplo 3.1.

3.3 Autômato Verificador

Conforme mencionado anteriormente, a construção do autômato verificador tem como finalidade reduzir o tempo gasto para a construção de um autômato capaz de verificar a diagnosticabilidade de falhas de uma linguagem.

O autômato verificador G_V é construído seguindo-se o seguinte algoritmo [13]:

Algoritmo 3.1 *(Construção do Verificador)*

- *Passo 1: Construa o autômato G_N, que modela o comportamento normal de G.*
  - *Defina $\Sigma_N = \Sigma \setminus \Sigma_f$.*
  - *Construa o autômato A_N (figura 3.5), responsável por nomear os estados normais. Ele é caracterizado por um único estado, que também é seu estado inicial, e com um autolaço rotulado pelos eventos de Σ_N.*

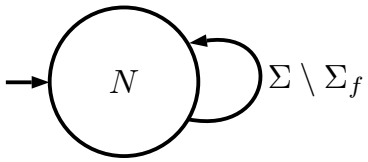


Figura 3.5: Autômato A_N.

  - *Componha o autômato $G_N = G \times A_N = (Q_N, \Sigma, \phi_N, \Gamma_N, q_{0,N})$.*
  - *Redefina o conjunto de eventos de G_N como sendo Σ_N, ou seja, $G_N = (Q_N, \Sigma_N, \phi_N, \Gamma_N, q_{0,N})$.*
- *Passo 2: Calcule o autômato G_F, que representa o comportamento de falha de G.*

  - *Modele um autômato A_l da seguinte forma: um estado inicial, chamado de N, com transição para um segundo estado, nomeado de F, rotulado por eventos de falha (σ_f). Por fim, um autolaço em F também rotulado pelos eventos de falha σ_f é inserido. Este autômato pode ser visto na figura 3.1.*
  - *Faça a composição paralela $G_l = G \parallel A_l$.*
  - *Marque todos os estados de G_l que contenham o rótulo F.*
  - *Obtenha o autômato de falha da seguinte forma: $G_F = CoAc(G_l)$.*

- *Passo 3: Construa o autômato G_{NR}, a partir de G_N, renomeando os eventos não-observáveis deste autômato. A renomeação deve ser dada da seguinte forma:*

$$R(\sigma) = \begin{cases} \sigma & , \text{ se } \sigma \in \Sigma_o, \\ \sigma_R & , \text{ se } \sigma \in \Sigma_{uo}. \end{cases} \tag{3.2}$$

- *Passo 4: Compute o autômato verificador $G_V = G_{NR} \parallel G_F = (Q_V, \Sigma_R \cup \Sigma, \phi_V, q_{0,V})$.*

- *Passo 5: Verifique a existência de um caminho cíclico $cc := (q_V^k, \sigma_k, q_V^{k+1}, ..., q_V^l, \sigma_l, q_V^k)$, em que $l \geq k > 0$, em G_V satisfazendo a seguinte condição:*

$$\exists j \in \{k, k+1, ..., l\} \text{ tal que, para algum } q_V^j, (q_l^j = F) \wedge (\sigma_j \in \Sigma). \tag{3.3}$$

  Se um caminho cíclico for encontrado, a linguagem L não será diagnosticável em relação a P_o e Σ_f. Caso contrário, L será diagnosticável.

Como pode ser visto na construção do verificador, a linguagem gerada pelo autômato G_N possui todas as sequências de eventos de G que não contém eventos de falha. Por outro lado, o autômato G_F possui, em sua linguagem gerada, todas as sequências de eventos de G em que estão contidos eventos de falha de Σ_f, isto é, $\mathcal{L}(G_F) = L \setminus L_N$. Observe que a renomeação de eventos realizada no passo 3 é necessária para, ao realizar a composição paralela de G_N e G_F, somente os eventos observáveis sejam comuns aos dois autômatos, fazendo, assim, com que o verificador represente apenas as sequências de eventos de G_N e G_F que possuam as mesmas projeções.

O seguinte teorema apresenta uma condição necessária e suficiente para a verificação da diagnosticabilidade de falhas utilizando o autômato verificador [13]:

Teorema 3.2 *Sejam L e L_N as linguagens prefixo-fechadas geradas por G e G_N, respectivamente. Considere a projeção $P_o : \Sigma^* \to \Sigma$ e o conjunto de eventos de falha 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, $cc := (q_V^k, \sigma_k, q_V^{k+1}, ..., q_V^l, \sigma_l, q_V^k)$, *em que* $l \geq k > 0$, *satisfazendo a seguinte condição:*

$$\exists j \in \{k, k+1, ..., l\} \text{ tal que, para algum } q_V^j, (q_l^j = F) \wedge (\sigma_j \in \Sigma). \tag{3.4}$$

Observação 3.3 *De acordo com o teorema 3.2, para cada caminho cíclico cc existem duas sequências* $s_N \in L_N$ *e* $s_F = st \in L \setminus L_N$, *em que st possui comprimento arbitrariamente longo após a falha, tais que* $P_o(s_N) = P_o(st)$.

A seguir, um exemplo será proposto para ilustrar a verificação da diagnosticabilidade de falhas utilizando o autômato verificador.

Exemplo 3.2 *Considere o autômato G da figura 3.2. Após a execução do passo 1, o autômato* G_N, *que representa o comportamento normal da planta, é construído. Este autômato pode ser visto na figura 3.6. O próximo passo a ser seguido leva à construção do autômato* G_F, *que descrevem o comportamento de falha de G e pode ser visualizado na figura 3.7. No passo 3 o evento* σ_u *é renomeado para* σ_{uR}, *pois ele é um evento não-observável e não representa uma falha. O autômato* G_V *é computado no passo 4, ao realizar a composição paralela de* G_{NR} *e* G_F, *e pode ser visto na figura 3.8.*

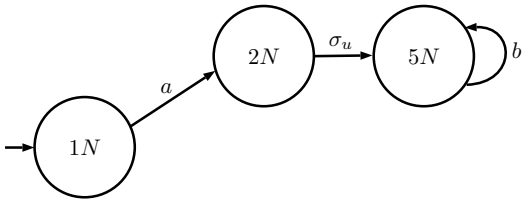


Figura 3.6: Autômato G_N do Exemplo 3.2.

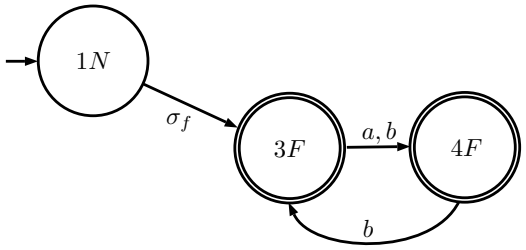


Figura 3.7: Autômato G_F do Exemplo 3.2.

Note que existe um caminho cíclico $cc = (\{5N, 3F\}, b, \{5N, 4F\}, b, \{5N, 3F\})$ *em* G_V *que não satisfaz o teorema 3.2, indicando, assim, que a linguagem L, gerada por G, é não-diagnosticável.*

Como esperado, tanto o diagnosticador quanto o verificador mostraram que a linguagem gerada pelo autômato G é não-diagnosticável, diferindo apenas dos métodos

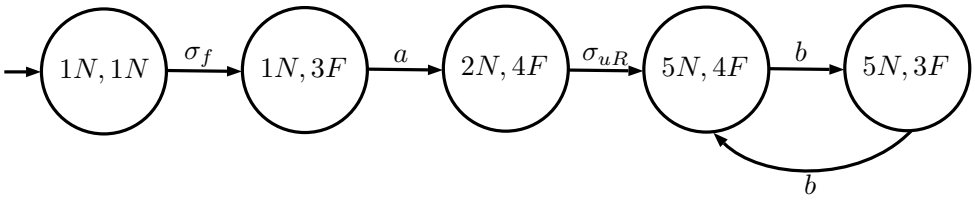


Figura 3.8: Autômato G_V do Exemplo 3.2.

empregados para se chegar a esta conclusão. Como foi dito e pode ser verificado em [13], o tempo de execução do algoritmo de construção do verificador é menor que o de construção do diagnosticador, possuindo ordem de complexidade $O(|Q|^2 \times |\Sigma|)$, polinomial com o número de estados e linear com o número de eventos.

Entretanto, mesmo que a linguagem de um sistema seja diagnosticável, o tempo de atraso até a detecção da falha deve ser levado em consideração. Se esse tempo for elevado demais, a integridade de pessoas ou, até mesmo, do próprio sistema pode ser comprometida. Para esse tipo de linguagem, e, também, para as não-diagnosticáveis, é necessário que um revisão seja feita no modelo do sistema. Um acréscimo de informações ao modelo pode fazer com que a linguagem se torne diagnosticável ou que o atraso até o diagnóstico da falha seja reduzido para um valor aceitável.

Com o objetivo de ser ter mais informações que possam contribuir para uma melhor diagnosticabilidade de falhas, alguns sistemas, quando necessário, passaram a ser modelados como sistemas híbridos, ou seja, parte do sistema é regido por eventos discretos e outra parte por uma dinâmica contínua. A inserção de informações da dinâmica contínua do sistema é capaz de tornar a linguagem do sistema diagnosticável ou reduzir o atraso para o diagnóstico da falha.

O diagnóstico de falhas em sistemas híbridos será abordado no próximo capítulo.

Capítulo 4

Diagnóstico de Falhas em Sistemas Híbridos

Neste capítulo, será apresentada a modelagem de sistemas híbridos e a teoria de diagnosticabilidade destes sistemas. Após isso, será apresentada uma metodologia para se verificar a diagnosticabilidade de falhas apresentada em [29, 32] e uma nova forma de realizar essa verificação será proposta. Essa nova forma de verificação da diagnosticabilidade é uma das constribuições deste trabalho e possui como princípio a construção de um novo autômato verificador. Por fim, uma comparação entre os dois métodos é feita e a complexidade computacional do algoritmo de construção do novo verificador é calculada.

4.1 Sistemas Híbridos

A grande maioria dos sistemas reais podem ser modelados como sistemas híbridos (SH), isto é, sistemas que possuem variáveis discretas e contínuas combinadas [27, 28, 30, 31, 37].

Na área de estudo de diagnóstico de falhas, informações das duas dinâmicas são utilizadas para que, em conjunto, se possa afirmar sobre a ocorrência de um evento de falha do sistema. Por natureza, estados discretos podem mudar de valor através de "saltos"e estados contínuos evoluem no tempo, segundo uma equação diferencial, se o sistema for contínuo no tempo, ou a diferenças finitas, se o sistema for discreto no tempo. Sistemas híbridos possuem estes dois tipos de dinâmica: saltos discretos e evoluções contínuas.

Neste trabalho, sistemas híbridos serão modelados por autômatos híbridos, cuja definição é apresentada a seguir [19, 30].

Definição 4.1 *Um autômato híbrido é uma tupla:*

$$H = (Q, X, Y, \Sigma, U, f, g, \phi, \Gamma, Inv, guard, \rho, q_0, \mathbf{x}_0), \tag{4.1}$$

em que
Q *é o conjunto de estados discretos;*
$X = \mathbb{R}^n$ *é o espaço de estados contínuo;*
$Y \subseteq \mathbb{R}^p$ *é o conjunto de possíveis saídas;*
Σ *é o conjunto finito de eventos;*
$U \subseteq \mathbb{R}^m$ *é o conjunto de controles admissíveis;*
$f : Q \times X \times U \to X$ *é o vetor de campo;*
$g : Q \times X \times U \to Y$ *é a função de saída;*
$\phi : Q \times \Sigma \to Q$ *é a função de transição de estados discretos;*
$\Gamma : Q \to 2^{\Sigma}$ *é a função de eventos ativos;*
$Inv : Q \times X$ *é o conjunto que define uma condição invariante;*
$guard \subseteq Q \times Q \times X$ *é o conjunto que define uma condição de guarda;*
$\rho : Q \times Q \times X \times \Sigma \to X$ *é a função de reset;*
$q_0 \in Q$ *é o estado inicial discreto;*
$\mathbf{x}_0 \in X$ *é o estado inicial contínuo.*

Observação 4.1 *Neste trabalho, a invariância (Inv) e a guarda (guard) não serão utilizados.*

Exemplo 4.1 *Considere uma sala sendo refrigerada por um ar-condicionado controlado por seu termostato. Suponha que quando o compressor esteja ligado, a temperatura,* $t \in \mathbb{R}$*, decresce exponencialmente até* 0 *graus de acordo com a equação diferencial*

$$\dot{t} = -at \tag{4.2}$$

para algum $a > 0$*.*

Quando o termostato desliga o compressor do ar-condicionado, a temperatura aumenta exponencialmente até 15 *graus, segundo a equação diferencial*

$$\dot{t} = -a(t - 15). \tag{4.3}$$

Suponha, também, que a temperatura desejada seja 7 *graus.*

Para impedir que o compressor ligue e desligue com uma frequência muito alta, o termostato só irá ligar (desligar) o compressor quando a temperatura alcançar 8 *(*6*) graus.*

Note que este sistema possui tanto estados contínuos quanto estado discretos. A parte contínua está presente na temperatura $t \in \mathbb{R}$*, em conjunto com as equações*

diferenciais 4.2 e 4.3, e a parte discreta pode ser modelada por um autômato G, que pode ser visto na figura 4.1.

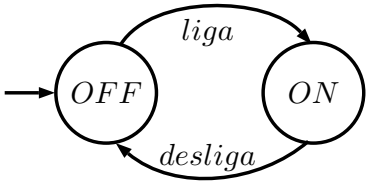


Figura 4.1: Autômato G do Exemplo 4.1.

Quando a temperatura atingir 6 *graus, o evento "desliga" será disparado e o compressor do ar-condicionado será desligado, "saltando"do estado* $\{OFF\}$ *para o estado* $\{ON\}$*. Por outro lado, quando a temperatura alcançar* 8 *graus, o evento "liga" ocorrerá e o compressor ligará, "saltando"do estado* $\{ON\}$ *para o estado* $\{OFF\}$*.*

Um estado $(q_i, \mathbf{x})$ de um sistema híbrido modelado por H é composto de uma parte discreta q_i e uma parte contínua $\mathbf{x}$, chamados, respectivamente, de modo e medida [30, 31, 37]. É possível extrair de um SH um SED, levando em consideração somente a parte dirigida por eventos do SH.

Toda a teoria e definições apresentadas nas seções anteriores são válidas para o SED associado ao SH. O comportamento discreto deste sistema pode ser modelado por um autômato $G = (Q, \Sigma, \phi, \Gamma, q_0)$.

Para a parte de sistema a eventos discretos G, é possível definir uma função que retorna os possíveis estados atuais de G após a ocorrência de uma sequência de eventos observáveis. Essa estimativa é denotada por $Reach(G, \nu)$, em que $\nu = \upsilon\sigma_o = P_o(s)$ é o traço observado pelo autômato G após a execução de um traço $s \in \mathcal{L}(G) = L$ em que o último evento observável é σ_o, e pode ser calculada recursivamente da seguinte forma [34]:

$$Reach(G, \varepsilon) = UR(q_0), \tag{4.4}$$

$$Reach(G, \upsilon\sigma_o) = UR(\Delta(Reach(G, \upsilon), \sigma_o)), \tag{4.5}$$

em que $\Delta(Reach(G, \upsilon), \sigma_o) = \bigcup_{i=1}^{\kappa} \delta(q_i, \sigma_o)$, com $q_i \in Reach(G, \upsilon)$, $\kappa = |Reach(G, \upsilon)|$, e $\delta(q_i, \sigma_o) = \phi(q_i, \sigma_o)$, se $\phi(q_i, \sigma_o)$ é definido, e $\delta(q_i, \sigma_o) = \emptyset$, caso contrário.

Exemplo 4.2 *Considere o autômato G da figura 4.2 e que o seu conjunto de eventos seja* $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$*, em que* $\Sigma_o = \{a, c, f\}$ *e* $\Sigma_{uo} = \{b, d, e\}$*. Suponha um traço* $s = abc$*. A estimativa* $Reach(G, P_o(s))$ *pode ser calculada recursivamente da seguinte forma:*

$Reach(G, \varepsilon) = UR(q_o) = \{1\}$*;*

$Reach(G, \varepsilon a) = UR(\Delta(Reach(G, \varepsilon), a))$;
$\Delta(Reach(G, \varepsilon), a) = \bigcup_{i=1}^{1} \delta(\{1\}, a) = \phi(\{1\}, a) = \{2\}$;
$Reach(G, a) = UR(\Delta(Reach(G, \varepsilon), a)) = UR(\{2\}) = \{\{2\}, \{3\}\}$;
$Reach(G, ac) = UR(\Delta(Reach(G, a), c))$;
$\Delta(Reach(G, a), c)) = \bigcup_{i=1}^{2} \delta(q_i, c) = \delta(\{2\}, c) \cup \delta(\{3\}, c) = \emptyset \cup \phi(\{3\}, c) = \{4\}$;
$Reach(G, ac) = UR(\Delta(Reach(G, a), c)) = UR(\{4\}) = \{\{4\}, \{5\}, \{6\}\}$

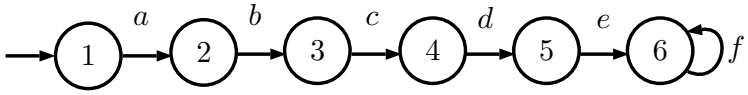


Figura 4.2: Autômato G do Exemplo 4.2.

Um caminho em G é representado por uma sequência $\mathcal{P} = (q_{p1}, \sigma_{p_1}, q_{p_2}, ..., \sigma_{p_{n-1}}, q_{p_n})$, em que $\sigma_{p_i} \in \Sigma$ e $\sigma_{p_{i+1}} = \phi(q_{p_i}, \sigma_{p_i})$, $i = 1, 2, ..., n-1$.

Denote G_P como sendo um autômato de estados finitos formado apenas pelo caminho $\mathcal{P}$, tal que $G_P = (Q_P, \Sigma_P, \phi_P, \Gamma_P, q_{p_0})$, em que Q_P é o conjunto de estados de $\mathcal{P}$, Σ_P o conjunto formado pelos eventos de $\mathcal{P}$, ϕ_P a função de transição definida por $\mathcal{P}$, Γ_P a função de eventos ativos e q_{0_P} o estado inicial de $\mathcal{P}$.

Exemplo 4.3 *Considere o autômato G da figura 4.3. Três exemplos de autômatos associados a caminhos de G estão representados na figura 4.4: G_{P_1} associado a $\mathcal{P}_1 = (1, a, 2, b, 3, a, 4)$ (a), G_{P_2} associado a $\mathcal{P}_2 = (1, b, 5, a, 6, b, 7)$ (b) e G_{P_3} associado a $\mathcal{P}_3 = (1, b, 5, a, 6, a, 8)$ (c).*

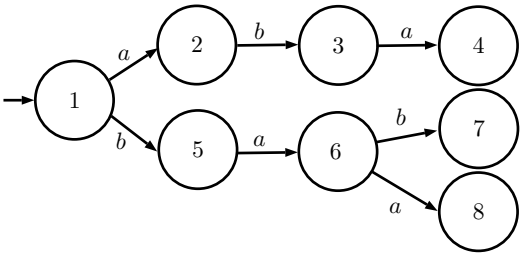


Figura 4.3: Autômato G do Exemplo 4.3.

A dinâmica contínua de um SH pode ser descrita por um sistema multimodo $M = (Q, X, Y, U, f, g, \mathbf{x}_0)$ [28]. Neste trabalho, os sistemas são lineares e invariantes no tempo, com representação por espaço de estados e é suposto que o vetor de campo $f(q_i, \mathbf{x}, \mathbf{u})$ e a função de saída $g(q_i, \mathbf{x}, \mathbf{u})$ são dados por modelos contínuos associados a cada modo q_i, conforme:

$$\begin{cases} \mathbf{x}(n+1) = \mathbf{A}_i\mathbf{x}(n) + \mathbf{B}_i\mathbf{u}(n) \\ \mathbf{y}(n) = \mathbf{C}_i\mathbf{x}(n) + \mathbf{D}_i\mathbf{u}(n) \end{cases}, \tag{4.6}$$

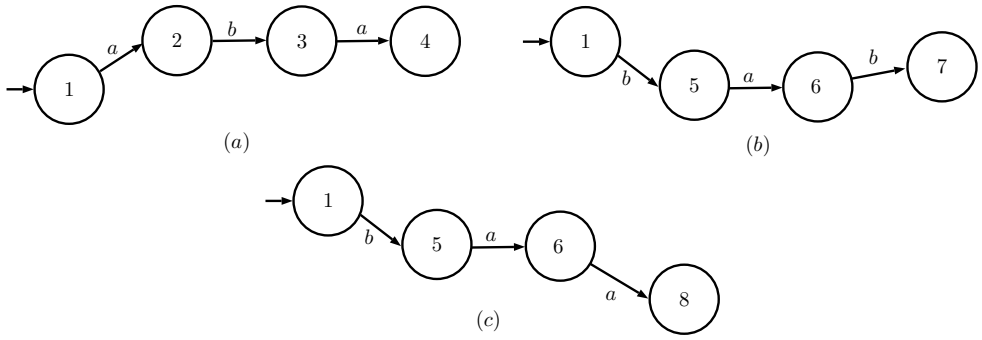


Figura 4.4: Autômatos G_{P_1}, G_{P_2} e G_{P_3} do Exemplo 4.3.

em que $\mathbf{x}(n)$ é o vetor de estado em um instante de tempo nT_s, $\mathbf{u}(n)$ é o vetor de entrada em um tempo nT_s, $\mathbf{y}(n)$ é o vetor de saída no tempo nT_s, T_s é o período de amostragem e $\mathbf{A}_i$, $\mathbf{B}_i$, $\mathbf{C}_i$ e $\mathbf{D}_i$ são matrizes constantes.

Neste trabalho é necessário apresentar a seguinte definição:

Definição 4.2 *Sejam q_i e q_j dois estados discretos diferentes de um sistema híbrido H, cujos modelos contínuos são representados por $\mathcal{M}_i$ e $\mathcal{M}_j$, respectivamente. Então, os modos q_i e q_j são ditos serem distinguíveis com respeito a uma medida ψ e tolerância μ, se $\psi(\mathcal{M}_i, \mathcal{M}_j, \mathbf{u}, \mathbf{x}_0) > \mu$, em que $\mathbf{u}$ representa a entrada definida sobre um intervalo de tempo I e $\mathbf{x}_0$ a condição inicial dos modelos. Caso contrário, os estados q_i e q_j são ditos serem não-distinguíveis.*

A medida ψ representa um cálculo efetuado a partir dos modelos de estados contínuos utilizados como argumentos da função. Seu objetivo é verificar a consistência das equações matemáticas dos estados q_i e q_j. Todavia, como modelos são imprecisos e as medidas dos estados estão sujeitas à ruídos, é preciso que seja definida uma tolerância μ para esta comparação.

Como a verificação da diagnosticabilidade é feita de forma *offline*, é pressuposto que a distinguibilidade entre os estados de um sistema já tenha sido calculada previamente. Métodos para se verificar a consistência das equações matemáticas dos estados do sistemas podem ser vistos em [29, 32, 38].

Neste trabalho, estados distinguíveis serão denotados por $q_i \nsim q_j$ e estados não-distinguíveis por $q_i \sim q_j$.

4.2 *h*-Diagnosticabilidade

Em [29], a diagnosticabilidade de um SH, chamada h-diagnosticabilidade, é definida da seguinte forma:

Definição 4.3 *Seja L a linguagem gerada pelo sistema a evento discreto associado a um SH e $L_N \subset L$ a linguagem normal de G. O sistema híbrido H é dito ser h-diagnosticável se*

$$(\exists n \in \mathbb{N})(\forall s \in L \setminus L_N)(\forall st \in L \setminus L_N, |t| \geq n) \Rightarrow D_1 \vee D_2, \tag{4.7}$$

em que D_1 é dado por

$$(\forall w \in P_o^{-1}(P_o(st)) \cap L, w \in L \setminus L_N) \tag{4.8}$$

e D_2 por

$$(\forall s_N \in L_N, P_o(s_N) = P_o(st) = \nu)(\exists \eta\sigma_o \in \overline{\nu}, q_F \nsim q_N), \tag{4.9}$$

em que $q_F \in \Delta(Reach(G_{P_F}, \eta), \sigma_o)$ (resp. $q_N \in \Delta(Reach(G_{P_N}, \eta), \sigma_o)$), e G_{P_F} (resp. G_{P_N}) é o subautômato de G associado ao caminho $\mathcal{P}_F$ (resp. $\mathcal{P}_N$), cujo traço é st (resp. s_N). $\mathcal{P}$ representa os caminhos associados a G, sejam eles normais ($\mathcal{P}_N$) ou de falha ($\mathcal{P}_F$), e G_{P_N} e G_{P_F}, os autômatos de estados finitos formados a partir desses caminhos, respectivamente.

De acordo com a definição 4.3, é possível perceber que L será h-diagnosticável se for possível distinguir, após a ocorrência de um número finito de eventos, todos os caminhos normais dos caminhos de falha. Para tal, existem duas maneiras de se realizar essa distinção:(i) as projeções das sequências de eventos associadas aos caminhos são diferentes (condição D_1), (ii) os estados alcançados, em ambos os caminhos, a partir da ocorrência de um evento observável são distinguíveis (condição D_2).

Observe que se para todo $st \in L \setminus L_N$ a condição D_1 for atendida, a linguagem do sistema a eventos discretos associado ao sistema híbrido será diagnosticável segundo a definição 3.1, fazendo com que o SH seja h-diagnosticável. Entretanto, se para algum $st \in L \setminus L_N$ a condição D_1 for falsa, é possível que, após a ocorrência de um evento observável, os estados alcançados após uma sequência de falha sejam distinguíveis dos estados alcançados após uma sequência normal, atendendo, desta forma, à condição D_2.

Para que seja possível utilizar as informações da dinâmica contínua, é preciso que seja definido um momento em que a distinção entre os estados tenha seu cálculo iniciado. Como o autômato que modela o sistema possuiu eventos observáveis e não-observáveis, o momento ideal para que o cálculo da distinguibilidade ocorra é depois da ocorrência de um evento observável.

Deve ser suposto, assim como em todos os demais trabalhos que envolvem diagnóstico de falhas em sistemas híbridos, que, uma vez iniciado o cálculo da distinguibilidade entre os estados estimados, nenhum evento, seja ele observável ou não,

irá ocorrer até que o cálculo seja finalizado. Caso essa suposição não fosse feita, o estado do sistema poderia mudar durante o cálculo da distinguibilidade, acarretando em um resultado incorreto, afetando a diagnosticabilidade do sistema.

Nas próximas seções serão apresentados dois métodos para a verificação da h-diagnosticabilidade de um sistema híbrido.

4.3 Autômato Diagnosticador

Visando realizar a diagnosticabilidade de um SH, é preciso integrar a dinâmica associada às variáveis contínuas à construção do diagnosticador. Em [29, 32] é proposto um algoritmo para construção do autômato G_{CD}, que representa o autômato diagnosticador do SED associado ao SH acrescido das informações da dinâmica associada às variáveis contínuas do sistema e infere sobre a diagnosticabilidade de G.

Antes de apresentar o algoritmo, é necessário fazer algumas definições.

Definição 4.4 *Uma sequência de estados ou conjunto de estados de um autômato G é dada por $h = (\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_n)$, em que $\mathbf{q}_i$ são subconjuntos de Q, isto é, $\mathbf{q}_i \subseteq Q$.*

Definição 4.5 *A sequência de estados vazia é denotada por λ.*

Da mesma forma que foi definido para sequência de eventos, o número de elementos $\mathbf{q}_i$ de uma sequência de estados ou conjunto de estados h é dado por $|h|$. Para a sequência vazia tem-se que $|h| = 0$.

Definição 4.6 *A concatenação de duas sequências de estados h_1 e h_2, denotada por $h_1 \cdot h_2$, possui como resultado uma sequência $h = h_1 h_2$, cujo comprimento é $|h| = |h_1| + |h_2|$, em que os $|h_1|$ primeiros elementos são provenientes do primeiro conjunto e estes serão imediatamente seguidos dos $|h_2|$ elementos do segundo.*

Exemplo 4.4 *Sejam duas sequências $h_1 = (\{1\}, \{2, 4\}, \{3\})$, com $|h_1| = 3$, e $h_2 = (\{3, 1\}, \{5\})$, com $|h_2| = 2$. A concatenação de h_1 com h_2 dá origem à sequência $h_3 = h_1 \cdot h_2 = (\{1\}, \{2, 4\}, \{3\}, \{3, 1\}, \{5\})$, com $|h_3| = 5$.*

A função a seguir possui como objetivo separar estados distinguíveis de um conjunto de estados de G e será utilizado para a construção do autômato G_{CD} [32]. É importante ressaltar que a distinguibilidade dos estados é supostamente conhecida.

Função 4.1 $[h_C, s_C] = Distinguish(H_{cl})$

- *Passo 1: Crie uma sequência de estados h com todos os elementos de H_{cl}, sem repetições. Faça $h_C = \lambda$ e $s_C = \varepsilon$.*

- *Passo 2: Enquanto $h \neq \lambda$ faça*
  - *Passo 2.1: $\tilde{H} = \emptyset$.*
  - *Passo 2.2: Seja h_j o j-ésimo elemento da sequência h. Para $j = 1$ até $|h|$ faça*
    - *Passo 2.2.1: Se $h_1 \sim h_j$, então $\tilde{H} \leftarrow \tilde{H} \cup \{h_j\}$.*
  - *Passo 2.3: Se $\tilde{H} \neq H_{cl}$:*
    - *Passo 2.3.1: Crie um evento cl_{h_1}.*
    - *Passo 2.3.2: $s_C \leftarrow s_C \cdot cl_{h_1}$.*
    - *Passo 2.3.3: $h_C \leftarrow h_C \cdot \tilde{H}$.*
  - *Passo 2.4: Construa uma nova sequência h removendo de h os estados de $\tilde{H}$.*

Antes de apresentar o algoritmo de construção do diagnosticador híbrido G_{CD} é necessário definir o autômato rotulador $A_\ell^{HT} = (\{H, T\}, \Sigma, \phi_\ell^{HT}, \Gamma_\ell^{HT}, \{H\})$, em que $\phi_\ell^{HT}(H, \sigma) = \phi_\ell^{HT}(T, \sigma) = H$, se $\sigma \in \Sigma_o$, e $\phi_\ell^{HT}(H, \sigma) = \phi_\ell^{HT}(T, \sigma) = T$ se $\sigma \in \Sigma_{uo}$; $\Gamma_\ell^{HT}(T) = \Gamma_\ell^{HT}(H) = \Sigma$. Esse autômato pode ser visualizado na figura 4.5.

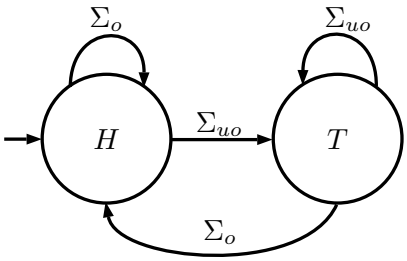


Figura 4.5: Autômato A_ℓ^{HT}.

Conforme foi dito, o gatilho para que o cálculo da distinguibilidade inicie é a ocorrência de um evento observável. Assim sendo, foi proposto o autômato A_ℓ^{HT} que, após a realização de uma composição desse autômato com a planta G, é inserido, em G, uma rotulação nos estados alcançados após a ocorrência de um evento observável com "H" e outra rotulação em estados alcançados após a ocorrência de eventos não-observáveis com "T". Estados rotulados por "H" irão servir de gatilho para o início do cálculo da distinguibilidade.

Neste trabalho, estados rotulados por "H" são chamados de estados cabeça e estados rotulados por "T" são chamados de estados cauda.

A construção do autômato $G_{CD} = (Q_{CD}, \Sigma_{CD}, \phi_{CD}, \Gamma_{CD}, q_{0_{CD}})$ é feita de acordo com o seguinte algoritmo [32].

Algoritmo 4.1

- *Passo 1: Calcule G_ℓ tal que*
  - *Passo 1.1:* $G_{\ell_1} = G \| A_\ell^{HT} = (Q_{\ell_1}, \Sigma, \Gamma_{\ell_1}, \phi_{\ell_1}, q_{\ell_{1,0}})$
  - *Passo 1.2:* $G_\ell = G_{\ell_1} \| A_l = (Q_\ell, \Sigma, \Gamma_\ell, \phi_\ell, q_{\ell_0})$
- *Passo 2: Defina* $q_{0_{CD}} = UR(q_{\ell_0})$, $\Sigma_{CD} = \Sigma_o$ *e* $Q_{CD} = \emptyset$
- *Passo 3: Defina* $\tilde{Q} = \{q_{0_{CD}}\}$
- *Passo 4:* $Q_{CD} \leftarrow Q_{CD} \cup \tilde{Q}$
- *Passo 5: Defina* $\hat{Q} = \tilde{Q}$ *e* $\tilde{Q} = \emptyset$
- *Passo 6: Para cada* $B \in \hat{Q}$*:*
  - *Passo 6.1:* $\Gamma_{CD}(B) = (\bigcup_{q_\ell \in B} \Gamma_\ell(q_\ell)) \cap \Sigma_o$
  - *Passo 6.2: Para cada* $e \in \Gamma_{CD}(B)$*:*
    - *Passo 6.2.1:* $H_{cl} = \{q_\ell \in Q_\ell : (\exists \tilde{q} \in B)[q_\ell = \phi_\ell(\tilde{q}, e)]\}$
    - *Passo 6.2.2:* $\phi_{CD}(B, e) = UR(H_{cl})$ *e* $\bar{Q} = UR(H_{cl})$
    - *Passo 6.2.3:* $Q_{CD} \leftarrow Q_{CD} \cup \{\phi_{CD}(B, e)\}$
    - *Passo 6.2.4: Calcule* $[h_C, s_C] = Distinguish(H_{cl})$
    - *Passo 6.2.5: Se* $|h_C| = 1$*, então* $\tilde{Q} \leftarrow \tilde{Q} \cup \{\phi_{CD}(B, e)\}$.
    - *Passo 6.2.6: Se* $|h_C| > 1$*, então*
      - *Passo 6.2.6.1:* $\Sigma_C = \bigcup_{i=1}^{|s_C|} \{cl_i\}$*, em que* cl_i *representa o i-ésimo elemento de* s_C.
      - *Passo 6.2.6.2: Defina* $\Sigma_{CD} \leftarrow \Sigma_{CD} \cup \Sigma_C$.
      - *Passo 6.2.6.3: Para* $i = 1$ *até* $|h_C|$ *faça*
        - *Passo 6.2.6.3.1:* $Q_{CD} \leftarrow Q_{CD} \cup \{UR(H_i)\}$*, em que* H_i *representa o i-ésimo elemento da sequência* h_C.
        - *Passo 6.2.6.3.2:* $\phi_{CD}(\bar{Q}, cl_i) = UR(H_i)$.
        - *Passo 6.2.6.3.3:* $\tilde{Q} \leftarrow \tilde{Q} \cup \{\phi_{CD}(\bar{Q}, cl_i)\}$.
      - *Passo 6.2.6.4:* $\Gamma_{CD}(\bar{Q}) = \Sigma_C$.
- *Passo 7: Repita os passos 5 e 6 até que seja finalizada a construção de* $Ac(G_{CD})$.

A ideia por trás do algoritmo 4.1 consiste em verificar a distinguibilidade de estados cabeças (*heads*), isto é, rotulados por "H", em que as projeções se confundem enquanto é feita a construção do diagnosticador. Quando a distinção entre estados puder ser realizada, eventos cl (*clusters*) são inseridos, indicando que, através da

análise das dinâmicas contínuas, o comportamento desses estados são diferenciados. Esses eventos *cl* representam as informações adicionais inseridas no modelo a eventos discretos do SH necessárias para tentar tornar a linguagem diagnosticável ou reduzir o tempo de atraso entre a ocorrência e a detecção da falha.

A diagnosticabilidade é verificada de forma análoga ao caso puramente discreto, através da busca por ciclos indeterminados no autômato G_{CD}.

Antes de apresentar uma condição necessária e suficiente para a h-diagnosticabilidade de um SH, é preciso que algumas definições sejam feitas.

Definição 4.7 *Seja* $q_{CD} = ((q_1, \ell_{1_1}, \ell_{2_1}), (q_2, \ell_{1_2}, \ell_{2_2}), \cdots, (q_n, \ell_{1_n}, \ell_{2_n})) \in Q_{CD}$, *em que* $q_k \in Q$, $\ell_{1_k} \in \{H, T\}$ *e* $\ell_{2_k} \in \{F, N\}$, *para* $k = 1, 2, \cdots, n$. *Então,* q_{CD} *é dito ser um estado discreto de falha, se* $\ell_{2_k} = F$ *para* $k = 1, 2, \ldots, n$, *e um estado discreto normal se* $\ell_{2_k} = N$ *para* $k = 1, 2, \ldots, n$. *Se existir* $(q_i, \ell_{1_i}, \ell_{2_i}), (q_j, \ell_{1_j}, \ell_{2_j}) \in q_{CD}$, (q_i, ℓ_{1_i}) *não necessariamente distinto de* (q_j, ℓ_{1_j}), *tal que* $\ell_{2_i} = F$ *and* $\ell_{2_j} = N$, *então* q_{CD} *é um estado discreto incerto de* G_{CD}.

Definição 4.8 *Um conjunto de estados discretos incertos* $\{q_{CD_1}, q_{CD_2}, \ldots, q_{CD_n}\} \subseteq Q_{CD}$ *forma um ciclo indeterminado no diagnosticador* G_{CD}, *se as seguintes condições forem verdadeiras:*

i) existe um caminho cíclico $\mathcal{P}_{CD} = (q_{CD_1}, \sigma_1, q_{CD_2}, \sigma_2, \ldots, q_{CD_n}, \sigma_n, q_{CD_1})$ *em* G_{CD};

ii) $\exists (q_l^{k_l}, \ell_{1_l}^{k_l}, \ell_{2_l}^{k_l}), (\tilde{q}_l^{r_l}, \tilde{\ell}_{1_l}^{r_l}, \tilde{\ell}_{2_l}^{r_l}) \in q_{CD_l}$, $(q_l^{k_l}, \ell_{1_l}^{k_l})$ *não necessariamente distinto de* $(\tilde{q}_l^{r_l}, \tilde{\ell}_{1_l}^{r_l})$, $l = 1, 2, \ldots, n$, $k_l = 1, 2, \ldots, m_l$, *e* $r_l = 1, 2, \ldots, \tilde{m}_l$ *tal que*

a) $\ell_{2_l}^{k_l} = F$, $\tilde{\ell}_{2_l}^{r_l} = N$, *para todo* l, k *e* r;

b) Os estados $\{(q_l^{k_l}, \ell_{1_l}^{k_l}, \ell_{2_l}^{k_l})\}$, $l = 1, 2, \ldots, n$, $k_l = 1, 2, \ldots, m_l$ *e* $\{(\tilde{q}_l^{r_l}, \tilde{\ell}_{1_l}^{r_l}, \tilde{\ell}_{2_l}^{r_l})\}$, $l = 1, 2, \ldots, n$, $r_l = 1, 2, \ldots, \tilde{m}_l$ *podem ser reorganizados para formar um caminho cíclico em* G_ℓ, *tal que os traços correspondentes* s *e* $\tilde{s}$ *satisfaçam* $P_o(s) = P_o(\tilde{s}) = P(\bar{s})$, *em que* $P : \Sigma_{CD}^\star \to \Sigma^\star$ *e* $\bar{s} = \sigma_1 \sigma_2 \ldots \sigma_n$, *em que* σ_1, σ_2, $\ldots$ σ_n *são definidos em i).*

Uma condição necessária e suficiente para a h-diagnosticabilidade de sistemas híbridos é estabelecida pelo seguinte teorema [29, 32].

Teorema 4.1 *Seja* G_{CD} *o diagnosticador do sistema híbrido* H. *Então,* H *é* h-*diagnosticável se, e somente se, o diagnosticador* G_{CD} *não tiver ciclos indeterminados.*

Exemplo 4.5 *Considere o autômato* G *da figura 4.6 como sendo o modelo da parte discreta de um SH e que o conjunto de eventos observáveis seja* $\Sigma_o = \{b, c\}$. *Após a execução do passo 1 do algoritmo 4.1, o autômato* G_ℓ *pode ser visto na figura 4.7.*

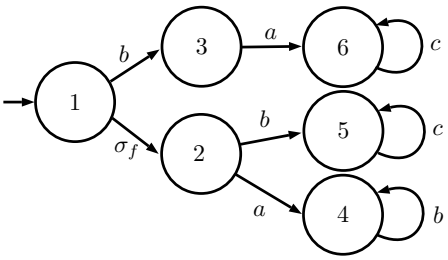


Figura 4.6: Autômato G do exemplo 4.5.

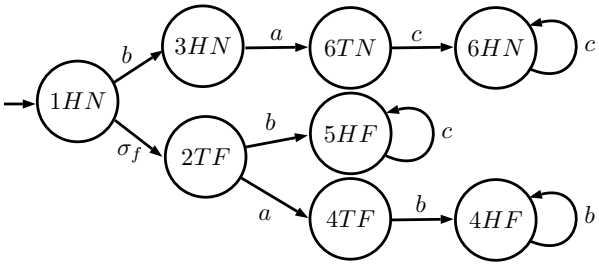


Figura 4.7: Autômato G_ℓ do exemplo 4.5.

Suponha, inicialmente, que nenhum dos estados cabeças sejam distinguíveis. Então a linguagem desse sistema é não-diagnosticável por haver um ciclo indeterminado $(\{5HF, 6HN\}, c, \{5HF, 6HN\})$ *em* G_{CD} *(figura 4.8).*

Suponha agora que os estados $\{5\}$ *e* $\{6\}$ *sejam distinguíveis por suas dinâmicas contínuas. Isso faria com que eventos cl fossem inseridos no diagnosticador e o ciclo que tornaria a linguagem não-diagnosticável deixaria de existir. Esse novo autômato* G_{CD} *pode ser visto na figura 4.9.*

Apesar de diagnosticar corretamente a ocorrência de uma falha, o diagnosticador proposto em [32] necessita de tempo computacional elevado para a construção do autômato G_{CD} e para a busca por ciclos indeterminados em G_{CD}. Visando reduzir este tempo, na próxima seção será proposto um novo método para a verificação da h-diagnosticabilidade, através da construção de um autômato verificador G_{VH}.

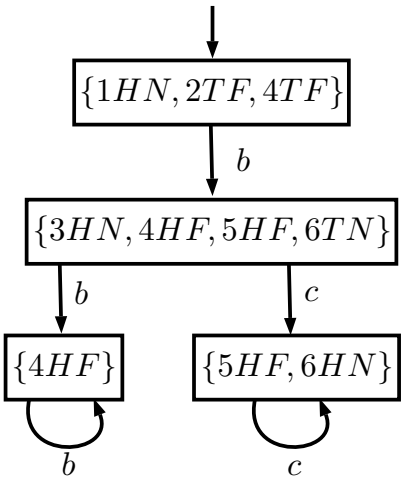


Figura 4.8: Autômato G_{CD} sem distinguibilidade do exemplo 4.5.

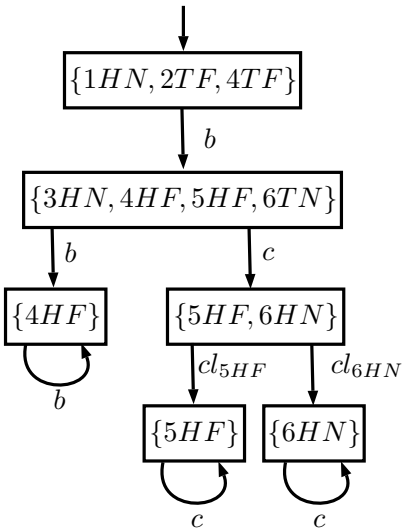


Figura 4.9: Autômato G_{CD} com distinguibilidade do exemplo 4.5.

4.4 Autômato Verificador

Uma contribuição deste trabalho é a elaboração do algoritmo para construção do autômato verificador G_{VH}, capaz de verificar a diagnosticabilidade de SH em tempo de execução polinomial. Para a construção deste autômato, foi utilizado como base o verificador para SEDs proposto em [13].

O algoritmo proposto em [13] consiste em construir um autômato que representa os comportamentos normal e de falha do sistema enquanto as projeções das sequências de eventos dos dois comportamentos forem iguais. Caso o verificador alcance um estado de bloqueio, isso significa que, a partir daqueles dois estados representados (um normal e um de falha), não há mais ambiguidade entre as duas sequências, ou seja, as projeções se tornam distintas.

Visando incorporar a distinguibilidade ao processo de verificação da diagnosticabilidade, os estados do sistema possuirão a rotulação H e T introduzida pelo autômato A_ℓ^{HT}. Assim como no algoritmo 4.1 proposto em [32], estados cabeças passarão pelo processo de distinguibilidade e influenciarão a construção do autômato verificador G_{VH}, podendo contribuir com a verificação da diagnosticabilidade do sistema.

Antes de apresentar o algoritmo de construção do autômato verificador, é preciso que a seguinte função seja definida.

Função 4.2 $[G_{VH}] = paralleld(G_{NR}, G_F)$

- *Passo 1: Defina* $q_{0,VH} = (q_{0,N}, q_{0,F})$ *e faça* $Q_{VH} = \{q_{0,VH}\}$ *e* $\tilde{Q}_{VH} = Q_{VH}$
- *Passo 2: Faça* $\hat{Q}_{VH} \leftarrow \tilde{Q}_{VH}$ *e* $\tilde{Q}_{VH} \leftarrow \emptyset$
- *Passo 3: Para cada* $B = (q_N, q_F) \in \hat{Q}_{VH}$
  - *Passo 3.1 Se* $q_N \sim q_F$ *ou se* q_N *ou* q_F *são estados cauda:*
    - *Passo 3.1.1:* $\Gamma_{VH}(B) = [\Gamma_N(q_N) \cap \Gamma_F(q_F)] \cup [\Gamma_N(q_N) \setminus \Sigma] \cup [\Gamma_F(q_F) \setminus \Sigma_R]$.
    - *Passo 3.1.2: Para cada* $e \in \Gamma_{VH}(B)$*, faça:*
      - *Passo 3.1.2.1:*

$$\phi_{VH}(B,e) = \begin{cases} (\phi_N(q_N,e), \phi_F(q_F,e)), & \text{se } e \in \Gamma_N(q_N) \cap \Gamma_F(q_F) \\ (\phi_N(q_N,e), q_F), & \text{se } e \in \Gamma_N(q_N) \setminus \Sigma \\ (q_N, \phi_F(q_F,e)), & \text{se } e \in \Gamma_F(q_F) \setminus \Sigma_R \\ \text{Não definida}, & \text{caso contrário.} \end{cases}$$

      - *Passo 3.1.2.2:* $\tilde{Q}_{VH} \leftarrow \tilde{Q}_{VH} \cup \phi_{VH}(B,e)$
- *Passo 4:* $Q_{VH} \leftarrow Q_{VH} \cup \tilde{Q}_{VH}$

- *Passo 5: Repita os passos 2 a 4 até que toda parte acessível de G_{VH} tenha sido obtida.*

Observação 4.2 *A função 4.2 possui como objetivo impedir que estados distinguíveis tenham sua função ϕ construída. Caso contrário, ou seja, caso nenhum estado seja distinguível dos demais, esse algoritmo realiza a mesma operação que a operação de composição paralela entre dois autômatos.*

A construção do autômato G_{VH} e sua verificação de diagnosticabilidade é dada da seguinte forma:

Algoritmo 4.2 *(Cálculo do autômato G_{VH})*

- *Passo 1: Obtenha $G_h = G \times A_\ell^{HT}$, em que A_ℓ^{HT} é o autômato apresentado na figura 4.5.*
- *Passo 2: Compute o autômato G_N, que modela o comportamento normal de G_h.*
  - *Passo 2.1: Defina $\Sigma_N = \Sigma \setminus \Sigma_f$.*
  - *Passo 2.2: Construa o autômato A_N (figura 3.5)*
  - *Passo 2.3: Construa o autômato $G_N = G_h \times A_N = (Q_N, \Sigma, \phi_N, \Gamma_N, q_{0,N})$.*
  - *Passo 2.4: Defina o conjunto de eventos de G_N como sendo Σ_N, ou seja, $G_N = (Q_N, \Sigma_N, \phi_N, \Gamma_N, q_{0,N})$.*
- *Passo 3: Obtenha o autômato G_F, que representa o comportamento de falha do sistema.*
  - *Passo 3.1: Construa um autômato A_l, representado na figura 3.1.*
  - *Passo 3.2: Faça $G_l = G_h \parallel A_l$.*
  - *Passo 3.3: Marque todos os estados de G_l que contenham o rótulo F.*
  - *Passo 3.4: $G_F = CoAc(G_l) = (Q_F, \Sigma, \phi_F, \Gamma_F, q_{0,F})$.*
- *Passo 4: Construa o autômato $G_{NR} = (Q_N, \Sigma_R, \phi_N, \Gamma_N, q_{0,N})$, a partir de G_N, renomeando os eventos não-observáveis de G_N. A renomeação deve ser realizada da seguinte forma:*

$$R(\sigma) = \begin{cases} \sigma & , \text{ se } \sigma \in \Sigma_o, \\ \sigma_R & , \text{ se } \sigma \in \Sigma_{uo}. \end{cases} \tag{4.10}$$

- *Passo 5: Calcule $G_{VH} = paralleld(G_{NR}, G_F)$.*

- *Passo 6: Procure pela existência de um caminho cíclico em G_{VH}*

  $$cc := (q_V^k, \sigma_k, q_V^{k+1}, ..., q_V^m, \sigma_m, q_V^k),$$

  em que $m \geq k > 0$, satisfazendo a seguinte condição:

  $$\exists j \in \{k, k+1, ..., m\} \text{ tal que para algum } q_V^j, \ (q_F^j = (q, l_1, F)) \wedge (\sigma_j \in \Sigma),$$

  em que $q \in Q$ e $l_1 \in \{H, T\}$.

  Se não existir um caminho cíclico, então o sistema é dito ser h-diagnosticável. Caso contrário, o sistema não é h-diagnosticável.

Observação 4.3 *Caso nenhum estado seja distinguível dos demais, o verificador G_{VH} funciona da mesma forma que o verificador puramente discreto, uma vez que a função 4.2 irá construir o paralelo entre os autômatos G_{NR} e G_F sem impedir nenhuma transição de ser construída.*

Neste trabalho, caminhos cíclicos de G_{VH} que satisfaçam a condição 4.11, são chamados de caminhos cíclicos ambíguos [39].

Uma condição necessária e suficiente para a h-diagnosticabilidade de um sistema híbrido é apresentada a seguir.

Teorema 4.2 *Seja G_{VH} o autômato verificador de um sistema híbrido H. Então H é h-diagnosticável se, e somente se, não existe em G_{VH} um caminho cíclico*

$$cc := (q_V^k, \sigma_k, q_V^{k+1}, ..., q_V^m, \sigma_m, q_V^k), \tag{4.11}$$

em que $m \geq k > 0$, satisfazendo a seguinte condição:

$$\exists j \in \{k, k+1, ..., m\} \text{ tal que para algum } q_V^j, (q_F^j = (q, l_1, F)) \wedge (\sigma_j \in \Sigma), \tag{4.12}$$

em que $q \in Q$ e $l_1 \in \{H, T\}$.

Prova Suponha que exista em G_{VH} um caminho cíclico ambíguo e que H seja h-diagnosticável. Se G_{VH} possui um caminho cíclico ambíguo, então:

- Note que todos os caminhos que existem em G_{VH} também existem no autômato $G_V = G_{NR} \parallel G_F$, obtido sem considerar a distinguibilidade de estados, uma vez que G_{VH} é obtido por meio da composição de G_{NR} e G_F, assim como na composição paralela, eliminando-se caminhos que não se confundem pela distinguibilidade de estados. Assim, se G_{VH} possui um caminho cíclico ambíguo então G_V possui o mesmo caminho cíclico, o que, de acordo

com o teorema 3.2 e a observação 3.3, implica na existência de uma sequência de falha $s_F \in L \setminus L_N$ e uma sequência normal $s_N \in L_N$, com comprimento arbitrariamente longo após a falha, tais que $P_o(s_F) = P_o(s_N)$, violando, portanto, a condição D_1 da definição 4.3.

- Uma vez que a condição D_1 não é satisfeita, para que o sistema seja h-diagnosticável é necessário que a condição D_2, também da definição 4.3, seja verificada. Sejam $\mathcal{P}_N$ e $\mathcal{P}_F$ os caminhos normal e de falha associados a s_N e s_F, respectivamente, e seja $P_o(s_F) = P_o(s_N) = \nu$. De acordo com a condição D_2, H é h-diagnosticável se existe $\eta\sigma_o \in \bar{\nu}$, tal que $q_F \nsim q_N$, em que $q_F \in \Delta(Reach(G_{P_F}, \eta), \sigma_o)$ e $q_N \in \Delta(Reach(G_{P_N}, \eta), \sigma_o)$. De acordo com o algoritmo 4.2, quando um estado (q_N, q_F) de G_{VH} é alcançado após a ocorrência de um evento observável, isto é, q_N e q_F são estados cabeça, é feita a verificação da distinguibilidade e o caminho de G_{VH} é interrompido caso q_N e q_F sejam distinguíveis. Como, por hipótese, existe um caminho cíclico ambíguo em G_{VH}, então a condição D_2 não é verificada, implicando assim em H não ser h-diagnosticável.

Suponha agora que H não seja h-diagnosticável. Se H não é h-diagnosticável, então:

- Existem $s_N \in L_N$ e $s_F \in L \setminus L_N$ tais que $P_o(s_N) = P_o(s_F) = \nu$. Além disso, os caminhos associados a s_N e s_F, $\mathcal{P}_N$ e $\mathcal{P}_F$, respectivamente, não possuem estados cabeça distinguíveis após a observação de uma sequência $\eta\sigma_o \in \bar{\nu}$, o que leva ao resultado da composição $paralleld(G_{P_{NR}}, G_{P_F})$, em que $G_{P_{NR}}$ e G_{P_F} são os autômatos associados aos caminhos $\mathcal{P}_N$ e $\mathcal{P}_F$, respectivamente, a ser igual ao resultado da composição paralela entre $G_{P_{NR}}$ e G_{P_F}. Logo, como mostrado em [13], existe em G_{VH} um caminho cíclico ambíguo associado aos caminhos $\mathcal{P}_N$ e $\mathcal{P}_F$.

□

Exemplo 4.6 *Para ilustrar o resultado do algoritmo proposto, considere o seguinte sistema: Uma esteira, após receber uma garrafa, é acionada para movimentar esta garrafa ao local onde ela será preenchida com água. Após essa última etapa, a garrafa é empurrada por um atuador para um lugar de armazenagem. Esse sistema simples pode ser modelado conforme a figura 4.10, em que o evento "a" representa a chegada de uma nova garrafa, identificada através de um sensor, o evento "m" a movimentação da esteira, o evento "p" inicia o preenchimento da garrafa, o evento "d" ativa o atuador que irá empurrar a garrafa ao local de estoque, o evento 'σ_f', que simboliza uma falha que faz a esteira emperrar e se movimentar com velocidade*

reduzida, e os estados possuem as seguintes informações: i)estados da esteira: emperrada (E_E), em movimento (E_M) e com defeito (E_D); ii)estados da garra: sem garrafa (G_0), garrafa na posição de entrada (G_1), garrafa sendo movimentada (G_2) e garrafa na posição de preenchimento (G_3); iii) estados de um controlador que executa a sequência ciclicamente: C_0, C_1, C_2, C_3. A falha σ_f é intermitente e representa

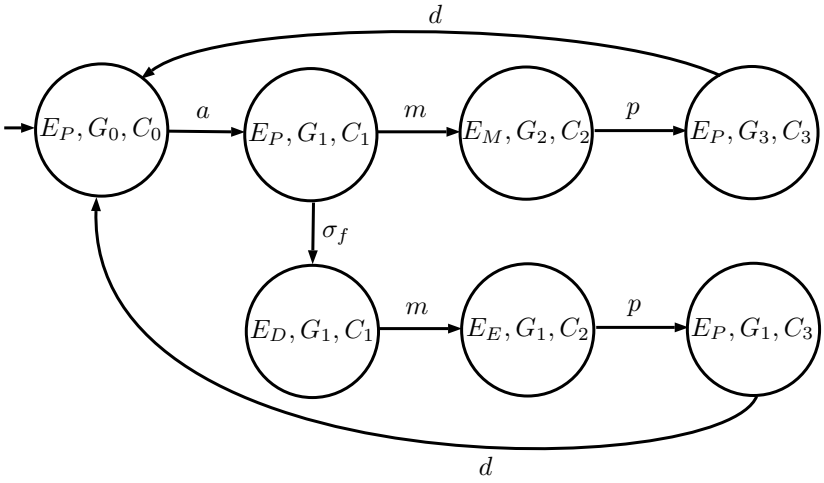


Figura 4.10: Autômato G do Exemplo 4.6.

um desalinhamento temporário da esteira, causado por algum desgaste interno, e o evento "d" retira a garrafa da esteira independentemente de sua posição.

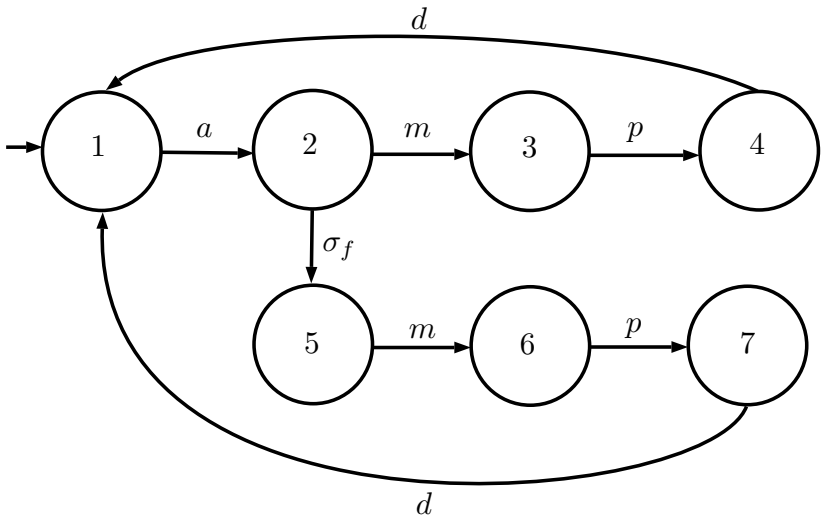


Figura 4.11: Autômato G do Exemplo 4.6 com estados renomeados.

Por simplicidade, os estados serão renomeados para uma sequência numérica de 1 a 7. O autômato renomeado pode ser visto na figura 4.11.

Considere que todos os eventos do sistema, exceto σ_f, sejam observáveis, i.e., $\Sigma_o = \{a, m, p, d\}$ e $\Sigma_{uo} = \{\sigma_f\}$.

De acordo com o passo 1 do algoritmo 4.2, o autômato G_h é obtido fazendo-se $G_h = G \parallel A_\ell^{HT}$, em que A_ℓ^{HT} é apresentado na figura 4.5. O resultado dessa

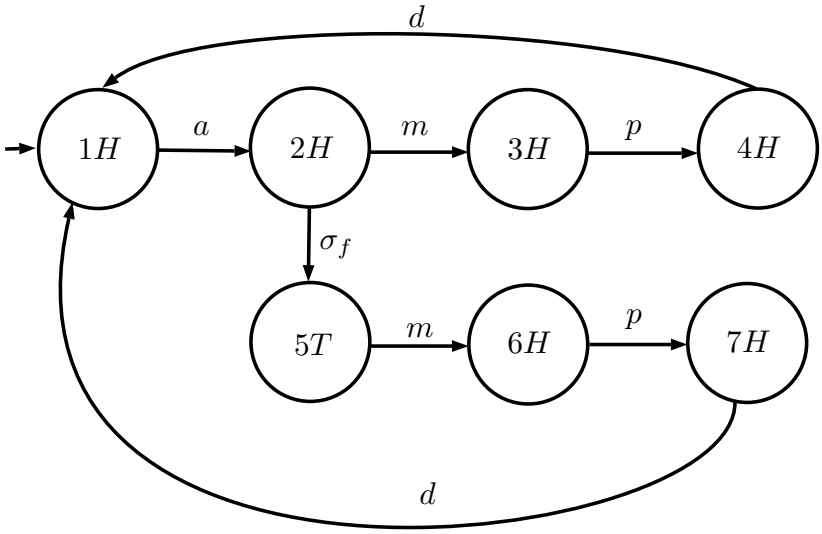


Figura 4.12: Autômato G_h do Exemplo 4.6.

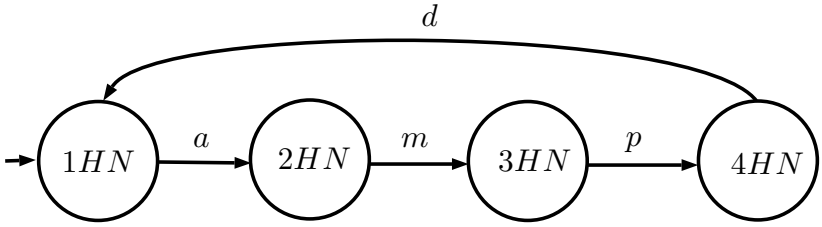


Figura 4.13: Autômato G_N do Exemplo 4.6.

operação pode ser visto na figura 4.12. Uma vez construído G_h, os passos 2 e 3 levam à construção de um autômato que representa a dinâmica normal do sistema (G_N) e um que representa a dinâmica de falha (G_F). Os autômatos G_N e G_F são mostrados nas figuras 4.13 e 4.14, respectivamente.

Note que neste exemplo,é suposto que todos os eventos são observáveis, com exceção apenas do evento de falha, o que faz com que G_{NR} seja igual a G_N. O próximo passo é a construção do autômato verificador G_{VH} e, inicialmente, será suposto que os estados não são distinguíveis uns dos outros. Como consequência, o autômato verificador resultante seria o representado na figura 4.15.

Note que há um caminho cíclico formado por $(\{1HN, 1HF\}, a, \{2HN, 2HF\}, m, \{3HN, 3HF\}, p, \{4HN, 4HF\}, d, \{1HN, 1HF\})$ *em que os estados do caminho possuem rótulos N e F. Esse caminho indicaria que o sistema não é h-diagnosticável.*

Suponha agora que a dinâmica contínua da esteira foi modelada através da sua velocidade, fazendo com que os estados $\{3\}$ *e* $\{6\}$ *sejam distinguíveis. O estado* $\{3\}$ *atribuiria à esteira uma velocidade maior quando comparada à velocidade do estado* $\{6\}$*. Desse modo, seria obtido, de acordo com o passo 5 do algoritmo 4.2, o autômato verificador da figura 4.16.*

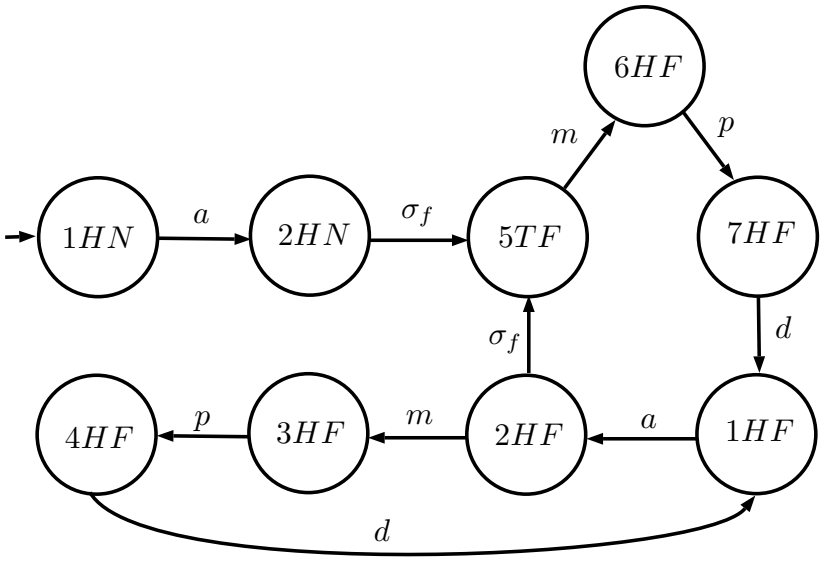


Figura 4.14: Autômato G_F do Exemplo 4.6.

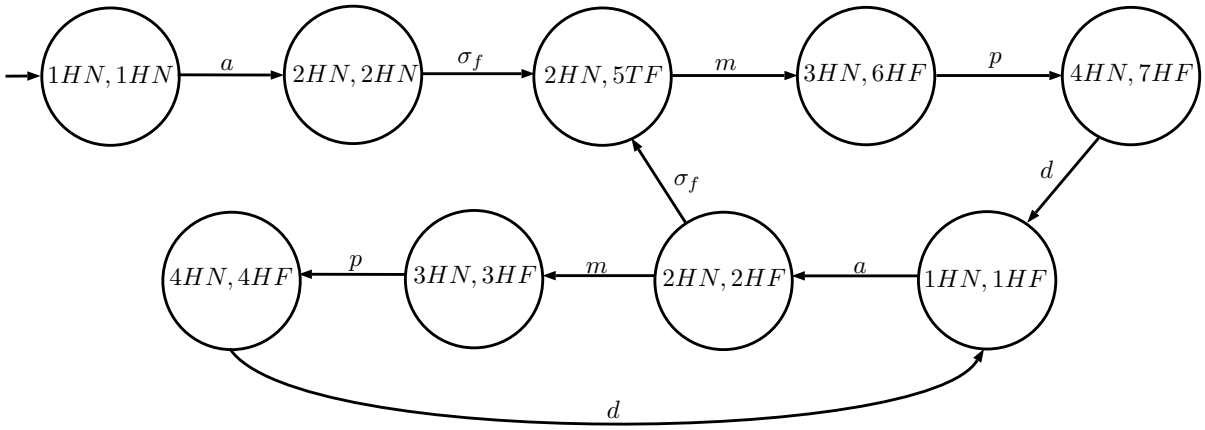


Figura 4.15: Autômato G_{VH}, sem estados distinguíveis, do Exemplo 4.6.

É válido ressaltar que apesar de os estados $\{3\}$ *e* $\{6\}$ *serem distinguíveis, só foi possível utilizar esta informação durante a construção do verificador pois estes estados foram alcançados a partir de um evento observável, ou seja, possuíam a rotulação* H*.*

Observe que a partir do momento em que houve distinção entre os estados $\{3\}$ *e* $\{6\}$*, não existem, no verificador, caminhos cíclicos ambíguos. Deste modo pode-se afirmar que este sistema é h-diagnosticável.*

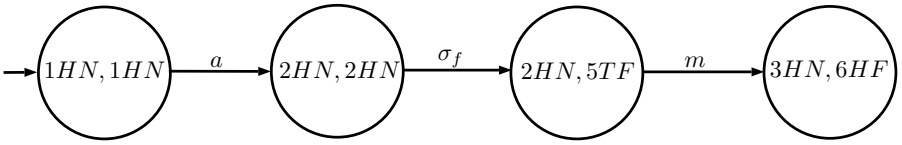


Figura 4.16: Autômato G_{VH}, com estados $\{3HN\}$ e $\{6HF\}$ distinguíveis, do Exemplo 4.6.

4.5 Comparação entre a Verificação da Diagnosticabilidade Híbrida utilizando Diagnosticadores e Verificadores

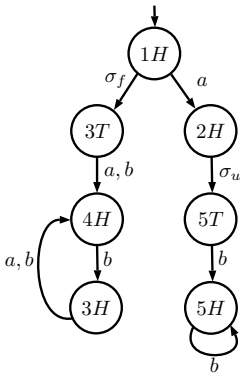


Figura 4.17: Autômato G_h para Comparação.

De modo que se possa comparar a verificação da diagnosticabilidade híbrida utilizando diagnosticadores com a verificação usando verificadores, os dois métodos serão aplicados ao mesmo sistema no exemplo a seguir.

Exemplo 4.7 *Considere o autômato G da figura 3.2 como o autômato que descreve o SED associado a um sistema híbrido H e que, inicialmente, nenhum estado de G é distinguível dos demais. Seguindo os passos do algoritmo 4.2, primeiramente é preciso realizar a composição paralela entre G e* A_ℓ^{HT}*, construindo o autômato* G_h *(figura 4.17). A partir de* G_h *são extraídos os autômatos que representam o comportamento normal e o comportamento de falha deste sistema,* G_N *e* G_F*, respectivamente. Estes dois autômatos podem ser vistos nas figuras 4.18 e 4.19, respectivamente.*

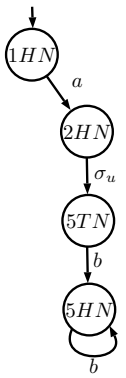


Figura 4.18: Autômato G_N para Comparação.

Considerando os conjuntos de eventos $\Sigma_o = \{a, b\}$ *e* $\Sigma_{uo} = \{\sigma_u, \sigma_f\}$*, o autômato* G_{NR} *será diferente de* G_N*. Esse autômato, construído após a renomeação* $R(\sigma)$*,*

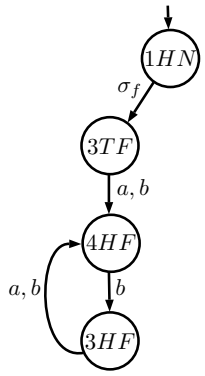


Figura 4.19: Autômato G_F para Comparação.

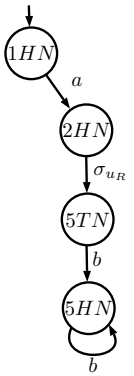


Figura 4.20: Autômato G_{NR} para Comparação.

pode ser visto na figura 4.20. O verificador e o diagnosticador, obtidos de acordo com os algoritmos 4.2 e 4.1, respectivamente, são mostrados nas figuras 4.21 e 4.22.

Analisando o verificador e o diagnosticador, é possível perceber que, em ambos, a sequência de eventos $a\sigma_u b^n$ do comportamento normal se confunde com a sequência $\sigma_f a b^n$. Essa confusão de estados é indicada no diagnosticador pelo ciclo indeterminado $\{3HF, 5TN\}, \{4HF, 5TN\}$ e no verificador pelo caminho cíclico ambíguo ($\{5HN, 3HF\}, b, \{5HN, 4HF\}, b, \{5HN, 3HF\}$). Quando o evento "b" ocorre antes do evento "a", nenhuma das soluções indica confusão no diagnóstico. O diagnosticador leva a estados em que a falha certamente ocorreu ($\{3HF\}$ e $\{4HF\}$) e o verificador não apresenta esta sequência, pois em sua concepção, ele só indica as sequências de eventos de falha e normais que possuem a mesma projeção. Portanto, as duas soluções indicam que o sistema não é h-diagnosticável.

Suponha agora que, os estados $\{2\}$ e $\{4\}$ sejam distinguíveis através de análise feita em suas dinâmicas contínuas. Através do algoritmo 4.1, o autômato G_{CD} será acrescido dos eventos cl_{2HN} e cl_{4HF}. O autômato G_{CD} pode ser observado na figura 4.23. Quanto ao verificador, quando o algoritmo 4.2 é utilizado adicionando-se as informações de distinguibilidade entre estados, o autômato indicado na figura 4.24 é obtido.

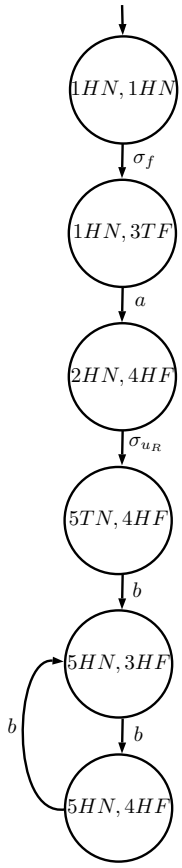


Figura 4.21: Autômato verificador G_{VH}, sem estados distinguíveis, para Comparação.

Observe que a distinção entre os estados $\{2\}$ *e* $\{4\}$ *somente é realizada quando esses estados são alcançados a partir de um evento observável, ou seja, ambos são rotulados por* H*.*

Comparando os dois autômatos finais de cada solução, é possível notar que ambos indicam que o sistema é h-diagnosticável. O diagnosticador G_{CD} *não possui nenhum ciclo indeterminado e o verificador, por sua vez, não contém nenhum caminho cíclico ambíguo.*

Realizar uma comparação acerca das duas soluções em um exemplo isolado pode levar a afirmações que nem sempre serão verdadeiras. O que pode ser dito com certeza, é que o verificador híbrido possui, no pior caso, complexidade polinomial, que é menor que a complexidade exponencial do diagnosticador.

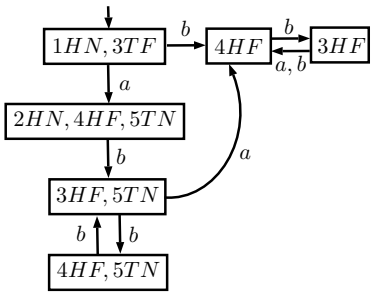


Figura 4.22: Autômato diagnosticador G_{CD}, sem estados distinguíveis, para Comparação.

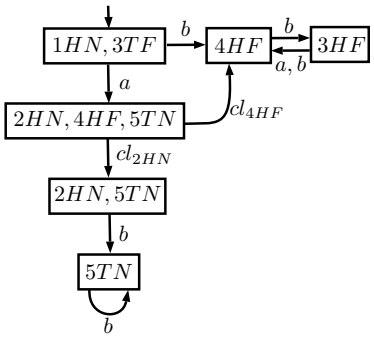


Figura 4.23: Autômato diagnosticador G_{CD} para Comparação.

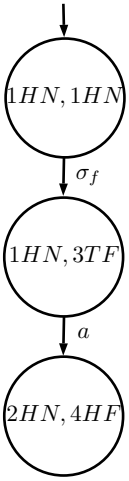


Figura 4.24: Autômato verificador G_{VH}, com estados 2 e 4 distinguíveis, para Comparação.

4.6 Complexidade Computacional

Os algoritmos apresentados neste capítulo são baseados na construção de autômatos diagnosticadores e verificadores e não procura por ciclos nesses autômatos. Por essa razão, a complexidade computacional será expressa em termos dos tamanhos desses grafos.

A tabela 4.6 mostra a complexidade computacional do algoritmo 4.2, através de análise dos autômatos que são gerados em cada passo do algoritmo, até a obtenção de G_{VH}. De modo a se detectar a diagnosticabilidade, o passo 6 propõe uma busca por caminhos cíclicos. Essa busca pode ser feita através da decomposição do grafo criado no algoritmo 4.2 em elementos fortemente conexos [25]. De acordo com o algoritmo em [25], se um dos vértices de um componente fortemente conexo tiver rotulação F, todos os vértices também terão. Como consequência, é preciso apenas verificar se existe uma aresta rotulada com um evento de Σ interligando os vértices com rótulo F de um componente fortemente conexo. Utilizando este artifício, a busca por caminhos cíclicos requer complexidade linear com o número de estados e transições de G_{VH}.

É importante ressaltar que a busca por elementos fortemente conexos não pode ser feita em G_{CD}.

A tabela 4.6 indica o número máximo de estados e transições presentes em cada autômato gerado para a obtenção de G_{VH}, ou seja, nela estão representados o pior caso para cada um dos autômatos. O primeiro passo do algoritmo 4.2 é a construção do autômato A_ℓ^{HT}, que possui dois estados e transições rotuladas por todos os eventos de Σ, e sua composição com G para formar o autômato $G_h = G \times A_\ell^{HT}$ é calculado. O pior caso para este passo resulta em um autômato em que todos os estados pertencentes a Q podem ser rotulados por T e H, e com todos os eventos de Σ viáveis para cada um dos estados, ou seja, número máximo de estados $2|Q|$ e número máximo de transições $2|Q| \times |\Sigma|$. O próximo passo se refere à construção de $G_N = G_h \times A_N$, em que A_N é modelado por um autômato de um único estado e com $|\Sigma - \Sigma_f|$ transições de autolaço , acarrentando em um número máximo de estados e transições de G_N iguais a $2|Q|$ e $2|Q| \times (|\Sigma| - |\Sigma_f|)$, respectivamente. O passo 3 se refere à construção de G_F, e, para tal, é preciso, primeiramente, construir o autômato A_l com dois estados e transições rotuladas por eventos de Σ_f. Após isto, é necessário calcular $G_l = G_h \parallel A_l$. Observe que a linguagem de G_l é igual a de G_h, que, por sua vez, é igual a de G, e que os estados de G_l são os estados de G_h rotulados por N ou F. Como resultado, o número máximo de estados G_l é o dobro do número de estados de G_h, $4|Q|$. O autômato G_F, que representa o comportamento de falha do sistema, é obtido através da operação $CoAc(G_l)$, que, no pior caso, possui o mesmo número de estados e transições de G_l. Em seguida, o autômato

G_{NR} é obtido. Como ele é construído renomeando os eventos não-observáveis de Σ, exceto os eventos de falha, o número máximo de estados e transições é igual a $2|Q|$ e $2|Q| \times (|\Sigma| - |\Sigma_f|)$. Por fim, G_{VH} é construído, no pior caso, em que não há distinguibilidade entre os estados de G, através da composição paralela de G_{NR} com G_F, que possuem número máximo de estados $2|Q|$ e $4|Q|$, respectivamente, acarretando em G_{VH} ter o número máximo de estados $8|Q|^2$. Como durante a construção de G_{NR} novos eventos podem ser criados, o número máximo de transições de G_{VH} é igual a $8|Q|^2 \times [|\Sigma| + (|\Sigma| - |\Sigma_f|)]$. Por consequência, a complexidade computacional do algoritmo proposto é $O(|Q|^2 \times (|\Sigma|)$, mostrando ser polinomial com o número de estados e eventos de G e igual a complexidade do algoritmo 3.1 para a obtenção do verificador da diagnosticabilidade de SED.

Por outro lado, o número de estados para o pior caso na construção do diagnosticador G_{CD} é $2^{4|Q|}$, possuindo crescimento muito superior quando comparado com o pior caso do verificador, $8|Q|^2$. Deste modo, pode ser constatado que o algoritmo para verificação da diagnosticabilidade de falhas em SH utilizando o autômato verificador é muito mais vantajoso quando levado em consideração o tempo computacional gasto.

Tabela 4.1: Complexidade Computacional do Algoritmo 4.2

Autômato	Nr. de Estados	Nr. de Transições
G	$\|Q\|$	$\|\Sigma\|$
A_ℓ^{HT}	2	$\|\Sigma\|$
G_h	$2\|Q\|$	$2\|Q\| \times \|\Sigma\|$
A_N	1	$\|\Sigma\| - \|\Sigma_F\|$
G_N	$2\|Q\|$	$2\|Q\| \times (\|\Sigma - \Sigma_f\|)$
A_l	2	$2\|\Sigma_f\|$
G_l	$4\|Q\|$	$4\|Q\| \times \|\Sigma\|$
G_F	$4\|Q\|$	$4\|Q\| \times \|\Sigma\|$
G_{NR}	$2\|Q\|$	$2\|Q\| \times (\|\Sigma - \Sigma_f\|)$
G_{VH}	$8\|Q\|^2$	$8\|Q\|^2 \times [\|\Sigma\| + (\|\Sigma\| - \|\Sigma_f\|)]$
Complexidade		$O(\|Q\|^2 \times (\|\Sigma\|)$

Capítulo 5

Diagnóstico Online de Falhas em Sistemas Híbridos

Se após a construção do autômato verificador, for constatado que o sistema híbrido é h-diagnosticável, é necessário implementar um diagnosticador online para detectar as falhas do sistema, uma vez que o autômato verificador não pode ser utilizado para esse propósito. Deste modo, neste capítulo é proposta uma alternativa para se realizar o diagnóstico de falhas *online*, possuindo como base a rede de Petri diagnosticadora proposta em [34]. Este novo método para o diagnóstico de falhas *online* é outra contribuição deste trabalho.

Este capítulo está dividido do seguinte modo: na seção 5.1 será apresentada a construção da rede de Petri diagnosticadora, na seção 5.2 informações sobre as dinâmicas contínuas do sistema serão inseridas na rede de Petri diagnosticadora, dando origem a uma rede de Petri diagnosticadora para SH e, por fim, na seção 5.3 o diagnóstico online será aplicado em um exemplo.

5.1 Rede de Petri Diagnosticadora

Antes de apresentar o diagnóstico online de sistemas híbridos, é preciso que sejam descritos os algoritmos acerca do diagnóstico online de SED.

Em [34, 40] é proposta uma sequência de algoritmos que têm como finalidade construir uma rede de Petri diagnosticadora online, denominada $\mathcal{N}_D$, que é capaz de sinalizar quando uma falha é detectada.

A sequência de algoritmos possui como ponto de partida a conversão de um autômato em uma rede de Petri máquina de estados. Após essa etapa inicial, essa rede dará origem a uma rede de Petri observadora de estados, que irá fornecer a estimativa dos estados do sistema após a observação de uma sequência de eventos. A rede de Petri diagnosticadora será construída com base na rede de Petri observadora

de estados. Essa rede irá indicar que uma falha ocorreu quando os estados associados ao comportamento normal do sistema não fizerem parte da sua estimativa de estados.

Os algoritmos necessários para a construção da rede de Petri diagnosticadora serão vistos a seguir [40].

Algoritmo 5.1 *(Rede de Petri máquina de estados) Seja* $G_l = (Q_l, \Sigma, \phi_l, \Gamma_l, q_{0,l})$ *o autômato com as dinâmicas normal e de falha do sistema. Uma rede de Petri máquina de estados* $\mathcal{N}_C = (P_C, T_C, Pre_C, Post_C, m_{0,C}, l_C)$ *pode ser obtida da seguinte forma:*

- *Passo 1: Crie um lugar* $p_{C_i} \in P_C$ *para cada estado* $q_{l_i} \in Q_l$.
- *Passo 2: Crie uma transição* $t_{C_j} \in T_C$ *para cada transição* $q_{l_\ell} = \phi(q_{l_i}, \sigma)$ *definida em* G_l*, para todo* $q_{l_i} \in Q_l$ *e* $\sigma \in \Gamma_l(q_{l_i})$*, e rotule* t_{C_j} *como* $l_C(t_{C_j}) = \{\sigma\}$.
- *Passo 3: Defina* $Pre_C(p_{C_i}, t_{C_j}) = Post_C(t_{C_j}, p_{C_\ell}) = 1$ *para cada transição* $t_{C_j} \in T_C$*, se a transição* $q_{l_\ell} = \phi(q_{l_i}, \sigma)$ *é definida em* G_l*. Caso contrário, faça* $Pre_C(p_{C_i}, t_{C_j}) = Post_C(t_{C_j}, p_{C_\ell}) = 0$
- *Passo 4: Faça* $m_{0,C}(p_{C_0}) = 1$ *e* $m_{0,C}(p_{C_i}) = 0$*, para todo* $p_{C_i} \in P_C \setminus \{p_{C_0}\}$*, em que* p_{C_0} *denota o lugar associado ao estado inicial de* G_l.

A próxima etapa consiste em criar novos arcos na rede de Petri $\mathcal{N}_C$ tendo como base o alcance não-observável dos lugares após o disparo de uma transição observável. Para isso, defina $T_{C_o} \subseteq T_C$ o conjunto de todas as transições de $\mathcal{N}_C$ rotuladas por eventos observáveis e a função $Reach_T : T_{C_o} \to 2^{P_C}$. O conjunto dos lugares $Reach_T(t_{C_j})$, em que $t_{C_j} \in T_{C_o}$, pode ser calculado conforme o algoritmo a seguir [40].

Algoritmo 5.2 *(Cálculo de* $Reach_T(t_{C_j})$*) Sejam* $O(t)$ *e* $O(p)$ *o conjunto de todos os lugares de saída de* t *e o conjunto de todas as transições de saída de* p*, respectivamente. Seja também* $O(P) = \bigcup_{p \in P} O(p)$ *e* $O(T) = \bigcup_{t \in T} O(t)$.

- *Passo 1: Defina* $p_{out} = O(t_{C_j})$*,* $P'_r = \{p_{out}\}$ *e* $P_r = P'_r$.
- *Passo 2: Forme o conjunto* T'_u *com todas as transições de* $O(P'_r)$ *associadas a eventos não-observáveis. Se* $T'_u = \emptyset$*,* $Reach_T(t_{C_j}) = P_r$ *e pare.*
- *Passo 3: Faça* $P'_r = O(T'_u), P_r \leftarrow P_r \cup P'_r$*, e retorne ao Passo 2.*

Uma vez obtido o alcance não-observável, é preciso que um arco de peso unitário seja inserido, conectando cada transição $t_{C_j} \in T_{C_o}$ a cada lugar $p_{C_i} \in Reach_T(t_{C_j})$, em $\mathcal{N}_C$, gerando uma nova rede de Petri. Após isso, todas as transições rotuladas

por eventos não-observáveis precisam ser removidas, bem como os arcos interligados a elas, dando origem a uma rede de Petri $\mathcal{N}_{C_o}$ com transições rotuladas somente por eventos observáveis, isto é, pertencentes a Σ_o. É importante ressaltar que a rede de Petri $\mathcal{N}_{C_o}$ deve ser binária, e para tal, é necessário que sua regra de evolução de estados seja definida conforme explicitado na subseção 2.5.5.

O algoritmo para o cálculo de $\mathcal{N}_{C_o}$ pode ser visto a seguir [40].

Algoritmo 5.3 *Cálculo de* $\mathcal{N}_{C_o} = (P_C, T_{C_o}, Pre_{C_o}, Post_{C_o}, m_{0,C}, l_{C_o})$.

- *Passo 1: Seja* $T_{C_o} \subseteq T_C$ *o conjunto de todas as transições de* $\mathcal{N}_C$ *rotuladas com eventos observáveis. Defina uma nova função* $Post'_C : T_C \times P_C \to \mathbb{N}$ *tal que* $Post'_C(t_{C_j}, p_{C_i}) = 1$*, se* $t_{C_j} \in T_{C_o}$ *e* $p_{C_i} \in Reach_T(t_{C_j})$*, e* $Post'_C(t_{C_j}, p_{C_i}) = Post_C(t_{C_j}, p_{C_i})$*, caso contrário.*

- *Passo 2: Defina as funções* $Pre_{C_o} : P_C \times T_{C_o} \to \mathbb{N}$ *e* $Post_{C_o} : T_{C_o} \times P_C \to \mathbb{N}$ *em que* $Pre_{C_o}(p_{C_i}, t_{C_j}) = Pre_C(p_{C_i}, t_{C_j}$ *e* $Post_{C_o}(t_{C_j}, p_{C_\ell}) = Post'_C(t_{C_j}, p_{C_\ell})$ *para todo* $t_{C_j} \in T_{C_o}$ *e* $p_{C_i}, p_{C_\ell} \in P_C$.

- *Passo 3: Defina uma nova função de rotulação* $l_{C_o} : T_{C_o} \to 2^{\Sigma_o}$ *tal que* $l_{C_o}(t_{C_j}) = l_C(t_{C_j})$ *para todo* $t_{C_j} \in T_{C_o}$.

Para que possa ser criada uma rede de Petri observadora de estados a partir de $\mathcal{N}_{C_o}$ é necessário adicionar um arco conectando cada lugar p_{C_i} de $\mathcal{N}_{C_o}$ a uma nova transição, rotulada com os eventos observáveis de Σ_o que não fazem parte do conjunto de eventos ativos do estado q_{l_i} de G_l associados a p_{C_i}. Dessa forma, após o disparo de uma transição, fichas serão removidas dos lugares que representam estados que não fazem parte da estimativa. O conjunto dessas novas transições será representado por T'_{C_o}. Devem ser removidos os arcos associados às transições de autolaço, uma vez que o disparo de uma transição de autolaço não altera a estimativa do estado.

Os passos que envolvem a criação da rede de Petri observadora $\mathcal{N}_{SO}$ a partir da rede de Petri $\mathcal{N}_{C_o}$ visam manter a estimativa de estados sempre fiel a cada ocorrência de um evento observável.

A construção da rede de Petri observadora de estados $\mathcal{N}_{SO}$ é obtida a partir do seguinte algoritmo [40].

Algoritmo 5.4 *(Rede de Petri observadora de estados) Seja* $\mathcal{N}_{SO} = (P_C, T_{SO}, Pre_{SO}, Post_{SO}, m_{0,SO}, l_{SO})$.

- *Passo 1: Faça* $T'_{C_o} = \emptyset$*. Para todo* $q_{l_i} \in Q_l$ *tal que* $\Gamma_l(q_{l_i}) \cap \Sigma_o \neq \Sigma_o$*, crie uma nova transição* t^i_C *e faça* $T'_{C_o} = T'_{C_o} \cup \{t^i_C\}$.

- *Passo 2: Faça* $T_{SO} = T_{C_o} \cup T'_{C_o}$.

- *Passo 3: Defina a nova função de rotulação* $l_{SO} : T_{SO} \rightarrow 2^{\Sigma_o}$*, em que* $l_{SO}(t_{C_j}) = l_{C_o}(t_{C_j})$*, se* $t_{C_j} \in T_{C_o}$ *e* $l_{SO}(t^i_C) = \Sigma_o \setminus (\Gamma_l(q_{l_i}) \cap \Sigma_o)$ *se* $t^i_C \in T'_{C_o}$*.*

- *Passo 4: Defina as funções* $Pre_{SO} : P_C \times T_{SO} \rightarrow \mathbb{N}$ *e* $Post_{SO} : T_{SO} \times P_C \rightarrow \mathbb{N}$ *em que* $Pre_{SO}(p_{C_i}, t_{C_j}) = Pre_{C_o}(p_{C_i}, t_{C_j})$ *e* $Post_{SO}(t_{C_j}, p_{C_\ell}) = Post_{C_o}(t_{C_j}, p_{C_\ell})$ *para todo* $t_{C_j} \in T_{C_o}$ *e* $p_{C_i}, p_{C_\ell} \in P_C$*, e* $Pre_{SO}(p_{C_i}, t^i_C) = 1$*,* $Pre_{SO}(p_{C_\ell}, t^i_C) = 0$ *e* $Post_{SO}(t^i_C, p_{C_\ell}) = Post_{SO}(t^i_C, p_{C_i}) = 0$*, para todo* $t^i_C \in T'_{C_o}$ *e* $p_{C_i}, p_{C_\ell} \in P_C$*, em que* $i \neq \ell$*.*

- *Passo 5: Defina o estado inicial de* $\mathcal{N}_{SO}$ *atribuindo uma ficha a cada lugar associado ao estado* $UR(q_{0,l})$ *e nenhuma ficha aos demais.*

- *Passo 6: Redefina* T_{SO}*,* Pre_{SO} *e* $Post_{SO}$ *eliminando as transições de autolaço e seus arcos associados.*

Após a obtenção de $\mathcal{N}_{SO}$, a rede de Petri diagnosticadora $\mathcal{N}_D$ pode ser calculada adicionando-se à $\mathcal{N}_{SO}$ uma transição t_f que possui um lugar de entrada p_N, inicialmente com uma ficha, e um lugar de saída p_F, sem fichas, ambos conectados a t_f por arcos de peso unitário. Arcos inibidores de peso igual a um são utilizados para conectar cada lugar associado a um estado de G_l que tenha rótulo N à transição t_f. Uma vez que o arco inibidor habilita a transição apenas quando o número de fichas do lugar de entrada é igual a zero, t_f será habilitada somente quando todos os lugares associados aos estados de G_l que possuem rotulação N não possuírem fichas, ou seja, quando os estados com o comportamento normal do sistema não mais fizerem parte da estimativa, indicando, assim, a ocorrência de uma falha.

A transição t_f é rotulada com o evento sempre ocorrente θ para representar que t_f dispara imediatamente após ter sido habilitada, removendo a ficha do lugar p_N e adicionando uma ficha ao lugar p_F.

O algoritmo a seguir define os passos para a construção da rede de Petri diagnosticadora $\mathcal{N}_D$ a partir de $\mathcal{N}_{SO}$ [40].

Algoritmo 5.5 *(Rede de Petri diagnosticadora) Seja* $\mathcal{N}_D = (P_D, T_D, Pre_D, Post_D, In_D, m_{0,D}, \Sigma_o \cup \{\theta\}, l_D)$

- *Passo 1: Defina* $T_D = T_{SO} \cup \{t_f\}$*.*

- *Passo 2: Defina a função de rotulação* $l_D : T_D \rightarrow 2^{\Sigma_o \cup \{\theta\}}$*, tal que* $l_D(t_D) = l_{SO}(t_D)$ *para todo* $t_D \in T_{SO}$*, e* $l_D(t_D) = \{\theta\}$ *para todo* $t_D = t_f$*.*

- *Passo 3: Defina* $P_D = P_C \cup \{p_N\} \cup \{p_F\}$*.*

- *Passo 4: Defina as funções* $Pre_D : P_D \times T_D \rightarrow \mathbb{N}$ *e* $Post_D : T_D \times P_D \rightarrow \mathbb{N}$ *em que* $Pre_D(p_{C_i}, t_D) = Pre_{SO}(p_{C_i}, t_D)$ *e* $Post_D(t_D, p_{C_i}) = Post_{SO}(t_D, p_{C_i})$

para todo $t_D \in T_{SO}$ *e* $p_{C_i} \in P_C$, $Pre_D(p_N, t_f) = Post_D(t_f, p_F) = 1$, *e* $Pre_D(p_{C_i}, t_f) = Post_D(t_f, p_{C_i}) = 0$, *para todo* $p_{C_i} \in P_C$.

- *Passo 5: Defina a função dos arcos inibidores* $In_D : P_D \times T_D \rightarrow \{0, 1\}$, *em que* $In_D(p_D, t_f) = 1$ *para todo lugar* $p_D \in P_D$ *associado a um estado de* G_l *rotulado por* N *e* $In_D(p_D, t_D) = 0$ *para todos os demais lugares* $p_D \in P_D$ *e transições* $t_D \in T_D$.

- *Passo 6: Defina o estado inicial dos lugares* P_N *como um e do lugar* P_F *como zero. Os demais lugares possuem a mesma condição inicial definida por* $m_{0,SO}$.

O algoritmo 5.6 resume os passos necessários para a construção da rede de Petri diagnosticadora $\mathcal{N}_D$.

Algoritmo 5.6

- *Passo 1: Calcule a rede de Petri máquina de estados rotulada* $\mathcal{N}_C = (P_C, T_C, Pre_C, Post_C, m_{0,C}, l_C)$ *utilizando o algoritmo 5.1.*

- *Passo 2: Calcule a rede de Petri* $\mathcal{N}_{C_o} = (P_C, T_{C_o}, Pre_{C_o}, Post_{C_o}, m_{0,C}, l_{C_o})$ *conforme o algoritmo 5.3.*

- *Passo 3: Calcule a rede de Petri observadora de estados binária* $\mathcal{N}_{SO} = (P_C, T_{SO}, Pre_{SO}, Post_{SO}, m_{0,SO}, l_{SO})$ *por meio do algoritmo 5.4.*

- *Passo 4: Calcule a rede de Petri diagnosticadora* $\mathcal{N}_D = (P_D, T_D, Pre_D, Post_D, In_D, m_{0,D}, \Sigma_o \cup \{\theta\}, l_D)$ *conforme o algoritmo 5.5.*

Dada a construção da rede de Petri diagnosticadora $\mathcal{N}_D$, é possível afirmar a seguinte condição necessária e suficiente para o diagnóstico online de um sistema [34].

Teorema 5.1 *Seja* L *a linguagem gerada pelo sistema* G *e suponha* L *diagnosticável em relação a* P_o *e* Σ_f. *Seja* $s \in L \setminus L_N$ *tal que* $\forall w \in L$ *satisfazendo* $P_o(w) = P_o(s), w \in L \setminus L_N$. *Então, o número de fichas no lugar* p_F *de* $\mathcal{N}_D$, *após a observação de uma sequência* $P_o(s)$ *é igual a um.*

5.2 Rede de Petri Diagnosticadora para um Sistema Híbrido

Nessa seção serão apresentados os algoritmos que irão construir uma rede de Petri diagnosticadora para um sistema híbrido. Esses algoritmos possuem o objetivo

de incrementar a rede de Petri diagnosticadora $\mathcal{N}_D$, apresentada em [34], com informações da dinâmica contínua do sistema. Para que essa integração ocorra, a solução será composta de duas partes: i) uma parte de estimativa de estados e diagnóstico; ii) um analisador de consistência de estados. A divisão da solução pode ser visualizada na figura 5.1.

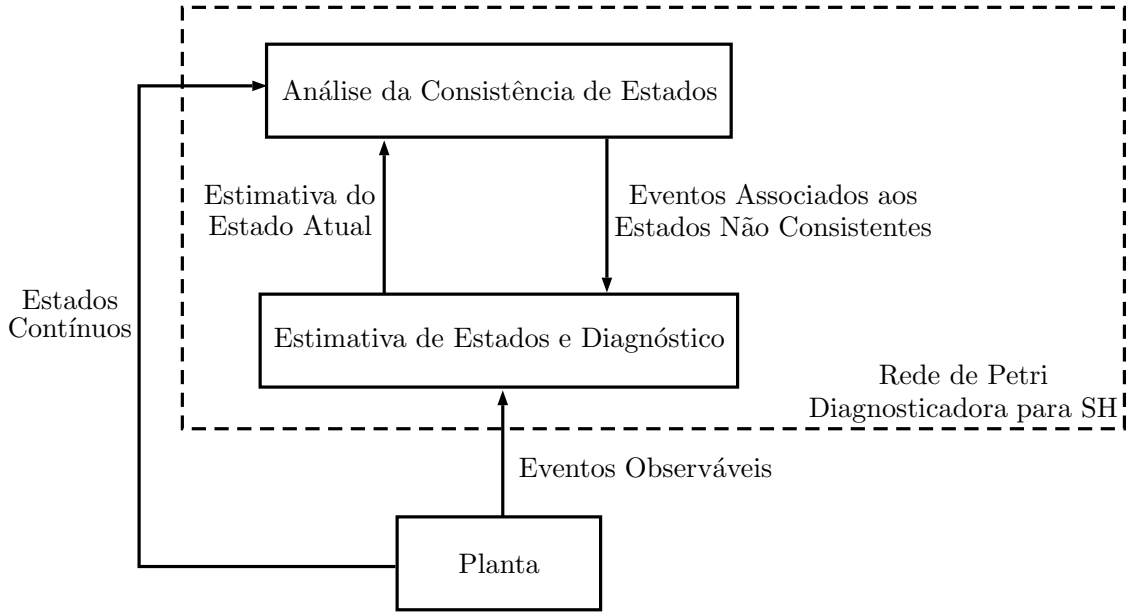


Figura 5.1: Esquemático: Rede de Petri Diagnosticadora para SH.

Essa rede de Petri diagnosticadora para SH funciona como uma rede de Petri interpretada [23]. A planta envia ao estimador informações de ocorrência de eventos observáveis para que a estimativa de estados seja atualizada a cada nova ocorrência, ao mesmo tempo que envia para o analisador de consistência os valores dos estados contínuos. Após a ocorrência de um evento observável a estimativa de estados é atualizada e os estados estimados são enviados para o analisador de consistência, que, junto com as informações dos estados contínuos, irá verificar a consistência dos estados estimados. Estados que não forem consistentes deverão ser removidos da estimativa.

Para a parte de estimativa de estados e diagnóstico, o ponto de partida é a construção de uma rede de Petri diagnosticadora $\mathcal{N}_D$, por meio do algoritmo 5.6. Uma vez construída, os comandos para o início do algoritmo de análise da consistência de estados deverão ser inseridos, bem como as transições que serão habilitadas pelos eventos provenientes do analisador. O algoritmo de construção dessa rede de Petri diagnosticadora para SH, denominada $\mathcal{N}_{DH}$, pode ser visualizado a seguir.

Algoritmo 5.7 *Rede de Petri diagnosticadora para um SH* $\mathcal{N}_{DH} = (P_D, T_{DH}, Pre_{DH}, Post_{DH}, In_D, m_{0,D}, \Sigma_o \cup \{r_k\}, l_{DH})$

- *Passo 1: Seja* $T_r = \bigcup_{k=1}^{n} t_{r_k}$*, em que* n *é o número de lugares na rede e* tr_k *é uma transição criada para remover fichas de lugares conforme ação do analisador da consistência de estados.* $T_{DH} = T_D \cup T_r$*.*

- *Passo 2: Defina a função de rotulação* $l_{DH} : T_{DH} \rightarrow 2^{\Sigma_o \cup \{r_k\}}$*, tal que* $l_{DH}(t_{DH}) = l_D(t_{DH})$ *para todo* $t_{DH} \in T_D$*, e* $l_{DH}(t_{DH}) = \{r_k\}$ *para todo* $t_{DH} \in t_{r_k}$*.*

- *Passo 3: Defina as funções* $Pre_{DH} : P_D \times T_{DH} \rightarrow \mathbb{N}$ *e* $Post_{DH} : T_{DH} \times P_D \rightarrow \mathbb{N}$ *em que* $Pre_{DH}(p_{D_i}, t_{DH}) = Pre_D(p_{D_i}, t_{DH})$ *e* $Post_{DH}(t_{DH}, p_{D_i}) = Post_D(t_{DH}, p_{D_i})$ *para todo* $t_{DH} \in T_D$ *e* $p_{D_i} \in P_D$*,* $Pre_{DH}(p_{D_i}, r_k) = 1$ *para todo* $p_{D_i} \in P_{DH}$ *e* $r_k \in T_r$*, e* $Post_{DH}(r_k, p_{D_i}) = 0$*, para todo* $p_{D_i} \in P_{DH}$ *e* $r_k \in T_r$*.*

- *Passo 4: Seja* P_h *o conjunto de lugares* $p_{D_i} \in P_{DH}$ *relacionados aos estados de* G_l *rotulados por* H*. Atribua o comando de iniciar análise de consistência a todos os lugares de* P_h*.*

O algoritmo 5.7 insere uma ação "iniciar verificação"em todos os lugares correspondentes aos estados de G_l que são rotulados por H. Dessa forma, sempre que uma ficha for inserida em um desses lugares, essa ação irá disparar o algoritmo de verificação da consistência dos estados estimados. Além disso, são inseridas transições r_k que são habilitadas pelo analisador da consistência quando algum dos estados estimados está inconsistente com informação da dinâmica contínua do sistema. Uma vez que o analisador da consistência de estados habilita uma transição r_k, ela irá disparar e uma ficha será removida do lugar interligado a ela.

O algoritmo presente no analisador da consistência de estados é apresentado a seguir.

Algoritmo 5.8 *Seja* Q_E *a estimativa de estados obtida de* $\mathcal{N}_{DH}$*, o comando iniciar verificação* (Q_E) *é executado da seguinte forma.*

- *Passo 1: Faça* $l = 0$ *e* $Q_{\bar{C}} \leftarrow Q_E$*.*

- *Passo 2: Forme uma sequência de estados* Q_H *com os estados de* Q_E *rotulados por* H*.*

- *Passo 3: Se* $|Q_H| > 1$ *faça:*

  - *Passo 3.1:* $l = l + 1$
  - *Passo 3.2: Se* $CONSISTÊNCIA(q_{H_\ell}) = true$*, em que* $q_{H_\ell} \in Q_H$*:*
    - *Passo 3.2.1:* $Q_{\bar{C}} \leftarrow Q_{\bar{C}} \setminus UR(q_{H_\ell})$*.*

– *Passo 3.3: Se $l < |Q_H|$, retorne ao passo 3.1*

- *Passo 4: Gere eventos r_k associados aos estados não consistentes de Q_C.*

Observação 5.1 *A função CONSISTÊNCIA que retorna se um estado estimado é consistente com as medidas do estado atual pode ser implementada de diversas formas [38], como exemplo pode-se citar o método de espaço de paridade. O foco deste trabalho não está na implementação dessa função, e sim no resultado proveniente da verificação da consistência de um estado visando a distinguibilidade entre estados.*

A ação "iniciar verificação"funciona da seguinte forma: uma vez ativada, a estimativa de estados e os estados cabeça rotulados por H presentes na estimava são armazenados em $Q_{\bar{C}}$ e Q_H, respectivamente. Se o número de estados cabeça for maior do que um, a consistência dos estados presentes em Q_H é verificada, um a um. Caso a consistência seja verdadeira, isto é, a informação da dinâmica contínua do estado atual do sistema esteja de acordo com a dinâmica da estimativa, esse estado e seu respectivo alcance não-observável são removidos de $Q_{\bar{C}}$. Esse processo se repete até que todos os estados de Q_H tenham sua consistência verificada e uma vez terminado, os eventos associados aos estados de $Q_{\bar{C}}$, denominados r_k, são gerados.

Uma vez que esses eventos r_k tenham sido gerados, as fichas dos lugares não-consistentes de $\mathcal{N}_{DH}$ serão removidas e a estimativa de estados é atualizada. Caso os estados que representam o comportamento normal do sistema não contenham ficha, os arcos inibidores irão habilitar a transição t_f, fazendo com que a ficha seja transferida do lugar p_N para o lugar p_F, indicando a ocorrência de uma falha.

Dada a construção da rede de Petri diagnosticadora para SH $\mathcal{N}_{DH}$, é possível afirmar a seguinte condição necessária e suficiente para o diagnóstico online.

Teorema 5.2 *Seja L a linguagem gerada pelo sistema a eventos discretos G associado ao sistema híbrido H. Seja $s_F \in L \setminus L_N$ tal que para todo $w \in L$ satisfazendo $P_o(w) = P_o(s_F), w \in L \setminus L_N$, ou $\forall s_N \in L_N$ tal que $P_o(s_N) = P_o(s_F) = \nu$, existe $\eta\sigma_o \in \overline{\nu}$ tal que $q_F \nsim q_N$ em que $q_F \in \Delta(Reach(G_{P_F}, \eta), \sigma_o)$ e $q_N \in \Delta(Reach(G_{P_N}, \eta), \sigma_o)$. Então, o número de fichas no lugar p_F de $\mathcal{N}_{DH}$, após a observação de uma sequência $P_o(s_F)$, é igual a um.*

Prova Note, de acordo com o algoritmo 5.7, que a estimativa de estado de $\mathcal{N}_{DH}$ após a ocorrência de um evento observável é igual à estimativa de estado de $\mathcal{N}_D$, caso não haja estados distinguíveis rotulados por H em G_l. Como, de acordo com o teorema 5.1, a rede de Petri $\mathcal{N}_D$ fornece, a cada ocorrência de um evento observável, a estimativa correta de estado do sistema, a estimativa obtida a partir de $\mathcal{N}_{DH}$ é também correta caso não haja estados distinguíveis. Logo, se existe uma sequência s_F de comprimento arbitrariamente longo após a falha tal que não existe

uma sequência normal w com a mesma projeção que s_F, nenhum estado rotulado por N em $\mathcal{N}_{DH}$ possuirá fichas após a ocorrência de s_F, o que implica no disparo da transição t_f e consequente marcação 1 no lugar p_F, indicando a ocorrência do evento de falha.

Considere agora o caso em que existem estados rotulados por H distinguíveis em uma estimativa de estado de $\mathcal{N}_{DH}$. Nesse caso, o algoritmo 5.8 retorna os eventos de retirada de ficha r_k associados aos lugares que não são consistentes ou que não estão associados aos estados do alcance não-observável dos estados consistentes, corrigindo a estimativa de estado do sistema. Logo, se após a ocorrência de uma sequência de falha s_F de comprimento arbitrariamente longo após a falha, associada a um caminho $\mathcal{P}_F$, todos os caminhos normais cujas sequências possuem mesma projeção que s_F, são distinguíveis de $\mathcal{P}_F$ após a sequência observável $\eta\sigma_o$, então nenhum lugar associado a um estado rotulado por N de G_l possui ficha após a observação de $\eta\sigma_o$. Logo, a transição t_f será habilitada e disparará, colocando uma ficha no lugar p_F, indicando a ocorrência da falha.

□

Na próxima seção será apresentado um exemplo ilustrando os passos para a obtenção de uma rede de Petri diagnosticadora híbrida $\mathcal{N}_{DH}$ e o processo de diagnóstico online do autômato G_l construído no exemplo 4.6.

5.3 Exemplo de uma Rede de Petri Diagnosticadora para um Sistema Híbrido

De modo a ilustrar o funcionamento dos algoritmos apresentados na seção 5.2, utilizar-se-a o autômato G_l, construído a partir de G no Passo 5 do algoritmo 4.2, do exemplo 4.6. O autômato G_l pode ser visualizado na figura 5.2.

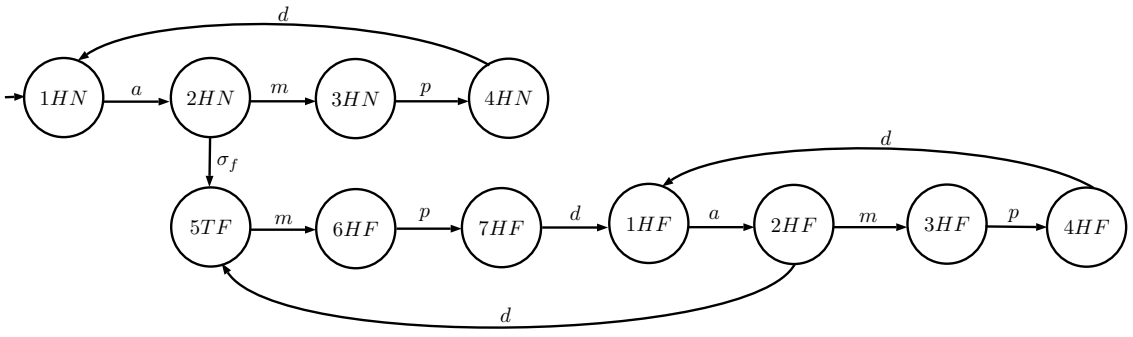


Figura 5.2: Autômato G_l.

Seguindo as orientações do algoritmo 5.6, o primeiro passo se baseia na construção de uma rede de Petri máquina de estados $\mathcal{N}_C$, seguindo-se o algoritmo 5.1,

que pode ser vista na figura 5.3. No segundo passo, a rede de Petri $\mathcal{N}_{C_o}$ é calculada através da execução do algoritmo 5.3. Essa rede de Petri pode ser visualizada na figura 5.4.

Em seguida, realizando os passos do algoritmo 5.4, a rede de Petri observadora de estados $\mathcal{N}_{SO}$, é calculada, e está representada na figura 5.5. A partir de $\mathcal{N}_{SO}$, a rede de Petri diagnosticadora $\mathcal{N}_D$, que pode ser vista na figura 5.6, é construída através do algoritmo 5.5.

Uma vez construída a rede de Petri $\mathcal{N}_D$, resta apenas a execução do algoritmo 5.7 para que seja calculada a rede de Petri diagnosticadora híbrida. Essa rede de Petri está representada na figura 5.7.

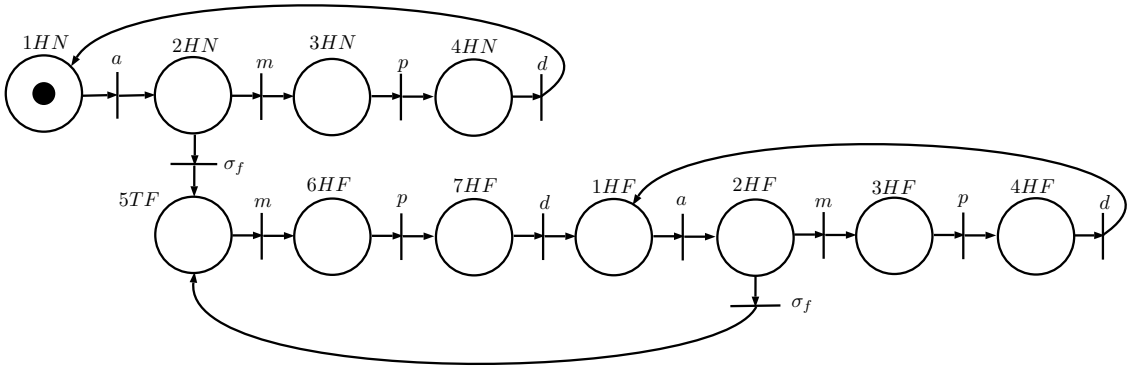


Figura 5.3: Rede de Petri $\mathcal{N}_C$.

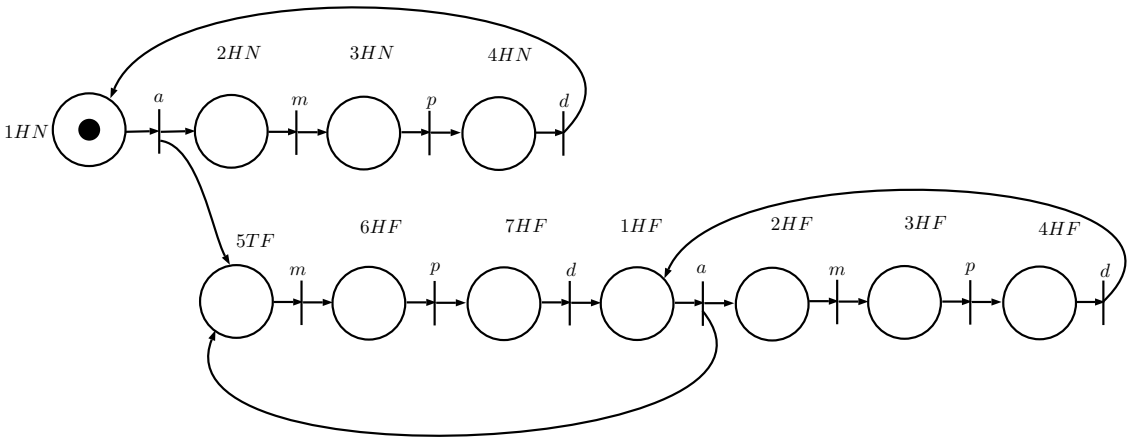


Figura 5.4: Rede de Petri $\mathcal{N}_{C_o}$.

Para que a evolução da rede de Petri diagnosticadora para SH $\mathcal{N}_{DH}$, construída anteriormente e indicada na figura 5.7, possa ser visualizada, será considerado que os estados $\{3\}$ e $\{6\}$ são distinguíveis, assim como no exemplo 4.6. Suponha também que a sequência de eventos seja $a\sigma_f m$, ocasionando o disparo da transição t_{SO_0} e, em seguida, as transições t_{SO_2} e t_{SO_8}.

Após o disparo da transição t_{SO_0}, rotulada por "a", uma ficha é inserida nos lugares rotulados por $2HN$ e $5TF$, habilitando as transições t_{SO_2} e t_{SO_8}, rotuladas

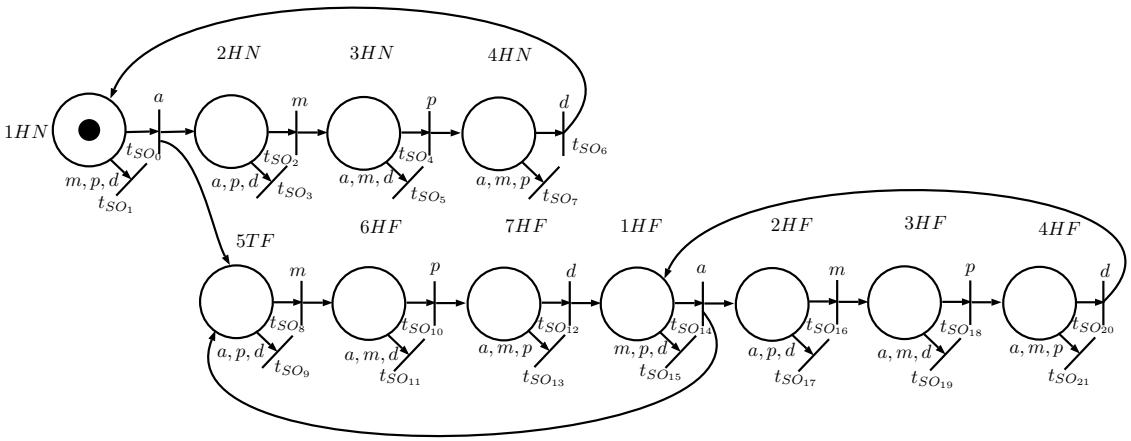


Figura 5.5: Rede de Petri $\mathcal{N}_{SO}$.

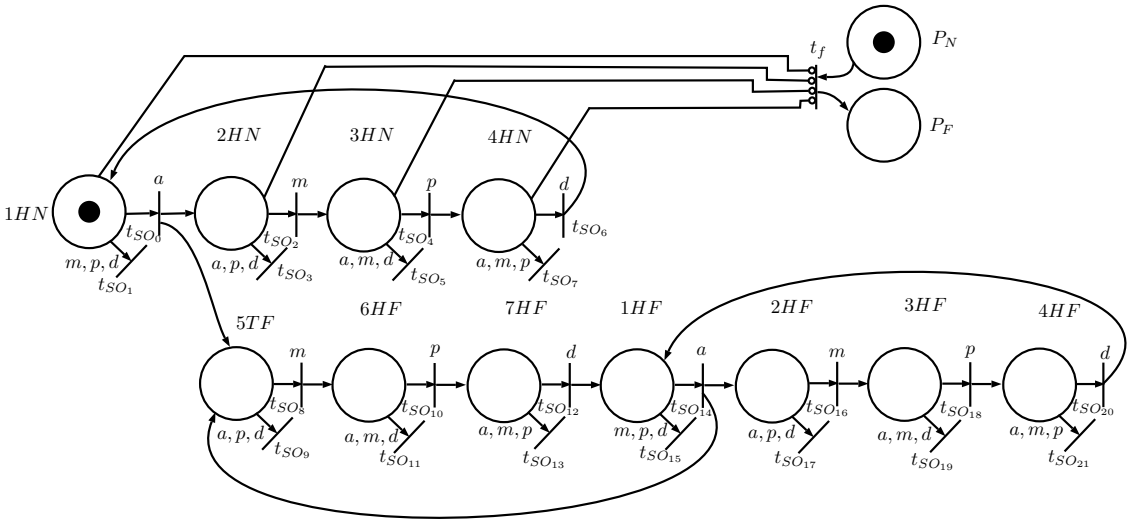


Figura 5.6: Rede de Petri $\mathcal{N}_D$.

por "m". Como há uma ficha no lugar rotulado por $2HN$, a transição t_f não está habilitada e a rede de Petri indica que o comportamento do sistema ainda está normal. Além disso, o recebimento de ficha em um lugar que possui rótulo H aciona a ação "iniciar verificação". Como só há um lugar da rede com rótulo H, o algoritmo 5.8 não irá verificar a consistência de nenhum estado. O posicionamento das fichas pode ser visto na figura 5.8.

As próximas transições disparadas são as rotuladas por "m", fazendo com que as fichas sejam retiradas dos lugares rotulados por $2HN$ e $5TF$ e inseridas nos rotulados por $3HN$ e $6HF$. Neste momento, quando a ação "iniciar verificação"for executada, o algoritmo 5.8 irá verificar a consistência dos estados estimados, e, como foi suposto que os estados distinguíveis são $\{3\}$ e $\{6\}$, e a falha σ_f ocorreu, a transição r_3 será habilitada pelo analisador da consistência e irá disparar, fazendo com que uma ficha seja removida do lugar rotulado por $3HN$, restando apenas a ficha do lugar rotulado por $6HF$.

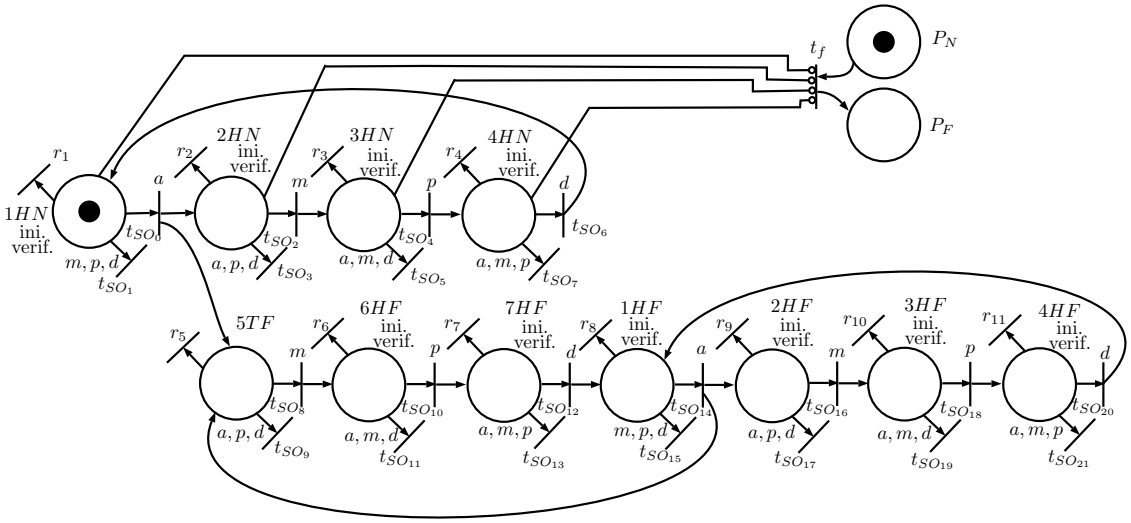


Figura 5.7: Rede de Petri $\mathcal{N}_{DH}$

Uma vez que não há fichas em nenhum lugar rotulado por N e que a transição t_f dispara imediatamente após sua habilitação, a ficha do lugar P_N é transferida para o lugar P_F, indicando a ocorrência da falha σ_f. A rede de Petri antes e depois da ação do analisador da consistência podem ser vistas nas figuras 5.9 e 5.10, respectivamente.

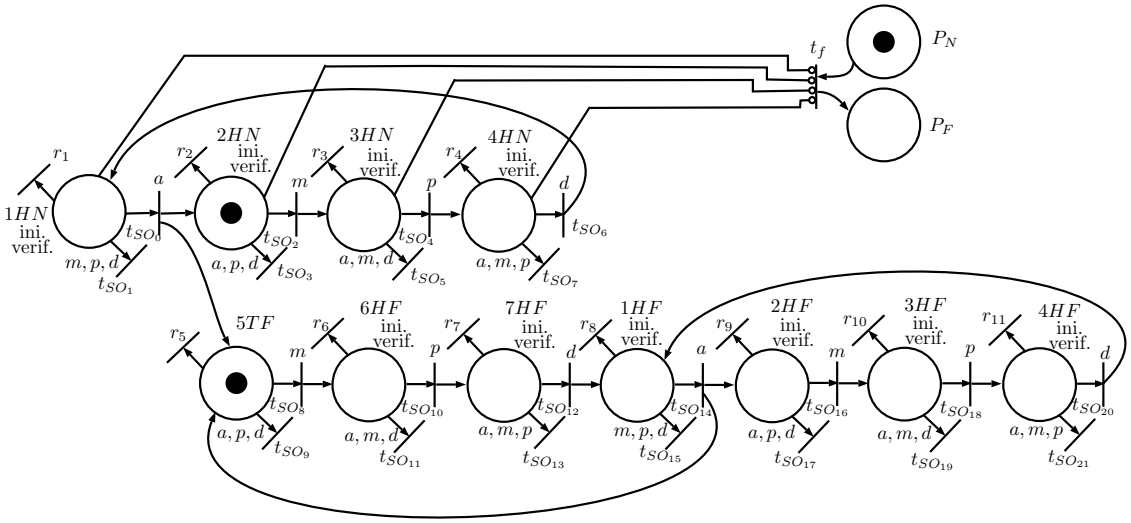


Figura 5.8: Rede de Petri $\mathcal{N}_{DH}$ após o disparo da transição t_{SO_0}.

Caso o analisador da dinâmica não tivesse conseguido distinguir os lugares rotulados por $3HN$ e $6HF$, a próxima dupla de lugares com ficha seriam os rotulados por $4HN$ e $7HF$, após o disparo das transições rotuladas por p. Nesses dois novos lugares, uma vez que possuem rótulos H, o analisador teria mais uma chance de realizar a distinguibilidade.

Observação 5.2 *É importante ressaltar que essa rede de Petri diagnosticadora para*

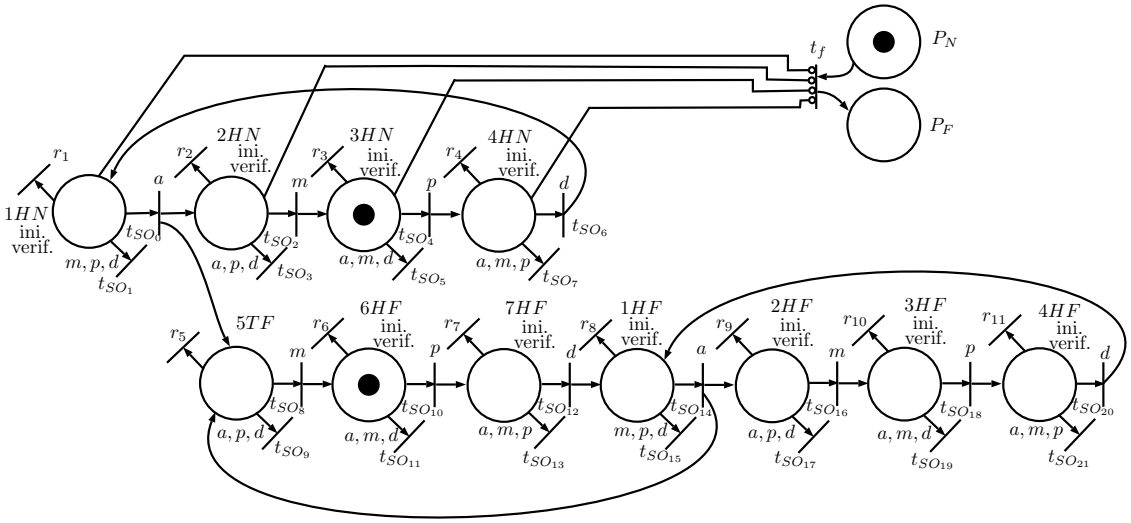


Figura 5.9: Rede de Petri $\mathcal{N}_{DH}$ após o disparo das transições t_{SO_2} e t_{SO_8}.

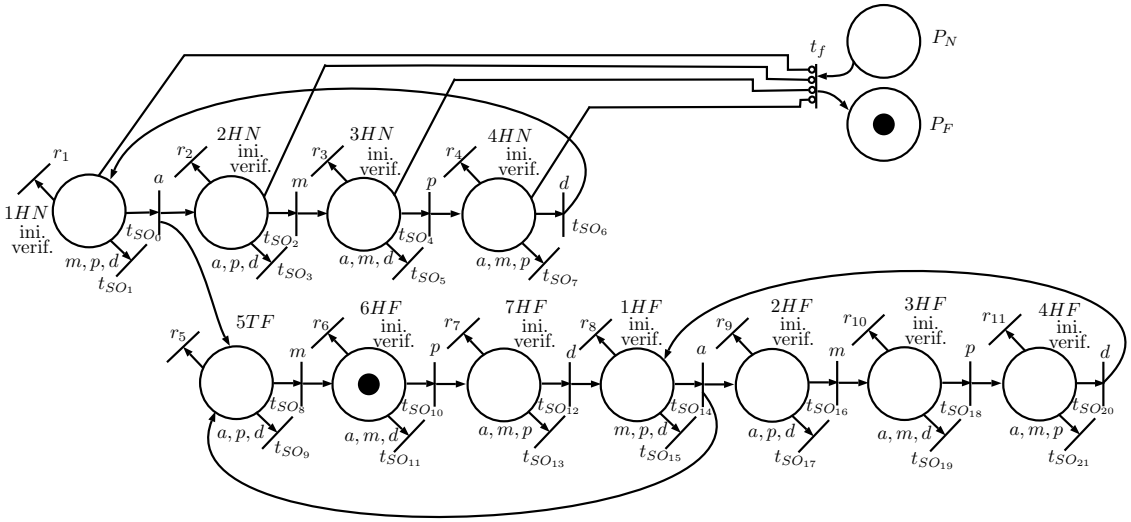


Figura 5.10: Rede de Petri $\mathcal{N}_{DH}$ após ação do analisador da consistência e disparo das transições r_3 e t_f.

SH é construída somente após a constatação da h-diagnosticabilidade do sistema híbrido.

É possível notar que se não houver distinção das dinâmicas contínuas dos pares de estados $\{3HN, 6HF\}$ e $\{4HN, 7HF\}$, a rede de petri diagnosticadora para SH nunca irá indicar a ocorrência da falha, assim como o verificador e o diagnosticador apresentados no capítulo anterior iriam constatar que o sistema não seria h-diagnosticável.

Capítulo 6

Conclusão e Possíveis Trabalhos

Neste trabalho, uma nova abordagem para a verificação da diagnosticabilidade de falhas em sistemas híbridos foi proposta. Através da construção de um autômato verificador G_{VH}, é possível verificar se a linguagem do sistema é h-diagnosticável buscando-se por componentes fortemente conexas que possuem caminhos cíclicos ambíguos. Quando comparado às demais soluções propostas por outros autores [27–29, 32], é notável a redução de tempo computacional gasto para inferir a diagnosticabilidade do sistema, uma vez que as demais soluções possuem como base a construção de autômatos diagnosticadores. Diagnosticadores possuem, no pior caso, crescimento exponencial com o número de estados da planta. Desta forma, o autômato proposto neste trabalho apresenta vantagens evidentes quando o tempo computacional é levado em consideração, dado que sua complexidade é polinomial.

Diferentemente de diagnosticadores, verificadores não podem ser usados para se realizar o diagnóstico online de falhas do sistema, e para atender a essa necessidade, neste trabalho é proposto um algoritmo para a construção de uma rede de Petri diagnosticadora para SH, capaz de informar a estimativa de estado do sistema baseado na observação de eventos e na consistência de estados discretos a partir das variáveis contínuas do sistema.

Quando uma comparação de custo computacional é realizada tomando como base a soma das soluções (verificador e rede de Petri diagnosticadora para SH), as soluções apresentadas neste trabalho se mostram mais eficientes. Dado que a rede de Petri é construída a partir do autômato G_l, o seu número de lugares no pior caso é $O(4|Q|)$. Somando a contabilização de pior caso da rede de Petri diagnosticadora para SH à construção do verificador, é possível concluir que o custo computacional da soma das soluções é reduzido quando comparado à construção do autômato diagnosticador.

Por fim, continuações deste trabalho podem ser sugeridas, como a utilização do autômato verificador para análise da diagnosticabilidade descentralizada de falhas em sistemas distribuídos, a implementação da rede de Petri diagnosticadora para SH utilizando uma linguagem de alto nível, ou até mesmo integrando uma

programação em um controlador lógico programável com algum sistema capaz de realizar o cálculo da distinguibilidade. Além dessas possibilidades, ainda é possível citar o desenvolvimento de métodos para a distinguibilidade de estados, visando simplificar o diagnóstico de falhas e, reduzir o custo computacional para o diagnóstico de falhas.

Referências Bibliográficas

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

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

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

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

[5] 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, pp. 33–86, 2000.

[6] SAMPATH, M., LAFORTUNE, S., TENEKETZIS, D., "A Hybrid Aproach to Failure Diagnosis of Industrial Systems", *Proc. of the American Control Conference*, v. 3, pp. 2077–2082, 2001.

[7] SENGUPTA, R., TRIPAKIS, S., "Decentralized diagnosability of regular languages is undecidable", *In Proc. 41st IEEE Conference on Decision and Control*, v. 1, pp. 423–428, 2002.

[8] ZAD, S. H., KWONG, R. H., WONHAM, W. M., "Fault diagnosis in discrete-event systems: Framework and model reduction", *IEEE Transactions on Automatic Control*, v. 48, pp. 1199–1212, 2003.

[9] SU, R., WONHAM, W., "A model of component consistency in distributed diagnosis", *International Workshop on Discrete Event Systems*, v. 16, pp. 427–432, 2004.

[10] SU, R., WONHAM, W. M., "Global and local consistencies in distributed fault diagnosis for discrete-event systems", *IEEE Transactions on Automatic Control*, v. 32, pp. 1923–1935, 2005.

[11] PAOLI, A., LAFORTUNE, S., "Diagnosability analysis of a class of hierarchical state machines", *Discrete Event Dynamic Systems: Theory and Applications*, v. 18, pp. 385–413, 2008.

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

[13] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C., "Polynomial time verification of decentralized diagnosability of discrete event systems", *IEEE Transactions on Automatic Control*, v. 56, pp. 1679–1684, 2011.

[14] CARVALHO, L. K., BASILIO, J. C., MOREIRA, M. V., "Robust diagnosis of discrete event systems against intermittent loss of observations", *Automatica*, v. 48, pp. 2068–2078, 2012.

[15] BASILIO, J. C., SOUZA LIMA, S. T., LAFORTUNE, S., et al., "Computation of minimal event bases that ensure diagnosability", *Discrete Event Dynamic Systems: Theory and Applications*, v. 22, 2012.

[16] BASILIO, J. C., LAFORTUNE, S., "Robust codiagnosability of discrete event systems", *American control conference*, v. 10, pp. 2202–2209, 2009.

[17] CARVALHO, L., MOREIRA, M., BASILIO, J., "Generalized robust diagnosability of discrete event systems", *In Proc. IFAC World Congress*, v. 18, pp. 8737–8742, 2011.

[18] ZAYTOON, J., LAFORTUNE, S., "Overview of Fault Diagnosis Methods for Discrete Event Systems", *Annual Reviews in Control*, v. 37, pp. 308–320, 2013.

[19] CASSANDRAS, C. G., LAFORTUNE, S., *Introduction to Discrete Events Systems*. 2 ed. New York, NY : USA, Springer, 2008.

[20] HOPCROFT, J., MOTWANI, R., ULLMAN, J., *Introduction to Automata Theory, Languages, and Computation*. 3 ed. 2006.

[21] MURATA, T., "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, v. 77, pp. 541–580, 1989.

[22] PETERSON, J. L., *Petri net theory and the modeling of systems*. Upper Saddle River, NJ, Prentice Hall, 1981.

[23] DAVID, R., ALLA, H., "Petri nets for Modeling of Dynamic Systems - A Survey", *Automatica*, v. 30, pp. 175–202, 1995.

[24] MOREIRA, M. V., BASILIO, J. C., "Bridging the Gap Between Design and Implementation of Discrete-Event Controllers", *IEEE Transactions on Automation Science and Engineering*, v. 11, pp. 48–65, 2014.

[25] VALIANT, G. L., "Graph-theoretic properties in computational complexity", *Journal of Computer and System Sciences*, v. 13, pp. 278–285, 1976.

[26] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al., *Introduction to Algorithms*. 2 ed. 2007.

[27] DAIGLE, M. J., KOUTSOUKOS, X. D., BISWAS, G., "An event-based approach to integrated parametric and discrete fault diagnosis in hybrid systems", *Transactions of the Institute of Measurement and Control*, v. 22, pp. 487–510, 2010.

[28] BAYOUDH, M., TRAVE-MASSUYES, L., "Diagnosability analysis of hybrid systems cast in a discrete-event framework", *Discrete Event Dynamic Systems: Theory and Applications*, v. 24, pp. 309–338, 2014.

[29] DIENE, O., SILVA, E. R., MOREIRA, M. V., "Analysis and Verification of the Diagnosability of Hybrid Systems", *IEEE 53rd Annual Conference on Decision and Control (CDC)*, pp. 1–6, 2014.

[30] LYGEROS, J., *Lecture Notes on Hybrid Systems*, Lecture notes, University of Patras, 2004.

[31] ABRAHAM, E., *Modelling and Analysis of Hybrid Systems*, Lecture notes, RWTH Aachen University, 2012.

[32] DOS REIS SILVA, E. A., *Diagnosticabilidade de Falhas em Sistemas Híbridos*, Dissertação de mestrado, UFRJ, COPPE, 2015.

[33] TARJAN, R., "Depth first search and linear graph algorithms", *SIAM Journal of Computer*, v. 1, pp. 146–160, 1972.

[34] CABRAL, F. G., MOREIRA, M. V., DIENE, O., et al., "A Petri Net Diagnoser for Discrete Event Systems Modeled by Finite State Automata", *IEEE Transactions on Automatic Control*, v. 60, pp. 59–71, 2013.

[35] RAMADGE, P. J., WONHAM, W. M., "The control of discrete-event systems", *Proceedings of the IEEE*, v. 77, pp. 81–98, 1989.

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

[37] VAN DER SCHAFT, A., SCHUMACHER, H., "An Introduction to Hybrid Dynamical Systems", *London Berlin Heidelberg: Springer-Verlag*, v. 2, 2000.

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

[39] SANTORO, L. P. M., *Obtenção de Bases Mínimas para o Diagnóstico de Falhas de Sistemas a Eventos Discretos Utilizando Verificadores*, Dissertação de mestrado, UFRJ, COPPE, 2013.

[40] DE OLIVEIRA CABRAL, F. G., *Diagnóstico Online de Falhas em Sistemas a Eventos Discretos Modelados por Autômatos Finitos: Uma Abordagem Utilizando Redes de Petri*, Dissertação de mestrado, UFRJ, COPPE, 2014.