CODIAGNOSTICABILIDADE DE SISTEMAS A EVENTOS DISCRETOS COM OBSERVAÇÃO DINÂMICA

Wesley Rodrigues Silveira

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

Orientador: Marcos Vicente de Brito Moreira

Rio de Janeiro
Julho de 2017

CODIAGNOSTICABILIDADE DE SISTEMAS A EVENTOS DISCRETOS COM OBSERVAÇÃO DINÂMICA

Wesley Rodrigues Silveira

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. João Carlos dos Santos Basílio, Ph.D

Prof. Lilian Kawakami Carvalho, D.Sc.

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

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2017

Silveira, Wesley Rodrigues

Codiagnosticabilidade de Sistemas a Eventos Discretos com Observação Dinâmica/Wesley Rodrigues Silveira. – Rio de Janeiro: UFRJ/COPPE, 2017.

XVII, 74 p.: il.; 29, 7cm.

Orientador: Marcos Vicente de Brito Moreira

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2017.

Referências Bibliográficas: p. 68 – 71.

1. Diagnosticabilidade. 2. Codiagnosticabilidade. 3. Sistemas a eventos discretos. 4. Observação dinâmica. 5. Verificadores. I. Moreira, Marcos Vicente de Brito. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Dedico esse trabalho ao Espírito Santo, meu amigo, sem o qual seria impossível a conclusão. Como disse Jesus:

"Todavia digo-vos a verdade, que vos convém que eu vá; porque, se eu não for, o Consolador não virá a vós, mas se eu for, enviar-vo-lo-ei."
João 16:7

Agradecimentos

Agradeço, antes de tudo, a ti Senhor da minha vida, meu Deus e meu Pai. Agradeço-te por ter me enchido de força para chegar até aqui e por nunca me desamparar, pois "os que confiam em ti não ficarão desapontados"[1]. Nos momentos difíceis me deste vitória e me fez maior que os meus inimigos. A ti toda glória e louvor.

Agradeço a minha família pelo seu apoio incondicional. Agradeço com todo meu amor a minha mãe Sueli, que sempre me apoiou e me deu forças nos momentos difíceis. A minha irmã Suellen e meu cunhado Rodrigo, que sempre me alegraram e me fizeram rir nas dificuldades. Agradeço também pelas vezes que me tirarem da frente dos livros para o meu próprio bem :). Agradeço ao meu pastor e amigo Rafael e aos demais amigos Rafaela e Alan, que sempre estiveram perto.

Agradeço aos meus amigos do LCA, Tiago, Públio, Felipe e Juliano, pelas conversas, ajudas e papos descontraídos que animavam minhas idas ao laboratório. Agradeço especialmente ao professor Marcos Vicente, meu orientador, de quem veio a ideia deste trabalho. Agradeço pelo amplo apoio na escrita e nas ideias, mesmo cheio de responsabilidades com a sua ida para França. Deixo aqui meu sincero agradecimento e o meu desejo de que o senhor tenha uma excelente estadia em terras estrangeiras.

[1] Isaías 49.23

"Não a nós, Senhor, não a nós, mas ao teu nome dá glória, por amor da tua benignidade e da tua verdade."

Salmos 115:1

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

CODIAGNOSTICABILIDADE DE SISTEMAS A EVENTOS DISCRETOS COM OBSERVAÇÃO DINÂMICA

Wesley Rodrigues Silveira

Julho/2017

Orientador: Marcos Vicente de Brito Moreira

Programa: Engenharia Elétrica

Neste trabalho é proposto um novo algoritmo para verificação da codiagnosticabilidade de uma linguagem em sistemas a eventos discretos (SED) com observação dinâmica.

Este problema da codiagnosticabilidade em SEDs com observação dinâmica é abordado por outros trabalhos na literatura. Cada um dos métodos existentes atualmente na literatura capazes de lidar com esse problema são abordados e comparados com o método aqui proposto. Essa comparação valida os resultados, uma vez que esse novo verificador, em geral, possui um menor custo computacional do que os outros, apesar de todos terem complexidade polinomial.

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

CODIAGNOSABILITY OF DISCRETE EVENT SYSTEMS WITH DYNAMIC OBSERVATION

Wesley Rodrigues Silveira

July/2017

Advisor: Marcos Vicente de Brito Moreira

Department: Electrical Engineering

In this work, a new algorithm for verifying the codiagnosticability of a language in discrete event systems (DES) with dynamic observation is proposed.

Codiagnosticability in DESs with dynamic observation has been addressed by other studies in the literature. Each of the methods currently available in the literature capable of dealing with this problem are considered and compared with the method proposed here. This comparison validates the results, since this new verifier, in general, has a lower computational cost than the others, although all have polynomial complexity.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

A	Conjunto de agentes locais, p. 21
$Ac(\cdot)$	Parte acessível, p. 11
$CoAc(\cdot)$	Parte coacessível, p. 11
G	Autômato determinístico de estados finitos, p. 9
G^{NF}	Autômato *F-livre*, p. 33
G_F	Autômato que modela o comportamento de falha do sistema, p. 23
G_N	Autômato que modela o comportamento normal do sistema, p. 22
G_V	Autômato verificador, p. 22
G_i	Autômato que representa o autômato G sob a projeção P_{Ω_i}, isto é, observação para o agente i, p. 32
G_{FI}	Autômato que modela o comportamento de falha do sistema acrescido de índices, p. 49
G_{VI}	Verificador Estendido, p. 51
$I_i(\cdot)$	Função de indexação, p. 32
$I_p(\cdot)$	Função de pseudo-indexação, p. 36
$I_{nd}(\cdot)$	Função de índices, p. 48
L	Linguagem, p. 5
L/t	Pós Linguagem de L após a sequência t, p. 7
L^*	Fecho de Kleene de uma linguagem L, p. 6
L_N	Linguagem normal do sistema, p. 20

$O(\cdot)$	Notação assintótica, p. 18
$Obs(G, \Sigma_o)$	Automato observador de G com relação a Σ_o, p. 17
$P(\cdot)$	Projeção natural, p. 7
$P^{-1}(\cdot)$	Projeção inversa, p. 8
$P_{\Omega_i}(\cdot)$	Projeção no contexto da observação dinâmica, p. 31
$R(\sigma, c_i(x))$	Alcance de um *cluster*, p. 35
$R_I^{-1}(\cdot)$	Função de renomeação inversa, p. 55
$TR(G)$	Conjunto das transições do autômato G, p. 30
$Trim(\cdot)$	Operação Trim, p. 12
$UR(\cdot)$	Alcance não observável, p. 16
V	Autômato *T-Verifier*, p. 42
X	Conjunto de estados, p. 9
$X^F_{c_p(x)}$	Conjunto de estados do *cluster* $c_p(x)$ alcançados por sequências com eventos de falha, p. 37
$X^{NF}_{c_p(x)}$	Conjunto de estados do *cluster* $c_p(x)$ alcançados por sequências sem eventos de falha, p. 37
X_m	Conjunto de estados marcados, p. 9
$\Gamma(\cdot)$	Função de eventos ativos, p. 9
Ω_i	Conjunto das transições observáveis, p. 31
Σ	Conjunto de eventos, p. 5
Σ^*	Fecho de Kleene de um conjunto de eventos Σ, p. 5
Σ_f	Conjunto dos eventos de falha, p. 20
Σ_o	Conjunto de eventos observáveis, p. 15
Σ_{uo}	Conjunto de eventos não observáveis, p. 15
$\emptyset$	Conjunto vazio, p. 5
$\mathbb{N}$	Conjunto dos números naturais, p. 18

$\mathbb{R}$	Conjunto dos números reais, p. 18
$\mathcal{C}_p^{F-cycle}$	Conjunto de *cluster* cujos elementos $c_p(x)$ contém ciclos com eventos de falha ou ciclos alcançados por sequências de falha, p. 37
$\mathcal{C}_i^{NF}$	Conjunto dos autômatos *clusters* para G^{NF}, p. 35
$\mathcal{C}_p^{cycle}$	Conjunto de *cluster* cujos elementos $c_p(x)$ contém ciclos, p. 37
$\mathcal{C}_i$	Conjunto dos autômatos *clusters* para G, p. 32
$\mathcal{C}_p$	Conjunto dos autômatos *clusters* para G sob a função de pseudo-indexação, p. 36
$\mathcal{L}(\cdot)$	Linguagem gerada por um autômato, p. 10
$\mathcal{L}_m(\cdot)$	Linguagem marcada por um autômato, p. 10
$\omega_i(\cdot)$	Mapeamento da observação, p. 30
$\overline{L}$	Fecho de Prefixo de uma linguagem L, p. 6
$\overline{\Gamma}(\cdot)$	Função de transição do autômato *C-Verifier*, p. 37
σ	Evento genérico, p. 9
σ_f	Evento de falha, p. 20
$\theta_i(\cdot)$	Projeção, p. 44
$\theta_{I_0}(\cdot)$	Projeção, p. 55
$\theta_{I_i}(\cdot)$	Projeção, p. 55
$\tilde{G}_{Ni}$	Autômato que modela o comportamento normal do sistema, levando em consideração a observação dinâmica de cada agente, p. 50
ε	Sequência vazia, p. 5
$c_i(x)$	Autômato *cluster*, p. 32
$c_p(x)$	Autômato *clusters* com relação a função de pseudo-indexação, p. 36
$f(\cdot)$	Função de transição, p. 9
s	Sequência genérica, p. 5

t	Sequência genérica, p. 5
x	Estado genérico, p. 9
x_0	Estado inicial, p. 9
y	Estado genérico, p. 16

Capítulo 1

Introdução

1.1 Motivação

A automatização de trabalhos anteriormente realizados pelo homem tem se tornado algo cada vez mais frequente em nossos dias. Com essa crescente substituição da mão de obra humana por sistemas automatizados, pode-se destacar melhorias como: aumento da produtividade, redução de custos e melhoria da qualidade. Um dos motivos que levam à essa melhoria de qualidade, vem pela diminuição das falhas que anteriormente eram causadas por ações humanas. Vale ressaltar que, mesmo com a utilização de sistemas automatizados, as falhas são factíveis e nenhum sistema é invulnerável à ocorrência delas. Daí, a necessidade de detectá-las e isolá-las, ou seja, de diagnosticá-las.

1.2 Objetivo

Alguns sistemas automatizados têm como característica a não dependência temporal, e a sua evolução se dá, a partir da ocorrência de eventos. Exemplos de eventos são a borda de subida de um sensor, a finalização de um processo ou ainda um botão sendo pressionado. Sistemas que são dirigidos por eventos são denominados sistemas a eventos discretos (SED). A modelagem utilizada para o controle clássico, isto é, modelagem baseada em equações diferenciais não é a mais adequada para analisar SEDs. Assim, necessitam-se de técnicas para poder representá-los melhor.

Os dois formalismos mais utilizados atualmente para se modelar sistemas a eventos discretos são as redes de Petri e os autômatos. As redes de Petri possuem a característica de ter o estado do sistema distribuído pela estrutura da rede [1]. Essa característica faz com que a rede de Petri seja vantajosa para sistemas com um grande número de estados. Os autômatos possuem o conceito de estado concentrado, o que facilita a sua utilização quando o objetivo é o diagnóstico de falha

[1–15].

A falha em SEDs é representada pela ocorrência de um evento não observável, ou seja, a ocorrência da falha não pode ser detectada e transmitida a terceiros. Outra característica das falhas em SEDs é que a sua ocorrência, não causa a parada imediata do sistema. Sendo assim, para se detectar uma falha é necessário observar a ocorrência dos eventos após a falha, e ser capaz de informar que a falha ocorreu.

Os estudos sobre diagnóstico de falhas utilizam, na sua maioria, modelos baseados em autômatos, apesar de modelos com redes de Petri também poderem ser utilizados [16, 17]. O problema da abordagem com redes de Petri é que esses métodos não atendem todas as classes de redes de Petri [18].

O problema de diagnóstico de falha foi inicialmente proposto em [19]. Em seguida, em [2] foram apresentadas as condições necessárias e suficientes para o diagnóstico de falhas em SEDs, assim como, a construção do diagnosticador, que permite tanto verificar se um sistema é diagnosticável ou não, como o seu diagnóstico em tempo real.

O diagnóstico de falhas em sistemas a eventos discretos pode ser executado utilizando-se arquitetura centralizada ou descentralizada. Na arquitetura centralizada há apenas um diagnosticador central que irá inferir se houve ou não a falha com base na ocorrência dos eventos observáveis executados pela planta. Já na arquitetura descentralizada, existem diversos diagnosticadores locais, cada um com um conjunto de eventos observáveis.

Em [6] foram propostos vários protocolos para diagnose descentralizada. O protocolo 3 que descreve a arquitetura descentralizada disjuntiva será utilizado nesse trabalho. A arquitetura descentralizada disjuntiva considera que não há troca de informação entre os diagnosticadores locais e a falha será diagnosticada quando ao menos um dos agentes locais informar a ocorrência da falha.

Os sistemas propostos nos trabalhos [2–7, 9–15] consideram o problema de diagnóstico considerando observação estática. Entretanto, em muitas situações, a observação de eventos pode não ser fixa como, por exemplo, em problemas de ativação de sensor, em que cada agente pode ligar ou desligar seus sensores dinamicamente durante a evolução do sistema com base em seu histórico de observação, seja por uma questão de restrição de energia, ou largura de banda, ou mesmo de segurança [20–23]. Uma outra situação em que a observação pode ser dinâmica ocorre quando há comunicação entre diferentes agentes, isto é, a comunicação entre agentes pode levar um evento, que é observado na ocorrência de uma transição, não ser observado na ocorrência de uma transição diferente [24, 25].

A diagnosticabilidade de falhas sob observação dinâmica já foi investigada em trabalhos anteriores [20, 21, 26, 27], em que os trabalhos [20] e [21] abordam o caso centralizado e os trabalhos [26] e [27] o caso descentralizado.

O objetivo deste trabalho é apresentar um novo verificador capaz de diagnosticar uma falha, considerando que o sistema trabalha com observação dinâmica. Esse novo verificador será baseado no verificador para observação estática apresentado em [15], uma vez que este é o verificador que apresenta a menor complexidade computacional. Como consequência, esse novo verificador, em geral, apresenta um menor custo computacional do que os outros, apesar de todos terem complexidade polinomial. Para se obter este resultado, esse novo verificador será comparado com os verificadores apresentados em [26] e [27], por serem os verificadores, encontrados na literatura, que visam solucionar o mesmo problema.

1.3 Estrutura

Este trabalho está organizado da seguinte maneira: no capítulo 2 serão apresentados os fundamentos teóricos de sistemas a eventos discretos e como representá-los a partir de autômatos. Além disso, será apresentada uma maneira de se calcular a complexidade computacional de algoritmos. Por fim, será apresentado o diagnóstico de falhas para sistemas a eventos discretos.

No capítulo 3 será apresentada a diferença entre sistemas a eventos discretos com observação estática e com observação dinâmica. Também será apresentado o diagnóstico de falhas em sistemas a eventos discretos com observação dinâmica. Por fim, serão apresentados os verificadores encontrados atualmente na literatura capazes de verificar a diagnosticabilidade de um sistema com observação dinâmica.

No capítulo 4 será apresentado um novo verificador capaz de verificar a diagnosticabilidade de um sistema com observação dinâmica. Além disso, será apresentada a sua complexidade computacional. Por fim, será feita uma comparação entre os verificadores apresentados no capítulo 3, encontrados atualmente na literatura, com o verificador apresentado no capítulo 4.

No capítulo 5 serão apresentadas as conclusões gerais do trabalho, bem como sugestões para trabalhos futuros.

Capítulo 2

Fundamentos Teóricos

Neste capítulo serão apresentadas as bases teóricas para o entendimento deste trabalho, começando pelos sistemas a eventos discretos, linguagens, autômatos, complexidade computacional e diagnóstico de falhas.

2.1 Sistemas a Eventos Discretos

Sistemas a eventos discretos são sistemas que, em geral, não dependem da passagem do tempo para que haja a mudança de um estado para outro, pois essa mudança será feita com a ocorrência de eventos. Esses eventos são por natureza discretos como, por exemplo, a borda de subida de um sensor ou o fim de uma determinada tarefa.

De um modo geral, um SED é constituído basicamente de dois elementos, estados e eventos. O estado de um sistema dinâmico é definido a seguir.

Definição 2.1. *O estado de um sistema dinâmico, em um instante de tempo t_0, é o conjunto de informações que junto do conhecimento da entrada do sistema, para todo $t \geq t_0$, é suficiente para determinar unicamente a resposta do sistema, para todo $t \geq t_0$ [1].*

Uma definição mais intuitiva para evento seria dada como: a percepção de estímulos pelo sistema em questão. Em termos mais formais e utilizando-se do conceito de estados tem-se a seguinte definição de eventos.

Definição 2.2. *Um evento é uma ocorrência instantânea que pode promover uma mudança no estado do sistema.*

Algo que deve ser salientado é a questão do evento ser caracterizado por ocorrências abruptas e instantâneas, e ao perceber o evento, o sistema pode se acomodar num novo estado, até que ocorra um novo evento. Assim, a simples passagem do tempo não é suficiente para garantir a evolução do sistema sendo, para isso, necessária a ocorrência de eventos [28].

Com as informações e definições acima, é possível definir um SED da seguinte forma.

Definição 2.3. *Um Sistema a eventos discretos é um sistema dinâmico que evolui de acordo com a ocorrência abrupta de eventos físicos, em intervalos de tempo, em geral, irregulares e desconhecidos [28].*

2.2 Linguagens

Uma das maneiras de se estudar o comportamento de SED é por intermédio da teoria de linguagens e autômatos [1]. Para se estudar SED a partir dessas ferramentas deve-se ter em mente que todo SED tem um conjunto de eventos e que neste trabalho esse conjunto será denotado por Σ. Os eventos desse conjunto são denominados o alfabeto do sistema e a sequência desses eventos são denominados palavras ou simplesmente sequências. O conjunto dessas palavras gera a linguagem que o sistema pode realizar. A definição formal de linguagem é dada a seguir.

Definição 2.4. *Uma linguagem L definida em um conjunto de eventos Σ é um conjunto de sequências de eventos de comprimento finito formadas a partir dos eventos em Σ.*

O comprimento de uma sequência de eventos é o número de eventos da sequência, contando as múltiplas ocorrências de um mesmo evento. O comprimento de uma sequência s será denotado por $||s||$.

A sequência que não possui eventos é chamada de sequência vazia e é denotada por ε, cujo comprimento é zero e é o elemento neutro da concatenação, isto é, uma sequência $s = ab$ é igual a $\varepsilon ab = a\varepsilon b = ab\varepsilon$ e o comprimento de s é $||s|| = ||ab|| = ||ab\varepsilon|| = 2$. A concatenação é a operação mais básica entre sequências e eventos, por exemplo, $s = ab$ é formado pela concatenação do evento a com o evento b, formando ab, mas pode-se também concatenar s com $s_1 = bc$ formando assim a sequência $s_2 = ss_1 = abbc$.

A linguagem $L = \{\varepsilon, a, ab, acb\}$ é um exemplo de linguagem formada por quatro sequências, incluindo a sequência vazia, geradas a partir do conjunto de eventos $\Sigma = \{a, b, c, d\}$. Os conjuntos $\emptyset$, o próprio Σ e Σ^* também são exemplos de linguagens, em que o conjunto Σ^* é formado por todas as sequências de comprimento finito formados com eventos de Σ mais a sequência vazia ε; a operação * é chamada de fecho de Kleene.

Seja $s = abc$, com $a, b, c \in \Sigma$. Então, a é chamada de prefixo de s, b é chamada de subsequência de s e c é chamada de sufixo de s. Por fim a notação s/t é utilizada para se referir ao sufixo após a ocorrência de t, em que $t \in \Sigma^*$.

2.2.1 Operações com Linguagens

As operações básicas para conjuntos são aplicáveis às linguagens, uma vez que essas são representadas matematicamente por conjuntos. Outras operações também são comumente definidas para linguagens como pode ser visto a seguir.

Concatenação

Como mencionado a concatenação é a operação mais básica entre eventos e sequências, e a mesma pode também ser aplicada às linguagens.

Definição 2.5. *Sejam L_a e $L_b \subseteq \Sigma^*$ então a concatenação entre L_a e L_b é definida por:*

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

A concatenação de duas linguagens será a concatenação das sequências da linguagem L_a com todas as sequências de L_b.

Fecho de Prefixo

Além da concatenação existe a operação fecho de prefixo de uma linguagem L, denotada por $\overline{L}$.

Definição 2.6. *Seja $L \subseteq \Sigma^*$, então o fecho de prefixo de L é definido por:*

$$\overline{L} = \{s \in \Sigma^* : (\exists t \in \Sigma^*) \wedge (st \in L)\}.$$

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

O fecho de prefixo de uma linguagem L é dado por todos os prefixos de cada sequência que a compõe. Pode-se observar que $L \subseteq \overline{L}$.

Fecho de Kleene

O fecho de Kleene de uma linguagem L é mais uma das possíveis operações feitas com linguagens e é denotada por L^*.

Definição 2.7. *Seja $L \subseteq \Sigma^*$, então o fecho de Kleene de L é definido por:*

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

O fecho de Kleene de uma linguagem L é o conjunto formado pela concatenação de um número finito de elementos de L.

Note que a operação fecho de Kleene é idempotente, isto é, $(L^*)^* = L^*$.

Pós-Linguagem

A Pós-linguagem de uma linguagem L após uma sequência t será denotada por L/t.

Definição 2.8. *Seja $L \subseteq \Sigma^*$ e $t \in L$ então a pós-linguagem de L após t denotado por L/t e é definida como:*

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

A Pós-linguagem de uma linguagem L após uma sequência t é o conjunto de todas as sequências formadas a partir de t que pertençam à linguagem. Por definição $L/t = \emptyset$ se $t \notin \overline{L}$.

O Exemplo 2.1 ilustra as operações já descritas.

Exemplo 2.1. *Seja o conjunto de eventos $\Sigma = \{a, b, c\}$, e as linguagens $L_1 = \{\varepsilon, a, abc\}$ e $L_2 = \{a, c\}$, então, tem-se que $L_1L_2 = \{a, c, aa, ac, abca, abcc\}$. Pode-se perceber que ambas as linguagens não são prefixo-fechadas, pois $ab \notin L_1$ e $\varepsilon \notin L_2$. Logo $\overline{L_1} = \{\varepsilon, a, ab, abc\}$ e $\overline{L_2} = \{\varepsilon, a, c\}$. O fecho de Kleene de L_1 e L_2 será $L_1^* = \{\varepsilon, a, abc, aa, aabc, abca, abcabc, \dots\}$ e $L_2^* = \{\varepsilon, a, c, aa, ac, ca, cc, \dots\}$, respectivamente e, por fim, $L_1/a = \{\varepsilon, bc\}$ enquanto que $L_2/b = \emptyset$.*

Projeção

Outra operação com linguagens usualmente utilizada é a *projeção natural* ou simplesmente *projeção*, denotada por P.

Definição 2.9. *Seja Σ_l e Σ_s conjuntos de eventos, tais que, $\Sigma_s \subset \Sigma_l$, então a projeção $P : \Sigma_l^* \to \Sigma_s^*$ definida de modo recursivo para sequências é:*

$$P(\varepsilon) = \varepsilon$$

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

$$P(s\sigma) = P(s)P(\sigma), \text{ para todo } s \in \Sigma_l^*, \ \sigma \in \Sigma_l.$$

A definição de projeção para linguagem estendida a partir da projeção para sequências, em que $L_l \subseteq \Sigma_l^*$, é dada por:

$$P(L_l) = \{P(s) : s \in L_l\}.$$

Como pode se ver na definição de projeção de uma sequência, após aplicá-la a uma sequência do conjunto maior, ter-se-á como resultado uma sequência com

apenas os eventos do conjunto menor. Já a projeção aplicada a uma linguagem nada mais é do que aplicar à projeção a cada sequência da linguagem. Com isso, a projeção pode ser utilizada para representar a linguagem observada em um sistema, quando este tem parte dos seus eventos não observáveis, isto é, o conjunto maior seria todo o conjunto de eventos do sistema e o menor seria a conjunto de eventos observáveis. Logo, a projeção da linguagem desse sistema daria apenas a linguagem observada pelo mesmo.

É possível observar que duas sequências distintas podem se confundir em um sistema que possui apenas parte dos seus eventos observáveis, uma vez que duas sequências distintas podem possuir a mesma projeção.

Projeção Inversa

Além da projeção sobre linguagens há também a projeção inversa sobre linguagens.

Definição 2.10. *Sejam Σ_l e Σ_s conjuntos de eventos tais que $\Sigma_s \subset \Sigma_l$, então, a projeção inversa $P^{-1} : \Sigma_s^* \to 2^{\Sigma_l^*}$ sobre uma sequência, é definida como:*

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

A projeção inversa para linguagens, em que $L_s \subseteq \Sigma_s^*$, é dada por:

$$P^{-1}(L_s) = \{s \in \Sigma_l^* : \exists t \in L_s \text{ tal que } P(s) = t\}.$$

Dada uma sequência $t \in \Sigma_s^*$, então a projeção inversa de t será o conjunto formado por todas as sequências de Σ_l^* que têm a mesma projeção de t. Já a projeção inversa sobre L_s será o conjunto formado pela união de todas as sequências que tenham a mesma projeção que cada elemento de L_s.

Exemplo 2.2. *Seja $\Sigma_l = \{a, b, c\}$ e sejam $\Sigma_{s1} = \{a, b\}$ e $\Sigma_{s2} = \{b, c\}$. Dado $L_l = \{a, abc, ccbc, abbcbcca\} \subset \Sigma_l^*$, e definindo-se:*

$$P_1 : \Sigma_l^* \to \Sigma_{s1}^*,$$

$$P_2 : \Sigma_l^* \to \Sigma_{s2}^*.$$

Então,

- $P_1(L_l) = \{a, ab, b, abbba\}$;
- $P_2(L_l) = \{\varepsilon, bc, ccbc, bbcbcc\}$;
- $P_1^{-1}(\varepsilon) = \{c\}^*$;

- $P_1^{-1}(\{a\}) = \{c\}^*\{a\}\{c\}^*$;
- $P_2^{-1}(\{bc, b\}) = \{a\}^*\{b\}\{a\}^*\{c\}\{a\}^* \cup \{a\}^*\{b\}\{a\}^*$.

2.3 Autômatos

Um automato é um dispositivo capaz de representar uma linguagem de acordo com regras bem definidas [1]. Os autômatos podem ser determinísticos ou não determinísticos. A definição formal de um autômato determinístico é apresentada a seguir:

Definição 2.11. *Um autômato determinístico, denotado por G, é uma sêxtupla:*

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

sendo X o conjunto de estados, Σ o conjunto de eventos, $f : X \times \Sigma \to X$ a função de transição, $\Gamma : X \to 2^{\Sigma}$ a função de eventos ativos (viáveis), $x_0 \in X$ o estado inicial e $X_m \subseteq X$ o conjunto de estados marcados.

Além dessas regras, os autômatos possuem uma representação gráfica, em que os vértices são representado por círculos, e as arestas por arcos direcionados. Os estados são associados aos círculos e os eventos aos arcos. Os estados marcados são representados por dois círculos concêntricos.

O Exemplo 2.3 representa o grafo de um autômato.

Exemplo 2.3. *Seja G o automato representado pelo grafo da figura 2.1. O conjunto dos estados de G é dado por $X = \{0, 1, 2\}$, o conjunto de eventos é dado por $\Sigma = \{a, b, c\}$. A função de transição de estados de G é da seguinte forma: $f(0, a) = 1$, $f(0, c) = 2$, $f(1, a) = 1$, $f(1, b) = 0$, $f(2, a) = 1$ e $f(2, b) = 1$. Note que a função de transição não está definida para $f(0, b)$, $f(1, c)$ e $f(2, c)$, daí a função é dita ser parcial em seu domínio. A função de eventos ativos de G é da seguinte forma: $\Gamma(0) = \{a, c\}$, $\Gamma(1) = \{a, b\}$ e $\Gamma(2) = \{a, b\}$. O estado inicial é identificado por uma seta e, neste exemplo, $x_0 = 0$. Por fim, o conjunto de estados marcados é $X_m = \{1\}$.*

Usualmente se estende o domínio da função de transição como sendo $f : X \times \Sigma^* \to X$ ao invés de $f : X \times \Sigma \to X$ e sua lei de formação é definida de forma recursiva como mostrado a seguir:

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

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

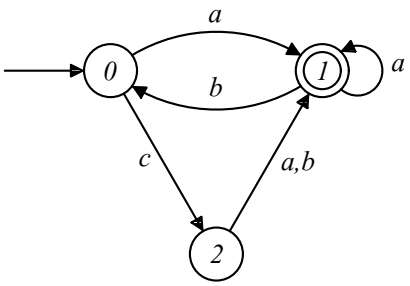


Figura 2.1: Grafo do autômato G.

Observação 2.1. *Um autômato é dito ser determinístico quando para todo estado $x \in X$ e para todo evento $\sigma \in \Gamma(x)$, existe um único estado $y \in X$ tal que $f(x, \sigma) = y$. Note que, se um autômato for não determinístico, $f(x, \sigma)$ poderá levar não só a um novo estado, mas sim a um conjunto de estados. Essa diferença pode ser ressaltada a partir da função de transição de um autômato, em que $f : X \times \Sigma \to X$ denota a função de transição para um autômato determinístico enquanto que $f : X \times \Sigma \to 2^X$ para um autômatos não determinísticos.*

Um autômato possui duas linguagens associadas a ele, que são a linguagem gerada e linguagem marcada. As definições a seguir descrevem matematicamente cada uma delas.

Definição 2.12. *A linguagem gerada por um autômato G é dada por:*

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

Pode-se notar que $\mathcal{L}(G) = \overline{\mathcal{L}(G)}$, isto é, a linguagem gerada por um autômato é prefixo-fechada e $\mathcal{L}(G) = \Sigma^*$ quando a função de transição é definida em todo o seu domínio.

Definição 2.13. *A linguagem marcada por um autômato G é dada por:*

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

A linguagem $\mathcal{L}_m(G)$ não é necessariamente prefixo-fechada e, evidentemente $\mathcal{L}_m(G) \subseteq \mathcal{L}(G)$.

Associa-se também aos autômatos as definições de caminho e caminho cíclico que serão usadas neste trabalho.

Definição 2.14. *Caminho é um sequência formada por estados e eventos:*

$$c = (x_1, \sigma_1, x_2, \sigma_2, \ldots, \sigma_{k-1}, x_k)$$

em que $x_1, x_2, \ldots, x_k \in X$, $\sigma_1, \sigma_2, \ldots, \sigma_{k-1} \in \Sigma$ *e* $x_{i+1} = f(x_i, \sigma_i)$ *para* $i = 1, \ldots, k-1$.

Um caminho $(x_1, \sigma_1, x_2, \sigma_2, \ldots, \sigma_{k-1}, x_k)$ *é dito ser cíclico se* $x_k = x_1$.

2.3.1 Operações com Autômatos

Um sistema pode ser representado por um autômato ou pela combinação de diversos autômatos que modelam cada parte individualmente desse sistema. Para se combinar essas diversas partes modeladas individualmente existem as operações de composição, que são a composição produto e a composição paralela. Além delas existem algumas operações que são aplicáveis apenas sobre um autômato, que são chamadas de operações unárias. A seguir será definida cada uma dessas operações começando pelas unárias e posteriormente as de composição.

Parte Acessível

A parte acessível de um autômato G é denotada neste trabalho por $Ac(G)$.

Definição 2.15. *A parte acessível de um autômato* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ *será:*

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

em que $X_{ac} = \{x \in X : \exists s \in \Sigma^* \text{ tal que } f(x_0, s) = x\}$, $f_{ac} : X_{ac} \times \Sigma^* \to X_{ac}$ *é a função de transição,* $\Gamma_{ac} : X_{ac} \to 2^{\Sigma}$ *é a função de eventos ativos (viáveis) e* $X_{m,ac} = X_m \cap X_{ac}$.

A parte acessível de um autômato G resultará no autômato que mantém todos os estados alcançáveis a partir do estado inicial.

Um autômato G que seja igual à sua parte acessível, isto é, $G = Ac(G)$ é dito ser acessível.

O Exemplo 2.4 ilustra a parte acessível de um automato G.

Exemplo 2.4. *Seja* G *o autômato dado pela figura 2.2, a parte acessível será dada pela figura 2.3.*

Parte Coacessível

A parte Coacessível de um autômato G é denotada por $CoAc(G)$.

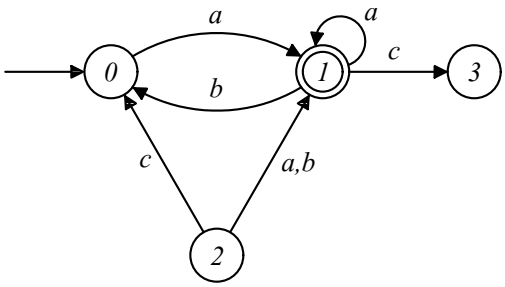


Figura 2.2: Autômato G.

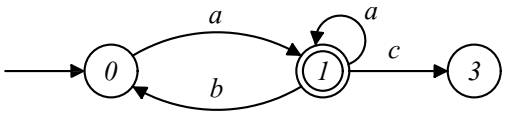


Figura 2.3: Parte acessível do autômato G.

Definição 2.16. *A parte coacessível de um autômato* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ *é dada por:*

$$CoAc(G) = (X_{coac}, \Sigma, f_{coac}, \Gamma_{coac}, x_0, X_m),$$

sendo $X_{coac} = \{x \in X : \exists s \in \Sigma^* \text{ tal que } f(x_0, s) \in X_m\}$, $f_{coac} : X_{coac} \times \Sigma \to X_{coac}$ *a função de transição e* $\Gamma_{coac} : X_{coac} \to 2^{\Sigma}$ *a função de eventos ativos (viáveis).*

$$x_{0,coac} = \begin{cases} x_0, & \text{se } x_0 \in X_{coac}, \\ \text{indefinido}, & \text{caso contrário.} \end{cases}$$

A parte coacessível de um autômato G resultará no autômato que mantém todos os estados que a partir dele se é capaz de alcançar um estado marcado.

A linguagem marcada de G será igual à linguagem marcada pela sua parte coacessível, ou seja, $\mathcal{L}_m(G) = \mathcal{L}_m(CoAc(G))$. Contudo, $\mathcal{L}(CoAc(G))$ pode ser diferente de $\mathcal{L}(G)$, mais especificamente, $\mathcal{L}(CoAc(G)) \subseteq \mathcal{L}(G)$.

Um autômato G que é igual à sua parte coacessível, isto é, $G = CoAc(G)$ é dito ser coacessível. Logo, se G é coacessível então $\mathcal{L}(G) = \overline{\mathcal{L}_m(G)}$.

Exemplo 2.5. *Seja G o autômato dado pela figura 2.2. A parte coacessível de G é apresentada na figura 2.4.*

Trim

A operação Trim aplicada a um determinado autômato G é denotada por $Trim(G)$.

Definição 2.17. *A operação Trim aplicada a um autômato* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$

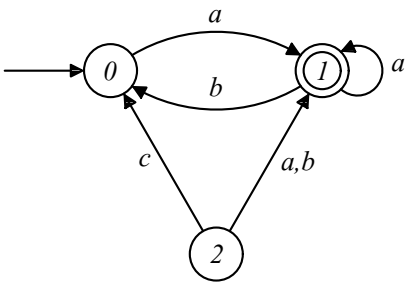


Figura 2.4: Parte coacessível do autômato G.

é definida como:

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

Um autômato G que é acessível e coacessível é dito ser trim, ou seja, $G = Trim(G)$.

Exemplo 2.6. *Seja G o autômato dado pela figura 2.2. Então $Trim(G)$ é apresentada na figura 2.5.*

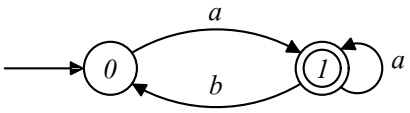


Figura 2.5: Operação trim sobre o autômato G.

Projeção e Projeção Inversa

As operações de projeção e projeção inversa também são operações unárias. A operação de projeção é responsável por gerar o autômato G_P a partir do autômato G, tal que, a linguagem gerada e marcada por G_P seja igual à projeção da linguagem gerada e marcada pelo autômato G, considerando dois conjuntos $\Sigma_s \subset \Sigma_l$. Em outras palavras, a projeção $P : \Sigma_l^* \to \Sigma_s^*$ aplicada sobre a linguagem $L \subseteq \Sigma_l^*$ gerada por G é obtida substituindo-se os eventos de G que fazem parte do conjunto $\Sigma_l \setminus \Sigma_s$ por ε, obtendo-se assim um autômato não determinístico.

A operação de projeção inversa é responsável por gerar o autômato G_{IP} a partir do autômato G, tal que, a linguagem gerada e marcada por G_{IP} seja igual à projeção inversa da linguagem gerada e marcada pelo autômato G. Para isso, basta acrescentar autolaços em cada estado de G com todos os eventos de $\Sigma_l \setminus \Sigma_s$.

Composição Produto

A operação produto é uma composição completamente síncrona, isto é, o autômato resultante só terá eventos que sejam comuns a todos os autômatos envolvidos na operação.

Definição 2.18. *Sejam* $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{0_1}, X_{m_1})$ *e* $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{0_2}, X_{m_2})$ *dois autômatos. Então, a composição produto é definida como:*

$$G_1 \times G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\times 2}, \Gamma_{1\times 2}, (x_{0_1}, x_{0_2}), X_{m_1} \times X_{m_2}),$$

em que

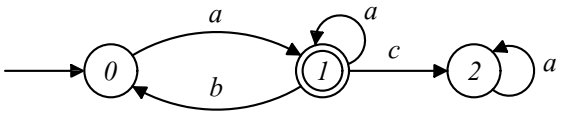
$$f_{1\times 2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), & \text{se } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \text{indefinido}, & \text{caso contrário} \end{cases}$$

e $\Gamma_{1\times 2}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$.

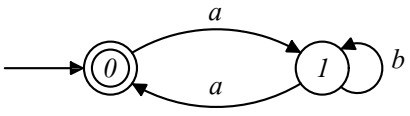
A linguagem gerada pelo autômato resultante da composição produto é a interseção das linguagens de G_1 e G_2, ou seja, $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$. O mesmo ocorre com a linguagem marcada, isto é, $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$.

A composição produto respeita a comutatividade, ou seja, $G_1 \times G_2 = G_2 \times G_1$, a menos de uma reordenação de estados, e a associatividade, logo $(G_1 \times G_2) \times G_3 = G_1 \times (G_2 \times G_3)$.

Exemplo 2.7. *Sejam* G_1 *e* G_2 *os autômatos apresentados nas figuras 2.6a e 2.6b, então a composição produto entre* G_1 *e* G_2 *é representado na figura 2.7.*



(a) Autômato G_1



(b) Autômato G_2

Figura 2.6: Autômatos que compõem um sistema.

Composição Paralela

A composição paralela é uma composição síncrona, isso significa que o autômato resultante sincroniza os eventos em comum que pertencem aos autômatos envolvidos na operação, além de caminhar com os eventos particulares de cada autômato.

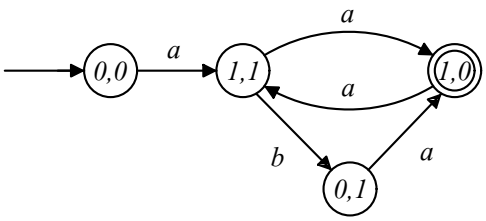


Figura 2.7: Composição produto entre G_1 e G_2.

Definição 2.19. *Sejam* $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{0_1}, X_{m_1})$ *e* $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{0_2}, X_{m_2})$ *dois autômatos. Então, a composição paralela é definida como:*

$$G_1||G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1||2}, \Gamma_{1||2}, (x_{0_1}, x_{0_2}), X_{m_1} \times X_{m_2}),$$

em que

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

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

Sejam $P_1 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_1^*$ e $P_2 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_2^*$. Então, a linguagem gerada pela composição paralela é dada por $\mathcal{L}(G_1||G_2) = P_1^{-1}(\mathcal{L}(G_1)) \cap P_2^{-1}(\mathcal{L}(G_2))$ e a linguagem marcada é $\mathcal{L}_m(G_1||G_2) = P_1^{-1}(\mathcal{L}_m(G_1)) \cap P_2^{-1}(\mathcal{L}_m(G_2))$.

A composição paralela respeita a comutatividade, ou seja, $G_1||G_2 = G_2||G_1$, a menos de uma reordenação de estados, e a associatividade, logo $(G_1||G_2)||G_3 = G_1||(G_2||G_3)$.

Exemplo 2.8. *Sejam* G_1 *e* G_2 *os autômatos apresentados nas figuras 2.6a e 2.6b, então a composição paralelo entre* G_1 *e* G_2 *é representado na figura 2.8.*

2.3.2 SEDs com Observação Parcial de Eventos

Até o momento foi considerado que o conjunto de eventos Σ possui todos os eventos perceptíveis pelo sistema, ou seja, qualquer evento de Σ pode ser observado de alguma maneira pelo sistema, o que é feito na prática através de sensores. O que ocorre é que nem todos os sistemas possuem todos os eventos observáveis, isso quer dizer que o conjunto Σ é particionado em um subconjunto de eventos observáveis, denotado por Σ_o, e outro de eventos não observáveis, denotado por Σ_{uo}, logo $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$.

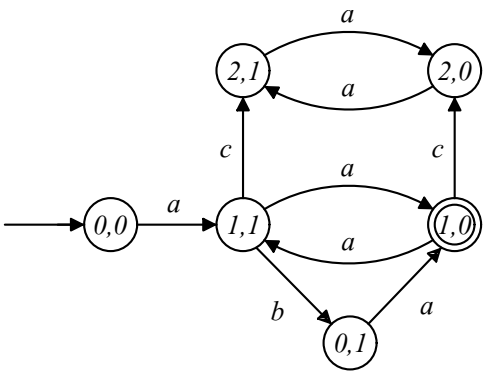


Figura 2.8: Composição paralelo entre G_1 e G_2.

Os eventos considerados como eventos de falha encontram-se no conjunto de eventos não observáveis, uma vez que, se os eventos de falha fossem observáveis, eles seriam trivialmente diagnosticados.

Para se obter a linguagem gerada e marcada observada por um sistema basta aplicar à projeção, $P_o : \Sigma^* \to \Sigma_o^*$ sobre a linguagem do sistema. Caso essa linguagem seja a linguagem gerada por um autômato G poder-se-ia aplicar à operação projeção no autômato, resultando em um autômato não determinístico.

Para se obter um autômato determinístico que possui a mesma linguagem gerada e marcada do autômato com observação reduzida é necessário apresentar o conceito de alcance não observável e de autômato observador.

Definição 2.20. *O alcance não observável de um estado $x \in X$, denotado por $UR(x)$, é definido por:*

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

A definição pode ser estendida para um conjunto de estados $B \subseteq X$ da seguinte forma:

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

Note que o alcance não observável de um estado x é o conjunto de estados alcançados a partir do estado x através de eventos não observáveis incluindo o próprio estado x, enquanto que o alcance para um conjunto de estados B é a união dos alcances não observáveis para cada estado de B.

Definição 2.21. *O observador de um autômato G com relação a um conjunto de eventos observáveis Σ_o, denotado por $Obs(G, \Sigma_o)$, é dado por:*

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

em que $X_{obs} \subseteq 2^X$, f_{obs}, Γ_{obs}, $x_{0,obs}$ *e* $X_{m,obs}$ *são obtidos seguindo os passos do algoritmo 2.1 [1, 29].*

Algoritmo 2.1. *Autômato Observador*

Entrada: *Autômato* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$*, e o conjunto de eventos observáveis* Σ_o*, em que* $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$*.*

Saída: *Autômato observador* $Obs(G, \Sigma_o) = (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, x_{0,obs}, X_{m,obs})$*.*

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

2: $\bar{X}_{obs} = \widetilde{X}_{obs}$*.* $\widetilde{X}_{obs} = \emptyset$*.*

3: Para cada $B \in \bar{X}_{obs}$ *faça*

3.1: $\Gamma_{obs}(B) = \left(\bigcup_{x \in B} \Gamma(x)\right) \cap \Sigma_o$*.*

3.2: Para cada $\sigma \in \Gamma_{obs}(B)$*,*

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

3.3: $\widetilde{X}_{obs} \leftarrow \widetilde{X}_{obs} \cup f_{obs}(B, \sigma)$*.*

4: $X_{obs} \leftarrow X_{obs} \cup \widetilde{X}_{obs}$*.*

5: Repita os passos de 2 a 4 até toda a parte acessível do $Obs(G, \Sigma_o)$ *ser construída.*

6: $X_{m,obs} = \{B \in X_{obs} : B \cap X_m \neq \emptyset\}$

Exemplo 2.9. *Seja* G *o autômato da figura 2.1. Suponha que* $\Sigma_o = \{a, b\}$ *e* $\Sigma_{uo} = \{c\}$*. O autômato observador de* G *obtido a partir do algoritmo 2.1 é mostrado na figura 2.9. Observe que o estado inicial do observador é o conjunto dos estados* $\{0, 2\}$*, uma vez que o evento* c *é não observável. Após a ocorrência dos eventos* a *ou* b*, há a evolução para o estado* 1*. Já no estado* 1*, após a ocorrência de* a*, permanece-se no estado* 1 *e com a ocorrência de* b *volta-se ao estado inicial* $\{0, 2\}$*.*

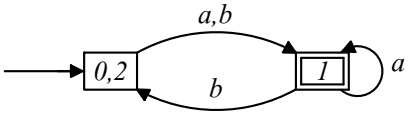


Figura 2.9: Observador de G.

2.4 Complexidade Computacional de um Algoritmo

Para determinar se um determinado algoritmo realiza uma tarefa de maneria mais eficiente do que outro, é necessário avaliar a taxa de crescimento de ambos os algoritmos, também chamada de complexidade computacional. Essa complexidade representa uma estimativa de quanto tempo um algoritmo precisará para processar uma entrada de tamanho n, uma vez que o tempo gasto por um algoritmo cresce com o número de entradas [30].

Tradicionalmente, a eficiência de um algoritmo é determinada considerando-se apenas os termos de maior ordem presentes na expressão matemática que define a eficiência de um algoritmo [30]. Uma vez que a análise é realizada para o pior cenário possível, as constantes e termos de baixa ordem tornam-se irrelevantes ao atribuir-se um valor elevado para o tamanho da entrada n. Essa é a ideia da análise de eficiência assintótica de um algoritmo. A seguir, é apresentada a notação assintótica O usada nesta dissertação para avaliar a complexidade dos algoritmos apresentados [31].

Definição 2.22. *Para uma dada função $g(n)$, denota-se $O(g(n))$ o seguinte conjunto de funções:*

$$O(g(n)) = \{f(n) : \exists\ c > 0\ e\ n_0 \in\ \mathbb{N}\ tal\ que\ 0 \leq f(n) \leq cg(n),\ \forall n \geq n_0\}.$$

A notação assintótica O é usada para atribuir um limitante superior para uma função, ou seja, $f(n) \leq cg(n),\ \forall n \geq n_o$. Diz-se que uma função $f(n) = O(g(n))$ para indicar que a função $f(n)$ pertence ao conjunto $O(g(n))$, ou ainda, $f(n) \in O(g(n))$. Por Exemplo, note que $(an^2 + bn + d) = O(n^2)$ se $c = 7a/4$ e $n_0 = 2max(|b|/a, \sqrt{|d|/a})$, sendo a, b e d constantes e $a > 0$. Quando se diz que a complexidade computacional de um algoritmo é da ordem de $O(g(n))$, quer dizer que, no pior caso, independentemente da forma da entrada de tamanho n, o tempo de execução desse algoritmo está limitado superiormente pelo valor de $g(n)$ e de sua taxa de crescimento. Além disso, é comum referenciar a ordem de crescimento de um algoritmo apenas pela categoria da função $g(n)$. Se um algoritmo possuir complexidade $O(g(n))$ e $g(n)$ for exponencial, por exemplo, $g(n) = a^n$, com $a \in \mathbb{R}$, diz-se que esse algoritmo tem complexidade exponencial. Da mesma forma, se $g(n)$ for um polinômio, por exemplo, $g(n) = n^k$, com $k \in \mathbb{N}$, diz-se que esse algoritmo tem complexidade polinomial [31].

No contexto de computação, o autômato pode ser visto como um grafo $G = (V, E)$, em que V é o conjunto de vértices e E é conjunto de arestas. Essa representação é útil para o cálculo da complexidade, uma vez que os algoritmos presentes nesta dissertação têm como finalidade informar se uma linguagem é diagnosticável

ou não. Para isso, esses algoritmos criam um autômato, denominado verificador, e buscam por caminhos cíclicos que violam a diagnosticabilidade da linguagem. Pela própria construção desses verificadores, essa busca por caminhos cíclicos pode ser simplificada buscando-se por componentes fortemente conexos. O conceito de componentes fortemente conexos é definido a seguir.

Definição 2.23. *O conjunto $U \subseteq V$ é dito ser um componente fortemente conexo se satisfaz às condições:*

(i) Para todo $u, v \in U$ existe um caminho de u para v e de v para u;

(ii) O conjunto U é maximal.

Exemplo 2.10. *Na figura 2.10 é possível perceber que o conjunto de estados $U = \{1, 2, 3\}$ satisfaz à definição de componente fortemente conexo, pois a partir de qualquer estado é possível alcançar qualquer outro do conjunto e o conjunto é maximal. Os conjuntos $\{1, 2\}$, $\{2, 3\}$ e $\{1, 3\}$ apesar de satisfazerem à primeira condição de definição 2.23, não são maximais, pois sempre é possível acrescentar mais um estado que satisfaça à primeira condição.*

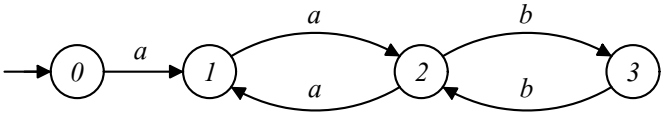


Figura 2.10: Autômato com componente fortemente conexo

A busca por componentes fortemente conexos tem complexidade linear no número de vértices e arestas, isto é, $O(|V| + |E|)$ (vide [30]). Para um autômato os vértices são os estados e as arestas as transições. Como, no pior caso, um autômato determinístico possui $|X| \times |\Sigma|$ transições, então, $|V| + |E|$ corresponde a $|X| + |X| \times |\Sigma|$. Como, $|X| + |X| \times |\Sigma| = |X|(1 + |\Sigma|) \leq 2|X| \times |\Sigma|$, a complexidade do algoritmo da busca por componentes fortemente conexos em um autômato determinístico é $O(|X| \times |\Sigma|)$, em que, $|X| \times |\Sigma|$ é o número de transições do autômato determinístico. Se o autômato for não determinístico, no pior caso, o autômato possui $|X|^2 \times |\Sigma|$ transições, daí, $|V| + |E|$ corresponde a $|X| + |X|^2 \times |\Sigma|$. Como, $|X| + |X|^2 \times |\Sigma| = |X|(1 + |X| \times |\Sigma|) \leq |X|(2|X| \times |\Sigma|)$, a complexidade do algoritmo da busca por componentes fortemente conexo para um autômato não determinístico é $O(|X|^2 \times |\Sigma|)$. Pode-se perceber que $|X|^2 \times |\Sigma|$ é o número de transições do autômato não determinístico. Assim, tanto o autômato determinístico como o não determinístico têm a sua complexidade computacional na busca por componentes fortemente conexos calculada sobre o seu número de transições. Essa informação será útil quando houver a necessidade de se comparar os verificadores apresentados nesta dissertação.

2.5 Diagnóstico de Falha

As falhas em um sistema nem sempre são detectáveis de maneira trivial. Na prática, isso significa que nem sempre é possível monitorar ou implementar um sensor que detecte a falha no exato momento em que ela ocorre. Essa incapacidade de monitorar o sistema pode ocorrer por questões econômicas, isto é, seria muito caro acrescentar um sensor no processo, ou pela própria inexistência de tal sensor.

Para se diagnosticar a ocorrência do evento de falha serão considerados os eventos observáveis subsequentes a esta e tentar inferir a partir desses eventos se a falha ocorreu ou não. Neste trabalho será considerado, sem perda de generalidade, a hipótese de que há apenas um evento de falha, isto é, $\Sigma_f = \{\sigma_f\} \subseteq \Sigma_{uo}$, sendo Σ_f o conjunto dos eventos de falha e σ_f o evento de falha. Essa hipótese é considerada apenas por uma questão de simplicidade, pois no caso da existência de mais de um evento de falha, bastaria analisar a diagnosticabilidade do sistema para cada um desses eventos individualmente, ou seja, considerar um deles como sendo de falha e os demais como eventos comuns não observáveis do sistema, e repetir o procedimento até que todo o conjunto que representa as falhas do sistema tenha sido verificado.

Anteriormente, como por exemplo em [2] e [3], outras hipóteses eram consideradas, tais como:

- A linguagem de G ser viva, isto é, $\Gamma(x) \neq \emptyset$ para todo $x \in X$;
- Não haver caminhos cíclicos compostos somente por eventos não observáveis.

Nos trabalhos mais atuais como em [11] e [15] essas hipóteses não são mais necessárias bastando colocar um autolaço rotulado por evento não observável nos estados em que $\Gamma(x) = \emptyset$. Já nos casos de haver caminhos cíclicos com eventos não observáveis a própria metodologia empregada em [11] e [15], por não utilizar os autômatos observadores, resolve o problema de diagnosticabilidade sem precisar levar em consideração essa hipótese.

Seja L a linguagem gerada por um autômato G e $L_N \subset L$ a linguagem formada por todas as sequências que não possuem um evento de falha, isto é, a linguagem normal do sistema. Então $L \setminus L_N$ é o conjunto formado por todas as sequências que possuem um evento de falha. Com essas definições é possível definir a diagnosticabilidade de uma linguagem.

Definição 2.24. *Seja L a linguagem prefixo fechada gerada por um autômato G e seja $L_N \subset L$ a linguagem prefixo fechada que representa o comportamento normal do sistema. Então, L é diagnosticável com relação à $P_o : \Sigma^* \to \Sigma_o^*$ e Σ_f se*

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

$$(P_o(st) \neq P_o(\omega),\ \forall \omega \in L_N).$$

De acordo com a definição 2.24 a linguagem L é diagnosticável se para toda a sequência de falha, a partir de um dado evento, após a ocorrência da falha, todas essas sequências não se confundem mais com nenhuma sequência normal do sistema em questão.

O diagnóstico de falhas em sistemas a eventos discretos pode ser executado utilizando-se arquiteturas centralizada ou descentralizada. Na arquitetura centralizada há apenas um diagnosticador central que irá inferir se houve ou não a falha baseado na ocorrência dos eventos observáveis executados pela planta. Já na arquitetura descentralizada existem diversos diagnosticadores locais, sendo que cada um com um conjunto de eventos observáveis. Neste trabalho serão considerados m diagnosticadores locais que serão chamados de agentes locais e serão denotados pelo conjunto $A = \{1, \ldots, m\}$.

Em [6] foram propostos vários protocolos para diagnose descentralizada. O protocolo 3 que descreve a arquitetura descentralizada disjuntiva que será utilizada nesse trabalho. A arquitetura descentralizada disjuntiva considera que não há troca de informação entre os diagnosticadores locais e a falha será diagnosticada quando ao menos um dos agentes locais informa a ocorrência da falha. Com essas considerações é possível definir a codiagnosticabilidade de uma linguagem.

Definição 2.25. *Seja L a linguagem prefixo fechada gerada por um autômato G e seja $L_N \subset L$ a linguagem prefixo fechada que representa o comportamento normal do sistema. Então, L é codiagnosticável com relação à $P_{oi} : \Sigma^* \to \Sigma_{o_i}^*$ com $i \in A$ e Σ_f se*

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

$$(\exists i \in A)(P_{oi}(st) \neq P_{oi}(\omega),\ \forall \omega \in L_N).$$

Conforme a definição acima, a linguagem L será codiagnosticável se para toda a sequência de falha, a partir da observação de um dado evento, após a ocorrência da falha, pelo menos um dos agentes locais for capaz de distinguí-las das sequências normais do sistema em questão.

2.5.1 Verificação da Codiagnosticabilidade

Com o objetivo de se diagnosticar a ocorrência de uma falha, a partir da observação dos eventos do sistema, foram propostos diversos trabalhos começando pelo diagnosticador apresentado em [2]. Contudo a verificação da diagnosticabilidade de uma linguagem proposta em [2], tem no pior caso, complexidade exponencial na cardinalidade do espaço de estado, uma vez que se baseia no observador.

Outros trabalhos visando melhorar a complexidade computacional da verificação da (co)diagnosticabilidade, como visto em [11], [15] e de [12, 32–34] foram propostos. Os trabalhos [12] e [32] desenvolveram métodos distintos na arquitetura centralizada mas que ainda precisava das hipóteses da linguagem ser viva e da não existência de ciclos formados por eventos não observáveis. Já os trabalhos [11], [33] e [34] foram desenvolvidos na arquitetura descentralizada e as hipóteses sobre a linguagem ser viva e a não existência de ciclos de eventos não observáveis foram relaxadas.

Dentre todos os trabalhos anteriormente citados o trabalho [15], além de ter a menor complexidade computacional (vide [35]), também não necessita das hipóteses da linguagem ser viva e da não existência de ciclos com eventos não observáveis. Com isso o verificador utilizado como base para o desenvolvimento deste trabalho será o apresentado em [15]. Como última observação quanto as hipóteses, vale ressaltar que caso a linguagem não seja viva, basta acrescentar um autolaço com um evento não observável no estado sem eventos ativos.

O algoritmo para a construção do autômato G_V para a verificação da codiagnosticabilidade proposta em [15] é apresentado abaixo.

Algoritmo 2.2. *Construção de G_V*

Entrada: *Autômato $G = (X, \Sigma, f, \Gamma, x_0, X_m)$, e os conjuntos Σ_f, Σ_{o_i} e $\Sigma_{uo_i} \forall i \in A$.*
Saída: *Existência de caminhos cíclicos no autômato G_V.*

1: *Construa o autômato G_N que modela o comportamento normal de G, da seguinte forma:*

   1.1: *Defina $\Sigma_N = \Sigma \setminus \Sigma_f$;*

   1.2: *Construa o autômato A_N formado por um único estado N (também seu estado inicial) com um autolaço com todos os eventos de Σ_N;*

   1.3: *Construa o autômato que modela o comportamento normal de G, onde $G_N = G \times A_N = (X_N, \Sigma, f_N, \Gamma_N, x_{0_N})$;*

   1.4: *Redefina o conjunto de eventos de G_N como sendo Σ_N, i.e., $G_N = (X_N, \Sigma_N, f_N, \Gamma_N, x_{0_N})$.*

2: *Construa o autômato G_F que modela o comportamento de falha, da seguinte forma:*

   2.1: *Construa o autômato $A_l = (X_l, \Sigma_f, f_l, \Gamma_l, x_{0_l})$, onde $X_l = \{N, F\}$, $x_{0_l} = \{N\}$, $\Gamma_l(N) = \Gamma_l(F) = \sigma_f$, $f_l(N, \sigma_f) = F$ e $f_l(F, \sigma_f) = F$, $\forall\ \sigma_f \in \Sigma_f$;*

   2.2: *Calcule $G_l = G||A_l$ e marque todos os estados de G_l que tem a segunda componente igual a F;*

   2.3: *Calcule G_F tomando a parte coacessível de G_l, i.e., $G_F = CoAc(G_l)$.*

3: Defina a função $R_i : \Sigma_N \to \Sigma_{R_i}$ *como:*

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

3.1: Construa o autômato $G_{Ni} = (X_N, \Sigma_{Ri}, f_{Ni}, \Gamma_{Ni}, x_{0_N})$ *onde:*

- Σ_{Ri} *é o conjunto de eventos de* G_N *renomeados, i.e.,* $\Sigma_{Ri} = R_i(\Sigma_N)$*;*
- $f_{Ni}(x_n, R_i(\sigma)) = f_N(x_n, \sigma)$, $\forall\ \sigma \in \Sigma_N$*;*
- $\Gamma_{Ni}(x_n) = R_i(\Gamma_N(x_n))$, $\forall\ x_n \in X_N$.

4: Construa o autômato verificador G_V *através da composição paralela entre os autômatos* G_{Ni} *e* G_F*, i.e.,* $G_V = G_{N1}||G_{N2}|| \cdots ||G_{Nm}||G_F$.

5: Verifique a existência de caminhos cíclicos $(x_{V_1}, \sigma_1, x_{V_2}, \sigma_2, \ldots, \sigma_{k-1}, x_{V_k})$ *em* G_V *com pelo menos um* $j \in \{1, 2, \ldots, k-1\}$ *tal que* $\sigma_j \in \Sigma$ *e o rótulo de* x_F *em* $x_{V_j} = (x_{N1}, x_{N2}, \ldots, x_{Nm}, x_F)$ *seja igual a* F.

No primeiro passo do algoritmo 2.2 é construído o autômato G_N que tem como linguagem L_N, isto é, todas as sequências que não possuem eventos de falha. Já no segundo passo é construído o autômato G_F que modela o comportamento de falha do sistema e tem como linguagem gerada $\mathcal{L}(G_F) = \overline{L \setminus L_N}$ e como linguagem marcada $\mathcal{L}_m(G_F) = L \setminus L_N$. No passo seguinte, a função de renomeação tem o papel de renomear em G_N apenas os eventos que são não observáveis para o agente em questão, fazendo com que na composição paralela do passo seguinte, esses eventos possam ocorrer sempre que possível, e o autômato G_V sincronize apenas os eventos observáveis comuns a G_{Ni}, para $i \in \{1, \ldots, m\}$, e G_F. Isso fará com que o verificador exiba apenas as sequências de G_F que possuam as mesmas projeções das sequências do autômato G_N, uma vez que apenas os eventos observáveis podem ocorrer simultaneamente. O último passo está relacionado com a questão da verificação da propriedade de codiagnosticabilidade de uma linguagem utilizando-se verificadores. O teorema a seguir mostra a necessidade deste último passo.

Teorema 2.1. *Sejam* L *e* L_N *linguagens prefixo-fechadas geradas por* G *e* G_N*, respectivamente. Considere a projeção* $P_{o_i} : \Sigma^* \to \Sigma_{o_i}^*$*, para* $i \in A$*, e o conjunto de eventos de falha* Σ_f*. Então,* L *não será codiagnosticável com relação à* P_{o_i} *e* Σ_f *se, e somente se, existir um caminho cíclico em* G_V*,* $(x_{V_1}, \sigma_1, x_{V_2}, \sigma_2, \ldots, \sigma_{k-1}, x_{V_k})$*, que satisfaz à seguinte condição:*

$$\begin{aligned} &\exists j \in \{1, 2, \ldots, k-1\}\ tal\ que\ \sigma_j \in \Sigma\ e\ o\ rótulo\ de\ x_F\ em \\ &x_{V_j} = (x_{N1}, x_{N2}, \ldots, x_{Nm}, x_F)\ seja\ igual\ a\ F. \end{aligned} \tag{2.1}$$

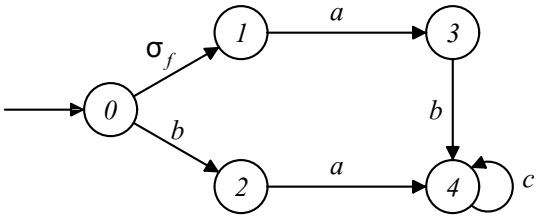


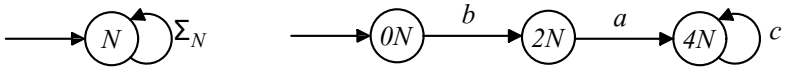
Figura 2.11: Autômato G.

Demonstração. Vide [15]. ■

O exemplo a seguir apresenta a aplicação do algoritmo 2.2 considerando um único agente local. Logo, o exemplo verifica a diagnosticabilidade do sistema em questão.

Exemplo 2.11. *Considere o autômato representado pela figura 2.11 cujo conjunto de eventos observáveis é* $\Sigma_o = \{a, c\}$ *e de não observáveis é* $\Sigma_{uo} = \{b, \sigma_f\}$*, em que* $\Sigma_f = \{\sigma_f\}$*. O primeiro passo do algoritmo 2.2 consiste em usar* A_N *(vide figura 2.12a) e através da composição produto encontrar* G_N *(vide figura 2.12b). A figura 2.13 mostra os autômatos calculados no segundo passo desde* A_l *até se obter* G_F*. A figura 2.14a mostra o autômato* G_N *após a renomeação dos eventos não observáveis e a figura 2.14b mostra* G_V*, isto é, a composição paralela de* G_{Nr} *com* G_F*.*

Pode-se notar que o estado $(4N, 4F)$ de G_V possui um ciclo com um evento de Σ e que o estado de G_F é rotulado por F. Logo, pelo teorema 2.1, a linguagem L gerada por G não é diagnosticável com relação à $P_o : \Sigma^* \to \Sigma_o^*$ e Σ_f.

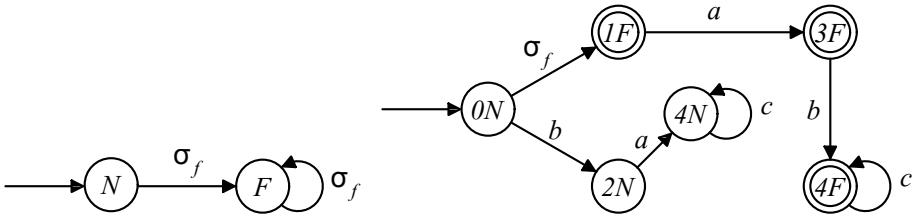


(a) Autômato A_N (b) Autômato G_N

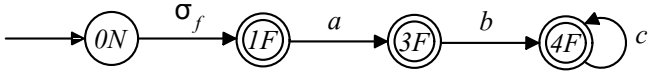
Figura 2.12: Obtenção de G_N.

O exemplo a seguir apresenta a aplicação do algoritmo 2.2 considerando mais de um agente local. Logo, o exemplo verifica a codiagnosticabilidade do sistema em questão.

Exemplo 2.12. *Considere o autômato representado na figura 2.15 cujo conjunto de eventos observáveis para o primeiro agente é* $\Sigma_{o_1} = \{a, b\}$ *e para o segundo agente é* $\Sigma_{o_2} = \{a, c\}$*. O conjunto de eventos não observáveis é* $\Sigma_{uo} = \{\sigma_f, \sigma_u\}$*, em que* $\Sigma_f = \{\sigma_f\}$*. O primeiro passo do algoritmo 2.2 consiste em encontrar* G_N *e, a partir da função de renomeação, encontrar os autômatos* G_{N1} *e* G_{N2} *(vide figura 2.16). A figura 2.17 mostra os autômatos* G_l *e* G_F *calculados no segundo passo. A figura 2.18 mostra* G_V*, isto é, a composição paralela entre* G_{N1}*,* G_{N2} *e* G_F*.*

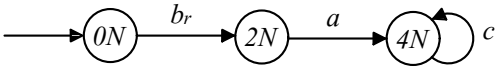


(a) Autômato A_l (b) Autômato G_l

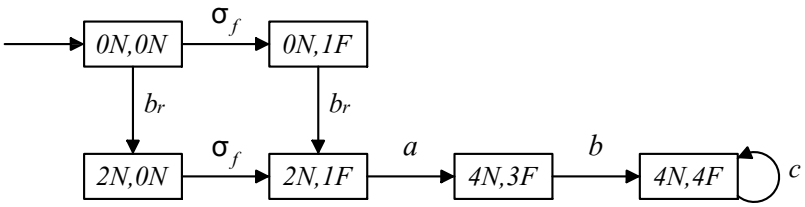


(c) Autômato G_F

Figura 2.13: Obtenção de G_F.



(a) Autômato G_{Nr}



(b) Autômato G_V

Figura 2.14: Obtenção de G_V.

Pode-se notar que o estado $(2N, 2N, 6F)$ de G_V possui um ciclo com um evento de Σ, isto é, σ_u e que o estado de G_F é rotulado por F. Logo, pelo teorema 2.1, a linguagem L gerada por G não é codiagnosticável com relação à $P_{o_i} : \Sigma^* \to \Sigma_{o_i}^*$, para $i \in A$, e Σ_f.

Até agora todos os trabalhos mencionados e as considerações de diagnosticabilidade e codiagnosticabilidade consideraram o caso de observação estática, isto é, o conjunto de eventos observáveis Σ_o e o conjunto de eventos não observáveis Σ_{uo} são fixos. Entretanto, em muitas aplicações, um mesmo agente local pode ter um evento observável em um dado estado, e em outro estado esse mesmo evento não ser mais observável. Essa condição caracteriza o que se denomina como observação dinâmica. No próximo capítulo será considerado o problema da verificação da codiagnosticabilidade de uma linguagem considerando o caso de observação dinâmica.

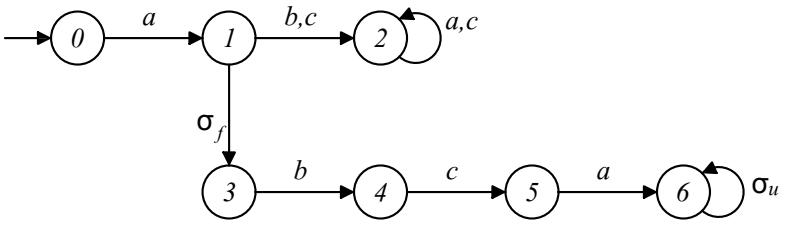
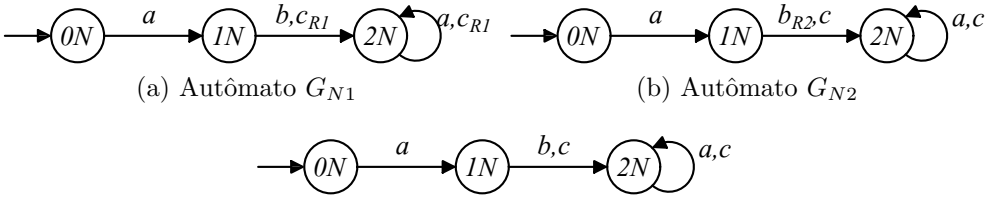


Figura 2.15: Autômato G.

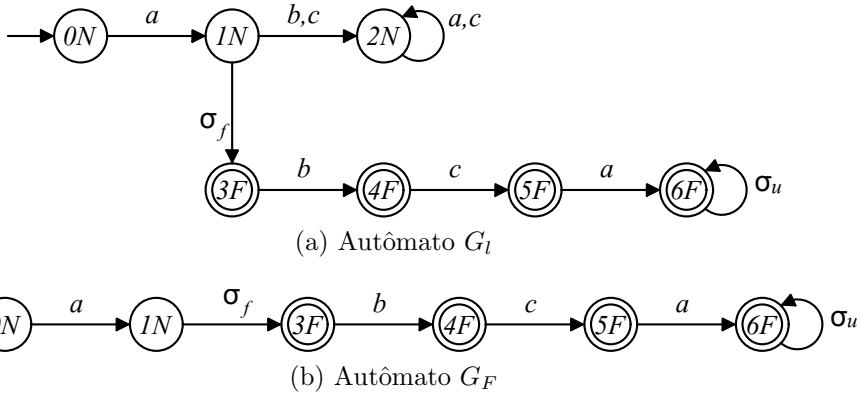


(a) Autômato G_{N1}

(b) Autômato G_{N2}

(c) Autômato G_N

Figura 2.16: Obtenção dos Autômatos de comportamento normal.



(a) Autômato G_l

(b) Autômato G_F

Figura 2.17: Obtenção de G_F.

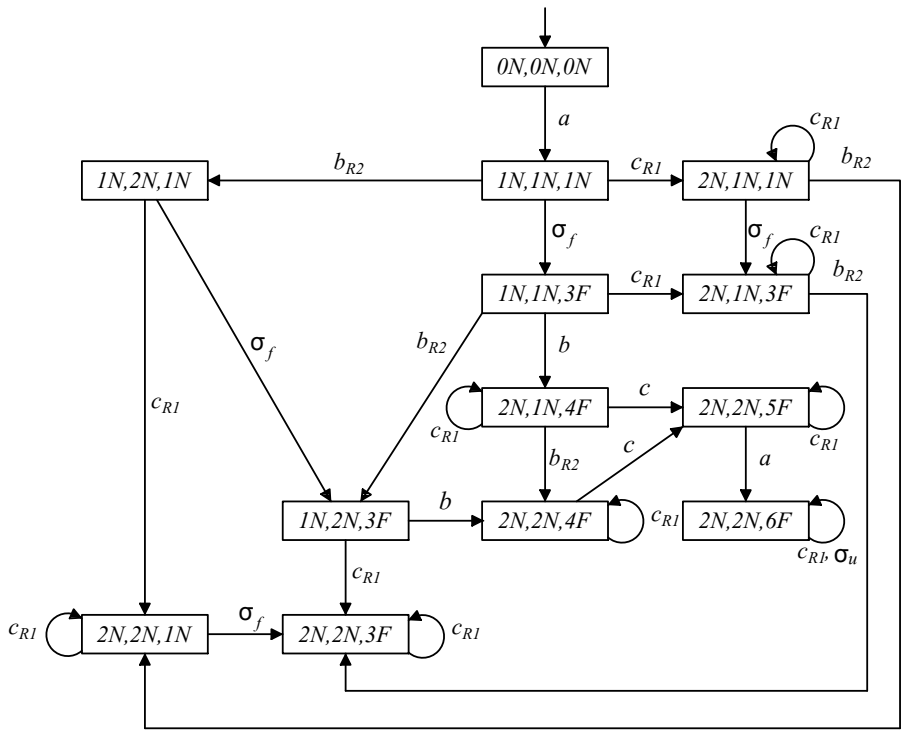


Figura 2.18: Autômato G_V.

Capítulo 3

Diagnóstico de Falhas em Sistemas a Eventos Discretos com Observação Dinâmica

Em muitas situações, a observação de eventos pode não ser fixa como, por exemplo, em problemas de ativação de sensor, em que cada agente pode ligar ou desligar seus sensores dinamicamente durante a evolução do sistema com base em seu histórico de observação, seja por uma questão de restrição de energia, ou largura de banda, ou mesmo de segurança (vide [20–23]). Uma outra situação em que a observação pode ser dinâmica ocorre quando há comunicação entre diferentes agentes, isto é, a comunicação entre agentes pode levar um evento que é observado na ocorrência de uma transição, não ser observado na ocorrência de uma transição diferente. (vide [24, 25]).

No capítulo anterior, foram abordadas a diagnosticabilidade e a codiagnosticabilidade de uma linguagem, assim como a verificação dessas propriedades considerando o caso de observação estática. Neste capítulo será considerado o caso de observação dinâmica e suas implicações sobre as definições de diagnosticabilidade e codiagnosticabilidade, bem como a verificação dessas propriedades utilizando as abordagens apresentadas na literatura.

3.1 Diagnosticabilidade de Falhas sob Observação Dinâmica

No caso de observação estática é considerado que se $\sigma \in \Sigma_{uo}$, então σ será não observável em todas as transições rotuladas por σ. Por outro lado, se $\sigma \in \Sigma_o$, então σ será observável em todas as transições rotuladas pelo evento σ. No caso da observação dinâmica, se um dado agente possui o evento σ observável em uma dada

transição, esse mesmo evento σ pode ser não observável em outra transição. Há ainda outras duas possibilidades: o evento σ ser observável em todas as transições rotuladas por ele, ou σ ser não observável em todas as transições rotuladas por ele. Mas a existência desse meio termo em que numa transição o evento σ é observável e em outra transição não, é que faz a diferença entre a observação estática da observação dinâmica. Assim, é fácil ver que a observação estática pode ser vista como um caso particular da observação dinâmica, em que só existe a possibilidade do evento ser observável ou não.

Na observação dinâmica cada agente possui um subconjunto de eventos observáveis, $\Sigma_{o_i} \subset \Sigma$, para $i \in \{1, \ldots, m\}$, mas como na observação dinâmica um evento pode ser observável ou não dependendo da transição, o conjunto Σ_{o_i} possuirá todos os eventos que são observáveis em pelo menos uma transição pelo agente i. Assim, o conjunto de eventos observáveis é definido como $\Sigma_o = \bigcup_{i=1}^{m} \Sigma_{o_i}$ e representa o conjunto de eventos que são potencialmente observáveis por pelo menos um agente diagnosticador. Já o conjunto de eventos não observáveis Σ_{uo} será definido como $\Sigma_{uo} = \Sigma \setminus \Sigma_o$ e representa o conjunto de eventos que nunca serão observados.

A diagnosticabilidade de falhas sob observação dinâmica já foi investigada em trabalhos anteriores [20, 21, 26, 27], em que os trabalhos [20] e [21] abordam o caso centralizado, e os trabalhos [26] e [27] o caso descentralizado.

Existem dois tipos de abordagem para a observação dinâmica: (i) baseada na linguagem; (ii) baseada na transição. A observação dinâmica baseada na linguagem considera que a observação de uma dada transição depende do caminho, a partir do estado inicial, que se toma para se chegar naquela transição. A observação dinâmica, baseada na transição, considera que uma determinada transição será observável ou não independentemente do caminho escolhido para se chegar até ela. Este trabalho se restringirá a observação dinâmica baseada na transição. O exemplo a seguir mostra a diferença entre os dois tipos de abordagens.

Exemplo 3.1. *Considere o autômato mostrado na figura 3.1, e suponha que o autolaço com o evento "a" no estado 1 possua observação dinâmica baseada na linguagem, isto é, o autolaço "a" será observável ou não dependendo da sequência que se faça para chegar no estado 1. Por exemplo, suponha que o autolaço "a" seja observável se for feita a sequência a partir do estado inicial "a" ou "ca", enquanto que se for feita a sequência "cb" o evento "a" é não observável. Considere agora a observação dinâmica baseada na transição. Uma vez definido que seja fixado que o evento "a" é não observável para um determinado agente, então, "a" será não observável independente do caminho que se tome a partir do estado inicial para se chegar no estado 1. Em outras palavras, independentemente se o estado 1 foi alcançado pela sequência "a", "ca", ou "cb", o autolaço "a" será não observável.*

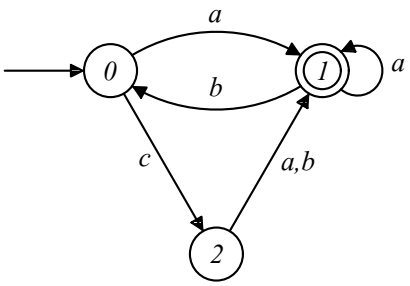


Figura 3.1: Grafo do autômato G.

O exemplo anterior mostra que na observação dinâmica baseada na linguagem uma determinada transição será observável ou não dependendo do caminho que se tome para se chegar àquela determinada transição, enquanto que na observação dinâmica baseada na transição, para um dado agente, uma transição será observável ou não independentemente do caminho que se tome para se chegar à ela.

A seguir são apresentadas algumas definições necessárias para definir a diagnosticabilidade com observação dinâmica baseada em transição.

Definição 3.1. *O conjunto das transições, denotado por $TR(G)$, de um autômato G é dado por:*

$$TR(G) = \{(x, \sigma) \in X \times \Sigma : f(x, \sigma) \text{ é definida}\}.$$

Definição 3.2. *O mapeamento da observação para o agente $i \in A$ é definido como a função $\omega_i : \mathcal{L}(G) \to 2^{\Sigma_{o_i}}$. Então, para uma trajetória $s \in \mathcal{L}(G)$, o mapeamento da observação $\omega_i(s)$ é o subconjunto de Σ_o que corresponde aos eventos observáveis após a sequência s.*

De acordo com a definição 3.2, o mapeamento da observação considera o caso da observação dinâmica baseada na linguagem. Para o caso da observação dinâmica baseado na transição, que é o objeto estudado neste trabalho, o mapeamento da observação deve satisfazer à seguinte sentença: $\forall s, t \in \mathcal{L}(G)$ *se* $f(x_0, s) = f(x_0, t)$, então $\omega_i(s) = \omega_i(t)$. A sentença anterior mostra que independentemente da sequência executada para se alcançar um estado $x \in X$, a observação dos eventos ativos deve ser a mesma, isto é, a observação de um evento não deve depender da sequência que se tome para se chegar a x.

Definição 3.3. *O conjunto das transições observáveis para o agente $i \in A$ é denotado por $\Omega_i \subseteq TR(G)$ e definido como:*

$$\Omega_i = \{(x, \sigma) \in TR(G) : \exists s \in \mathcal{L}(G) \text{ } t.q. \text{ } [f(x_0, s) = x] \wedge [\sigma \in \omega_i(s)]\}.$$

A partir da definição 3.3 pode-se definir projeção considerando-se observação dinâmica.

Definição 3.4. *A projeção* $P_{\Omega_i} : \mathcal{L}(G) \rightarrow \Sigma_{o_i}^*$ *pode ser definida recursivamente como:*

$$P_{\Omega_i}(\varepsilon) = \varepsilon,$$

$$P_{\Omega_i}(s\sigma) = \begin{cases} P_{\Omega_i}(s)\sigma, & se\ (f(x_0, s), \sigma) \in \Omega_i \\ P_{\Omega_i}(s), & se\ (f(x_0, s), \sigma) \notin \Omega_i. \end{cases}$$

No contexto da observação dinâmica, a definição de codiagnosticabilidade sofre pequenas alterações pelo fato de se usar a nova definição de projeção apresentada acima. A definição de codiagnosticabilidade no contexto da observação dinâmica baseada na transição é apresentada a seguir:

Definição 3.5. *Seja L a linguagem prefixo fechada gerada por um autômato G e seja* $L_N \subset L$ *a linguagem prefixo fechada que representa o comportamento normal do sistema. Então, L é codiagnosticável com relação à* $P_{\Omega_i} : \mathcal{L}(G) \rightarrow \Sigma_{o_i}^*$, *com* $i \in A$, *e* Σ_f *se*

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

$$(\exists i \in A)(P_{\Omega_i}(st) \neq P_{\Omega_i}(\omega),\ \forall \omega \in L_N).$$

3.1.1 Verificação da Codiagnosticabilidade de Sistemas a Eventos Discretos com Observação Dinâmica

Para verificar a codiagnosticabilidade, considerando a observação dinâmica baseada na transição, existem as abordagens dos trabalhos [26] e [27] atualmente na literatura. A partir desse ponto será apresentada a abordagem de cada um desses trabalhos com a finalidade de compará-los com o verificador proposto neste trabalho (capítulo 4).

C-Verifier

Em [26] é apresentado um verificador denominado *C-Verifier*. Antes de apresentá-lo é necessário apresentar algumas definições para que se possa entender o seu algoritmo de construção.

Definição 3.6. *A função de indexação* $I_i : TR(G) \rightarrow \{0, 1\}$, *é definida como:*

$$I_i(x, \sigma) = \begin{cases} 0, & se\ (x, \sigma) \notin \Omega_i, \\ 1, & se\ (x, \sigma) \in \Omega_i, \end{cases}$$

A função de indexação é definida apenas por conveniência para dizer se uma transição é observável ou não por um agente local. Se $I_i(x, \sigma) = 1$, então a transição (x, σ) é observável pelo agente i. Se $I_i(x, \sigma) = 0$, então a transição (x, σ) é não observável pelo agente i. Com a definição da função de indexação, é possível definir o autômato *cluster*.

Definição 3.7. *Seja G um autômato, com observação I_i do agente local $i \in A$. O autômato cluster $c_i(x)$ é um sub-autômato de G, formado pelo alcance não observável, a partir do estado x, em que x pode ser o estado inicial de G, ou qualquer outro estado de G alcançado por um evento observável.*

A partir da definição do autômato *cluster* é possível perceber que um autômato, pode ter mais de um *cluster*, pois o estado inicial gera um *cluster*, e cada estado alcançado por um evento observável gera outro. Assim, o conjunto $\mathcal{C}_i$ é o conjunto de todos os *clusters* de G com relação à função de indexação I_i, para o agente local $i \in A$.

O exemplo a seguir ilustra a construção dos *clusters* para o sistema mostrado na figura 3.2. A figura 3.3 mostra o sistema da figura 3.2 sob a observação de dois agentes locais. Os autômatos G_i, com $i \in A$, representam o autômato G sob a projeção P_{Ω_i}, isto é, representam o autômato G sob a observação do agente i. Os eventos entre colchetes serão usados para representar os eventos não observáveis.

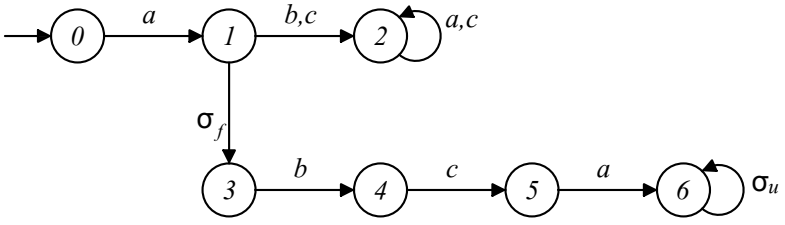
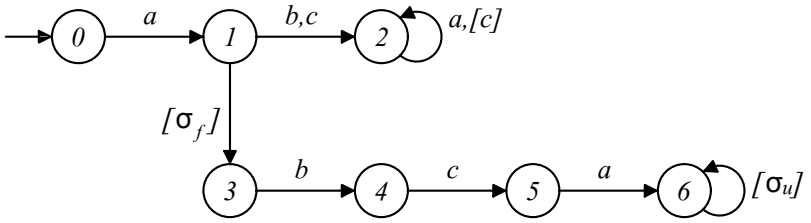
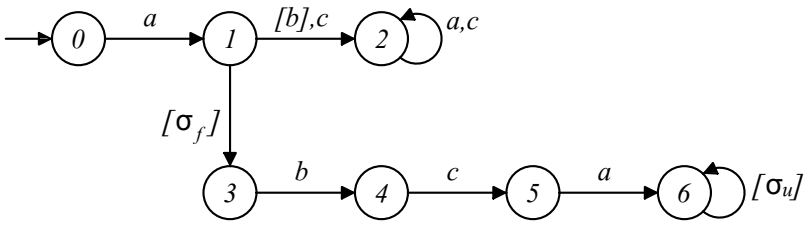


Figura 3.2: Autômato G.

Exemplo 3.2. *Considere o autômato G mostrado na figura 3.2, e a sua respectiva observação para os agentes 1 e 2 na figura 3.3. O conjunto de clusters $\mathcal{C}_1$, para o agente 1, é mostrado na figura 3.4 e o conjunto de clusters $\mathcal{C}_2$, para o agente 2, é mostrado na figura 3.5. As linhas tracejadas indicam as transições observáveis que levam ao estado inicial de outros clusters, enquanto que as linhas contínuas indicam as transições não observáveis. Por exemplo, na figura 3.4a o cluster $c_1(0)$ é iniciado pelo estado inicial. Como o alcance não observável do estado 0 é ele mesmo, então, $c_1(0)$ só possui o estado inicial. A transição $(0, a)$ leva o cluster $c_1(0)$ para o cluster $c_1(1)$, mostrado na figura 3.4b. O cluster $c_1(1)$ é formado pelos estados 1 e 3, pois o alcance não observável do estado 1 resulta nos estados 1 e 3. Após as transições $(1, b)$ ou $(1, c)$ o cluster $c_1(1)$ evolui para o cluster $c_1(2)$, mostrado na figura 3.4c.*



(a) Autômato G_1: Observação para o agente 1



(b) Autômato G_2: Observação para o agente 2

Figura 3.3: Autômato G com sua respectiva observação para o agente 1 e 2.

No cluster $c_1(2)$*, a transição* $(2, a)$ *leva para ele mesmo. Já a transição* $(2, c)$ *é não observável, e faz com que o alcance não observável do estado 2 seja ele mesmo. A transição* $(3, b)$ *leva o cluster* $c_1(1)$ *para o cluster* $c_1(4)$*, mostrado na figura 3.4d. No cluster* $c_1(4)$*, a transição* $(4, c)$ *leva para o cluster* $c_1(5)$*, mostrado na figura 3.4e. No cluster* $c_1(5)$*, a transição* $(5, a)$ *leva para o cluster* $c_1(6)$*, mostrado na figura 3.4f. No cluster* $c_1(6)$ *não há transições observáveis e a transição não observável* $(6, \sigma_u)$ *faz com que o alcance não observável do estado 6 seja ele mesmo. O conjunto de clusters para o agente 2 é construído da mesma maneira que o conjunto de clusters do agente 1. Para exemplificar, considere apenas o cluster* $c_2(1)$ *(figura 3.5b) que é formado pelos estados 1, 2 e 3, uma vez que o alcance não observável para o estado 1 do agente 2, resulta nos estados 1, 2 e 3. Observe, que as transições* $(1, c)$*,* $(2, a)$ *e* $(2, c)$ *levam o cluster* $c_2(1)$ *para o cluster* $c_2(2)$ *(figura 3.5c) e a transição* $(3, b)$ *para o cluster* $c_2(4)$ *(figura 3.5d).*

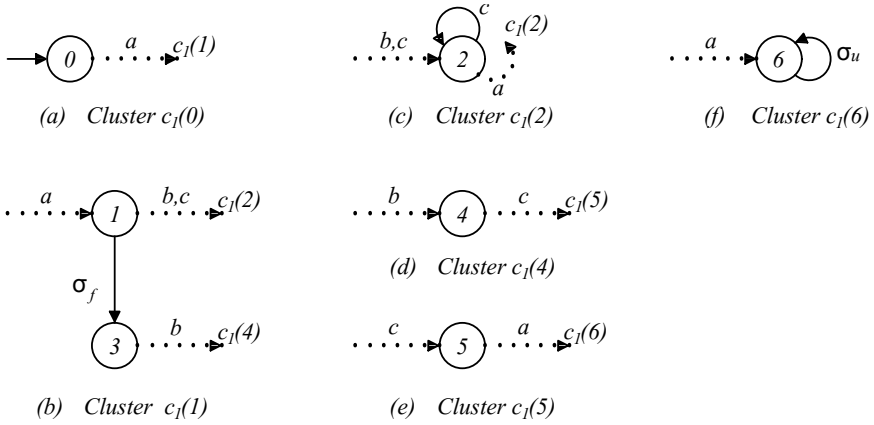


Figura 3.4: Conjunto de *clusters* $\mathcal{C}_1$ baseado na observação do agente 1.

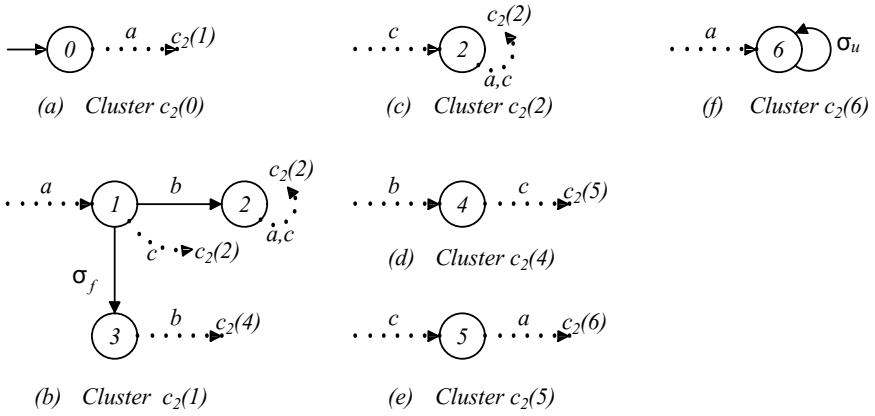


Figura 3.5: Conjunto de *clusters* $\mathcal{C}_1$ baseado na observação do agente 2.

Definição 3.8. *O autômato G^{NF} é denominado autômato F-livre e é um sub-autômato de G. G^{NF} é construído tomando-se a parte acessível de G após se remover as transições rotuladas pelos eventos de falha de G.*

Observe que o autômato *F-livre* G^{NF} é igual ao autômato G_N descrito no primeiro passo do algoritmo 2.2.

Como o algoritmo de construção do *C-Verifier* depende dos *clusters* construídos a partir de G^{NF}, então, o exemplo a seguir ilustra a construção dos autômatos *clusters*, para o sistema apresentado na figura 3.2, cujo autômato G^{NF} é mostrado na figura 3.6.

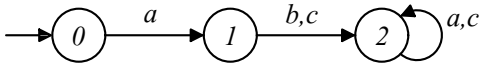


Figura 3.6: Autômato G^{NF}.

Exemplo 3.3. *Considere o autômato G mostrado na figura 3.2, e as suas respectivas observações para os agentes 1 e 2 nas figuras 3.3a e 3.3b. Os dois conjuntos de clusters, $\mathcal{C}_1^{NF}$ e $\mathcal{C}_2^{NF}$, mostrados nas figuras 3.7 e 3.8, representam o conjunto de clusters para os agentes 1 e 2, respectivamente, obtidos a partir do autômato G^{NF} (figura 3.6). O cluster $c_1(0)$ é iniciado pelo estado inicial e formado pelo alcance não observável do estado 0 no autômato G^{NF} com a observação do agente 1. Como o alcance não observável do estado 0 é ele mesmo, então, o cluster $c_1(0)$ possui apenas o estado inicial. A transição $(0, a)$ leva o cluster $c_1(0)$ para o cluster $c_1(1)$, mostrado na figura 3.7b. O alcance não observável do estado 1, considerando a observação do agente 1, é ele mesmo. As transições $(1, b)$ e $(1, c)$ levam o cluster $c_1(1)$ para o cluster $c_1(2)$, mostrado na figura 3.7c. A transição $(2, a)$ leva o cluster $c_1(2)$ nele mesmo, e a transição não observável $(2, c)$ faz com que o alcance não observável do estado 2 seja ele mesmo. O conjunto de clusters $\mathcal{C}_2^{NF}$ é construído da mesma*

maneira que o conjunto $\mathcal{C}_1^{NF}$, levando em consideração a observação do agente 2 no autômato G^{NF}. Para exemplificar, considere apenas o cluster $c_2(1)$ (figura 3.8b) que é formado pelos estados 1 e 2, uma vez que o alcance não observável para o estado 1 do agente 2, resulta nos estados 1 e 2. Observe, que as transições $(1, c)$, $(2, a)$ e $(2, c)$ levam o cluster $c_2(1)$ para o cluster $c_2(2)$, mostrado na figura 3.8c.

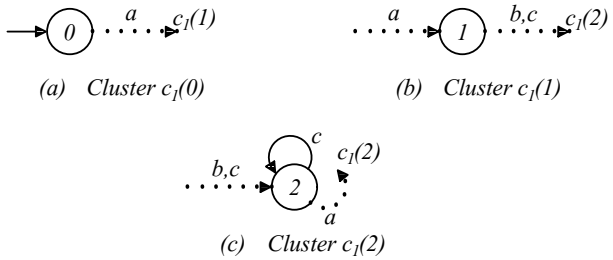


Figura 3.7: Conjunto de *clusters* $\mathcal{C}_1^{NF}$ baseado na observação do agente 1.

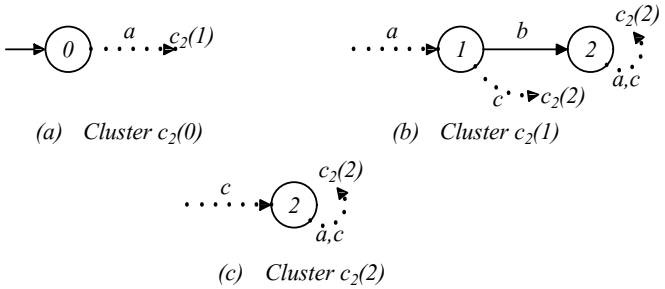


Figura 3.8: Conjunto de *clusters* $\mathcal{C}_2^{NF}$ baseado na observação do agente 2.

O exemplo 3.3 leva à definição do alcance de um *cluster*.

Definição 3.9. *Para o autômato F-livre G^{NF} e a observação I_i do agente $i \in A$, então, o alcance de um cluster $R(\sigma, c_i(x)) \subseteq \mathcal{C}_i^{NF}$ é o conjunto de todos os clusters $c_i(z)$ cujo estado inicial z segue de alguma transição (y, σ) observável para o agente i, de algum estado $y \in X|_{c_i(x)}$, sendo $X|_{c_i(x)}$ o conjunto de estados do cluster $c_i(x)$, ou pode ser dito ainda como o conjunto de estados X restrito ao cluster $c_i(x)$.*

Intuitivamente, o alcance de um *cluster*, nada mais é, do que os *clusters* alcançados, a partir das transições observáveis nos seus estados. Para se entender melhor a definição anterior, segue um exemplo.

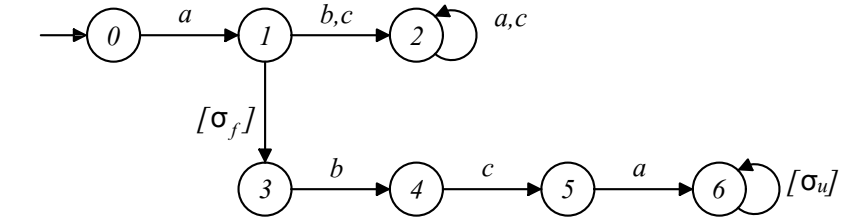
Exemplo 3.4. *Considere o cluster $c_2(1)$ apresentado na figura 3.8b. Então $R(c, c_2(1)) = \{c_2(2)\}$, o que significa que estando no cluster $c_2(1)$, após a ocorrência do evento c, alcança-se o cluster $c_2(2)$ (figura 3.8c) tanto pela transição $(1, c)$ quanto que pela transição $(2, c)$ do cluster $c_2(1)$. Para o cluster $c_2(2)$, na figura 3.8c, tem-se que $R(a, c_2(2)) = R(c, c_2(2)) = \{c_2(2)\}$.*

Definição 3.10. *Dados G e I_i, com $i \in A$, a função de pseudo-indexação I_p : $TR(G) \to \{0,1\}$ é dada por:*

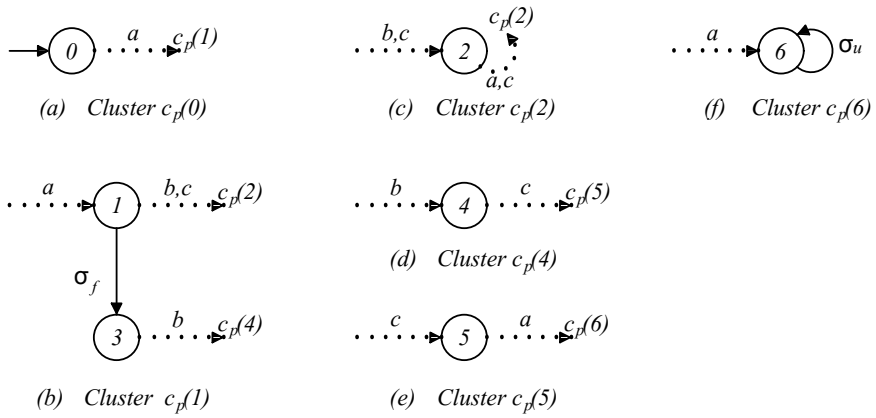
$$I_p(x,\sigma) = \begin{cases} 1, & \text{se } \exists i \in A \text{ tal que } I_i(x,\sigma) = 1, \\ 0, & \text{caso contrário.} \end{cases}$$

A função de pseudo-indexação é útil para a construção do *C-Verifier*. Como veremos mais à frente, o verificador atualiza sua estimativa do sistema pela ocorrência de um evento σ quando $I_p(x,\sigma) = 1$. Assim, defini-se $c_p(x)$ sendo o *cluster* do estado x com relação à função de pseudo-indexação I_p e $\mathcal{C}_p$ denota o conjunto de todos os $c_p(x)$.

Exemplo 3.5. *Para o sistema mostrado na figura 3.2, a figura 3.9a mostra sua observação baseada na função de pseudo-indexação I_p, isto é, G mesclando a observação dos seus dois agentes. A figura 3.9b mostra o conjunto de todos os clusters $\mathcal{C}_p$. O conjunto de clusters $\mathcal{C}_p$ é construído, a partir do sistema da figura 3.9a, da mesma maneira que os conjuntos de cluster $\mathcal{C}_1$ e $\mathcal{C}_2$ no exemplo 3.2 e os conjuntos $\mathcal{C}_1^{NF}$ e $\mathcal{C}_2^{NF}$ no exemplo 3.3.*



(a) Observação baseada em I_p



(b) *Clusters* $\mathcal{C}_p$ baseados em I_p

Figura 3.9: Observação baseada na função de pseudo-indexação I_p.

Definição 3.11. *Para um dado cluster $c_p(x)$, o conjunto $X|^F_{c_p(x)} \subseteq X|_{c_p(x)}$ contém todos os estados de $X|_{c_p(x)}$ que são alcançáveis a partir do estado x por alguma sequência $s \in \mathcal{L}(c_p(x))$ que possui pelo menos um evento de falha. Similarmente, o*

conjunto $X|^{NF}_{c_p(x)} \subseteq X|_{c_p(x)}$ *contém todos os estados de* $X|_{c_p(x)}$*, que são alcançáveis a partir do estado* x*, por alguma sequência* $s \in \mathcal{L}(c_p(x))$ *que não possui eventos de falha.*

A partir da definição 3.11, pode-se observar que o estado 1 do *cluster* $c_p(1)$ pertence ao conjunto $X|^{NF}_{c_p(x)}$, pois não é alcançado por sequências que possuem eventos de falha, enquanto que o estado 3 pertence ao conjunto $X|^{F}_{c_p(x)}$, pois é alcançado por uma sequência que possui um evento de falha, a saber $s = \sigma_f$.

As definições a seguir são importantes para a verificação da codiagnosticabilidade após a construção do autômato *C-Verifier*.

Definição 3.12. *Sob a função de pseudo-indexação* I_p, $\mathcal{C}_p^{F-cycle} \subseteq \mathcal{C}_p$ *é o conjunto de todos os clusters cujos elementos* $c_p(x)$ *contém um ciclo que é alcançado por uma sequência* $s \in \mathcal{L}(c_p(x))$ *que possui um evento de falha, ou o próprio ciclo possui um evento de falha. Matematicamente:*

$$\mathcal{C}_p^{F-cycle} = \{c_p(x) \in \mathcal{C}_p : (\exists ss' \in \mathcal{L}(c_p(x))) \text{ com } s' \neq \varepsilon \text{ tal que } f(x,s) \in X|^{F}_{c_p(x)} \wedge f(x,ss') = f(x,s)\}$$

Definição 3.13. *Sob a função de pseudo-indexação* I_p, $\mathcal{C}_p^{cycle} \subseteq \mathcal{C}_p$ *é o conjunto de todos os clusters que contém ciclos. Matematicamente:*

$$\mathcal{C}_p^{cycle} = \{c_p(x) \in \mathcal{C}_p : (\exists ss' \in \mathcal{L}(c_p(x))) \text{ com } s' \neq \varepsilon \text{ tal que } f(x,ss') = f(x,s)\}$$

O conjunto de rótulos $L = \{NF, F\}$ será usado para marcar se um evento de falha ocorreu ou não para os estados do verificador. A seguir será apresentada a definição da função de transição que rege a construção do *C-Verifier*.

Definição 3.14. *Para* $(x,\sigma) \in TR(G)$ *tal que* $x \in X|_{c_p(x_p)}$ *e* $I_p(x,\sigma) = 1$, $l \in L$ *e* $c_i(x_i) \in \mathcal{C}_i^{NF}$*, então, a função de transição do autômato C-Verifier* $\overline{\Gamma}([c_1(x_1), \ldots, c_m(x_m), (c_p(x_p), l)], (x,\sigma))$ *é definida como:*

$$\begin{aligned}
&\overline{\Gamma}([c_1(x_1), \ldots, c_m(x_m), (c_p(x_p), l)], (x,\sigma)) \\
&= \left\{ [\Gamma_1(c_1(x_1), (x,\sigma)), \ldots, \Gamma_m(c_m(x_m), (x,\sigma)), \Gamma_p((c_p(x_p), l), (x,\sigma))] : \right. \\
\Gamma_i(c_i(x_i), (x,\sigma)) &= \begin{cases} c_i(x_i), & \text{se } I_i(x,\sigma) = 0, \\ c_i(y_i), & \text{se } I_i(x,\sigma) = 1 \wedge c_i(y_i) \in R_i(\sigma, c_i(x_i)), \end{cases} \\
\Gamma_p((c_p(x_p), l), (x,\sigma)) &= \left\{ \begin{cases} (c_p(f(x,\sigma)), NF), & \text{se } l = NF \wedge x \in X|^{NF}_{c_p(x_p)}, \\ (c_p(f(x,\sigma)), F), & \text{se } l = F \vee x \in X|^{F}_{c_p(x_p)}. \end{cases} \right\}.
\end{aligned} \tag{3.1}$$

A função de transição $\overline{\Gamma}$, da maneira que é enunciada em [26], não contempla todos os casos possíveis para as transições observáveis que estão ativas nos estados do *cluster* $c_p(x_p)$. Assim, a função de transição poderia ter sido melhor definida como:

$$\begin{aligned}
&\overline{\Gamma}([c_1(x_1),\ldots,c_m(x_m),(c_p(x_p),l)],(x,\sigma)) \\
&= \Big\{ [\Gamma_1(c_1(x_1),(x,\sigma)),\ldots,\Gamma_m(c_m(x_m),(x,\sigma)),\Gamma_p((c_p(x_p),l),(x,\sigma))] : \\
\Gamma_i(c_i(x_i),(x,\sigma)) &= \begin{cases} c_i(x_i), & \textit{se } I_i(x,\sigma)=0, \\ c_i(y_i), & \textit{se } I_i(x,\sigma)=1 \wedge \exists c_i(y_i) \in R_i(\sigma,c_i(x_i)), \\ \overline{\Gamma}([\cdot],(x,\sigma))=\emptyset, & \text{Caso contrário,} \end{cases} \\
\Gamma_p((c_p(x_p),l),(x,\sigma)) &= \left\{ \begin{cases} (c_p(f(x,\sigma)),NF), & \textit{se } l=NF \wedge x \in X|^{NF}_{c_p(x_p)}, \\ (c_p(f(x,\sigma)),F), & \textit{se } l=F \vee x \in X|^{F}_{c_p(x_p)}. \end{cases} \right\},
\end{aligned} \tag{3.2}$$

em que $[\cdot]$ é o estado atual do *C-Verifier*, isto é, $[\cdot]=[c_1(x_1),\ldots,c_m(x_m),(c_p(x_p),l)]$.

O exemplo a seguir ilustra a função de transição para dois estados do *C-Verifier* e apresenta o ponto exato onde a função de transição $\overline{\Gamma}$ furaria.

Exemplo 3.6. *Considere o sistema apresentado na figura 3.2, e considere que a observação dinâmica é apresentada nas figuras 3.3a e 3.3b, pelos agentes 1 e 2, respectivamente. Suponha que σ_f seja o evento de falha. Então,*

$$\overline{\Gamma}([c_1(0),c_2(0),(c_p(0),NF)],(0,a)) = \{[c_1(1),c_2(1),(c_p(1),NF)]\}.$$

A transição $(0,a)$ leva o estado $[c_1(0),c_2(0),(c_p(0),NF)]$ do C-Verifier para o estado $[c_1(1),c_2(1),(c_p(1),NF)]$ pois $I_1(0,a)=1 \wedge [c_1(1) \in R_1(a,c_1(0))]$. Logo, o cluster $c_1(0)$ evolui para o cluster $c_1(1)$. A evolução do cluster $c_2(0)$ para o cluster $c_2(1)$ ocorre por motivo similar, isto é, $I_2(0,a)=1 \wedge [c_2(1) \in R_2(a,c_2(0))]$. O cluster $(c_p(0),NF)$ evolui para $(c_p(1),NF)$, mantendo o rótulo NF, pois $l=NF \wedge 0 \in X|^{NF}_{c_p(0)}$.

Para o estado alcançado $[c_1(1),c_2(1),(c_p(1),NF)]$ há três transições possíveis, uma vez que no cluster $c_p(1)$ há três transições observáveis saindo dos seus estados, $(1,b)$, $(1,c)$ e $(3,b)$:

$$\overline{\Gamma}([c_1(1),c_2(1),(c_p(1),NF)],(1,c)) = \{[c_1(2),c_2(2),(c_p(2),NF)]\},$$

$$\overline{\Gamma}([c_1(1),c_2(1),(c_p(1),NF)],(1,b)) = \{[c_1(2),c_2(1),(c_p(2),NF)]\}$$

e

$$\overline{\Gamma}([c_1(1),c_2(1),(c_p(1),NF)],(3,b)) = \emptyset.$$

A transição $(1, c)$ *leva o estado* $[c_1(1), c_2(1), (c_p(1), NF)]$ *do C-Verifier, para o estado* $[c_1(2), c_2(2), (c_p(2), NF)]$*, pelos mesmos motivos que a transição* $(0, a)$ *leva o estado* $[c_1(0), c_2(0), (c_p(0), NF)]$ *para o estado* $[c_1(1), c_2(1), (c_p(1), NF)]$*.*

A transição $(1, b)$ *leva o estado* $[c_1(1), c_2(1), (c_p(1), NF)]$ *do C-Verifier, para o estado* $[c_1(2), c_2(1), (c_p(2), NF)]$*, pois* $I_1(1, b) = 1 \wedge [c_1(2) \in R_1(b, c_1(1))]$*. Logo, o cluster* $c_1(1)$ *evolui para o cluster* $c_1(2)$*. O cluster* $c_2(1)$ *não evolui, pois* $I_2(1, b) = 0$*. O cluster* $(c_p(1), NF)$ *evolui para* $(c_p(2), NF)$*, mantendo o rótulo* NF*, pois* $l = NF \wedge 1 \in X|^{NF}_{c_p(1)}$*.*

A função de transição 3.1, para a transição $(3, b)$*, não é capaz de exprimir resultado, pois, apesar da transição* $(3, b)$ *levar o cluster* $c_1(1)$ *para o cluster* $c_1(2)$*, e levar o cluster* $(c_p(1), NF)$ *para o cluster* $(c_p(4), F)$*. A transição* $(3, b)$ *para o cluster* $c_2(1)$ *não está definida, uma vez que* $I_2(3, b) = 1$ *e* $R_2(b, c_2(1)) = \emptyset$*. Logo, a função de transição 3.1, para a transição* $(3, b)$ *no estado* $[c_1(1), c_2(1), (c_p(1), NF)]$ *do C-Verifier não está definida. Para a função de transição 3.2, o resultado será* $\overline{\Gamma}([c_1(1), c_2(1), (c_p(1), NF)], (3, b)) = \emptyset$*, significando que essa transição não deve ser realizada.*

Observação 3.1. *Como pode ser visto no exemplo 3.6 a função de transição 3.1 não contempla o caso da transição* $(3, b)$*, uma vez que* $I_2(3, b) = 1$ *e* $R_2(b, c_2(1)) = \emptyset$*. Para isso, deve-se considerar a função de transição 3.2.*

Considere um sistema G e m agentes locais, o conjunto de eventos de falha $\Sigma_f \subseteq \Sigma$ e a função de indexação I_i, com $i \in A$. O autômato *C-Verifier* é construído de acordo com o seguinte algoritmo.

Algoritmo 3.1. *Construção do C-Verifier*

Entrada: *Autômatos* G*,* G_1*,* $G_2, \ldots, G_m$*, e os conjuntos de clusters* $\mathcal{C}_1^{NF}$*,* $\mathcal{C}_2^{NF}, \ldots, \mathcal{C}_m^{NF}$*,* $\mathcal{C}_p$*.*

Saída: *Autômato C-Verifier e decisão da codiagnosticabilidade de sistemas a eventos discretos com observação dinâmica.*

1: *Faça* $x_0^D = [c_1(x_0), \ldots, c_m(x_0), (c_p(x_0), NF)]$ *como estado inicial e marque-o com 0, isto é,* $X^D = \{(x_0^D, 0)\}$*.*

2: *Encontre um estado* $x^D = [c_1(x_1), \ldots, c_m(x_m), (c_p(x_p), l)] \in X^D$ *marcado com 0. Então, para cada transição* $(x, \sigma) \in TR(G)$ *com o estado* $x \in X|_{c_p(x_p)}$ *e* $I_p(x, \sigma) = 1$*, calcule* $\overline{\Gamma}(x^D, (x, \sigma))$*. Adicione um novo estado* z^D *no C-Verifier se* $z^D \in \overline{\Gamma}(x^D, (x, \sigma)) \setminus X^D$ *e marque* z^D *com 0. Construa uma transição que começa em* x^D *e termina em cada* $y^D \in \overline{\Gamma}(x^D, (x, \sigma))$ *e rotule com o evento* σ*. Em seguida, mude a marcação de* x^D *de 0 para 1.*

3: Itere o passo 2 até todos os estados em X^D sejam marcados por 1. Então, vá para o passo 4.

4: Verifique se:

4.1: Um estado $(x^D, 1) \in X^D$ possui $c_p(x) \in \mathcal{C}_p^{F-cycle}$;

4.2: $c_p(x) \in \mathcal{C}_p^{cycle}$ com x^D rotulado por F;

4.3: Existe um ciclo rotulado por F no C-Verifier.

Caso ocorra um desses casos o sistema é não codiagnosticável com relação I_i, com $i \in A$ e Σ_f. Caso contrário, o sistema é codiagnosticável.

O automato *C-Verifier* observa todas as sequências $(t_1, t_2, \ldots, t_m)$ pertencentes ao produto cartesiano $\mathcal{L}(G^{NF}) \times \cdots \times \mathcal{L}(G^{NF})$, tais que t_i tem a mesma observação que $t_p \in \mathcal{L}(G)$, para o agente i. O autômato *C-Verifier* também marca se a sequência t_p possui ou não evento de falha, a partir dos rótulos F ou NF, respectivamente. Caso haja pelo menos uma das três condições do passo 4, do algoritmo 3.1, no autômato *C-Verifier*, significa que exite pelo menos uma sequência de falha que se confunde com sequência normais em todos os agentes $i \in A$.

O algoritmo anterior leva em consideração que diversos autômatos e *clusters* necessários para a construção do *C-Verifier* já foram calculados anteriormente. Contudo, poder-se-ia acrescentar mais quatro passos iniciais a esse algoritmo e ter como entrada apenas G e os autômatos que representam a observação para cada agente, G_i, com $i \in A$, obtendo-se, assim, um algoritmo que melhor mostra o passo a passo necessário para se construir o *C-Verifier*.

Os passos que poderiam ser acrescentados são:

1: Obter G^{NF} a partir de G;

2: A partir de G^{NF} e dos autômatos que representam a observação para cada agente, isto é, G_i, obter $\mathcal{C}_i^{NF}$, com $i \in A$;

3: A partir da função de pseudo-indexação I_p obter $G|_{I_p}$, isto é, G com observação baseada em I_p;

4: A partir de $G|_{I_p}$ obter $\mathcal{C}_p$.

O exemplo a seguir mostra a aplicação do algoritmo 3.1 para o sistema representado na figura 3.2.

Exemplo 3.7. *Considere o sistema mostrado na figura 3.2. A observação para os seus dois agentes locais é mostrado na figura 3.3. Os clusters $\mathcal{C}_i^{NF}$, com $i \in A$, e $\mathcal{C}_p$ já foram obtidos e encontram-se nas figuras 3.7, 3.8 e 3.9b, respectivamente. Suponha σ_f como único evento de falha, então o autômato C-Verifier é mostrado na figura 3.10.*

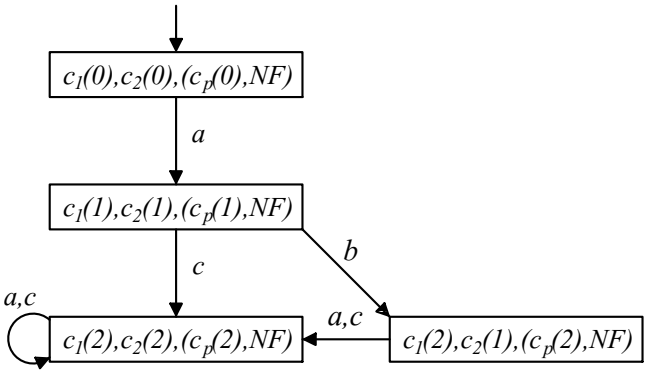
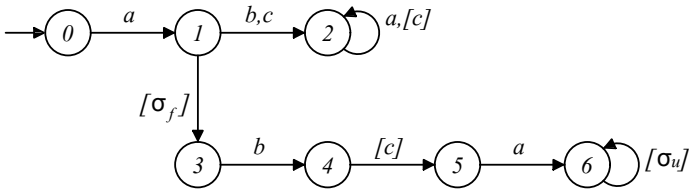


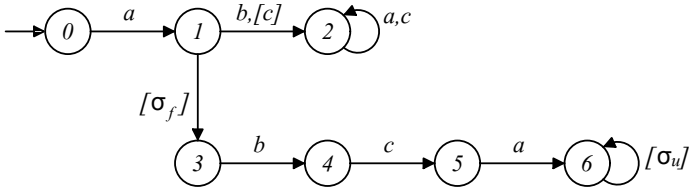
Figura 3.10: Autômato *C-Verifier*.

Ao se observar os *clusters* de $\mathcal{C}_p$ é possível observar que há um *cluster* com um ciclo, isto é, $c_p(6) \in \mathcal{C}_p^{cycle}$, mas esse ciclo não faz parte do *C-Verifier* e não há nenhum *cluster* pertencente ao conjunto $\mathcal{C}_p^{F-cycle}$. Assim, os passos 4.1 e 4.2 do algoritmo 3.1 não se verificam. Além disso, como no *C-Verifier* não há nenhum ciclo rotulado por F, o passo 4.3 também não se verifica e, portanto, pode-se concluir que o sistema apresentado na figura 3.2 é codiagnosticável com relação às observações dinâmica dos agentes 1 e 2, e o conjunto de eventos de falha Σ_f.

O exemplo a seguir considera o autômato mostrado na figura 3.2, com uma nova observação para os seus dois agentes mostrada na figura 3.11.



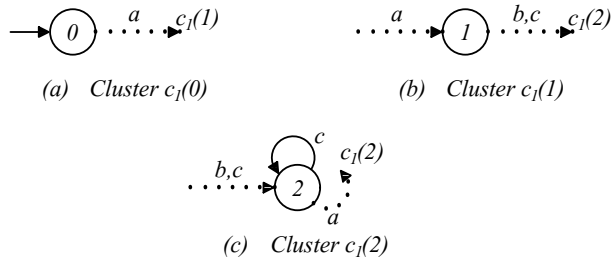
(a) Autômato G_1: Observação para o agente 1



(b) Autômato G_2: Observação para o agente 2

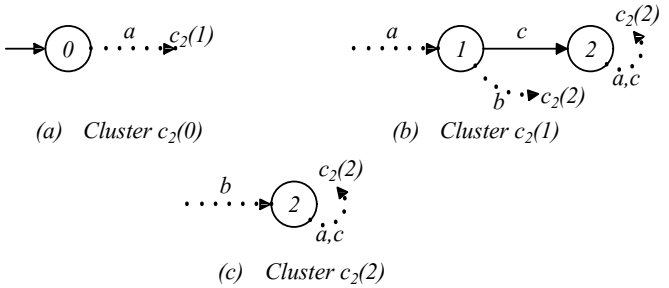
Figura 3.11: Autômato G com sua respectiva observação para o agente 1 e 2.

Exemplo 3.8. *Considere o sistema mostrado na figura 3.2, com a observação para os seus dois agentes locais mostrado na figura 3.11. Os clusters $\mathcal{C}_i^{NF}$, com $i \in A$ encontram-se na figura 3.12, e $\mathcal{C}_p$ encontra-se na figura 3.9b. Suponha σ_f como único evento de falha, então o autômato C-Verifier é mostrado na figura 3.13.*



(a) Cluster $c_1(0)$ (b) Cluster $c_1(1)$ (c) Cluster $c_1(2)$

(a) Conjunto de *clusters* $\mathcal{C}_1^{NF}$ baseado na observação do agente 1



(a) Cluster $c_2(0)$ (b) Cluster $c_2(1)$ (c) Cluster $c_2(2)$

(b) Conjunto de *clusters* $\mathcal{C}_2^{NF}$ baseado na observação do agente 2

Figura 3.12: Conjunto de *clusters* para o agente 1 e 2.

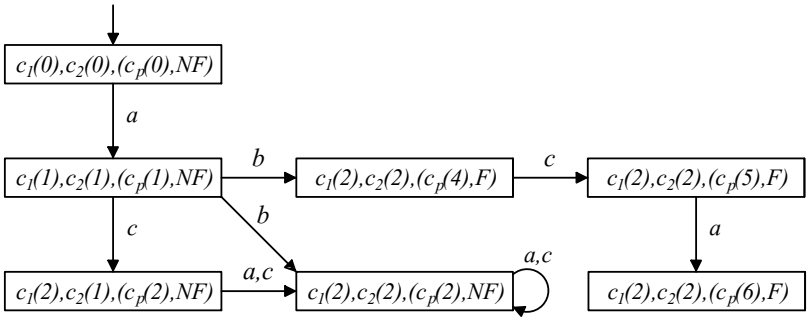


Figura 3.13: Autômato *C-Verifier*.

Ao se observar os *clusters* do conjunto $\mathcal{C}_p$ é possível notar que há um *cluster* com um ciclo, isto é, $c_p(6) \in \mathcal{C}_p^{cycle}$. Além disso, o estado que possui $c_p(6)$, que é o estado $c_1(2), c_2(2), (c_p(6), F)$, é rotulado com F. Logo, o passo 4.2 do algoritmo 3.1 se verifica. A partir do passo 4.2, pode-se concluir que o sistema apresentado na figura 3.2 é não codiagnosticável com relação às observações dinâmicas dos agentes 1 e 2, e o conjunto de eventos de falha Σ_f.

Observe ainda, que os passos 4.1 e 4.3 do algoritmo 3.1 não se verificam, pois não há nenhum *cluster* em $\mathcal{C}_p$ pertencente ao conjunto $\mathcal{C}_p^{F-cycle}$, e não há nenhum ciclo rotulado por F, no autômato *C-Verifier*.

T-Verifier

Nesta seção é apresentada a abordagem utilizada em [27], para verificar a codiagnosticabilidade, utilizando a construção de um autômato V denominado *T-Verifier*.

Para a construção do *T-Verifier* serão necessários os autômatos que modelam a planta G e os autômatos que representam a observação para cada agente G_i, para $i \in A$.

Algoritmo 3.2. *Construção do T-Verifier*

Entrada: *Autômatos G, G_1, $G_2, \ldots, G_m$ e o conjunto Σ_f.*
Saída: *Autômato V.*

1: *Construa o autômato G_N como a parte acessível de G removendo todos os eventos de falha em G (para se obter G_N basta seguir o primeiro passo do algoritmo 2.2).*

2: *Construa o autômato $V = (X^V, \Sigma^V, f^V, x_0^V)$, em que:*

2.1: $X^V = X \times \underbrace{X_N \times \cdots \times X_N}_{m \text{ vezes}} \times L$, *sendo* $L = \{N, F\}$ *o conjunto de rótulos;*

2.2: $\Sigma^V = \underbrace{(\Sigma \cup \{\varepsilon\}) \times \cdots \times (\Sigma \cup \{\varepsilon\})}_{(m+1) \text{ vezes}}$;

2.3: $x_0^V = (x_0, \underbrace{x_{0_N}, \ldots, x_{0_N}}_{m \text{ vezes}}, N)$ *é o estado inicial;*

2.4: $f^V : X^V \times \Sigma^V \to X^V$ *é a função de transição parcial (determinística) definida como se segue:*

(a) Se $\sigma \in \Sigma_f$, então:

$$f^V((x_0, x_1, \ldots, x_m, l), (\sigma, \varepsilon, \ldots, \varepsilon) = (f(x_0, \sigma), x_1, \ldots, x_m, F)$$

(b) Se $\sigma \in \Sigma \setminus \Sigma_f$, então particiona-se A, em que A é o conjunto de agentes locais, em quatro conjuntos disjuntos $A = A_1 \dot{\cup} A_2 \dot{\cup} A_3 \dot{\cup} A_4$ sendo:
$A_1 = \{i \in A : (x_0, \sigma) \in \Omega_i \wedge (x_i, \sigma) \in \Omega_i\}$,
$A_2 = \{i \in A : (x_0, \sigma) \in \Omega_i \wedge (x_i, \sigma) \notin \Omega_i\}$,

$A_3 = \{i \in A : (x_0, \sigma) \notin \Omega_i \wedge (x_i, \sigma) \in \Omega_i\}$,
$A_4 = \{i \in A : (x_0, \sigma) \notin \Omega_i \wedge (x_i, \sigma) \notin \Omega_i\}$,
e faz-se os dois passos abaixo:

(b1) *Se* $A_2 = \emptyset$, *então faz-se a seguinte transição*
$f^V((x_0, x_1, \ldots, x_m, l), (\sigma, e_1, \ldots, e_m))$
$= (f(x_0, \sigma), f_N(x_1, e_1), \ldots, f_N(x_m, e_m), l)$,
em que para cada $i \in A$, $e_i = \begin{cases} \sigma, & \text{se } i \in A_1, \\ \varepsilon, & \text{se } i \in A_3 \cup A_4, \end{cases}$

(b2) *e as seguintes transições para cada* $i \in A_2 \cup A_4$,
$f^V((x_0, x_1, \ldots, x_m, l), (\varepsilon, \varepsilon, \ldots, \varepsilon, \underbrace{\sigma}_{(i+1)\text{ésimo}}, \varepsilon, \ldots, \varepsilon))$
$= (x_0, x_1, \ldots, x_{i-1}, f_N(x_i, \sigma), x_{i+1}, \ldots, x_m, l)$.

Para verificar se a linguagem gerada $\mathcal{L}(G)$ é codiagnosticável é necessário introduzir as seguinte definições.

Definição 3.15. *Seja uma sequência* $t = \sigma_1 \ldots \sigma_n \in \mathcal{L}(V)$, *em que* $\sigma_k = (\sigma_k^0, \sigma_k^1, \sigma_k^2, \ldots, \sigma_k^m)$. *Defini-se* $\theta_i(t)$, $i = 0, 1, \ldots, m$, *sendo a restrição de t para cada um dos seus componentes. Por exemplo,* $\theta_i(t) = \sigma_1^i \ldots \sigma_n^i \in \mathcal{L}(G)$, $i = 0, 1, \ldots, m$.

O exemplo a seguir ilustra a função θ_i.

Exemplo 3.9. *Considere um sistema com a observação dinâmica de dois agentes locais, com um conjunto de eventos* $\Sigma = \{a, b, c, d, \sigma_f\}$. *Suponha que o autômato T-Verifier foi construindo segundo o algoritmo 3.2. Então, uma possível sequencia de eventos* $t = \sigma_1\sigma_2\sigma_3\sigma_4 \in \mathcal{L}(V)$ *seria* $t = (a, a, a)(b, \varepsilon, b)(\sigma_f, \varepsilon, \varepsilon)(\varepsilon, c, \varepsilon)$. *A função* $\theta_0(t)$ *observará apenas as sequências do autômato G, isto é,* $\theta_0(t) = ab\sigma_f\varepsilon$. *As funções* $\theta_1(t)$ *e* $\theta_2(t)$ *observam os autômatos* G_N, *cada um no seu respectivo componente, resultando em* $\theta_1(t) = a\varepsilon\varepsilon c$ *e* $\theta_2(t) = ab\varepsilon\varepsilon$.

A definição a seguir completa as definições necessárias para o enunciamento do teorema 3.1.

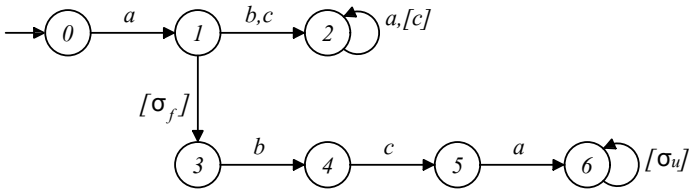
Definição 3.16. *Um ciclo* $(x_1, \sigma_1, x_2, \sigma_2, \ldots, \sigma_{k-1}, x_k)$ *é dito ser ciclo F-real se:*

- $(x_i)_l = F$, $\forall i \in \{1, 2, \ldots, k-1\}$, *em que* $(x_i)_l$ *significa o rótulo do componente de* x_i;
- $\exists j \in \{1, 2, \ldots, k-1\}$ *tal que* $\theta_0(\sigma_j) \neq \varepsilon$.

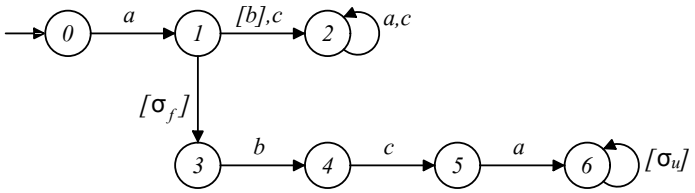
Teorema 3.1. $\mathcal{L}(G)$ *não é codiagnosticável com relação à* Ω_i, *para* $i \in A$, *e* Σ_f, *se e somente se, existe um F-real cycle em* V.

Demonstração. Vide [27]. ■

O exemplo a seguir mostra a aplicação do algoritmo 3.2 para o sistema representado na figura 3.2. A figura 3.3 será repetida na figura 3.14 com o intuito de facilitar o entendimento da construção do autômato V para o leitor.



(a) Autômato G_1: Observação para o agente 1



(b) Autômato G_2: Observação para o agente 2

Figura 3.14: Autômato G com sua respectiva observação para o agente 1 e 2.

Exemplo 3.10. *Seja o sistema mostrado na figura 3.2. A observação para os seus dois agentes locais é mostrado na figura 3.14. Suponha σ_f como único evento de falha, então o autômato T-Verifier é mostrado na figura 3.15. A partir do estado inicial $(0,0,0,N)$ o único evento habilitada é o evento a. Lembre que os estados tem a forma $(0,0,0,N) = (x_0, x_1, x_2, N)$ e precisa-se construir para cada evento de $\Sigma \backslash \Sigma_f$ os conjuntos A_1, A_2, A_3, e A_4. As transições $(0,a)$ e $(0,a)$ são observáveis para o agente 1, isto é, $(0,a)$, $(0,a) \in \Omega_1$ e as transições $(0,a)$ e $(0,a)$, são observáveis para o agente 2, isto é, $(0,a)$, $(0,a) \in \Omega_2$. Logo, para o evento a, o conjunto $A = A_1 = \{1,2\}$. A partir do passo 2.4(b1) do algoritmo 3.2, o sistema evolui, com o evento aaa, para o estado $(1,1,1,N)$. Nesse novo estado os eventos habilitados são b, c e σ_f. As transições $(1,b)$, $(1,b) \in \Omega_1$ e as transições $(1,b)$, $(1,b) \notin \Omega_2$. Logo, para o evento b, o conjunto $A = A_1 \dot{\cup} A_4$, em que $A_1 = \{1\}$ e $A_4 = \{2\}$. A partir do passo 2.4(b1) do algoritmo 3.2, o sistema evolui, com o evento $bb\varepsilon$, para o estado $(2,2,1,N)$. A partir do passo 2.4(b2) o sistema evolui, com o evento $\varepsilon\varepsilon b$ para o estado $(1,1,2,N)$. Para o evento c, as transições $(1,c)$, $(1,c) \in \Omega_1$ e as transições $(1,c)$, $(1,c) \in \Omega_2$. Logo, para o evento c, o conjunto $A = A_1 = \{1,2\}$. A partir do passo 2.4(b1), o sistema evolui, com o evento ccc, para o estado $(2,2,2,N)$. Para o evento σ_f, a partir do passo 2.4a o sistema evolui com o evento $\sigma_f\varepsilon\varepsilon$ para o estado $(3,1,1,F)$. Os demais estados e transições do autômato V seguem os mesmos passos descrito acima.*

Pode-se observar que no autômato V, da figura 3.15, não há ciclos F-real, uma vez que os únicos ciclos estão no comportamento normal do sistema. Logo, de acordo

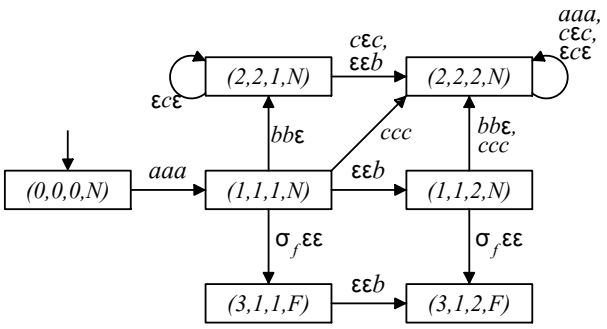


Figura 3.15: Autômato *T-Verifier*.

com o teorema 3.1, o sistema é codiagnosticável com relação à observação dinâmica dos agentes 1 e 2 e o conjunto de falhas Σ_f.

Observação 3.2. *Considere, o estado* $(2, 2, 1, N)$ *alcançado pelo evento* $bb\varepsilon$*, na figura 3.15. Os eventos habilitados nesse estado são a b e c. Para o evento a as transições* $(2, a)$*,* $(2, a) \in \Omega_1$ *enquanto que a transição* $(2, a) \in \Omega_2$*. Repare que a transição* $(1, a)$ *não é definida no autômato G, logo, essa transição trivialmente não pertence ao conjunto* Ω_2*, isto é,* $(1, a) \notin \Omega_2$*. Assim, para o evento a, o conjunto* $A = A_1 \dot{\cup} A_2$*, em que* $A_1 = \{1\}$ *e* $A_2 = \{2\}$*. A partir do passo 2.4(b2), o sistema tenta evoluir, com o evento* $\varepsilon\varepsilon a$*, mas essa transição não faz parte da evolução do sistema, pois a transição* $(1, a)$ *não existe. O mesmo ocorre para o evento b. As transições* $b\varepsilon\varepsilon$*,* $\varepsilon b\varepsilon$ *e* $\varepsilon\varepsilon b$*, são geradas, e apenas a transição* $\varepsilon\varepsilon b$ *evolui normalmente, para o estado* $(2, 2, 2, N)$*. As transições* $b\varepsilon\varepsilon$ *e* $\varepsilon b\varepsilon$ *não fazem parte da evolução do sistema, pois a transição* $(2, b)$ *não existe. Logo, o algoritmo do T-Verifier induz a criação de transições que não são capazes de serem evoluídos com os eventos que a rotulam. Então, devem ser descartadas, pois não fazem parte das possibilidades do sistema.*

O *T-Verifier*, baseia-se no verificador apresentado em [11]. Logo, o *T-Verifier* busca sequências de L_N que tem a mesma projeção das sequências de L. Em outras palavras, o *T-Verifier* acompanha sequências em $\mathcal{L}(G_N)$, para os m agentes, comparando-as com sequências em $\mathcal{L}(G)$. Caso hajam ciclos rotuladas por F, significa que exite pelo menos uma sequência de falha que se confunde com sequência normais em todos os agentes $i \in A$.

O exemplo a seguir considera uma nova observação para o sistema da figura 3.2, resultando em um sistema não codiagnosticável com a observação dinâmica apresentada na figura 3.11.

Exemplo 3.11. *Seja o sistema mostrado na figura 3.2. A observação para os seus dois agentes locais é mostrado na figura 3.11. Suponha* σ_f *como único evento de falha, então o autômato T-Verifier é mostrado na figura 3.16.*

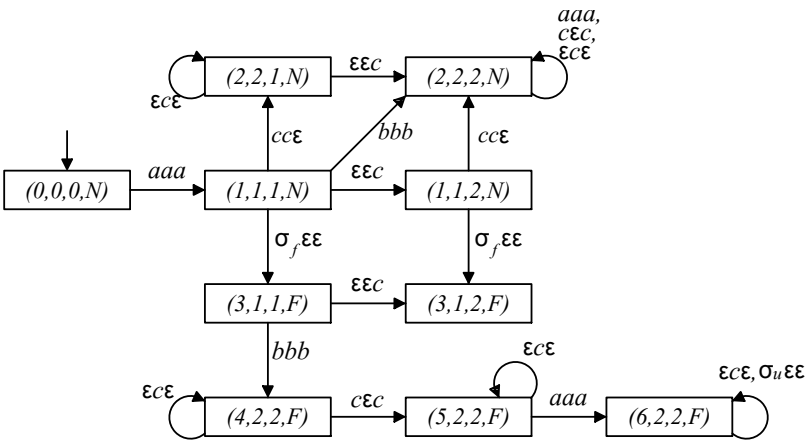


Figura 3.16: Autômato *T-Verifier*.

Pode-se observar que no autômato V, da figura 3.16, há um ciclo F-real, uma vez que o estado $(6, 2, 2, F)$ possui um autolaço com o evento $\sigma_u\varepsilon\varepsilon$. Logo, de acordo com o teorema 3.1, o sistema é não codiagnosticável com relação à observação dinâmica dos agentes 1 e 2 e o conjunto de falhas Σ_f. A não codiagnosticabilidade pode ser percebida, pois existe uma sequência de falha, $s_f = a\sigma_f bca\sigma_u^n$, com $n \in \mathbb{N}$, que se confunde com a sequência normal para o agente 1, $s_{N1} = aba$, e para o agente 2, $s_{N2} = abca$, como pode ser vista no T-Verifier.

No próximo capítulo será mostrado um novo algoritmo para a construção de um verificador para a verificação da codiagnosticabilidade da linguagem de um sistema com observação dinâmica, baseado no verificador para sistemas com observação estática apresentado em [15]. Também será feita uma comparação entre o *C-Verifier*, o *T-Verifier* e esse novo verificador.

Capítulo 4

Verificador Estendido para Sistemas a Eventos Discretos com Observação Dinâmica

No capítulo anterior foram apresentados os verificadores disponíveis na literatura capazes de verificar a codiagnosticabilidade da linguagem de um sistema, considerando o caso da observação dinâmica baseada na transição. Neste capítulo será apresentado um novo verificador, denominado Verificador Estendido que recebe este nome pois é uma extensão do verificador apresentado em [15], e é capaz de verificar a codiagnosticabilidade de sistemas com observação dinâmica.

4.1 Verificador Estendido

Esta seção visa apresentar os passos necessários para a construção do Verificador Estendido. Para isso, serão apresentadas algumas definições necessárias para o seu entendimento.

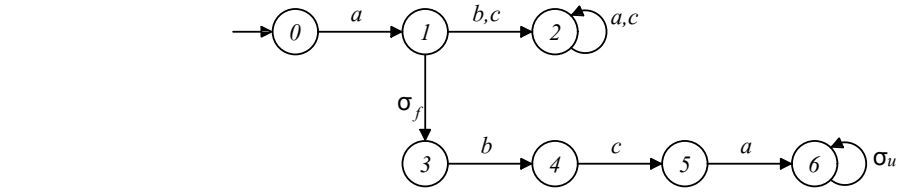
Definição 4.1. *A função de índices $I_{nd} : TR(G) \rightarrow 2^A$, é definida como o conjunto que contém os agentes locais capazes de observar a transição $(x, \sigma) \in TR(G)$. Matematicamente a função de índices é definida como:*

$$I_{nd}(x, \sigma) = \{i \in A : (x, \sigma) \in \Omega_i\}$$

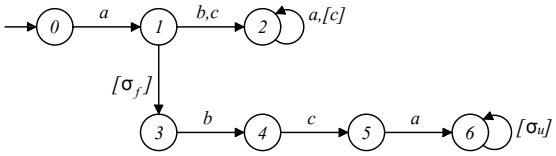
O exemplo a seguir ilustrará a função de índices de algumas transições para o sistema apresentado na figura 4.1, que será o sistema utilizado para os exemplos neste capítulo.

Exemplo 4.1. *Seja o sistema apresentado na figura 4.1a considerando dois agentes locais, com a observação dinâmica para cada agente mostrada nas figuras 4.1b e 4.1c.*

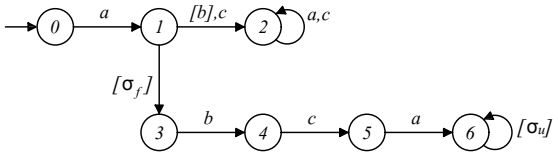
Então, a função de índices para a transição $(0, a)$*, do autômato* G*, é* $I_{nd}(0, a) = \{1, 2\}$ *uma vez que tanto o agente 1 como o agente 2 são capazes de observar essa transição. Já para a transição* $(1, b)$*, a função de índices é* $I_{nd}(1, b) = \{1\}$*, já que apenas o agente 1 é capaz de observar essa transição. Por fim, a função de índices para a transição* $(6, \sigma_u)$ *é* $I_{nd}(6, \sigma_u) = \emptyset$*, uma vez que nenhum agente é capaz de observar essa transição.*



(a) Autômato G



(b) Autômato G_1: Observação para o agente 1



(c) Autômato G_2: Observação para o agente 2

Figura 4.1: Autômato G com sua respectiva observação para o agente 1 e 2.

Com o entendimento da função $I_{nd}(x, \sigma)$ é possível construir o autômato G_{FI}. O autômato G_{FI} será obtido seguindo o algoritmo 4.1.

Algoritmo 4.1. *Autômato* G_{FI}

Entrada: *Autômato* G_F*.*

Saída: *Autômato* G_{FI}*.*

1: Construa o autômato G_{FI} *a partir do autômato* G_F*:*

1.1: Defina a função $R : X_F \times \Sigma \to \Sigma_{FI}$*, em que:*

$$R((x, \sigma)) = \begin{cases} \sigma, & se\ I_{nd}(x, \sigma) = \emptyset, \\ \sigma^{I_{nd}(x,\sigma)}, & se\ I_{nd}(x, \sigma) \neq \emptyset. \end{cases}$$

1.2: Construa o autômato $G_{FI} = (X_F, \Sigma_{FI}, f_{FI}, \Gamma_{FI}, x_{0_F})$*, em que:*

- $\Sigma_{FI} = \{\tilde{\sigma} = R((x_f, \sigma)) : (x_f \in X_F) \wedge (\sigma \in \Sigma)\}$;
- $f_{FI}(x_f, R(x_f, \sigma)) = f_F(x_f, \sigma)$, $\forall\ (x_f, \sigma) \in X_F \times \Sigma$;
- $\Gamma_{FI}(x_f) = R(x_f, \Gamma_F(x_f))$, $\forall\ x_f \in X_F$.

Note que o autômato G_{FI} é obtido a partir do autômato G_F, descrito pelo passo 2 do algoritmo 2.2, com os seus eventos renomeados de acordo com a seguinte regra:

Para toda transição (x, σ) de G_F, se $I_{nd}(x, \sigma)$ for vazio, então mantenha o evento como σ, caso contrário σ será substituído por $\sigma^{I_{nd}(x,\sigma)}$. O exemplo a seguir ilustra a obtenção do autômato G_{FI}.

Exemplo 4.2. *Considere o autômato G_F mostrado na figura 2.17b obtido a partir do sistema da figura 4.1. O autômato G_{FI} será obtido seguindo o algoritmo 4.1 considerando que a observação dinâmica é dada nas figuras 4.1b e 4.1c. A figura 4.2 mostra o autômato G_{FI}, em que o evento a da transição $(0N, a)$ foi rotulado com o conjunto $\{1, 2\}$, já que ambos os agentes são capazes de visualizar a ocorrência desse evento nesta transição, e o mesmo ocorreu com os eventos das transições $(3F, b)$, $(4F, c)$ e $(5F, a)$. Já o evento σ_f e σ_u não foram rotulados, já que nenhum dos agentes é capaz de observar as transições $(1N, \sigma_f)$ e $(6F, \sigma_u)$. É importante ressaltar que nas figuras foram suprimidos os parênteses do conjunto $Ind(x, \sigma)$, por simplicidade, ou seja, o evento $a^{\{1,2\}}$, por exemplo, está representado apenas como $a^{1,2}$.*

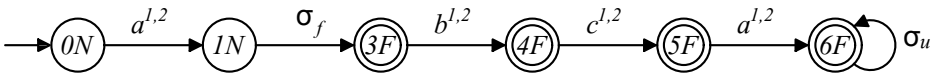


Figura 4.2: Autômato G_{FI}.

O autômato G_{FI} será utilizado para o cálculo do verificador. O cálculo do verificador será realizado entre os autômatos que representam o comportamento sem falha do sistema, renomeados levando-se em consideração a observação parcial de cada agente, com o autômato G_{FI}.

Os autômatos que representam o comportamento sem falha do sistema, renomeados levando em consideração a observação parcial de cada agente serão denotados por $\tilde{G}_{Ni}$, com $i \in A$. Os autômatos $\tilde{G}_{Ni}$, com $i \in A$, serão obtidos seguindo o algoritmo 4.2.

Algoritmo 4.2. *Autômatos $\tilde{G}_{Ni}$, com $i \in A$*

Entrada: *Autômato G_N.*
Saída: *Autômatos $\tilde{G}_{N1}, \tilde{G}_{N2}, \ldots, \tilde{G}_{Nm}$.*

1: Construa os autômatos $\tilde{G}_{Ni}$, para todo $i \in A$, a partir do autômato G_N:

1.1: Defina a função $R_i : X_N \times \Sigma_N \to \tilde{\Sigma}_{Ri}$, em que:

$$R_i(x, \sigma) = \begin{cases} \sigma, & se\ (x, \sigma) \in \Omega_i, \\ \sigma_{Ri}, & se\ (x, \sigma) \notin \Omega_i. \end{cases}$$

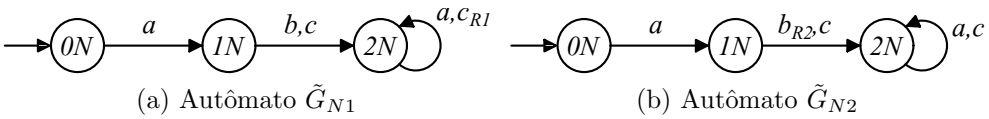
1.2: Construa o autômato $\tilde{G}_{Ni} = (X_N, \tilde{\Sigma}_{Ri}, \tilde{f}_{Ni}, \tilde{\Gamma}_{Ni}, x_{0_N})$*, em que:*

- $\tilde{\Sigma}_{Ri} = \{\tilde{\sigma} = R_i((x_n, \sigma)) : (x_n \in X_N) \wedge (\sigma \in \Sigma_N)\}$;
- $\tilde{f}_{Ni}(x_n, R_i(x_n, \sigma)) = f_N(x_n, \sigma)$, $\forall\ (x_n, \sigma) \in X_N \times \Sigma_N$;
- $\tilde{\Gamma}_{Ni}(x_n) = R_i(x_n, \Gamma_N(x_n))$, $\forall\ x_n \in X_N$.

Os autômatos $\tilde{G}_{Ni}$, com $i \in A$, seguem o mesmo princípio dos autômatos G_{Ni} descritos no passo 3 do algoritmo 2.2, ou seja, a partir do autômato G_N, obtido no passo 1 do algoritmo 2.2, renomeiam-se os eventos não observáveis levando-se em consideração a observação de cada agente local. Logo, para cada agente será obtido um $\tilde{G}_{Ni}$, com $i \in A$. A diferença nos $\tilde{G}_{Ni}$, com $i \in A$, obtidos a partir da observação dinâmica com relação aos G_{Ni}, com $i \in A$, obtidos para o caso com observação estática, é que na observação dinâmica deve-se renomear cada transição não observável, ao invés de se renomear todos os eventos não observáveis, como era no caso da observação estática.

O exemplo a seguir ilustra os autômatos $\tilde{G}_{N1}$ e $\tilde{G}_{N2}$ para o sistema mostrado na figura 4.1, levando em consideração a observação dinâmica apresentada nas figuras 4.1b e 4.1c.

Exemplo 4.3. *Considere o autômato* G_N *mostrado na figura 2.16c obtido a partir do sistema da figura 4.1. Então os autômatos* $\tilde{G}_{N1}$ *e* $\tilde{G}_{N2}$*, considerando a observação dinâmica mostrada nas figuras 4.1b e 4.1c, são dados na figura 4.3. Pode-se observar que* $\tilde{G}_{N1}$ *mostrado na figura 4.3a diferencia-se do* G_{N1} *mostrado na figura 2.16a apenas na transição* $(1N, c)$*, pois para o sistema com observação estática, o evento c era não observável para o agente 1, logo qualquer ocorrência de c seria renomeada, enquanto que para o sistema com observação dinâmica o evento c é observável na transição* $(1N, c)$ *e não observável na transição* $(2N, c)$*. Já o autômato* $\tilde{G}_{N2}$ *mostrado na figura 4.3b e o mostrado na figura 2.16b são idênticos, uma vez que na observação estática para o agente 2, o evento b é não observável e na observação dinâmica a transição* $(1N, b)$ *também possui o evento b não observável.*



(a) Autômato $\tilde{G}_{N1}$ (b) Autômato $\tilde{G}_{N2}$

Figura 4.3: Obtenção dos Autômatos de comportamento normal.

A partir desse ponto será apresentada a construção do autômato verificador G_{VI}, denominado Verificado Estendido, considerando um sistema com observação de dois agentes locais, isto é, $\tilde{G}_{Ni}$, com $i \in \{1, 2\}$, e G_{FI}.

Algoritmo 4.3. *Autômato G_{VI}*

Entrada: *Autômatos $\tilde{G}_{N1}, \tilde{G}_{N2}$ e G_{FI}.*

Saída: *Autômato G_{VI}.*

1: Construa o autômato verificador G_{VI} a partir de $\tilde{G}_{Ni}$, com $i \in \{1,2\}$, e G_{FI}. $G_{VI} = Ac(X_{VI}, \Sigma_{VI}, f_{VI}, \Gamma_{VI}, x_{0_{VI}})$, em que:

1.1: $X_{VI} = X_{N1} \times X_{N2} \times X_{FI}$;

1.2: $\Sigma_{VI} = \tilde{\Sigma}_{R1} \cup \tilde{\Sigma}_{R2} \cup \Sigma_{FI}$;

1.3: $x_{0_{VI}} = (x_{0_{N1}}, x_{0_{N2}}, x_{0_{FI}})$;

1.4: $f_{VI}((x_{N1}, x_{N2}, x_{FI}), \sigma)$

$$= \begin{cases} (\tilde{f}_{N1}(x_{N1},\sigma), \tilde{f}_{N2}(x_{N2},\sigma), f_{FI}(x_{FI},\sigma)), & se\ [I_{nd}(x_{FI},\sigma) = \{1,2\}] \wedge \\ & [\sigma \in \tilde{\Gamma}_{N1}(x_{N1})] \wedge \\ & [\sigma \in \tilde{\Gamma}_{N2}(x_{N2})], \\ (\tilde{f}_{N1}(x_{N1},\sigma), x_{N2}, f_{FI}(x_{FI},\sigma)), & se\ [I_{nd}(x_{FI},\sigma) = \{1\}] \wedge \\ & [\sigma \in \tilde{\Gamma}_{N1}(x_{N1})], \\ (x_{N1}, \tilde{f}_{N2}(x_{N2},\sigma), f_{FI}(x_{FI},\sigma)), & se\ [I_{nd}(x_{FI},\sigma) = \{2\}] \wedge \\ & [\sigma \in \tilde{\Gamma}_{N2}(x_{N2})], \\ (x_{N1}, x_{N2}, f_{FI}(x_{FI},\sigma)), & se\ I_{nd}(x_{FI},\sigma) = \emptyset, \\ (\tilde{f}_{N1}(x_{N1},\sigma), x_{N2}, x_{FI}), & se\ \sigma \in \tilde{\Sigma}_{R1} \setminus \Sigma, \\ (x_{N1}, \tilde{f}_{N2}(x_{N2},\sigma), x_{FI}), & se\ \sigma \in \tilde{\Sigma}_{R2} \setminus \Sigma, \\ \textit{Indefinido}, & \textit{Caso contrário;} \end{cases}$$

1.5: $\Gamma_{VI}((x_{N1}, x_{N2}, x_{FI}))$

$= \{\sigma \in \Gamma_{FI}(x_{FI}) : I_{nd}(x_{FI},\sigma) = \emptyset \vee [I_{nd}(x_{FI},\sigma) \neq \emptyset \quad t.q.\ \forall i \in I_{nd}(x_{FI},\sigma),\ \sigma \in \tilde{\Gamma}_{Ni}(x_{Ni})]\} \cup [(\tilde{\Gamma}_{N1}(x_{N1}) \setminus \Sigma) \cup (\tilde{\Gamma}_{N2}(x_{N2}) \setminus \Sigma)]$.

A função de transição de G_{VI} segue o mesmo princípio da composição paralela, ou seja, evolui com os eventos comuns entre o G_{FI} e $\tilde{G}_{Ni}$, com $i \in A$ que serão sincronizados pelos índices dos eventos de G_{FI}, mas se o evento é particular a evolução ocorre apenas no autômato que o possui. Isso pode ser percebido avaliando os casos da função de transição de G_{VI}. Do primeiro ao terceiro caso da função de transição de G_{VI}, o sistema evolui com o evento $\sigma \in \Sigma_{FI}$ nos autômatos $\tilde{G}_{Ni}$ e G_{FI} se o evento σ está ativo no estado atual de cada $\tilde{G}_{Ni}$ com $i \in I_{nd}(x_{FI},\sigma)$. Por exemplo, suponha um sistema com observação de dois agentes locais e que o estado atual do verificador seja (x_{N1}, x_{N2}, x_{FI}) e que o evento $\sigma^{\{1\}}$ está ativo no estado x_{FI}. Então, deve-se ver se o evento σ está ativo em x_{N1}, caso esteja, deve-se alcançar o estado $((\tilde{f}_{N1}(x_{N1},\sigma), x_{N2}, f_{FI}(x_{FI},\sigma))$. Caso o evento $\sigma^{\{1,2\}}$ esteja ativo no estado x_{FI}, então deve-se ver se o evento σ está ativo em x_{N1} e em x_{N2}. Caso

esteja, deve-se alcançar o estado $((\tilde{f}_{N1}(x_{N1},\sigma),\tilde{f}_{N2}(x_{N2},\sigma),f_{FI}(x_{FI},\sigma))$. Caso esteja ativo apenas em x_{N1} e não em x_{N2}, ou esteja ativo em x_{N2} e não em x_{N1}, ou não esteja ativo em nenhum dos dois estados, a transição não existirá. O quarto caso da função de transição é responsável por evoluir com os eventos particulares do autômato G_{FI}. Por exemplo, suponha novamente que o estado atual do verificador seja (x_{N1},x_{N2},x_{FI}) e que o evento σ está ativo no estado x_{FI}. Como σ não possui índices rotulando-o, isto é, $I_{nd}(x_{FI},\sigma)=\emptyset$, então σ não precisa sincronizar a sua evolução com nenhum outro autômato que faz parte do estado de G_{VI}. Logo, o estado alcançado será $(x_{N1},x_{N2},f_{FI}(x_{FI},\sigma))$. O quinto e sexto caso da função de transição segue o mesmo princípio que o anterior, ou seja, evolui com os eventos particulares, mas dessa vez com os eventos particulares dos autômatos $\tilde{G}_{Ni}$, com $i\in A$. Por exemplo, para o estado (x_{N1},x_{N2},x_{FI}), se o evento σ_{R1} está ativo em x_{N1}, então o sistema alcançará o estado $((\tilde{f}_{N1}(x_{N1},\sigma_{R1}),x_{N2},x_{FI})$.

A função de eventos ativos é dividida basicamente em dois conjuntos:

(i) $\{\sigma\in\Gamma_{FI}(x_{FI}) : I_{nd}(x_{FI},\sigma)=\emptyset \vee [I_{nd}(x_{FI},\sigma)\neq\emptyset \text{ t.q. } \forall i\in I_{nd}(x_{FI},\sigma),\ \sigma\in\tilde{\Gamma}_{Ni}(x_{Ni})]\}$;

(ii) $(\tilde{\Gamma}_{N1}(x_{N1})\setminus\Sigma)\cup(\tilde{\Gamma}_{N2}(x_{N2})\setminus\Sigma)$.

O conjunto *(i)* possui os eventos particulares de G_{FI} além dos eventos $\sigma\in\Sigma_{FI}$ que sincronizam com os eventos dos autômatos $\tilde{G}_{Ni}$, em que $i\in I_{nd}(x_{FI},\sigma)$. Já o conjunto *(ii)* possui todos os eventos renomeados de cada $\tilde{G}_{Ni}$, com $i\in A$.

A função de transição de G_{VI} pode ser facilmente estendida para m agentes locais, sendo que os três primeiros casos se resumem a um único caso, em que o estado x_{FI} evolui com o evento σ, junto com os estados dos autômatos $\tilde{G}_{Ni}$, desde que a seguinte sentença matemática seja satisfeita: $I_{nd}(x_{FI},\sigma)\neq\emptyset \ \wedge\ [\forall i\in I_{nd}(x_{FI},\sigma),\ \sigma\in\tilde{\Gamma}_{Ni}(x_{Ni})]$. Já o quinto e o sexto caso da função de transição resumem-se a um único caso, em que os estados x_{Ni}, com $i\in A$, evoluem com o evento σ sempre que $\sigma\in\tilde{\Sigma}_{Ri}\setminus\Sigma$.

O algoritmo 4.4[1] resume todos os passos para a obtenção de G_{VI} a partir de G, $G_1, G_2,\ldots,G_m$, e representa a condição necessária e suficiente para a verificação da codiagnosticabilidade de sistemas a eventos discretos com observação dinâmica.

Algoritmo 4.4. *Construção do Verificador Estendido*

Entrada: *Autômatos G, G_1, $G_2,\ldots,G_m$ e o conjunto Σ_f.*
Saída: *Existência de caminhos cíclicos no autômato G_{VI}.*

1: Construa o autômato G_N que modela o comportamento sem falha de G (para se obter G_N siga o primeiro passo do algoritmo 2.2);

[1] O algoritmo 4.4 é apresentado novamente no Apêndice A com todos os passos destrinchados.

2: Construa o autômato G_F que modela o comportamento de falha de G (para se obter G_F siga o segundo passo do algoritmo 2.2);

3: Construa o autômato G_{FI} de acordo com o algoritmo 4.1;

4: Construa os autômatos $\tilde{G}_{Ni}$, para todo $i \in A$, de acordo com o algoritmo 4.2;

5: Construa o autômato verificador G_{VI} de acordo com o algoritmo 4.3;

6: Verifique a existência de caminhos cíclicos $(x_{VI_1}, \sigma_1, x_{VI_2}, \sigma_2, \ldots, \sigma_{k-1}, x_{VI_k})$ em G_{VI} com pelo menos um $j \in \{1, 2, \ldots, k-1\}$ tal que $\sigma_j \in \Sigma_{FI}$, e o rótulo de x_{FI} em $x_{VI_j} = (x_{N1}, x_{N2}, \ldots, x_{Nm}, x_{FI})$ seja igual a F.

Observação 4.1. *É importante ressaltar que se o sistema possui observação estática os passos 3 e 5 do algoritmo 4.4 culminam no passo 4 do algoritmo 2.2. Em outras palavras, se a observação for estática, o algoritmo 4.3, responsável pela construção de G_{VI}, recai na composição paralela, e o algoritmo 4.4 é capaz de diagnosticar o sistema no caso estática.*

Comparando o algoritmo 2.2 com o algoritmo 4.4 pode-se ver que os dois primeiros passos para ambos os algoritmos são os mesmos. Mas o algoritmo 2.2, pode ter o seu uso substituído pelo uso do algoritmo 4.4, uma vez que o algoritmo 4.4 pode ser aplicado tanto no caso estático como no caso dinâmico.

No primeiro passo do algoritmo 4.4 é construído o autômato G_N que tem como linguagem L_N, isto é, todas as sequências que não possuem eventos de falha. Já no segundo passo é construído o autômato G_F que modela o comportamento de falha do sistema e tem como linguagem gerada $\mathcal{L}(G_F) = \overline{L \setminus L_N}$, e como linguagem marcada $\mathcal{L}_m(G_F) = L \setminus L_N$. No terceiro passo, a função de índices rotula cada evento de G_F com os índices dos agentes que são capazes de observar a ocorrência daqueles eventos. No passo seguinte, a função de renomeação tem o papel de renomear em G_N apenas os eventos que são não observáveis para o agente em questão, fazendo com que na composição do passo seguinte esses eventos possam ocorrer sempre que possível, e o autômato G_{VI} sincronize apenas os eventos observáveis comuns a $\tilde{G}_{Ni}$, para $i \in A$, e G_{FI}. Isso fará com que o verificador exiba apenas as sequências de G_{FI} que possuam as mesmas projeções das sequências do autômato G_N, uma vez que apenas os eventos observáveis podem ocorrer simultaneamente. O último passo está relacionado com a questão da verificação da propriedade de codiagnosticabilidade da linguagem, de um sistema a eventos discretos, com observação dinâmica, utilizando-se verificadores. As definições a seguir darão base para a demonstração do teorema que justifica esse último passo.

Definição 4.2. *A projeção $\theta_{I_0} : \mathcal{L}(G_{VI}) \rightarrow \Sigma_{FI}^*$ pode ser definida recursivamente como:*

$$\theta_{I_0}(\varepsilon) = \varepsilon,$$

$$\theta_{I_0}(s\sigma) = \begin{cases} \theta_{I_0}(s), & se\ \sigma \in \tilde{\Sigma}_{Ri} \setminus \Sigma,\ \forall i \in A \\ \theta_{I_0}(s)\sigma, & caso\ contrário \end{cases}$$

Definição 4.3. *A projeção $\theta_{I_i} : \mathcal{L}(G_{VI}) \rightarrow \Sigma_{VI}^*$, com $i \in A$, pode ser definida recursivamente como:*

$$\theta_{I_i}(\varepsilon) = \varepsilon,$$

$$\theta_{I_i}(s\sigma) = \begin{cases} \theta_{I_i}(s)\sigma, & se\ [i \in I_{nd}(f_{VI}(x_{0_{VI}}, s), \sigma)] \vee [\sigma \in \tilde{\Sigma}_{Ri} \setminus \Sigma] \\ \theta_{I_i}(s), & caso\ contrário \end{cases}$$

Definição 4.4. *A função de renomeação inversa é definida como:*

$$R_I^{-1} : \Sigma_{VI} \rightarrow \Sigma$$
$$\sigma_{Ri} \mapsto \sigma$$
$$\sigma^{I_{nd}} \mapsto \sigma$$

em que $\sigma_{Ri} = R_i(\sigma)$ e $\sigma^{I_{nd}} = R(\sigma)$. A função pode ser facilmente estendido para o domínio Σ_{VI}^*, como $R_I^{-1}(s_{VI}\sigma_{VI}) = R_I^{-1}(s_{VI})R_I^{-1}(\sigma_{VI})$ para todo $s_{VI} \in \Sigma_{VI}^*$ e $\sigma_{VI} \in \Sigma_{VI}$. Além disso, $R_I^{-1}(\varepsilon) = \varepsilon$.

A função R_I^{-1} retira todos os índices e renomeações que as funções R e R_i tenham colocado. Como as sequências resultado das projeções θ_{I_0} e θ_{I_i}, com $i \in A$, possuem índices e renomeações das funções R e R_i, as seguintes notações serão adotadas: $\theta_{I_{R0}}(t) = R_I^{-1}(\theta_{I_0}(t))$ e $\theta_{I_{Ri}}(t) = R_I^{-1}(\theta_{I_i}(t))$, para denotar as sequências resultado das projeções θ_{I_0} e θ_{I_i} sem os índices e as renomeações. Com essas definições segue o lema.

Lema 4.1. *Seja $t \in \mathcal{L}(G_{VI})$, então para todo $i \in A$, tem-se:*

$$P_{\Omega_i}(\theta_{I_{R0}}(t)) = P_{\Omega_i}(\theta_{I_{Ri}}(t)). \tag{4.1}$$

Demonstração. A prova é realizada para $i = 1$ e o mesmo procedimento pode ser realizado para todo $i \in A$. A prova é feita por indução no comprimento da sequência $t \in \mathcal{L}(G_{VI})$.

Se $||t|| = 0$, isto é, $t = \varepsilon$, então 4.1 se mantém, pois $\theta_{I_{R0}}(\varepsilon) = \varepsilon$ e $\theta_{I_{R1}}(\varepsilon) = \varepsilon$.

Hipótese de Indução: Suponha que 4.1 seja verdade para $||t|| = p$, isto é, $P_{\Omega_1}(\theta_{I_{R0}}(t)) = P_{\Omega_1}(\theta_{I_{R1}}(t))$.

Considere a sequência $t\tilde{\sigma} \in \mathcal{L}(G_{VI})$, com $\tilde{\sigma} \in \Sigma_{VI}$, Logo, $||t\tilde{\sigma}|| = p + 1$. Tem-se que:

$$P_{\Omega_1}(\theta_{I_{R0}}(t\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R0}}(t)\theta_{I_{R0}}(\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R0}}(t))P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})),$$

e

$$P_{\Omega_1}(\theta_{I_{R1}}(t\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(t)\theta_{I_{R1}}(\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(t))P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma})).$$

Se $P_{\Omega_1}(\theta_{I_{R0}}(t\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(t\tilde{\sigma}))$, então $P_{\Omega_1}(\theta_{I_{R0}}(t))P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(t))P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma}))$. Como a H.I. mostra que $P_{\Omega_1}(\theta_{I_{R0}}(t)) = P_{\Omega_1}(\theta_{I_{R1}}(t))$, logo, basta mostrar que $P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma}))$.

Existem cinco casos para $\tilde{\sigma}$: (i) $\tilde{\sigma} = \sigma^{1,2}$ ou (ii) $\tilde{\sigma} = \sigma^1$ ou (iii) $\tilde{\sigma} = \sigma^2$ ou (iv) $\tilde{\sigma} = \sigma$ ou (v) $\tilde{\sigma} = \sigma_{Ri}$.

Para os casos (i) e (ii) tem-se que $\theta_{I_0}(\tilde{\sigma}) = \sigma^{1,2}$ e $\theta_{I_0}(\tilde{\sigma}) = \sigma^1$, respectivamente. Então, $P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})) = \sigma$, pois $\tilde{\sigma}$ tem o índice 1, logo, é observável para o agente 1. Como $\theta_{I_{R1}}(\tilde{\sigma}) = \sigma$, logo, $P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma})) = \sigma$. Então $P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma})) = \sigma$.

Para os casos (iii) e (iv), tem-se que $\theta_{I_0}(\tilde{\sigma}) = \sigma^2$ e $\theta_{I_0}(\tilde{\sigma}) = \sigma$, respectivamente. Então, $P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})) = \varepsilon$, pois $\tilde{\sigma}$ não tem o índice 1, logo, não é observável para o agente 1. Como $\theta_{I_{R1}}(\tilde{\sigma}) = \varepsilon$, logo, $P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma})) = \varepsilon$. Então $P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma})) = \varepsilon$.

Para o caso (v), tem-se que $\theta_{I_0}(\tilde{\sigma}) = \varepsilon$, logo, $P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})) = \varepsilon$. Se $\tilde{\sigma} = \sigma_{R1}$, então $\theta_{I_{R1}}(\tilde{\sigma}) = \sigma$, mas $P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma})) = \varepsilon$. Se $\tilde{\sigma} = \sigma_{R2}$, então $\theta_{I_{R1}}(\tilde{\sigma}) = \varepsilon$, e $P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma})) = \varepsilon$. Então, $P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma})) = \varepsilon$.

Assim, como $P_{\Omega_1}(\theta_{I_{R0}}(\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(\tilde{\sigma}))$ para os cinco casos. Então, $P_{\Omega_1}(\theta_{I_{R0}}(t\tilde{\sigma})) = P_{\Omega_1}(\theta_{I_{R1}}(t\tilde{\sigma}))$, que completa a prova por indução. ■

Teorema 4.1. *Sejam L e L_N linguagens prefixo-fechadas geradas por G e G_N, respectivamente. Considere a projeção $P_{\Omega_i} : \mathcal{L}(G) \to \Sigma_{o_i}^*$, para $i \in A$, e o conjunto de eventos de falha Σ_f. Então, L não será codiagnosticável com relação à P_{Ω_i} e Σ_f se, e somente se, existir um caminho cíclico em G_{VI}, $(x_{VI_1}, \sigma_1, x_{VI_2}, \sigma_2, \ldots, \sigma_{k-1}, x_{VI_k})$, que satisfaz à seguinte condição:*

$$\begin{aligned} &\exists j \in \{1, 2, \ldots, k-1\} \text{ tal que } \sigma_j \in \Sigma_{FI}, \text{ e o rótulo de } x_{FI} \text{ em} \\ &x_{VI_j} = (x_{N1}, x_{N2}, \ldots, x_{Nm}, x_{FI}) \text{ seja igual a } F. \end{aligned} \tag{4.2}$$

Demonstração. Sem perda de generalidade, suponha que existam apenas dois agentes locais, isto é, $A = \{1, 2\}$

($\Leftarrow$) Suponha que exista um ciclo em G_{VI}, $(x_{VI_1}, \sigma_1, x_{VI_2}, \sigma_2, \ldots, \sigma_{k-1}, x_{VI_k})$, que satisfaz à condição 4.2. Uma vez que exista um rótulo de x_{FI} em $x_{VI_j} = (x_{N1}, x_{N2}, x_{FI})$ igual a F, então, pela construção de A_l e G_{VI} todos os rótulos dos x_{FI} em x_{VI_j} serão F, para todo $j \in \{1, \ldots, k-1\}$. Isso implica que existe uma

sequência de eventos $s_{VI}t_{VI} \in \mathcal{L}(G_{VI})$ tal que $\sigma_f \in s_{VI}$[2], em que $\sigma_f \in \Sigma_f$, e $t_{VI} = (\sigma_1, \ldots, \sigma_{k-1})^p$, $p \in \mathbb{N}$ e $||t_{VI}|| > n$, $\forall n \in \mathbb{N}$.

Considere a sequência $s_{VI}t_{VI}$. Uma vez que $\sigma_f \in s_{VI}$, então $\sigma_f \in \theta_{I_{R0}}(s_{VI})$. Além disso, a condição 4.2, diz que $\sigma_j \in \Sigma_{FI}$, logo, essa condição pode ser representada como $\theta_{I_{R0}}(t_{VI}) \neq \varepsilon$.

Definem-se as sequências $s_f = \theta_{I_{R0}}(s_{VI}t_{VI}) \in \mathcal{L}(G)$, $\omega_1 = \theta_{I_{R1}}(s_{VI}t_{VI})$ e $\omega_2 = \theta_{I_{R2}}(s_{VI}t_{VI}) \in \mathcal{L}(G_N)$. Observe que, $||\theta_{I_{R0}}(s_{VI}t_{VI})|| \geq 1 + n$. Pelo lema 4.1 tem-se que $P_{\Omega_1}(\theta_{I_{R0}}(s_{VI}t_{VI})) = P_{\Omega_1}(\theta_{I_{R1}}(s_{VI}t_{VI}))$ e $P_{\Omega_2}(\theta_{I_{R0}}(s_{VI}t_{VI})) = P_{\Omega_2}(\theta_{I_{R2}}(s_{VI}t_{VI}))$, isto é, $P_{\Omega_1}(s_f) = P_{\Omega_1}(\omega_1)$ e $P_{\Omega_2}(s_f) = P_{\Omega_2}(\omega_2)$. Uma vez que $||t_{VI}|| > n$, $\forall n \in \mathbb{N}$, então a sequência $s_f = \theta_{I_{R0}}(s_{VI}t_{VI})$ é arbitrariamente longa. Logo, isso leva à violação da codiagnosticabilidade.

($\Rightarrow$) Suponha que L é não codiagnosticável com relação P_{Ω_1}, P_{Ω_2} e Σ_f. Então, existe pelo menos uma sequência $s_f = st \in \mathcal{L}(G_F)$, em que $\sigma_f \in s$ e $||t|| > n$, $\forall n \in \mathbb{N}$, e ω_1, $\omega_2 \in \mathcal{L}(G_N)$, tal que $P_{\Omega_1}(s_f) = P_{\Omega_1}(\omega_1)$ e $P_{\Omega_2}(s_f) = P_{\Omega_2}(\omega_2)$.

Seja $s_f = s'\sigma$, em que $s' \in \Sigma^*$ e $\sigma \in \Sigma$, então $P_{\Omega_1}(s_f) = P_{\Omega_1}(s'\sigma) = P_{\Omega_1}(s')P_{\Omega_1}(\sigma) = P_{\Omega_1}(s')\sigma$, se $(f(x_{0_F}, s'), \sigma) \in \Omega_1$. Caso contrário, $P_{\Omega_1}(s_f) = P_{\Omega_1}(s')P_{\Omega_1}(\sigma) = P_{\Omega_1}(s')$.

Como $P_{\Omega_1}(s_f) = P_{\Omega_1}(\omega_1)$, então $P_{\Omega_1}(s_f) = \sigma_1 \ldots \sigma_k = P_{\Omega_1}(\omega_1)$. Seja s_i e $s_i' \in \Sigma^*$ de maneira que s_f seja representado como $s_f = s_i\sigma_i s_i'$. Então, para cada σ_i, com $i \in \{1, \ldots, k\}$, pela função R do algoritmo 4.1, σ_i será alterado para σ_i^1, pois $(f(x_{0_F}, s_i), \sigma_i) = (x_f, \sigma_i) \in \Omega_1$, isto é, $\{1\} \subseteq I_{nd}(x_f, \sigma_i)$, pelo menos.

O mesmo procedimento pode ser repetido para o caso em que $P_{\Omega_2}(s_f) = P_{\Omega_2}(\omega_2)$. Então, $P_{\Omega_2}(s_f) = e_1 \ldots e_l = P_{\Omega_2}(\omega_2)$, em que cada e_j, com $j \in \{1, \ldots, l\}$, será alterado, pela função R, para e_j^2.

Observe que se $s_f = s_i\sigma_i s_i'$, tal que, $P_{\Omega_1}(s_f) = P_{\Omega_1}(s_i)\sigma_i P_{\Omega_1}(s_i')$ e $P_{\Omega_2}(s_f) = P_{\Omega_2}(s_i)\sigma_i P_{\Omega_2}(s_i')$, então σ_i será alterado pela função R para $\sigma_i^{1,2}$, isto é, existe e_j em $P_{\Omega_2}(s_f) = e_1 \ldots e_l$, tal que, $\sigma_i = e_j$. Caso $P_{\Omega_1}(s_f) = P_{\Omega_1}(s_i)P_{\Omega_1}(s_i')$ e $P_{\Omega_2}(s_f) = P_{\Omega_2}(s_i)P_{\Omega_2}(s_i')$, então $(f(x_{0_F}, s_i), \sigma_i) = (x_f, \sigma_i) \notin \Omega_1 \cup \Omega_2$. Logo, a função R não altera o evento σ_i, pois $I_{nd}(x_f, \sigma_i) = \emptyset$.

Os passos anteriores mostram que a sequência $s_f \in \mathcal{L}(G_F)$ após o algoritmo 4.1 se torna a sequência $s_{f_I} \in \mathcal{L}(G_{FI})$, em que cada um dos seus eventos recebe índices a partir da função R, baseada na observação dos agentes 1 e 2.

Seja $s_{f_I} = s_I t_I$ de maneria que s_I e t_I sejam as sequências s e t, respectivamente, após o passo 1.1 do algoritmo 4.1, isto é, após o acréscimo dos índices pela função R. A partir do algoritmo 4.3, para cada evento $\tilde{\sigma} \in t_I$, existem quatro possibilidades: (i) $\tilde{\sigma} = \sigma^{1,2}$ ou (ii) $\tilde{\sigma} = \sigma^1$ ou (iii) $\tilde{\sigma} = \sigma^2$ ou (iv) $\tilde{\sigma} = \sigma$.

Para $\tilde{\sigma} = \sigma^{1,2}$, a função de transição f_{VI} do algoritmo 4.3 evolui do estado

[2]Esse abuso de notação é utilizado para representar que na sequência s_{VI} existe um evento σ_f, em que $\sigma_f \in \Sigma_f$.

atual $x_{VI} = (x_{N1}, x_{N2}, x_{FI})$, se existir o evento σ ativo no estado x_{N1} e no estado x_{N2}. Mas como $P_{\Omega_1}(s_f) = P_{\Omega_1}(\omega_1) \Rightarrow P_{\Omega_1}(s_f) = \sigma_1 \ldots \sigma_k = P_{\Omega_1}(\omega_1)$ e $P_{\Omega_2}(s_f) = P_{\Omega_2}(\omega_2) \Rightarrow P_{\Omega_2}(s_f) = e_1 \ldots e_l = P_{\Omega_2}(\omega_2)$, então, $\sigma^{1,2}$ tem um correspondente σ em ω_1 e um correspondente e em ω_2, em que $\sigma = e$. Logo, a transição $(x_{VI}, \sigma^{1,2})$ evolui para $(\tilde{f}_{N1}(x_{N1}, \sigma), \tilde{f}_{N2}(x_{N2}, \sigma), f_{FI}(x_{FI}, \sigma))$. Para $\tilde{\sigma} = \sigma^1$, a função de transição f_{VI} evolui do estado atual $x_{VI} = (x_{N1}, x_{N2}, x_{FI})$, se existir o evento σ ativo no estado x_{N1}. Mas como $P_{\Omega_1}(s_f) = P_{\Omega_1}(\omega_1) \Rightarrow P_{\Omega_1}(s_f) = \sigma_1 \ldots \sigma_k = P_{\Omega_1}(\omega_1)$, então, σ^1 tem um correspondente σ em ω_1. Logo, a transição (x_{VI}, σ^1) evolui para $(\tilde{f}_{N1}(x_{N1}, \sigma), x_{N2}, f_{FI}(x_{FI}, \sigma))$. O procedimento para $\tilde{\sigma} = \sigma^2$ é similar ao adotado para $\tilde{\sigma} = \sigma^1$. Para $\tilde{\sigma} = \sigma$, a função de transição f_{VI} evolui apenas com o estado de G_{FI}, isto é, o estado $x_{VI} = (x_{N1}, x_{N2}, x_{FI})$ evolui para $(x_{N1}, x_{N2}, f_{FI}(x_{FI}, \sigma))$.

Assim, para cada evento $\tilde{\sigma} \in t_I$ o sistema evolui, uma vez que $P_{\Omega_1}(s_f) = P_{\Omega_1}(\omega_1)$ e $P_{\Omega_2}(s_f) = P_{\Omega_2}(\omega_2)$. Como $||t|| > n$, $\forall n \in \mathbb{N}$, então $||t_I|| > n$, $\forall n \in \mathbb{N}$. Logo, haverá uma sequência arbitrariamente longa pertencente à $\mathcal{L}(G_{VI})$. Como existe uma sequência arbitrariamente longa em G_{VI} associada à sequencia $s_{fI} = s_I t_I$ e t_I é a sequência após a falha, então há um caminho cíclico com pelo menos um evento de Σ_{FI} que satisfaz à condição 4.2. ■

Exemplo 4.4. *Considere o autômato G apresentado na figura 4.1 e considere dois agentes locais, com a observação dinâmica para esse sistema, mostrados nas figuras 4.1a e 4.1b. A partir do primeiro passo do algoritmo 4.4, obtêm-se o autômato G_N, que é mostrado na figura 2.16c. O segundo passo consiste em obter o autômato G_F, que é apresentado na figura 2.17b. No terceiro passo calcula-se o autômato G_{FI} a partir de G_F, mostrado na figura 4.2. No quarto passo, a partir da observação dinâmica dos dois agentes que observam o sistema, obtêm-se os autômatos $\tilde{G}_{N1}$ e $\tilde{G}_{N2}$, dados pela figura 4.3. Com os autômatos $\tilde{G}_{N1}$, $\tilde{G}_{N2}$ e G_{FI} é possível seguir para o quinto passo, obtendo-se o autômato G_{VI} mostrado na figura 4.4.*

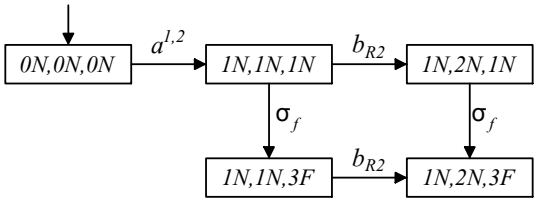


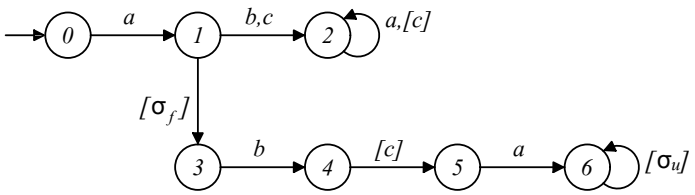
Figura 4.4: Verificador Estendido

O sexto passo consiste em buscar por ciclos em G_{VI} que sejam rotulados por F e tenham pelo menos um evento $\sigma \in \Sigma_{FI}$, $\forall i \in A$. Pode-se observar que o autômato G_{VI} não possui ciclos. Logo, esse sistema é codiagnosticável com relação à observação dinâmica dos agentes mostrados nas figuras 4.1a e 4.1b e o conjunto de falhas Σ_f, segundo o teorema 4.1.

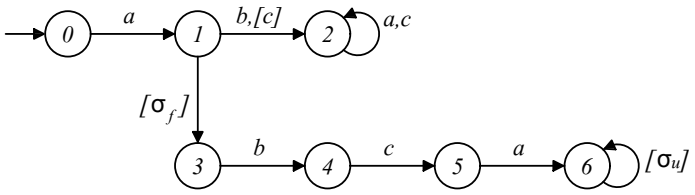
Observação 4.2. *É importante ressaltar que σ_f, apresentado na figura 4.1, considerando o caso da observação estática, é não codiagnosticável para os agentes 1 e 2, cujas observações eram $\Sigma_{o_1} = \{a, b\}$ e $\Sigma_{o_2} = \{a, c\}$, respectivamente, como pôde ser visto no exemplo 2.12. Além disso, ainda que houvesse um único agente que fosse capaz de observar todos os eventos, exceto σ_f e σ_u, esse sistema ainda seria não diagnosticável e por consequência não codiagnosticável, uma vez que codiagnosticabilidade implica em diagnosticabilidade. O interessante é que esse mesmo sistema, que era não diagnosticável e por consequência não codiagnosticável na observação estática, passou a ser diagnosticável considerando a observação dinâmica.*

A observação 4.2 constitui um interessante trabalho futuro, pois a observação dinâmica pode ser abordada como uma nova maneira para tornar um sistema diagnosticável. Até então, se um sistema fosse não diagnosticável, era possível torná-lo diagnosticável com o acréscimo de sensores ou a partir de uma remodelagem do sistema. Assim, com a utilização da observação dinâmica, um sistema anteriormente não diagnosticável pode se tornar diagnosticável, sem a necessidade dessas mudanças.

O exemplo a seguir considera o autômato mostrado na figura 4.1a, com uma nova observação para os seus dois agentes mostrada na figura 4.5.



(a) Autômato G_1: Observação para o agente 1

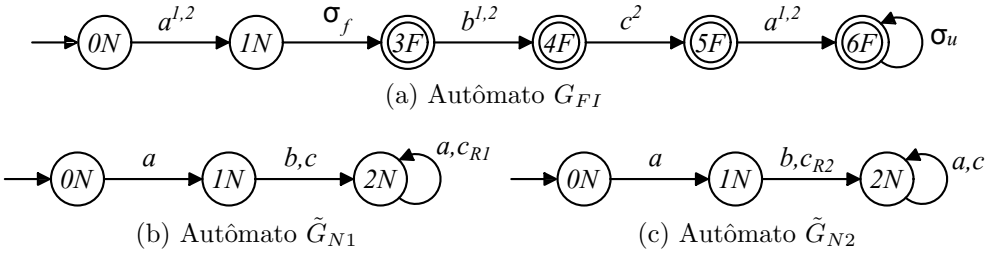


(b) Autômato G_2: Observação para o agente 2

Figura 4.5: Autômato G com sua respectiva observação para o agente 1 e 2.

Exemplo 4.5. *Considere o autômato G apresentado na figura 4.1a considerando dois agentes locais, com a observação dinâmica para cada agente mostrada na figura 4.5. Suponha σ_f como único evento de falha, então o autômato G_{FI}, assim como, os autômatos $\tilde{G}_{N1}$ e $\tilde{G}_{N2}$ são apresentados na figura 4.6. A partir dos autômatos da figura 4.6 é possível construir o autômato G_{VI}, que é mostrado na figura 4.7.*

Observe que a sequência de falha em G_{FI} é $s_f = a\sigma_f bca\sigma_u^n$, para $n \in \mathbb{N}$. Essa sequência se confunde com a sequência normal $s_{N1} = aba$, para o agente 1, e com



(a) Autômato G_{FI}

(b) Autômato $\tilde{G}_{N1}$

(c) Autômato $\tilde{G}_{N2}$

Figura 4.6: Autômatos necessários para a construção de G_{VI}.

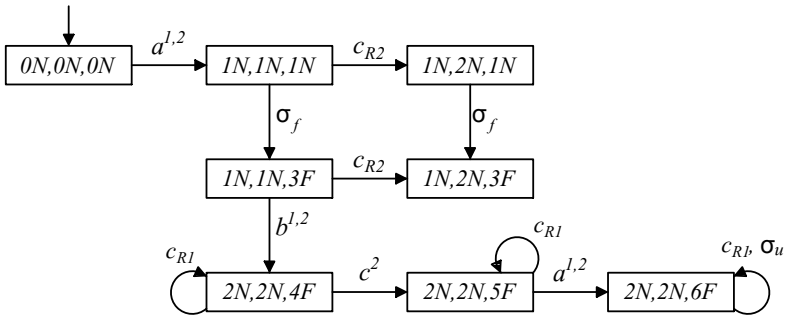


Figura 4.7: Verificador Estendido

a sequência normal $s_{N2} = abca$, para o agente 2. Essa confusão pode ser vista no Verificador Estendido, uma vez que no Verificador Estendido é possível ver a sequência $s_f = a^{\{1,2\}}\sigma_f b^{\{1,2\}}c^2a^{\{1,2\}}\sigma_u^n$.

4.2 Análise de Complexidade do Algoritmo 4.4

A complexidade computacional do algoritmo 4.4 pode ser calculada considerando que a existência de ciclos descritos pelo passo 6 é um problema de detecção de componentes fortemente conexos em grafos, uma vez que todos os estados desses ciclos são rotulados por F. Então, a complexidade computacional será linear com relação ao número de estados e transições de G_{VI}, já que buscas por componentes fortemente conexos tem complexidade $O(|V|+|E|)$, em que, $|V|$ o número de vértices (estados) e $|E|$ é o número de arestas (transições) em um grafo (vide [30]). Assim a complexidade computacional pode ser calculada analisando os passos necessários para a obtenção do autômato G_{VI}.

A tabela 4.1 mostra o número máximo de estados e transições para cada autômato que é calculado para a obtenção do Verificador Estendido considerando m agentes locais, isto é, $A = \{1, \ldots, m\}$. O primeiro passo do algoritmo 4.4 consiste em construir o autômato G_N a partir do autômato A_N através da composição produto. Como A_N possui um único estado com um autolaço rotulado com os eventos de $\Sigma_N = \Sigma \setminus \Sigma_f$, então o número máximo de estados e transições de G_N serão $|X|$ e $|X|(|\Sigma| - |\Sigma_f|)$, respectivamente. O segundo passo do algoritmo 4.4 é a construção

Tabela 4.1: Complexidade computacional do algoritmo 4.4

	N^o de Estados	N^o de Transições
G	$\|X\|$	$\|X\| \times \|\Sigma\|$
A_N	1	$\|\Sigma\| - \|\Sigma_f\|$
G_N	$\|X\|$	$\|X\|(\|\Sigma\| - \|\Sigma_f\|)$
A_l	2	$2\|\Sigma_f\|$
G_l	$2\|X\|$	$2\|X\| \times \|\Sigma\|$
G_F	$2\|X\|$	$2\|X\| \times \|\Sigma\|$
G_{FI}	$2\|X\|$	$2\|X\| \times \|\Sigma\|$
$\tilde{G}_{Ni}$	$\|X\|$	$\|X\|(\|\Sigma\| - \|\Sigma_f\|)$
G_{VI}	$2\|X\|^{m+1}$	$2\|X\|^{m+1}[\|\Sigma\| + m(\|\Sigma\| - \|\Sigma_f\|)]$
Complexidade		$O(m\|X\|^{m+1} \times \|\Sigma\|)$

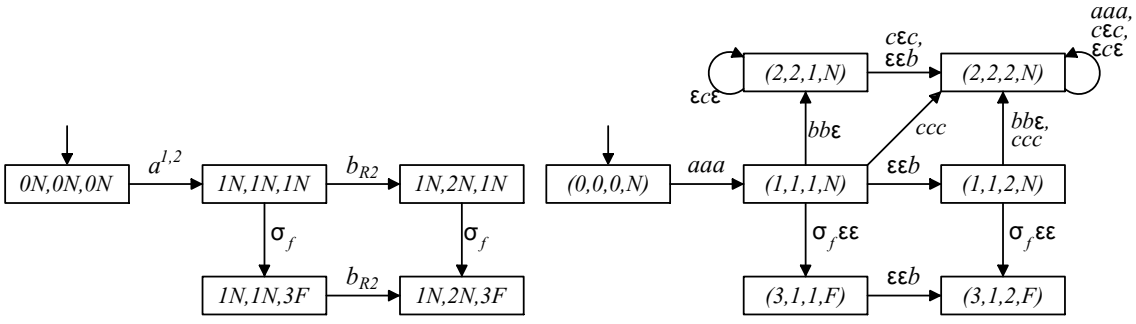
de G_F a partir do autômato G_l. Para se construir o autômato G_l, primeiro é necessário construir o autômato A_l que possui dois estados, N e F, e duas transições rotuladas pelos eventos de Σ_f. Na sequência, G_l é calculado como $G_l = G||A_l$. Observe que $\mathcal{L}(G_l) = \mathcal{L}(G)$ e que os estados de G_l são iguais a (x, N) ou (x, F), em que $x \in X$. Portanto, o número máximo de estados de G_l é $2|X|$. O autômato G_F é calculado tomando-se a parte coacessível de G_l, isto é, $G_F = CoAc(G_l)$. Logo, no pior caso G_F tem o mesmo número de estados e transições de G_l. No terceiro passo o autômato G_{FI} é obtido renomeando-se os eventos de G_F levando-se em consideração as observações dos agentes locais segundo a função R, definida no passo 1.1 do algoritmo 4.1. O autômato G_{FI}, por ser apenas a renomeação dos eventos de G_F, tem o mesmo número de estados e transições de G_F. No quarto passo os autômatos $\tilde{G}_{Ni}$, para $i \in A$, são obtidos a partir de G_N renomeando-se os eventos não observáveis segundo a função R_i, definida na passo 1.1 do algoritmo 4.2, gerando m autômatos com o mesmo número de estados e transições de G_N. Finalmente, no quinto passo o autômato G_{VI} é obtido a partir da função de transição no passo 1.4 do algoritmo 4.3. Observe que a função de transição de G_{VI} sincroniza os eventos em Σ_{FI} com os eventos de $\tilde{\Sigma}_{Ri}$, para $i \in A$, e os eventos particulares de G_{FI} e $\tilde{G}_{Ni}$ evoluem separadamente, assim como na composição paralela. Então, uma vez que o número de estados de $\tilde{G}_{Ni}$ e G_{FI} são $|X|$ e $2|X|$ respectivamente, o número de estados de G_{VI} no pior caso é igual a $2|X|^{m+1}$. Além disso, o número de transições de G_{VI}, no pior caso, é igual a $2|X|^{m+1}[|\Sigma| + m(|\Sigma| - |\Sigma_f|)]$, uma vez que para cada estado há pelo menos $|\Sigma| + m(|\Sigma| - |\Sigma_f|)$ transições saindo. Então a partir do número de transições de G_{VI} é possível calcular a complexidade computacional do algoritmo 4.4 que é $O(m|X|^{m+1} \times |\Sigma|)$, assim como a complexidade do algoritmo 2.2 [35][3].

[3]Os artigos [35] e [27] enunciam e demonstram a complexidade computacional do algoritmo 2.2 e do *T-Verifier*, respectivamente, como sendo $O(m|X|^{m+1}(|\Sigma| - |\Sigma_f|))$. Para se obter $O(m|X|^{m+1} \times |\Sigma|)$ a partir de $O(m|X|^{m+1}(|\Sigma| - |\Sigma_f|))$ basta observar que $(|\Sigma| - |\Sigma_f|) \leq |\Sigma|$.

4.3 Comparação entre Verificadores para Sistemas com Observação Dinâmica

Os verificadores apresentados no capítulo 3 e o Verificador Estendido, apresentado neste capítulo, possuem pequenas diferenças quanto a complexidade computacional, mas tanto o *C-Verifier*, quanto o *T-Verifier* e ainda o Verificador Estendido possuem complexidade computacional polinomial com relação ao número de estados e transições.

Apesar da complexidade computacional do *T-Verifier* e do Verificador Estendido serem exatamente iguais, isto é, $O(m|X|^{m+1} \times |\Sigma|)$ (vide [27][3]), o Verificador Estendido apresenta uma pequena vantagem com relação ao *T-Verifier* neste quesito, pois o Verificador Estendido busca sequências de L_N que têm a mesma projeção das sequências de $\mathcal{L}(G_F) = \overline{L \setminus L_N}$, enquanto que o *T-Verifier*, por ser baseado no verificador apresentado em [11], busca sequências de L_N que têm a mesma projeção das sequências de L. Assim, para os casos em que o fecho de prefixo da linguagem de falha é menor que a linguagem de G o Verificador Estendido será menor que o *T-Verifier*. A figura 4.8 ilustra essa diferença, comparando o Verificador Estendido e o *T-Verifier* para o sistema da figura 4.1, uma vez que para este sistema o autômato $G_F = CoAc(G_l)$ é menor que o autômato G_l, isto é, o fecho de prefixo da linguagem de falha é menor que a linguagem de G.

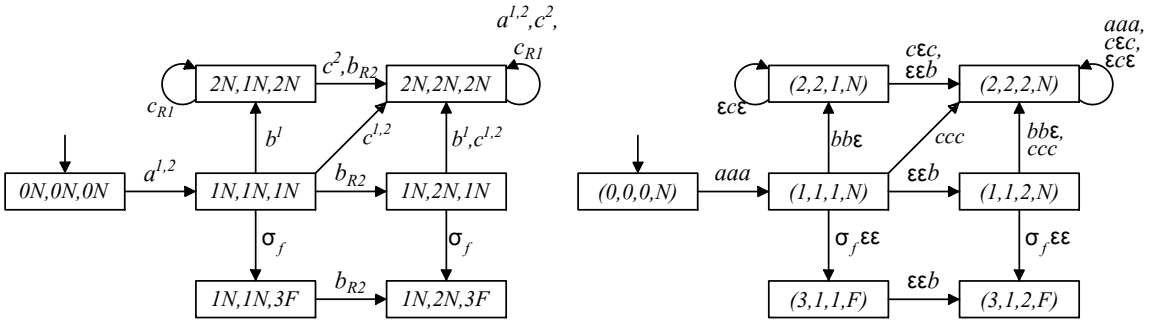


(a) Verificador Estendido (b) Autômato *T-Verifier*

Figura 4.8: Verificadores.

Se o Verificador Estendido utilizasse o $G_F = G_l$ ao invés de $G_F = CoAc(G_l)$, o Verificador Estendido seria praticamente igual ao *T-Verifier*, diferenciando-se apenas pelo nome dos estados e pela maneira que o *T-Verifier* rotula as transições. A figura 4.9 mostra o Verificador Estendido e o *T-Verifier* para o sistema da figura 4.1, em que se utilizou o $G_F = G_l$ ao invés de $G_F = CoAc(G_l)$ e como consequência o Verificador Estendido recaiu no *T-Verifier*.

Outra vantagem que o Verificador Estendido apresenta sobre o *T-Verifier* encontra-se justamente nos eventos que rotulam as transições. No Verificador Es-



(a) Verificador Estendido Modificado (b) Autômato *T-Verifier*

Figura 4.9: Verificadores.

tendido os eventos possuem índices ou rótulos que trazem a informação de quais autômatos estão sincronizando na ocorrência daquela transição. Por exemplo, no Verificador Estendido da figura 4.8a a transição $(0N, 0N, 0N, a^{1,2})$ o índice $1,2$ do evento a indica que tanto $\tilde{G}_{N1}$ como $\tilde{G}_{N2}$ observam a ocorrência desse evento. Para a transição $(1N, 1N, 1N, \sigma_f)$, como o evento σ_f não possui nem rótulo nem índice, então, apenas o autômato G_{FI} evolui, pois nenhum dos dois agentes é capaz de observar esse evento. Para a transição $(1N, 1N, 1N, b_{R2})$, o rótulo $R2$ indica que o evento b é não observável para o agente 2, logo o sistema evoluirá apenas com o agente 2. No *T-Verifier* o procedimento é o mesmo, mas há uma sequência de eventos que rotulam as transições. Por exemplo, no *T-Verifier* da figura 4.8b nas transições $((0,0,0,N), aaa)$, $((1,1,1,N), \sigma_f\varepsilon\varepsilon)$ e $((1,1,1,N), \varepsilon\varepsilon b)$ a sequência de eventos aaa, $\sigma_f\varepsilon\varepsilon$ e $\varepsilon\varepsilon b$ trazem as mesmas informações que os eventos $a^{1,2}$, σ_f e b_{R2}, mencionados anteriormente, respectivamente. A desvantagem de se utilizar essa sequencia de eventos é que a medida que se tem outros agentes locais, essa sequência irá aumentar e ficará cada vez mais difícil de se entender quais agentes sincronizam ou não naquele transição. Já no Verificador Estendido essa informação vem de maneira compacta através de dos índices e rótulos.

Há ainda uma outra desvantagem que *T-Verifier* apresenta frente ao Verificador Estendido. O algoritmo do *T-Verifier* induz a criação de transições que não são capazes de serem evoluídas com os eventos que as rotulam. Logo, devem ser descartadas pois não fazem parte das possibilidades do sistema. A figura 4.10 mostra as transições que o algoritmo do *T-Verifier* gera além daquelas que fazem parte da evolução do sistema. No estado $(3,1,2,F)$, por exemplo, o algoritmo gera as transições $a\varepsilon\varepsilon$, $\varepsilon a\varepsilon$, $c\varepsilon\varepsilon$ e $\varepsilon\varepsilon b$ que não tem condição de serem executadas neste estado, logo, devem ser descartadas. O mesmo ocorre para as transições $a\varepsilon\varepsilon$, $\varepsilon a\varepsilon$, $\varepsilon\varepsilon b$ no estado $(1,1,2,N)$, $\varepsilon\varepsilon a$, $b\varepsilon\varepsilon$, $\varepsilon b\varepsilon$ no estado $(2,2,1,N)$ e $c\varepsilon\varepsilon$ no estado $(3,1,1,F)$.

Com relação ao *C-Verifier* o Verificador Estendido também apresenta vantagens no quesito complexidade computacional. A complexidade computacional apresentada em [26] para o *C-verifier* é apresentada como $O(|X|^{m+1})$, uma vez que o *C-*

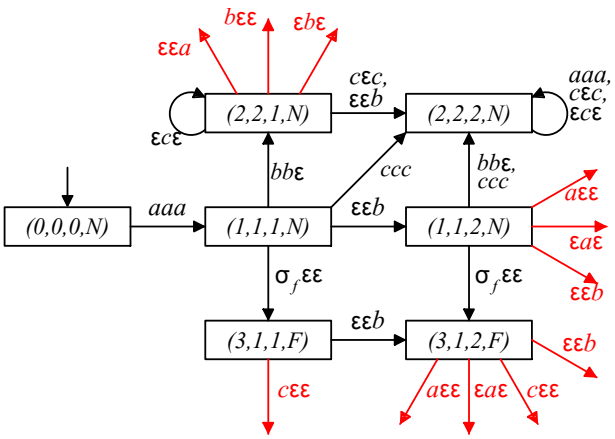


Figura 4.10: Autômato *T-Verifier*

Verifier pode apresentar $2|X|^{m+1}$ estados, em que, m é o número de agentes locais e $|X|$ a quantidade de estados de G. A questão é que a complexidade computacional da busca por componentes fortemente conexos, para a verificação da diagnosticabilidade de uma linguagem, não é calculada sobre a quantidade de estados que um autômato pode gerar, e sim sobre a quantidade de transições, como foi demonstrada o capítulo 2. Então, para a comparação entre o *C-Verifier* e o Verificador Estendido deve-se calcular a quantidade de transições.

A função de transição do *C-Verifier* evolui com cada autômato cluster a partir da transição (x, σ), isto é, $\overline{\Gamma}([c_1(x_1), \ldots, c_m(x_m), (c_p(x_p), l)], (x, \sigma))$. O estado x da transição (x, σ) deve ser um estado do conjunto de estados do cluster $c_p(x_p)$ e o evento σ deve ser observável por algum agente, isto é, $I_p(x, \sigma) = 1$. Como o cluster é o alcance não observável a partir de um estado alcançado por um evento observável, então o evento σ na transição (x, σ) do cluster $c_p(x_p)$ pode ocorrer, no pior caso, $|X|$ vezes. Como a função de transição $\overline{\Gamma}([\cdot], (x, \sigma))$ evolui com os demais clusters além do $c_p(x_p)$, isto é, $c_1(x_1), \ldots, c_m(x_m)$, e em cada cluster o evento σ pode ocorrer, no pior caso, $|X|$ vezes. Então o evento σ pode ocorrer $|X|^{m+1}$ vezes em um estado do *C-Verifier*. Repare que $|X|^{m+1}$ é a quantidade de vezes que um único evento σ pode ocorrer em um estado do *C-Verifier*, e como podem haver $|\Sigma| - |\Sigma_f|$ eventos habilitados em cada cluster. Então, no pior caso, podem haver $|X|^{m+1}(|\Sigma| - |\Sigma_f|)$ eventos habilitados em um único estado do *C-Verifier*. Por fim, como *C-Verifier* possui até $2|X|^{m+1}$ estados. Então, o número máximo de transições será a quantidade de estados vezes a quantidade de eventos habilitados por estado, ou seja, $(2|X|^{m+1}) \times [|X|^{m+1}(|\Sigma| - |\Sigma_f|)]$ que é igual a $2|X|^{2(m+1)}(|\Sigma| - |\Sigma_f|)$. O que já era de se esperar, uma vez que o *C-Verifier* é um autômato não determinístico, e um autômato não determinístico apresenta, no pior caso, $|X|^2 \times |\Sigma|$ transições.

Como no número de transições do *C-Verifier* é $2|X|^{2(m+1)}(|\Sigma| - |\Sigma_f|)$ a sua complexidade computacional é $O(|X|^{2(m+1)} \times |\Sigma|)$. Assim, o Verificador Estendido

possui uma complexidade computacional menor que o *C-Verifier*. Repare ainda que o Verificador Estendido além de apresentar essa vantagem na complexidade computacional sobre o *C-Verifier*, também depende basicamente de composições já bem fundamentadas na literatura, com exceção da função de transição de G_{VI} apresentada no algoritmo 4.3, enquanto que o *C-Verifier* necessita da criação de autômatos clusters e de uma função de transição, $\overline{\Gamma}([c_1(x_1), \ldots, c_m(x_m), (c_p(x_p), l)], (x, \sigma))$, de difícil entendimento.

A comparação entre os verificadores para sistemas com observação dinâmica apresentada nesta seção mostrou que o Verificador Estendido possui menor complexidade computacional que o *C-Verifier* e que apesar de possuir a mesma complexidade que o *T-Verifier*, é capaz de ser menor que este último, uma vez que busca sequências de L_N que tem a mesma projeção das sequências de $\mathcal{L}(G_F) = \overline{L \setminus L_N}$, ao invés de buscar sequências de L_N que tem a mesma projeção das sequências de L, como faz o *T-Verifier*. Além disso, o Verificador Estendido ainda apresenta um certa simplicidade se comparado com os demais, já que não depende da criação de clusters, não utiliza uma sequência de eventos para rotular suas transições e não induz a criação de transições que não podem ser executadas pelo sistema. Finalmente o Verificador Estendido pode ser utilizado para o caso estático, uma vez que nesse caso apresentará o mesmo resultado que o verificador em [15].

Capítulo 5

Conclusões

Neste trabalho foi proposto um novo algoritmo capaz de criar um verificador que verifica a codiagnosticabilidade de uma linguagem em sistemas a eventos discretos com observação dinâmica.

O verificador aqui apresentado, denominado Verificador Estendido, apresenta vantagens com relação aos verificadores existentes até então na literatura. Essas vantagens vêm do fato que o Verificador Estendido possui menor complexidade computacional que o verificador apresentado em [26], denominado *C-Verifier*, e iguala-se ao verificador apresentado em [27], denominado *T-Verifier*, apenas se o fecho de prefixo da linguagem de falha é menor que a linguagem de G. Outra vantagem do Verificador Estendido é que ele é uma extensão natural do trabalho de codiagnosticabilidade apresentado em [15] e, por consequência, pode ser utilizado para a verificação da codiagnosticabilidade no caso da observação estática. Essa última implicação vem do fato já mencionado de que a observação estática é um caso particular da observação dinâmica. Logo, uma vez implementado o algoritmo de verificação do Verificador Estendido, não importa se o sistema possui observação dinâmica ou estática o verificador será capaz de verificar se a linguagem é diagnosticável ou não, uma vez que se for um caso de observação estática o algoritmo apresentará o mesmo resultado que o algoritmo 2.2 mostrado em [15]. Assim, o algoritmo 2.2, pode ser substituído pelo uso do algoritmo 4.4, uma vez que o algoritmo 4.4 pode ser aplicado tanto no caso estático como no caso dinâmico, sem aumentar em nada o custo computacional.

Uma importante observação pode ser feita a partir deste trabalho. Um sistema que era não diagnosticável na observação estática pode se tornar diagnosticável considerando a observação dinâmica. Esse fato abre uma nova oportunidade de tornar sistemas que anteriormente eram não diagnosticáveis, para observação estática, em diagnosticáveis com a observação dinâmica, não necessitando assim, uma remodelagem ou um acréscimo de sensores no sistema.

Como trabalhos futuros, é desejado que a observação dinâmica seja abordada como uma nova maneira para tornar um sistema diagnosticável. Além disso, outro

ponto que pode ser tratado é a questão de bases mínimas, tratadas em [31, 36–39], pois uma vez que se encontre as bases mínimas na caso estático, pode ser que ao utilizar a abordagem da observação dinâmica consiga-se diminuir ainda mais a utilização de sensores.

Referências Bibliográficas

[1] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2 ed. New York, Springer, 2008.

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

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

[4] SAMPATH, M. "A hybrid approach to failure diagnosis of industrial systems". In: *American Control Conference, 2001. Proceedings of the 2001*, v. 3, pp. 2077–2082. IEEE, 2001.

[5] SAMPATH, M., LAFORTUNE, S., TENEKETZIS, D. "Active diagnosis of discrete-event systems", *IEEE Transactions on Automatic Control*, v. 43, n. 7, pp. 908–929, 1998.

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

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

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

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

[10] THORSLEY, D., TENEKETZIS, D. "Diagnosability of stochastic discrete-event systems", *IEEE Transactions on Automatic Control*, v. 50, n. 4, pp. 476–492, 2005.

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

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

[13] KILIC, E. "Diagnosability of fuzzy discrete event systems", *Information Sciences*, v. 178, n. 3, pp. 858–870, 2008.

[14] ZAD, S. H., KWONG, R., WONHAM, W. "Fault diagnosis in discrete-event systems: Incorporating timing information", *IEEE Transactions on Automatic Control*, v. 50, n. 7, pp. 1010–1015, 2005.

[15] 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, n. 7, pp. 1679–1684, 2011.

[16] FANTI, M. P., MANGINI, A. M., UKOVICH, W. "Fault detection by labeled Petri nets in centralized and distributed approaches", *IEEE Transactions on Automation Science and Engineering*, v. 10, n. 2, pp. 392–404, 2013.

[17] CABASINO, M. P., GIUA, A., LAFORTUNE, S., et al. "A new approach for diagnosability analysis of Petri nets using verifier nets", *IEEE Transactions on Automatic Control*, v. 57, n. 12, pp. 3104–3117, 2012.

[18] GAUBERT, S., GIUA, A. "Petri net languages and infinite subsets of Nm", *Journal of Computer and System Sciences*, v. 59, n. 3, pp. 373–391, 1999.

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

[20] CASSEZ, F., TRIPAKIS, S. "Fault diagnosis with static and dynamic observers", *Fundamenta Informaticae*, v. 88, n. 4, pp. 497–540, 2008.

[21] THORSLEY, D., TENEKETZIS, D. "Active acquisition of information for diagnosis and supervisory control of discrete event systems", *Discrete Event Dynamic Systems: Theory and Applications*, v. 17, n. 4, pp. 531–583, 2007.

[22] WANG, W., LAFORTUNE, S., GIRARD, A. R., et al. "Optimal sensor activation for diagnosing discrete event systems", *Automatica*, v. 46, n. 7, pp. 1165–1175, 2010.

[23] WANG, W., LAFORTUNE, S., LIN, F., et al. "Minimization of dynamic sensor activation in discrete event systems for the purpose of control", *IEEE Transactions on Automatic Control*, v. 55, n. 11, pp. 2447–2461, 2010.

[24] RUDIE, K., LAFORTUNE, S., LIN, F. "Minimal communication in a distributed discrete-event system", *IEEE Transactions on Automatic Control*, v. 48, n. 6, pp. 957–975, 2003.

[25] LIN, F. "Control of networked discrete event systems: dealing with communication delays and losses", *SIAM Journal on Control and Optimization*, v. 52, n. 2, pp. 1276–1298, 2014.

[26] WANG, W., GIRARD, A. R., LAFORTUNE, S., et al. "On codiagnosability and coobservability with dynamic observations", *IEEE Transactions on Automatic Control*, v. 56, n. 7, pp. 1551–1566, 2011.

[27] YIN, X., LAFORTUNE, S. "Codiagnosability and coobservability under dynamic observations: Transformation and verification", *Automatica*, v. 61, pp. 241–252, 2015.

[28] CURY, J. E. R. "Teoria de Controle Supervisório de Sistemas a Eventos Discretos". In: *V Simpósio Brasileiro de Automação Inteligente*, p. 7, Canela-Rio Grande do Sul, 2001.

[29] BASILIO, J. C., CARVALHO, L. K., MOREIRA, M. V. "Diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos", *Revista Controle & Automaçao*, v. 21, n. 5, pp. 510–533, 2010.

[30] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L. *Introduction to Algorithms*. 25 ed. Cambridge, MA: MIT Press, 2000.

[31] SANTORO, L. P. M. *Obtenção de Bases Mínimas para o Diagnóstico de Falhas de Sistemas a Eventos Discretos Utilizando Verificadores*. DSc dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro - RJ, BRA, 2013.

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

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

[34] BASILIO, J. C., LAFORTUNE, S. "Robust codiagnosability of discrete event systems". In: *American Control Conference, 2009. ACC'09.*, pp. 2202–2209. IEEE, 2009.

[35] MOREIRA, M. V., BASILIO, J. C., CABRAL, F. G. ""Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems" Versus "Decentralized Failure Diagnosis of Discrete Event Systems": A Critical Appraisal", *IEEE Transactions on Automatic Control*, v. 61, n. 1, pp. 178–181, 2016.

[36] DEBOUK, R., LAFORTUNE, S., TENEKETZIS, D. "On an optimization problem in sensor selection", *Discrete Event Dynamic Systems*, v. 12, n. 4, pp. 417–445, 2002.

[37] TRAVÉ-MASSUYES, L., ESCOBET, T., OLIVE, X. "Diagnosability analysis based on component-supported analytical redundancy relations", *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, v. 36, n. 6, pp. 1146–1160, 2006.

[38] BASILIO, J. C., LIMA, S. T. S., LAFORTUNE, S., et al. "Computation of minimal event bases that ensure diagnosability", *Discrete Event Dynamic Systems*, v. 22, n. 3, pp. 249–292, 2012.

[39] SANTORO, L. P., MOREIRA, M. V., BASILIO, J. C. "Computation of minimal diagnosis bases of Discrete-Event Systems using verifiers", *Automatica*, v. 77, pp. 93–102, 2017.

Apêndice A

Algoritmo do Verificador Estendido

Algoritmo A.1. *Construção do Verificador Estendido*

Entrada: *Autômatos G, G_1, $G_2, \ldots, G_m$ e o conjunto Σ_f.*
Saída: *Existência de caminhos cíclicos no autômato G_{VI}.*

1: Construa o autômato G_N que modela o comportamento normal de G, da seguinte forma:

1.1: Defina $\Sigma_N = \Sigma \setminus \Sigma_f$;

1.2: Construa o autômato A_N formado por um único estado N (também seu estado inicial) com um autolaço com todos os eventos de Σ_N;

1.3: Construa o autômato que modela o comportamento normal de G, onde $G_N = G \times A_N = (X_N, \Sigma, f_N, \Gamma_N, x_{0_N})$;

1.4: Redefina o conjunto de eventos de G_N como sendo Σ_N, i.e., $G_N = (X_N, \Sigma_N, f_N, \Gamma_N, x_{0_N})$.

2: Construa o autômato G_F que modela o comportamento de falha, da seguinte forma:

2.1: Construa o autômato $A_l = (X_l, \Sigma_f, f_l, \Gamma_l, x_{0_l})$, onde $X_l = \{N, F\}$, $x_{0_l} = \{N\}$, $\Gamma_l(N) = \Gamma_l(F) = \sigma_f$, $f_l(N, \sigma_f) = F$ e $f_l(F, \sigma_f) = F$, $\forall\ \sigma_f \in \Sigma_f$;

2.2: Calcule $G_l = G||A_l$ e marque todos os estados de G_l que tem a segunda componente igual a F;

2.3: Calcule G_F tomando a parte coacessível de G_l, i.e., $G_F = CoAc(G_l)$.

3: Construa o autômato G_{FI} a partir do autômato de G_F:

3.1: Defina a função de índices $I_{nd} : TR(G) \to 2^A$ *da seguinte maneira:*

$$I_{nd}(x, \sigma) = \{i \in A : (x, \sigma) \in \Omega_i\}$$

3.2 Defina a função $R : X_F \times \Sigma \to \Sigma_{FI}$*, em que:*

$$R((x, \sigma)) = \begin{cases} \sigma, & se\ I_{nd}(x, \sigma) = \emptyset, \\ \sigma^{I_{nd}(x,\sigma)}, & se\ I_{nd}(x, \sigma) \neq \emptyset. \end{cases}$$

3.3: Construa o autômato $G_{FI} = (X_F, \Sigma_{FI}, f_{FI}, \Gamma_{FI}, x_{0_F})$*, em que:*

- $\Sigma_{FI} = \{\tilde{\sigma} = R((x_f, \sigma)) : (x_f \in X_F) \wedge (\sigma \in \Sigma)\}$;
- $f_{FI}(x_f, R(x_f, \sigma)) = f_F(x_f, \sigma)$, $\forall\ (x_f, \sigma) \in X_F \times \Sigma$;
- $\Gamma_{FI}(x_f) = R(x_f, \Gamma_F(x_f))$, $\forall\ x_f \in X_F$.

4: Construa os autômatos $\tilde{G}_{Ni}$*, para todo* $i \in A$*:*

4.1: Defina a função $R_i : X_N \times \Sigma_N \to \tilde{\Sigma}_{Ri}$*, em que:*

$$R_i(x, \sigma) = \begin{cases} \sigma, & se\ (x, \sigma) \in \Omega_i, \\ \sigma_{Ri}, & se\ (x, \sigma) \notin \Omega_i. \end{cases}$$

4.2: Construa o autômato $\tilde{G}_{Ni} = (X_N, \tilde{\Sigma}_{Ri}, \tilde{f}_{Ni}, \tilde{\Gamma}_{Ni}, x_{0_N})$*, em que:*

- $\tilde{\Sigma}_{Ri} = \{\tilde{\sigma} = R_i((x_n, \sigma)) : (x_n \in X_N) \wedge (\sigma \in \Sigma_N)\}$;
- $\tilde{f}_{Ni}(x_n, R_i(x_n, \sigma)) = f_N(x_n, \sigma)$, $\forall\ (x_n, \sigma) \in X_N \times \Sigma_N$;
- $\tilde{\Gamma}_{Ni}(x_n) = R_i(x_n, \Gamma_N(x_n))$, $\forall\ x_n \in X_N$.

5: Construa o autômato verificador G_{VI} *a partir de* $\tilde{G}_{Ni}$*, com* $i \in A$*, e* G_{FI}*.* $G_{VI} = Ac(X_{VI}, \Sigma_{VI}, f_{VI}, \Gamma_{VI}, x_{0_{VI}})$*, em que:*

1.1: $X_{VI} = X_{N1} \times X_{N2} \times \cdots \times X_{Nm} \times X_{FI}$;

1.2: $\Sigma_{VI} = \tilde{\Sigma}_{R1} \cup \tilde{\Sigma}_{R2} \cup \cdots \cup \tilde{\Sigma}_{Rm} \cup \Sigma_{FI}$;

1.3: $x_{0_{VI}} = (x_{0_{N1}}, x_{0_{N2}}, \ldots, x_{0_{Nm}}, x_{0_{FI}})$;

1.4: $f_{VI}((x_{N1}, x_{N2}, \ldots, x_{Nm}, x_{FI}), \sigma)$

$$= \begin{cases} (\ldots, \tilde{f}_{Ni}(x_{Ni}, \sigma), \ldots, f_{FI}(x_{FI}, \sigma)), & se\ I_{nd}(x_{FI}, \sigma) \neq \emptyset \wedge \\ & [\forall i \in I_{nd}(x_{FI}, \sigma), \\ & \sigma \in \tilde{\Gamma}_{Ni}(x_{Ni})], \\ (x_{N1}, x_{N2}, \ldots, x_{Nm}, f_{FI}(x_{FI}, \sigma)), & se\ I_{nd}(x_{FI}, \sigma) = \emptyset, \\ (x_{N1}, x_{N2}, \ldots, \tilde{f}_{Ni}(x_{Ni}, \sigma), \ldots, x_{Nm}, x_{FI}), & se\ \sigma \in \tilde{\Sigma}_{Ri} \setminus \Sigma, \\ \textit{Indefinido}, & \textit{Caso contrário;} \end{cases}$$

1.5: $\Gamma_{VI}((x_{N1}, x_{N2}, \ldots, x_{Nm}, x_{FI}))$
$= \{\sigma \in \Gamma_{FI}(x_{FI}) : I_{nd}(x_{FI}, \sigma) = \emptyset \vee [I_{nd}(x_{FI}, \sigma) \neq \emptyset \quad t.q. \ \forall i \in I_{nd}(x_{FI}, \sigma), \ \sigma \in \tilde{\Gamma}_{Ni}(x_{Ni})]\} \cup [(\tilde{\Gamma}_{N1}(x_{N1}) \setminus \Sigma) \cup \cdots \cup (\tilde{\Gamma}_{Nm}(x_{Nm}) \setminus \Sigma)]$.

6: Verifique a existência de caminhos cíclicos $(x_{VI_1}, \sigma_1, x_{VI_2}, \sigma_2, \ldots, \sigma_{k-1}, x_{VI_k})$ *em* G_{VI} *com pelo menos um* $j \in \{1, 2, \ldots, k-1\}$ *tal que* $\sigma_j \in \Sigma_{FI}$*, e o rótulo de* x_{FI} *em* $x_{VI_j} = (x_{N1}, x_{N2}, \ldots, x_{Nm}, x_{FI})$ *seja igual a* F*.*
