DIAGNÓSTICO ONLINE DE FALHAS EM SISTEMAS A EVENTOS DISCRETOS MODELADOS POR AUTÔMATOS FINITOS: UMA ABORDAGEM UTILIZANDO REDES DE PETRI

Felipe Gomes de Oliveira Cabral

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

Orientadores: Marcos Vicente de Brito Moreira
Oumar Diene

Rio de Janeiro
Março de 2014

DIAGNÓSTICO ONLINE DE FALHAS EM SISTEMAS A EVENTOS DISCRETOS MODELADOS POR AUTÔMATOS FINITOS: UMA ABORDAGEM UTILIZANDO REDES DE PETRI

Felipe Gomes de Oliveira Cabral

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

Examinada por:

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

Prof. Oumar Diene, D.Sc.

Prof. Paulo Eigi Miyagi, Dr.Eng.

Prof. Patrícia Nascimento Pena, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2014

Cabral, Felipe Gomes de Oliveira

Diagnóstico online de falhas em sistemas a eventos discretos modelados por autômatos finitos: uma abordagem utilizando redes de Petri/Felipe Gomes de Oliveira Cabral. – Rio de Janeiro: UFRJ/COPPE, 2014.

XIV, 98 p.: il.; 29, 7cm.

Orientadores: Marcos Vicente de Brito Moreira
Oumar Diene

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2014.

Referências Bibliográficas: p. 94 – 98.

1. Sistemas a eventos discretos. 2. Redes de Petri. 3. Autômatos. 4. Diagnóstico de falha. 5. Controladores lógicos programáveis. I. Moreira, Marcos Vicente de Brito *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Agradecimentos

Agradeço a Deus. *Porque dele, e por meio dele, e para ele são todas as coisas. A ele, pois, a glória eternamente. Amém!* (Romanos 11. 36).

Agradeço aos meus pais Ronaldo Almeida Cabral e Denise Gomes de Oliveira Cabral porque sem eles esta conquista não seria possível.

Agradeço à minha noiva Julia Rodrigues Chagas pelo companheirismo e paciência durante a elaboração deste trabalho.

Agradeço aos meus professores e orientadores, Marcos Vicente de Brito Moreira e Oumar Diene, por todas as horas gastas de aconselhamento e orientação.

Agradeço à Eletrobras Eletronorte pelo apoio financeiro através do projeto Desenvolvimento de Sistema Inteligente de Diagnóstico de Faltas em Subestações, número 4500078819 e às equipes da UFRJ e da UFMA.

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

DIAGNÓSTICO ONLINE DE FALHAS EM SISTEMAS A EVENTOS DISCRETOS MODELADOS POR AUTÔMATOS FINITOS: UMA ABORDAGEM UTILIZANDO REDES DE PETRI

Felipe Gomes de Oliveira Cabral

Março/2014

Orientadores: Marcos Vicente de Brito Moreira
Oumar Diene

Programa: Engenharia Elétrica

Neste trabalho é proposto um método para o diagnóstico online de falhas em sistemas a eventos discretos modelados por autômatos finitos. O método de diagnóstico é baseado em uma rede de Petri diagnosticadora que é construída em tempo polinomial e requer menos memória do que outros métodos propostos na literatura. Métodos de conversão da rede de Petri diagnosticadora em SFC e em diagrama ladder para implementação em um computador lógico programável (CLP) são apresentados. Problemas relacionados à implementação também são abordados neste trabalho.

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

ONLINE FAULT DIAGNOSIS OF DISCRETE EVENT SYSTEMS MODELED BY FINITE STATE AUTOMATA: A PETRI NET APPROACH

Felipe Gomes de Oliveira Cabral

March/2014

Advisors: Marcos Vicente de Brito Moreira
Oumar Diene

Department: Electrical Engineering

In this work, we propose an online fault diagnosis method for discrete event systems modeled by finite state automata. The diagnosis method is based on the construction of a Petri net diagnoser and requires, in general, less memory than other methods proposed in the literature. In addition, methods for the conversion of the Petri net diagnoser into a sequential function chart and into a ladder diagram for implementation on a programmable logic controller (PLC) are presented. Implementation issues are also addressed in this work.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

σ	Evento genérico
σ_o	Evento observável genérico
σ_u	Evento não observável genérico
σ_{f_k}	Evento de falha do tipo k
λ	Evento sempre ocorrente
Σ	Conjunto de eventos de um SED
Σ_o	Conjunto de eventos observáveis
Σ_{uo}	Conjunto de eventos não observáveis
Σ_f	Conjunto de eventos de falha
Σ_{f_k}	Conjunto de eventos de falha do tipo k
Σ_{N_k}	Conjunto $\Sigma \setminus \Sigma_{f_k}$
Π_f	Partição genérica de Σ_f
$\Sigma^\star$	Fecho de Kleene de um conjunto de eventos Σ
ε	Sequência vazia
$\|\|s\|\|$	Comprimento de uma sequência s
G	Autômato determinístico de estados finitos
G_{nd}	Autômato não determinístico
G_C	Autômato que modela o comportamento normal de G em relação a cada tipo de falha
A_ℓ	Autômato rotulador
G_ℓ	Autômato rotulado
G_{diag}	Autômato diagnosticador
G_{N_k}	Autômato que modela o comportamento normal de G em relação à falha do tipo F_k
$G^a_{N_k}$	Autômato G_{N_k} aumentado
Q	Conjunto de estados de um autômato
$f(\cdot,\cdot)$	Função de transição de um autômato
$\Gamma(\cdot)$	Função de eventos ativos de um estado de um autômato
q_0	Estado inicial de um autômato
Q_m	Conjunto de estados marcados de um autômato
$\mathcal{L}(G)$	Linguagem gerada por um autômato G

L	Linguagem
$\mathcal{L}_m(G)$	Linguagem marcada por um autômato G
L_{N_k}	Linguagem formada por sequências que não contém o evento σ_{f_k}
$L_1 \cup L_2$	Operação de união sobre as linguagens L_1 e L_2
$L_1 \cap L_2$	Operação de interseção sobre as linguagens L_1 e L_2
$L_1 \setminus L_2$	Operação de subtração de conjuntos entre as linguagens L_1 e L_2
$L_1 L_2$	Operação de concatenação entre as linguagens L_1 e L_2
$L^\star$	Fecho de Kleene da linguagem L
$\overline{L}$	Fecho do prefixo da linguagem L
L/s	Pós linguagem de L após a sequência s
$P_s(\cdot)$	Projeção do conjunto de eventos Σ_l para o conjunto de eventos Σ_s
$P_o(\cdot)$	Projeção natural
$P_s^{-1}(\cdot)$	Projeção inversa
$Ac(\cdot)$	Operação de acessibilidade
$CoAc(\cdot)$	Operação de coacessibilidade
$Trim(\cdot)$	Operação *trim*
$G_1 \times G_2$	Composição produto entre os autômatos G_1 e G_2
$G_1 \| G_2$	Composição paralela entre os autômatos G_1 e G_2
$UR(\cdot)$	Alcance não observável de um estado
$Obs(G)$	Observador do autômato G em relação a Σ_o
$\mathcal{N}$	Rede de Petri
P	Conjunto de lugares de uma rede de Petri
T	Conjunto de transições de uma rede de Petri
$Pre(\cdot,\cdot)$	Função de arcos ordinários que conectam lugares a transições
$Post(\cdot,\cdot)$	Função de arcos ordinários que conectam transições a lugares
x_0	Função de marcação inicial do conjunto de lugares
$l(\cdot)$	Função de rotulagem
p_i	Lugar de uma rede de Petri
t_j	Transição de uma rede de Petri
n	Número de elementos do conjunto de lugares P
m	Número de elementos do conjunto de transições T
$x(p_i)$	Número de fichas atribuídas a p_i
$\underline{x}$	Marcação de uma rede de Petri
$\mathcal{N}_C$	Rede de Petri máquina de estados obtida a partir de G_C
$\mathcal{N}_{SO}$	Rede de Petri observadora de estados
$Reach_T(\cdot)$	Função de alcance de lugares
$In_D(\cdot,\cdot)$	Função de arcos inibidores
$\|A\|$	Cardinalidade do conjunto A

Capítulo 1

Introdução

Sistemas a eventos discretos (SEDs) [1, 2] são sistemas em que o conjunto de estados é discreto e cuja evolução se dá através da ocorrência de eventos e não pelo tempo. Exemplos de eventos são o início e o término de uma tarefa, uma mudança em um estado de um sensor ou o apertar de um botão por um operador. Sistemas desse tipo estão presentes em uma série de aplicações, como: sistemas de manufatura, robótica, supervisão de tráfego, sistemas operacionais, gerenciamento de dados, otimização de sistemas distribuídos e logística.

Em SEDs, os eventos são modelados como uma ocorrência instantânea e a natureza discreta desses sistemas faz com que modelos matemáticos baseados em equações diferenciais ou a diferenças não sejam adequados para descrevê-los e analisá-los. Assim, faz-se necessário um formalismo matemático que seja capaz de lidar com as características desses sistemas.

Entre as formas mais utilizadas para representar SEDs destacam-se os autômatos e as redes de Petri [1–4]. Os autômatos são grafos orientados, em que os vértices representam os estados do sistema e os arcos representam as transições que são ocasionadas pela ocorrência de eventos. As redes de Petri são grafos bipartidos cuja característica principal é a representação do estado do sistema, que é feita de maneira distribuída na estrutura da rede. Essa característica faz com que a rede de Petri seja uma representação mais vantajosa do que autômatos para sistemas com grande número de estados.

Sistemas de automação estão sujeitos à ocorrência de falhas que podem alterar o seu comportamento normal, diminuindo sua confiabilidade e desempenho na execução das tarefas para as quais foram projetados. Assim, o estudo sobre diagnóstico de falhas em SEDs é fundamental e consiste em indicar a ocorrência de uma falham, baseado na observação dos eventos gerados pelo sistema. Para isso, é necessário construir um modelo que represente o comportamento normal e o comportamento do sistema após a ocorrência da falha. Muitos trabalhos têm sido publicados na literatura com esse objetivo [5–14] para sistemas modelados tanto por autômatos quanto por redes de Petri.

Em [5, 6] é apresentada uma abordagem para o diagnóstico de falhas em sistemas a eventos discretos modelados por autômatos finitos. O diagnosticador proposto por SAMPATH *et al.* [5, 6] pode ser usado para detecção online de eventos de falha e para a verificação *off-line* da propriedade da diagnosticabilidade da linguagem gerada pelo sistema. Embora esse diagnosticador possa ser implementado diretamente em um computador, isso é geralmente evitado, uma vez que, no pior caso, o espaço de estados do diagnosticador cresce exponencialmente com a cardinalidade do espaço de estados do modelo do sistema [5–7, 15].

Em [5], é afirmado que o diagnóstico online pode ser realizado sem armazenar todo o espaço de estados do diagnosticador, sendo necessário armazenar apenas o seu estado atual e, após a ocorrência de um evento observável, atualizar esse estado. Entretanto, a maneira exata de realizar essa implementação não é detalhada em [5]. Em [7], um método para o diagnóstico online que evita a construção e o armazenamento completo do autômato diagnosticador é proposto. Para isso, um autômato não determinístico G_{nd}, cujo espaço de estados tem, no pior caso, cardinalidade $2 \times |X|$, em que X denota o espaço de estados de G e $|.|$ denota cardinalidade, é calculado. Nesse método, apenas o estado atual do diagnosticador e do autômato G_{nd} precisam ser armazenados para o diagnóstico online. Após a ocorrência de um evento observável, o próximo estado do diagnosticador pode ser obtido online a partir do seu estado atual e a partir do autômato G_{nd} em tempo polinomial. Entretanto,

os detalhes da implementação desse método de diagnóstico em um computador não são apresentados em [7]. Não é de conhecimento do autor nenhum outro trabalho na literatura que aborde o problema da implementação online de diagnosticadores para sistemas modelados por autômatos.

Por outro lado, para sistemas modelados por redes de Petri, existe uma vasta literatura sobre a construção de diagnosticadores online. A forma mais simples de realizar o diagnóstico em sistemas modelados por redes de Petri seria construir o grafo de alcançabilidade da rede de Petri que modela o sistema e, em seguida, obter seu diagnosticador, o que implicaria em substituir o modelo em rede de Petri por um modelo em autômato correspondente. Ao se utilizar essa abordagem, a vantagem de se utilizar redes de Petri para modelagem do sistema é perdida, gerando-se um modelo em autômato cujo grafo pode ser muito maior que a rede de Petri. Para superar essa limitação, diversos outros métodos que usam redes de Petri foram propostos.

Em [16], [17], [18] e [19] o cálculo de um observador de estados que não requer a construção completa do grafo de alcançabilidade da rede de Petri do sistema é apresentado. Para tanto, o conceito de marcação base, que é um subconjunto do espaço de alcançabilidade da rede de Petri, é proposto. Os vetores de marcação da rede de Petri podem ser obtidos como a solução de um sistema linear que depende da marcação base. Em [20], um diagnosticador interpretado baseado na solução online de problemas de computação é proposto. A planta é modelada por uma rede de Petri e é suposto que todos os eventos não observáveis podem ser detectados. Além disso, os autores supõem que duas transições diferentes não podem ser associadas ao mesmo evento. Em [21], um método para obter um modelo em rede de Petri interpretada (RPI) para o sistema, de acordo com uma metodologia *bottom-up*, é apresentado, em que é suposto que o modelo em RPI possui uma função de saída de estado e que duas transições distintas rotuladas pelo mesmo evento precisam sempre ser detectadas pelos símbolos de saída de estado.

Apesar de diversos trabalhos abordarem métodos de obtenção de diagnosticado-

res usando redes de Petri para modelar tanto o sistema quanto o diagnosticador, apenas alguns trabalhos tratam da implementação de um diagnosticador online em um controlador lógico programável (CLP). O CLP é a ferramenta mais usada para o controle discreto de sistemas automatizados e pode ser programado em cinco linguagens definidas na norma internacional IEC 61131-3 [22]: (*i*) diagrama ladder; (*ii*) diagrama de blocos de função; (*iii*) texto estruturado, (*iv*) lista de instruções e (*v*) sequenciamento gráfico de funções (em inglês, *sequential function chart* - SFC). Entre essas cinco linguagens, o diagrama ladder é o mais utilizado pela indústria e está disponível em quase todos os CLPs.

Um CLP pode ser usado exclusivamente para o diagnóstico online ou, dependendo das especificações do sistema, o diagnosticador online pode ser implementado no mesmo CLP usado no controle em malha fechada. A principal vantagem de se implementar o diagnosticador online no mesmo CLP usado para controle do sistema é a redução no número de equipamentos usados para o diagnóstico. Note que, nesse caso, todos os eventos de comando se tornam observáveis para o diagnosticador, sem a necessidade de utilizar sensores ou barramentos de comunicação entre o controlador e o observador.

Em [23], uma plataforma de CLP particular, chamada *softPLC Orchestra*, é usada para o diagnóstico. Nesse caso, o diagnosticador é uma tarefa do CLP, programado em linguagem C, que toma amostras das variáveis globais do CLP e acompanha a evolução do sistema através das transições de estado do autômato diagnosticador. Embora esse esquema de implementação tenha sido aplicado por LUCA *et al.* [23] com sucesso, a extensão desse método para outras plataformas de CLP, que não suportam programação em linguagem C, não é uma tarefa fácil.

Apesar do fato de não existir quase nenhuma literatura sobre implementação em CLPs de diagnosticadores online, existem diversos métodos de conversão de códigos de controle complexos em diagramas ladder [24–30]. Em [31, 32], um problema importante associado à implementação de códigos de controle modelados por autômatos e SFCs, chamado de efeito avalanche, é apresentado junto com um

método para evitá-lo. Em [33] e [34], um algoritmo de evolução que evita o efeito avalanche é apresentado para a conversão de sistemas modelados por redes de Petri em linguagem de texto estruturado. Em [30], um método geral para a conversão de redes de Petri interpretadas para controle em diagramas ladder é proposto, levando a um diagrama ladder que simula o comportamento da rede de Petri e evita o efeito avalanche.

Além disso, outro problema abordado em [30], associado à implementação de redes de Petri em diagramas ladder, ocorre quando um lugar recebe e perde uma ficha simultaneamente após o disparo de duas transições diferentes. Dependendo da implementação em ladder da rede de Petri, a marcação resultante pode ser errada, levando a uma representação incorreta da dinâmica da rede de Petri. Esse problema não é abordado nos demais trabalhos propostos na literatura sobre métodos de conversão de códigos de controle em diagramas ladder.

Este trabalho apresenta uma abordagem por redes de Petri para o diagnóstico online de falhas em um SED modelado por um autômato finito G, cujo conjunto de eventos de falha, Σ_f, pode ser particionado em diferentes conjuntos de falha $\Sigma_{f_k}, k = 1, \ldots, r$, em que r denota o número de tipos de falha. O método é baseado na construção de um autômato G_C, obtido a partir de G e de autômatos G_{N_k}, para $k = 1, \ldots, r$, em que cada autômato G_{N_k} modela o comportamento normal de G em relação ao conjunto de eventos de falha Σ_{f_k}. Em geral, G_{N_k} possui um número menor de estados e transições do que G, levando a uma redução da complexidade computacional do diagnóstico online em comparação com os métodos tradicionais que usam tanto o comportamento normal quanto o de falha do sistema [5, 7].

A técnica de diagnóstico proposta neste trabalho consiste em encontrar os estados alcançáveis de G_C após a observação de uma sequência de eventos e, baseado no conjunto de estados alcançáveis de G_C, verificar se a falha ocorreu ou não. Para tanto, uma rede de Petri diagnosticadora, obtida a partir de uma rede de Petri binária, que é capaz de estimar os estados alcançáveis de G_C após a observação de uma sequência de eventos, é proposta [35–38]. A rede de Petri diagnosticadora

provê uma estrutura para o procedimento de diagnóstico online que facilita a implementação do código do diagnosticador em um computador e é construída em tempo polinomial.

A diferença principal entre a abordagem proposta neste trabalho e as que são propostas em [16–19] é que, neste trabalho, o sistema é modelado por um autômato e a estimativa de estado pode ser eficientemente calculada após a ocorrência de um evento observável em tempo polinomial. Como consequência, não é necessário calcular as marcações base para o sistema. Este trabalho também se diferencia de [20] no sentido em que não é requerido que todos os eventos não observáveis possam ser detectados e que duas transições diferentes não possam ser associadas ao mesmo evento, como é suposto em [20]. Finalmente, a formulação do problema de diagnóstico de falhas apresentada em [21] não tem qualquer similaridade com a abordada nesta dissertação, já que o problema de diagnóstico de falhas abordado neste trabalho baseia-se apenas nas observações dos eventos do sistema.

Outras contribuições deste trabalho são métodos para conversão da rede de Petri diagnosticadora, que descreve o diagnosticador online, em um diagrama SFC e em um diagrama ladder para implementação em CLP. Como a rede de Petri diagnosticadora é uma rede de Petri binária, a conversão em um diagrama SFC é quase direta. A conversão em diagrama ladder para a implementação em CLPs que não suportam a programação em linguagem SFC é inspirada no método proposto em [30]. Como vantagens principais, esse método evita o efeito avalanche e gera um diagrama ladder bem estruturado. Além disso, o problema da implementação em diagrama ladder associado com a remoção e adição simultânea de uma ficha em um lugar após o disparo de duas transições diferentes é abordado e uma solução para esse problema, diferente da proposta em [30], é apresentada.

Este trabalho está organizado da seguinte forma: no capítulo 2 são apresentados conceitos preliminares necessários para o entendimento desta dissertação, bem como uma revisão sobre SEDs modelados por autômatos e redes de Petri e o conceito de diagnosticabilidade usado neste trabalho. No capítulo 3 a rede de Petri diagnosti-

cadora é apresentada e uma análise da complexidade do algoritmo de construção da rede e do processo de diagnóstico online é apresentada. Os fundamentos de CLP e das linguagens de programação SFC e diagrama ladder são apresentados no capítulo 4. No capítulo 5, as técnicas de conversão de redes de Petri diagnosticadoras em diagramas ladder e em SFC são apresentadas. Exemplos são apresentados para ilustrar os resultados obtidos. Finalmente, as conclusões e trabalhos futuros são apresentados no capítulo 6.

Capítulo 2

Sistemas a eventos discretos

Neste capítulo são apresentados fundamentos teóricos de sistemas a eventos discretos necessários para a compreensão e elaboração deste trabalho. Para tanto, este capítulo está estruturado com o objetivo de tratar a modelagem e os formalismos matemáticos usados para descrever sistemas a eventos discretos.

De um modo geral, um sistema é um conjunto de elementos combinados pela natureza, ou pelo homem, de maneira a formar um todo complexo, realizando uma função que não seria possível com nenhum dos componentes individualmente [1]. Os sistemas considerados neste trabalho são sistemas a eventos discretos cujo espaço de estados é um conjunto discreto e, cujas transições de estados são observadas na ocorrência de eventos. Eventos podem ser, por exemplo, uma ação específica (como alguém apertar um botão), uma ocorrência espontânea (como um sistema sair do ar por alguma razão desconhecida) ou o resultado de uma condição que seja satisfeita (como o nível de um líquido em um recipiente exceder um determinado valor).

Dessa forma, um SED é um sistema dinâmico que evolui de acordo com ocorrências de eventos e, assim, faz-se necessário um formalismo matemático capaz de descrever esse tipo de sistema. Esse formalismo deve ser capaz de determinar o estado atual do sistema e ter uma regra de evolução baseada na ocorrência de um evento, ou, de forma genérica, de uma sequência de eventos.

Analogamente, o conjunto de eventos de um SED pode ser considerado um alfabeto do sistema. Assim, sequências de eventos formam palavras e um conjunto

de palavras forma uma linguagem: neste sentido, o conjunto formado por todas as sequências possíveis de serem geradas por um sistema é chamado de linguagem gerada pelo sistema. As linguagens determinam a evolução de estados em um SED a partir da ocorrência de eventos e, portanto, possuem uma função semelhante às das equações diferenciais para descrever sistemas dinâmicos contínuos no tempo [1].

Embora o conhecimento do estado inicial e da linguagem sejam suficientes para modelar um SED, esse tipo de representação é muito complexa do ponto de vista prático. Para contornar esse problema, são usualmente utilizadas estruturas em grafos para representar sistemas e as linguagens geradas por esses sistemas. Para o presente trabalho são considerados dois tipos de formalismos: autômatos e redes de Petri.

2.1 Linguagens

Uma linguagem é um conjunto de sequências de eventos de comprimento finito geradas por um sistema e, dessa forma, constitui-se a informação que, junto com o estado inicial, é suficiente para descrever o comportamento futuro do sistema. Uma linguagem, portanto, é um formalismo matemático que pode ser usado para descrever um SED [1].

2.1.1 Notações e definições

Neste trabalho, a notação Σ representa o conjunto de eventos de um SED, ou seja, é o *alfabeto*. O símbolo e será usado para representar um evento genérico e ε representa a sequência vazia. Se s é uma sequência, seu comprimento será denotado por $\|s\|$. Por convenção, o comprimento da sequência vazia ε é zero. A definição formal de uma linguagem é dada a seguir [1].

Definição 2.1 *(Linguagem) Uma linguagem definida em um conjunto de eventos Σ é um conjunto de sequências de eventos de comprimento finito formadas a partir dos eventos em Σ.*

A linguagem $L = \{\varepsilon, a, b, ac, acb\}$, por exemplo, é formada a partir do conjunto de eventos $\Sigma = \{a, b, c\}$ e possui cinco sequências, incluindo a sequência vazia. Algumas operações envolvendo eventos e sequências podem ser usadas para a formação de novas sequências e, portanto, linguagens. A principal delas é a concatenação de uma ou mais sequências com o objetivo de gerar somente uma. A sequência acb, por exemplo, é formada pela concatenação da sequência ac com o evento, ou sequência de comprimento um, b. Sendo que a própria sequência ac é obtida a partir da concatenação dos eventos a e c, respectivamente. A sequência vazia ε é o elemento neutro da concatenação, de tal forma que $\varepsilon s = s\varepsilon = s$, em que s é uma sequência de eventos.

A linguagem gerada por um SED está contida no conjunto formado por todas as sequências de comprimento finito construídas com os elementos de Σ, incluindo a sequência vazia ε. Esse conjunto é denotado por $\Sigma^\star$, em que a operação $\star$ é chamada de fecho de Kleene. Além disso, em particular, $\emptyset$, Σ e $\Sigma^\star$ são linguagens.

Por fim, considere a sequência $s = tuv$, com t, u, $v \in \Sigma^\star$, então, t é chamada de prefixo de s, u é uma subsequência de s e v é chamada de sufixo de s. Além disso, será usada a notação s/t para denotar o sufixo de s após t. Se t não é um prefixo de s, então s/t não é definido.

2.1.2 Operações com linguagens

As operações de concatenação e fecho de Kleene são formalmente definidas sobre linguagens a seguir [1].

Definição 2.2 *(Concatenação) Sejam L_a, $L_b \subseteq \Sigma^\star$, então a concatenação L_aL_b é definida da seguinte forma:*

$$L_aL_b = \{s \in \Sigma^\star : (s = s_as_b)\ e\ (s_a \in L_a)\ e\ (s_b \in L_b)\}.$$

Uma sequência s pertence a L_aL_b se ela pode ser construída como uma concatenação de uma sequência de L_a com uma sequência de L_b, ou seja, quando aplicada

a linguagens, a operação de concatenação agrupa cada sequência de uma linguagem com cada sequência de outra linguagem.

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

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

Os elementos de $L^\star$ são formados pela concatenação de um número finito de elementos de L. Note que a operação fecho de Kleene é idempotente, ou seja, $(L^\star)^\star = L^\star$.

Além das definições de concatenação e fecho de Kleene, outras operações podem ser definidas sobre linguagens. Essas operações são definidas a seguir [1].

Definição 2.4 *(Fecho de prefixo) Seja $L \subseteq \Sigma^\star$, então*

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

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

O fecho do prefixo de uma linguagem L é uma linguagem denotada por $\overline{L}$ e é constituído de todos os prefixos de todas as sequências de L. Note que, em geral, $L \subseteq \overline{L}$. L é dita ser prefixo-fechada quando $L = \overline{L}$, ou seja, L é prefixo-fechada quando todos os prefixos de todas as linguagens de L também são elementos de L.

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

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

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

O exemplo a seguir ilustra as operações com linguagens apresentadas.

Exemplo 2.1 *Seja $\Sigma = \{a, b, c\}$ e considere as linguagens $L_1 = \{\varepsilon, a, ab, aab\}$ e $L_2 = \{c\}$. Note que L_1 e L_2 não são prefixo-fechadas, já que $aa \notin L_1$ e $\varepsilon \notin L_2$. Logo, tem-se que, $L_1L_2 = \{c, ac, abc, aabc\}$, $\overline{L}_1 = \{\varepsilon, a, ab, aa, aab\}$, $\overline{L}_2 = \{\varepsilon, c\}$, $L_1^\star = \{\varepsilon, c, cc, ccc, \ldots\}$ e $L_2^\star = \{\varepsilon, a, ab, aab, aa, aaab, aba, aaba, \ldots\}$.*

Observação 2.1 *É importante observar que para uma linguagem* $L = \emptyset$, $\overline{L} = \emptyset$, *mas se* $L \neq \emptyset$, *então necessariamente* $\varepsilon \in \overline{L}$. *Além disso,* $\emptyset^\star = \{\varepsilon\}$ *e* $\{\varepsilon\}^\star = \{\varepsilon\}$ *e a operação de concatenação envolvendo uma linguagem e o conjunto vazio tem como resultado o próprio conjunto vazio, ou seja,* $\emptyset L = L\emptyset = \emptyset$.

Além das operações definidas acima é possível realizar as operações de projeção e projeção inversa, que são definidas a seguir [1].

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

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

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

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

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

De acordo com a definição 2.6, a operação de projeção apaga todos os eventos $e \in \Sigma_l \setminus \Sigma_s$ das sequências de $s \in \Sigma_l^\star$. Considere, por exemplo, os conjuntos $\Sigma_l = \{a, b, c\}$ e $\Sigma_s = \{b\}$ e as sequências de eventos $s_1 = ac$ e $s_2 = acb$, em que, s_1, $s_2 \in \Sigma_l^\star$. Então, a projeção $P_s : \Sigma_l^\star \to \Sigma_s^\star$ de s_1 e s_2 são iguais a $P_s(s_1) = \varepsilon$ e $P_s(s_2) = b$.

Além da operação de projeção, a operação de projeção inversa é definida a seguir.

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

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

Para uma dada sequência s construída a partir do conjunto de eventos Σ_s, a operação de projeção inversa P_s^{-1} realizada sobre s gera como resultado um conjunto formado por todas as sequências a partir do conjunto Σ_l, cuja projeção P_s resulta em s.

A operação de projeção P_s e projeção inversa P_s^{-1} podem ser estendidas para linguagens. Para isso, basta aplicá-las a todas as sequências que formam a linguagem. Formalmente, as operações de projeção e projeção inversa aplicadas a linguagens são definidas da seguinte forma [1].

Definição 2.8 *(Projeção de linguagens) Seja $L \subseteq \Sigma_l^\star$, então, a projeção de L, $P_s(L)$, é definida como:*

$$P_s(L) = \{t \in \Sigma_s^\star : (\exists s \in L)[P_s(s) = t]\}.$$

Definição 2.9 *(Projeção inversa de linguagens) Seja $L_s \subseteq \Sigma_s^\star$, então a projeção inversa de L_s, $P_s^{-1}(L_s)$, é definida como:*

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

Uma das aplicações da operação de projeção é a de representar a linguagem observada de um sistema por um observador que só tem acesso a eventos registrados por sensores ou eventos de comando de um controlador. Os eventos que não são observados são apagados das sequências da linguagem gerada pelo sistema através da operação de projeção, obtendo-se assim a linguagem observada do sistema.

O exemplo a seguir ilustra a operação de projeção e projeção inversa aplicada a linguagens.

Exemplo 2.2 *Sejam $\Sigma_l = \{a, b, c\}$, $\Sigma_s = \{a, b\}$ e a linguagem $L = \{a, b, c, ac, abc\}$. Considere a projeção $P_s : \Sigma_l^\star \to \Sigma_s^\star$. Ao tirar-se a projeção P_s da linguagem L, deve-se apagar de todas as sequências de L os eventos que pertencem ao conjunto $\Sigma_l \setminus \Sigma_s$. Neste caso, $\Sigma_l \setminus \Sigma_s = \{c\}$, logo, temos $P_s(L) = \{\varepsilon, a, b, ab\}$. Além disso, temos que $P_s^{-1}(\{ab\}) = \{c\}^\star a\{c\}^\star b\{c\}^\star$.*

Note que, no exemplo 2.2, $P_s(\{abc\}) = \{ab\} \neq P_s^{-1}(\{ab\})$. Dessa forma, em geral, $P_s(L) \neq P_s^{-1}(L)$ para uma dada linguagem $L \subseteq \Sigma_l^\star$.

2.2 Autômatos

Um dos formalismos capazes de representar linguagens geradas por SEDs são os autômatos. Um autômato é um dispositivo capaz de representar uma linguagem com regras bem definidas. A definição formal de um autômato é apresentada a seguir [1].

Definição 2.10 *(Autômato determinístico) Um autômato determinístico, denotado por G, é uma sêxtupla:*

$$G = (Q, \Sigma, f, \Gamma, q_0, Q_m)$$

em que Q é o conjunto de estados, Σ é o conjunto de eventos associados a G, $f : Q \times \Sigma \to Q$ é a função de transição, que pode ser parcial no seu domínio, $\Gamma : Q \to 2^{\Sigma}$ é a função de eventos ativos, q_0 é o estado inicial e $Q_m \subseteq Q$ é o conjunto de estados marcados.

Um autômato pode ser representado graficamente através de um grafo orientado chamado de diagrama de transição de estados. Os vértices do grafo, representados por círculos, são os estados e as arestas, representadas por arcos, são as transições entre os estados. As transições são rotuladas pelos eventos em Σ responsáveis pela transição de estados.

O estado inicial de um autômato é indicado por uma seta sem estado de origem e os estados marcados são representados no diagrama de transição de estados por círculos duplos concêntricos. As arestas representam graficamente a função de transição do autômato, denotada por $f : Q \times \Sigma \to Q$.

Um exemplo de um autômato e seu diagrama de transição de estados é apresentado a seguir.

Exemplo 2.3 *Seja G um autômato cujo diagrama de estados pode ser visto na figura 2.1. O conjunto de estados de G é dado por $Q = \{0, 1, 2\}$ e o conjunto de eventos é dado por $\Sigma = \{a, b, c\}$. A função de transição de estados de G é definida da seguinte forma: $f(0, c) = 0$; $f(0, a) = f(0, b) = 1$; $f(1, c) = 1$; $f(1, b) = 2$;*

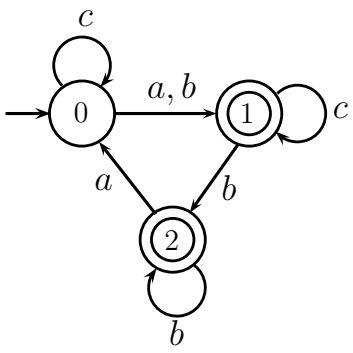


Figura 2.1: Diagrama de transição de estados do autômato G do exemplo 2.3.

$f(2,b) = 2$; $f(2,a) = 0$. *Note que* $f(1,a)$ *e* $f(2,c)$ *não são definidas. Assim, a função de eventos ativos de cada estado possui os seguintes resultados:* $\Gamma(0) = \{a,b,c\}$; $\Gamma(1) = \{b,c\}$; $\Gamma(2) = \{a,b\}$. *Por fim, o estado inicial de* G *é* $q_0 = 0$ *e o conjunto de estados marcados é* $Q_m = \{1,2\}$.

Por conveniência, a função de eventos ativos Γ será, por vezes, omitida dos autômatos definidos nesta dissertação.

As linguagens gerada e marcada por um autômato são descritas de acordo com as definições 2.11 e 2.12.

Definição 2.11 *(Linguagem gerada) A linguagem gerada por um autômato* $G = (Q, \Sigma, f, q_0, Q_m)$ *é dada por:*

$$\mathcal{L}(G) = \{s \in \Sigma^\star : f(q_0, s) \text{ é definida}\}.$$

Definição 2.12 *(Linguagem marcada) A linguagem marcada por um autômato* $G = (Q, \Sigma, f, q_0, Q_m)$ *é dada por:*

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

É importante ressaltar que nas definições 2.11 e 2.12 é suposto que a função de transição f é estendida, ou seja, $f : Q \times \Sigma^\star \to Q$. Além disso, para qualquer G que possua um conjunto de estados Q não vazio, $\varepsilon \in \mathcal{L}(G)$.

A linguagem gerada por G, $\mathcal{L}(G)$, é composta por todas as sequências que podem ser executadas no diagrama de transição de estados, partindo do estado inicial. A sequência de eventos que corresponde a um caminho é composta pela concatenação dos eventos que servem de rótulo das transições que compõem esse caminho. Assim, é importante observar que $\mathcal{L}(G)$ é prefixo-fechada por definição, uma vez que um caminho só é possível se todos os seus correspondentes prefixos são também possíveis. Além disso, é possível existirem eventos definidos em Σ que não fazem parte do diagrama de transição de estados de G e, portanto, não fazem parte de $\mathcal{L}(G)$.

A linguagem marcada por G, $\mathcal{L}_m(G)$, é um subconjunto de $\mathcal{L}(G)$, que corresponde a todas as sequências s tais que $f(q_0, s) \in Q_m$, ou seja, todas as sequências que levam a um estado marcado no diagrama de transição de estados de G. É importante observar que a linguagem marcada por G, $\mathcal{L}_m(G)$, não é necessariamente prefixo-fechada, já que nem todos os estados de Q precisam ser marcados.

2.2.1 Operações com autômatos

Para que seja possível realizar análises em um sistema a eventos discreto modelado por um autômato finito é preciso definir um conjunto de operações capazes de modificar o seu diagrama de transição de estados de acordo com alguma operação correspondente da linguagem gerada. Além disso, é necessário definir algumas operações que permitam combinar dois ou mais autômatos, para que modelos de sistemas complexos possam ser construídos a partir de modelos de componentes do sistema.

A parte acessível de um autômato G é uma operação unária que visa eliminar todos os estados de G que não são alcançáveis a partir do estado inicial q_0 e suas transições relacionadas. A definição formal de parte acessível de um autômato é apresentada a seguir [1].

Definição 2.13 *(Parte acessível) Seja $G = (Q, \Sigma, f, q_0, Q_m)$. A parte acessível de G, denotada por $Ac(G)$, é definida como:*

$$Ac(G) = (Q_{ac}, \Sigma, f_{ac}, q_0, Q_{ac,m}),$$

em que $Q_{ac} = \{q \in Q : (\exists s \in \Sigma^\star)[f(q_0, s) = q]\}$, $Q_{ac,m} = Q_m \cap Q_{ac}$ *e* $f_{ac} : Q_{ac} \times \Sigma^\star \to Q_{ac}$.

É importante notar que, ao realizar a operação de tomar a parte acessível de um autômato, a função de transição fica restrita a um domínio menor dos estados acessíveis Q_{ac}. Além disso, a parte acessível não altera as linguagens $\mathcal{L}(G)$ e $\mathcal{L}_m(G)$.

Um estado $q \in Q$ é dito ser coacessível se existir um caminho a partir do estado q que leve a um estado marcado, ou seja, um estado que pertença a Q_m. A operação de tomar a parte coacessível apaga todos os estados em G, e suas transições correspondentes, que não são coacessíveis. A definição formal de parte coacessível de um autômato é apresentada a seguir [1].

Definição 2.14 *(Parte Coacessível) Seja* $G = (Q, \Sigma, f, q_0, Q_m)$*, a parte coacessível de* G*, denotada por* $CoAc(G)$*, é definida como:*

$$CoAc(G) = (Q_{coac}, \Sigma, f_{coac}, q_{0,coac}, Q_m),$$

em que $Q_{coac} = \{q \in Q : (\exists s \in \Sigma^\star)[f(q, s) \in Q_m]\}$, $q_{0,coac} = q_0$*, se* $q_0 \in Q_{coac}$ *e* $q_{0,coac}$ *é indefinido, se* $q_0 \notin Q_{coac}$ *e* $f_{coac} : Q_{coac} \times \Sigma^\star \to Q_{coac}$.

É importante notar que, assim como ao realizar a operação de tomar a parte acessível de um autômato, ao realizar a operação de tomar a parte coacessível a função de transição fica restrita a um domínio menor dos estados coacessíveis Q_{coac}. Note que $\mathcal{L}(CoAc(G)) \subseteq \mathcal{L}(G)$, contudo $\mathcal{L}_m(CoAc(G)) = \mathcal{L}_m(G)$.

Um autômato que é tanto acessível quanto coacessível é chamado de Trim. A definição formal da operação trim é apresentada a seguir [1].

Definição 2.15 *(Operação trim) Seja* $G = (Q, \Sigma, f, q_0, Q_m)$*, a operação Trim pode ser definida da seguinte forma:*

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

O exemplo a seguir ilustra o resultado das operações de tomar a parte acessível, a parte coacessível e da operação trim em um autômato G.

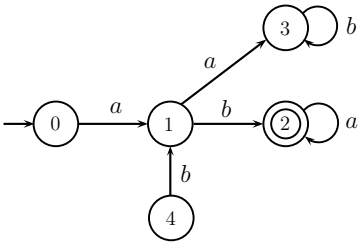


Figura 2.2: Autômato G do exemplo 2.4.

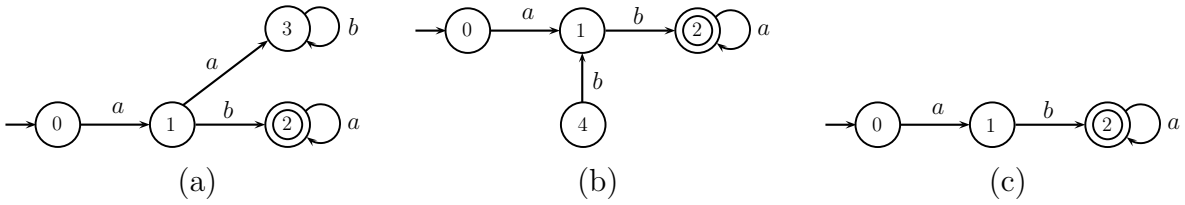


Figura 2.3: $Ac(G)$ do exemplo 2.4 (a), $CoAc(G)$ do exemplo 2.4 (b) e $Trim(G)$ do exemplo 2.4 (b).

Exemplo 2.4 *Considere o autômato G mostrado na figura 2.2. As figuras 2.3(a), 2.3(b) e 2.3(c) mostram os autômatos resultantes após as operações de parte acessível, coacessível e trim, respectivamente.*

As operações de projeção e projeção inversa de uma linguagem L também são operações unárias que podem ser aplicadas sobre o autômato G que gera a linguagem L. A projeção $P_s : \Sigma_l^\star \to \Sigma_s^\star$ pode ser aplicada ao autômato G substituindo-se todas as transições rotuladas por eventos que pertencem a $\Sigma_l \setminus \Sigma_s$ pela sequência vazia ε, gerando um autômato não determinístico G_{nd}, cuja linguagem gerada é $P_s(L)$. Um autômato determinístico pode ser obtido a partir de G_{nd}, que gera e marca a mesma linguagem, bastando, para isso, calcular o autômato observador de G_{nd} que será apresentado mais adiante nesta dissertação.

A projeção inversa $P_s^{-1} : \Sigma_s^\star \to 2^{\Sigma_l^\star}$ aplicada sobre uma linguagem $L \subseteq \Sigma_s^\star$ pode ser realizada em um autômato G que gera a linguagem L adicionando-se um autolaço rotulado por todos os eventos de $\Sigma_l \setminus \Sigma_s$ a todos os estados de G. A linguagem gerada do autômato resultante é igual à projeção inversa de L, $P_s^{-1}(L)$.

Além das operações unárias, existem operações que envolvem dois ou mais autômatos. Neste trabalho são apresentadas duas operações desse tipo: a com-

posição produto e a composição paralela.

A composição produto é chamada de composição completamente síncrona e é denotada por $\times$. A definição formal da composição produto é apresentada a seguir [1].

Definição 2.16 *(Composição produto) Sejam os autômatos* $G_1 = (Q_1, \Sigma_1, f_1, \Gamma_1, q_{01}, Q_{m1})$ *e* $G_2 = (Q_2, \Sigma_2, f_2, \Gamma_2, q_{02}, Q_{m2})$*, então, a composição produto entre* G_1 *e* G_2*,* $G_1 \times G_2$*, é dada por:*

$$G_1 \times G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f_{1\times2}, \Gamma_{1\times2}, (q_{01}, q_{02}), Q_{m1} \times Q_{m2}),$$

em que:

$$\begin{aligned} f_{1\times2}((q_1, q_2), e) &= \begin{cases} (f_1(q_1, e), f_2(q_2, e)), \textit{ se } e \in \Gamma_1(q_1) \cap \Gamma_2(q_2) \\ \textit{indefinido, caso contrário,} \end{cases} \\ \Gamma_{1\times2}(q_1, q_2) &= \Gamma_1(q_1) \cap \Gamma_2(q_2). \end{aligned}$$

De acordo com a definição 2.16, as transições dos dois autômatos precisam sempre ser sincronizadas com um evento em comum, ou seja, um evento que pertença a $\Sigma_1 \cap \Sigma_2$. Dessa forma, um evento ocorre em $G_1 \times G_2$ se e somente se o evento ocorrer em G_1 e G_2 ao mesmo tempo.

Os estados de $G_1 \times G_2$ são denotados em pares, em que o primeiro componente é um estado de G_1 e o segundo componente é um estado de G_2. Além disso, como a composição produto sincroniza a evolução dos autômatos, a linguagem gerada e a linguagem marcada por $G_1 \times G_2$ são $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$ e $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$, respectivamente.

A composição paralela, também chamada de composição síncrona, é representada por $\|$. Diferente da composição produto, que permite apenas transições rotuladas por eventos comuns, a composição paralela permite transições rotuladas por eventos particulares e sincroniza transições rotuladas por eventos comuns. A maneira mais comum de se obter o modelo de um sistema complexo, a partir dos modelos de seus

componentes, é através da composição paralela entre eles. A definição formal de composição paralela é apresentada a seguir [1].

Definição 2.17 *(Composição paralela) Sejam os autômatos* $G_1 = (Q_1, \Sigma_1, f_1, \Gamma_1, q_{01}, Q_{m1})$ *e* $G_2 = (Q_2, \Sigma_2, f_2, \Gamma_2, q_{02}, Q_{m2})$*. A composição paralela entre* G_1 *e* G_2*, denotada por* $G_1 \| G_2$*, é dada por:*

$$G_1 \| G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f_{1\|2}, \Gamma_{1\|2}, (q_{01}, q_{02}), Q_{m1} \times Q_{m2}),$$

em que

$$\begin{aligned} f((q_1,q_2),e) &= \begin{cases} (f_1(q_1,e), f_2(q_2,e)), \textit{se } e \in \Gamma_1(q_1) \cap \Gamma_2(q_2) \\ (f_1(q_1,e), q_2), \textit{se } e \in \Gamma_1(q_1) \setminus \Sigma_2 \\ (q_1, f_2(q_2,e)), \textit{se } e \in \Gamma_2(q_2) \setminus \Sigma_1 \\ \textit{indefinido, caso contrário,} \end{cases} \\ \Gamma_{1\|2}(q_1,q_2) &= [\Gamma_1(q_1) \cap \Gamma_2(q_2)] \cup [\Gamma_1(q_1) \setminus \Sigma_2] \cup [\Gamma_2(q_2) \setminus \Sigma_1]. \end{aligned}$$

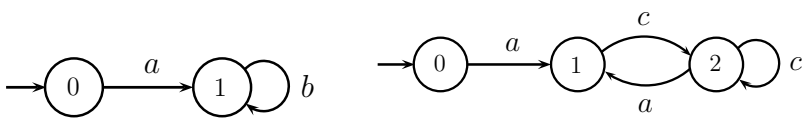
Assim, na composição paralela, um evento comum, ou seja, um evento que pertença a $\Sigma_1 \cap \Sigma_2$ pode ocorrer apenas se os dois autômatos o executam simultaneamente. Os eventos particulares, ou seja, eventos que pertencem a $(\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1)$ podem ocorrer sempre que forem possíveis. Assim, a composição paralela sincroniza apenas os eventos que são comuns aos dois autômatos.

Além disso, se $\Sigma_1 = \Sigma_2$, então o resultado da composição paralela é igual ao resultado da composição produto, uma vez que todas as transições serão sincronizadas.

Para caracterizar corretamente as linguagens gerada e marcada pelo autômato resultante da composição paralela é preciso definir as seguintes projeções:

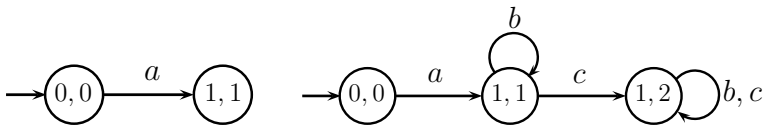
$$P_i : (\Sigma_1 \cup \Sigma_2)^\star \to \Sigma_i^\star \text{ para } i = 1, 2.$$

Com base nessas projeções, as linguagens gerada e marcada resultantes da composição paralela são $\mathcal{L}(G_1 \| G_2) = P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)]$ e $\mathcal{L}_m(G_1 \| G_2) =$



(a) Autômato G_1. (b) Autômato G_2.

Figura 2.4: Autômatos G_1 e G_2 do exemplo 2.5.



(a) $G_1 \times G_2$. (b) $G_1 \| G_2$

Figura 2.5: Resultados da composição produto e paralela do exemplo 2.5.

$P_1^{-1}[\mathcal{L}_m(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)]$, respectivamente. As operações de composição paralela e produto são ilustradas no exemplo a seguir.

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

2.2.2 Autômatos com observação parcial de eventos

Nas seções anteriores a esta, considerou-se que a ocorrência de todos os eventos de Σ é conhecida de alguma forma, ou seja, considerou-se que todos os eventos de Σ eram observáveis. Eventos não observáveis são eventos que ocorrem no sistema, mas que não são vistos, ou observados, por um observador externo ao comportamento do sistema. Essa não observação pode ocorrer devido à não existência de um sensor associado a esse evento ou o evento ocorreu em uma localização remota e sua ocorrência não foi comunicada. Além disso, eventos de falha que não causam nenhuma alteração imediata na leitura de sensores também são eventos não observáveis.

Dessa forma, o conjunto de eventos de G pode ser particionado em $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, em que Σ_o denota o conjunto de eventos observáveis e Σ_{uo} denota o conjunto de eventos não observáveis. Para um sistema G com eventos não observáveis, a linguagem

gerada observada de G é obtida aplicando-se a projeção P_o, em que $P_o : \Sigma^\star \to \Sigma_o^\star$, resultando em $P_o[\mathcal{L}(G)]$. Além disso, com o conjunto de eventos particionado entre eventos observáveis e eventos não observáveis, é necessário uma estrutura que identifique os possíveis estados do sistema após a observação de uma sequência de eventos. Essa estrutura é chamada de observador de G e é denotada por $Obs(G)$. Antes de apresentar o algoritmo de construção de $Obs(G)$ é necessário apresentar a seguinte definição de alcance não observável, denotado por $UR(q)$.

Definição 2.18 *(Alcance não observável) O alcance não observável de um estado $q \in Q$, denotado por $UR(q)$, é definido como:*

$$UR(q) = \{y \in Q : (\exists t \in \Sigma_{uo}^\star)(f(q,t) = y)\}. \tag{2.1}$$

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

$$UR(B) = \bigcup_{q \in B} UR(q). \tag{2.2}$$

O alcance não observável de um estado v gera um conjunto de estados que corresponde a todos os estados que são alcançáveis a partir de v através de transições rotuladas por eventos não observáveis. Em outras palavras, suponha que um sistema tenha gerado uma sequência $s \in \Sigma^\star$ de eventos e alcançado um estado $v \in Q$, então o alcance não observável de v, $UR(v)$, será igual ao conjunto de estados alcançáveis a partir do estado v por eventos, ou sequências de eventos, não observáveis.

Definição 2.19 *(Observador) O observador de um autômato G em relação a um conjunto de eventos observáveis Σ_o, denotado por $Obs(G)$, é dado por:*

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

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

Com base na definição 2.18, o observador de G, $Obs(G)$, pode ser construído de acordo com o algoritmo 2.1 [39].

Algoritmo 2.1 *Seja* $G = (Q, \Sigma, f, \Gamma, q_0, Q_m)$ *um autômato determinístico, sendo* $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. *Então,* $Obs(G) = (Q_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, q_{0,obs}, Q_{m,obs})$ *é construído da seguinte forma:*

- *Passo 1: Defina* $q_{0,obs} = UR(q_0)$. *Faça* $Q_{obs} = \{q_{0,obs}\}$ *e* $\widetilde{Q}_{obs} = Q_{obs}$.
- *Passo 2:* $\overline{Q}_{obs} = \widetilde{Q}_{obs}$ *e* $\widetilde{Q}_{obs} = \emptyset$.
- *Passo 3: Para cada* $B \in \overline{Q}_{obs}$,
  - *Passo 3.1:* $\Gamma_{obs}(B) = (\bigcup_{q \in B} \Gamma(q)) \cap \Sigma_o$.
  - *Passo 3.2: Para cada* $e \in \Gamma_{obs}(B)$,

$$f_{obs}(B, e) = UR(\{q \in Q : (\exists y \in B)[q = f(y, e)]\}).$$

  - *Passo 3.3:* $\widetilde{Q}_{obs} \leftarrow \widetilde{Q}_{obs} \cup f_{obs}(B, e)$.
- *Passo 4:* $Q_{obs} \leftarrow Q_{obs} \cup \widetilde{Q}_{obs}$.
- *Passo 5: Repita os passos 2 a 4 até que toda a parte acessível de* $Obs(G)$ *tenha sido construída.*
- *Passo 6:* $Q_{m,obs} = \{B \in Q_{obs} : B \cap Q_m \neq \emptyset\}$.

É importante ressaltar que as linguagens gerada e marcada pelo autômato $Obs(G)$ são: $\mathcal{L}(Obs(G)) = P_o[\mathcal{L}(G)]$ e $\mathcal{L}_m(Obs(G)) = P_o[\mathcal{L}_m(G)]$, sendo $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$.

Exemplo 2.6 *Seja* G *o autômato cujo diagrama de transição de estados pode ser visto na figura 2.6(a). O conjunto de estados de* G *é* $Q = \{0, 1, 2, 3\}$ *e o conjunto de eventos é* $\Sigma = \{a, b, \sigma_{uo}\}$, *em que* $\Sigma_o = \{a, b\}$ *e* $\Sigma_{uo} = \{\sigma_{uo}\}$. *O observador de* G, $Obs(G)$, *pode ser visualizado na figura 2.6(b). Considere que o sistema tenha executado a sequência de eventos* $t = a\sigma_{uo}b$, *a sequência observada será* $P_o(t) = ab$,

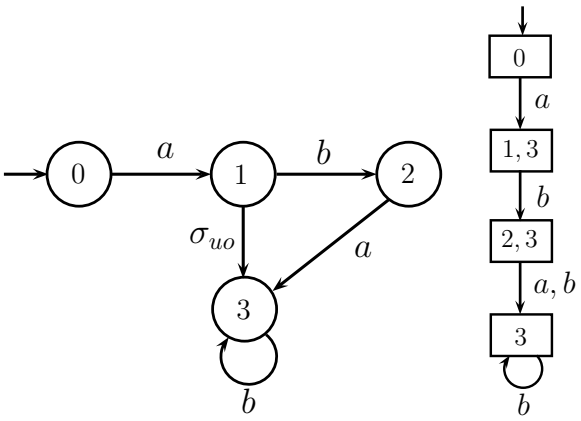


Figura 2.6: Diagrama de transição de estados do autômato G com eventos não observáveis (a), e autômato observador de G, $Obs(G)$, que fornece uma estimativa dos estados alcançados de G após a observação de uma sequência de eventos gerada pelo sistema (b).

para $P_o : \Sigma^\star \to \Sigma_o^\star$. *Ao acompanhar a sequência* $P_o(t)$ *em* $Obs(G)$*, é alcançado o estado* $\{2, 3\}$*, que corresponde à estimativa de estado de* G *após a observação dessa sequência.*

O exemplo 2.6 mostra o observador $Obs(G)$ de um autômato G com eventos não observáveis. Note que cada estado do observador $Obs(G)$ é um conjunto de estimativas do estado de G após a observação de uma sequência de eventos.

Na próxima seção, outro formalismo matemático capaz de representar SEDs é apresentado.

2.3 Redes de Petri

Uma rede de Petri é um formalismo usado como alternativa aos autômatos para descrever sistemas a eventos discretos. Assim como um autômato, uma rede de Petri é um dispositivo capaz de representar linguagens de acordo com regras bem definidas. As redes de Petri possuem condições explícitas para que as transições, rotuladas por eventos, ocorram. Aliado a isso, diferente dos autômatos, o estado na rede de Petri tem uma representação distribuída ao longo de sua estrutura, o que facilita a representação de sistemas complexos.

2.3.1 Fundamentos básicos das redes de Petri

No formalismo das redes de Petri, eventos são associados a transições e, para que determinada transição ocorra, é necessário que um conjunto de condições seja satisfeito. As informações relacionadas a essas condições estão nos lugares da rede. Cada transição possui um conjunto de lugares de entrada, que são associados com as condições necessárias para que a transição ocorra, e um conjunto de lugares de saída, que são associados com as condições que são afetadas pela ocorrência da transição.

Estrutura de uma rede de Petri

Lugares, transições e as relações entre eles formam o conjunto de informações básicas capaz de definir a estrutura de uma rede de Petri. A estrutura de uma rede de Petri possui dois tipos de vértices. Um tipo de vértice representa os lugares e o segundo tipo de vértice representa as transições. Como cada aresta não pode ligar vértices do mesmo tipo, uma rede de Petri é um grafo bipartido.

A definição formal da estrutura de uma rede de Petri é apresentada a seguir [1, 4].

Definição 2.20 *(Estrutura de uma rede de Petri) A estrutura de uma rede de Petri é um grafo bipartido ponderado*

$$(P, T, Pre, Post),$$

em que P é o conjunto finito de lugares (o primeiro tipo de vértice do grafo), T é o conjunto finito de transições (o segundo tipo de vértice do grafo), $Pre : (P \times T) \rightarrow \mathbb{N} = \{0, 1, 2, \ldots\}$ é a função de arcos ordinários que conectam lugares a transições e $Post : (T \times P) \rightarrow \mathbb{N}$ é a função de arcos ordinários que conectam transições a lugares.

O conjunto de lugares é representado por $P = \{p_1, p_2, ..., p_n\}$ e o conjunto de transições é representado por $T = \{t_1, t_2, ..., t_m\}$. Dessa forma, $|P| = n$ e $|T| = m$, em que, $|.|$ denota a cardinalidade dos conjuntos. O conjunto de lugares de entrada

(transições de entrada) de uma transição $t_j \in T$ (lugar $p_i \in P$) é denotado por $I(t_j)$ ($I(p_i)$) e é formado por lugares $p_i \in P$ (transições $t_j \in T$) tais que $Pre(p_i, t_j) > 0$ ($Post(t_j, p_i) > 0$).

Na representação gráfica de redes de Petri, os lugares são representados por círculos e as transições são representadas por barras. A quantidade e o sentido dos arcos que ligam lugares a transições e transições a lugares devem estar de acordo com as funções Pre e $Post$. Um exemplo de uma estrutura de uma rede de Petri pode ser visto a seguir.

Exemplo 2.7 *Seja a estrutura de uma rede de Petri definida por:* $P = \{p_1, p_2\}$, $T = \{t_1\}$, $Pre(p_1, t_1) = 1$ *e* $Post(t_1, p_2) = 2$. *Nesse caso,* $I(t_1) = \{p_1\}$ *e* $I(p_2) = \{t_1\}$. *Esse grafo é mostrado na figura 2.7.*

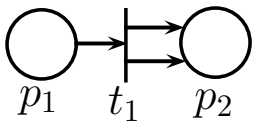


Figura 2.7: Estrutura de uma rede de Petri do exemplo 2.7.

Note que o fato de $Post(t_1, p_2)$ *ser igual a 2 é representado pela presença de dois arcos ligando a transição* t_1 *ao lugar* p_2. *Outra forma de representar seria adicionar a indicação do peso ao lado do arco ligando a transição* t_1 *ao lugar* p_2.

Marcação de uma rede de Petri

Para que cada transição dispare, é necessário que um conjunto de condições sejam satisfeitas. O mecanismo que indica se as condições para o disparo das transições são satisfeitas é obtido através da atribuição de fichas aos lugares. O número de fichas atribuídas a um lugar é representado por $x(p_i)$, em que $x : P \to \mathbb{N}$ é uma função de marcação. Logo, é possível definir uma marcação para a rede de Petri, representada pelo vetor coluna $\underline{x} = [x(p_1)\ x(p_2)\ \ldots\ x(p_n)]^T$, formado pelo número de fichas em cada lugar p_i, para $i = 1, ..., n$, como resultado da função de marcação. As fichas são representadas graficamente como pontos pretos dentro dos lugares.

Isso leva à seguinte definição de uma rede de Petri [1, 4].

Definição 2.21 *(Rede de Petri) Uma rede de Petri $\mathcal{N}$ é uma quíntupla $\mathcal{N} = (P, T, Pre, Post, x_0)$, em que, de acordo com a definição 2.20, $(P, T, Pre, Post)$ é a estrutura da rede de Petri e x_0 é a função de marcação inicial do conjunto de lugares.*

Dessa forma, em uma rede de Petri, o vetor de marcação de lugares $\underline{x}$ é o estado do sistema que a rede de Petri representa. A cada estado alcançado por um sistema há uma nova marcação de lugares na rede de Petri correspondente. O exemplo 2.8 ilustra uma rede de Petri.

Exemplo 2.8 *Considere a rede de Petri do exemplo 2.7 mostrada na figura 2.7. Suponha que, através da função de marcação inicial, o vetor de marcação de estados inicial seja $\underline{x}_0 = [1\ 0]^T$. A rede de Petri com a marcação correspondente pode ser vista na figura 2.8.*

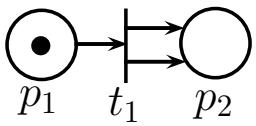


Figura 2.8: Rede de Petri com marcação inicial do exemplo 2.8.

Uma transição t_j em uma rede de Petri é dita estar habilitada quando o número de fichas em cada um dos lugares de entrada de t_j é maior ou igual aos pesos dos arcos que conectam os lugares à transição t_j. A definição de transição habilitada é apresentada a seguir.

Definição 2.22 *(Transição habilitada) Uma transição $t_j \in T$ é dita estar habilitada se e somente se*

$$x(p_i) \geq Pre(p_i, t_j), \text{ para todo } p_i \in I(t_j). \tag{2.3}$$

Dinâmica de uma rede de Petri

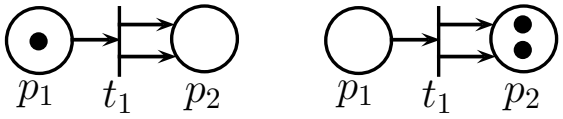
Quando uma transição t_j está habilitada, diz-se que ela pode disparar. As mudanças de estados em uma rede de Petri são dadas pela movimentação das fichas ao longo da rede em consequência do disparo de transições. Se uma transição t_j, que está habilitada para uma marcação $\underline{x}$, dispara, então a rede de Petri alcança uma nova marcação $\overline{\underline{x}}$, dada por:

$$\overline{x}(p_i) = x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i), i = 1, \ldots, n. \tag{2.4}$$

De acordo com a equação (2.4), se p_i é um lugar de entrada de t_j, ele perde uma quantidade de fichas igual ao peso do arco que conecta p_i a t_j. Se p_i for um lugar de saída de t_j, ele ganha uma quantidade de fichas igual ao peso do arco que conecta t_j a p_i. Note que é possível que p_i seja, ao mesmo tempo, um lugar de entrada e de saída de t_j, nesse caso, a partir da equação (2.4), são retiradas $Pre(p_i, t_j)$ fichas de p_i e então, imediatamente são colocadas $Post(t_j, p_i)$ fichas em p_i.

Se, em uma rede de Petri, um lugar sempre ficar com zero ou uma ficha, para qualquer marcação alcançada, esse lugar é chamado de seguro. O exemplo 2.9 ilustra o processo de disparo de uma transição, mostrando a distribuição de fichas antes e depois do disparo.

Exemplo 2.9 *Considere a rede de Petri da figura 2.9(a). Note que a transição* t_1 *está habilitada para a marcação inicial* $\underline{x}_0 = [1\ 0]^T$ *e, portanto, pode disparar. Suponha que* t_1 *dispare, logo, como o arco que conecta* p_1 *a* t_1 *tem peso 1, o lugar* p_1 *perde uma ficha e, como o arco que conecta* t_1 *a* p_2 *tem peso 2, duas fichas são colocadas no lugar* p_2*, resultando na rede de Petri com a marcação* $\underline{x} = [0\ 2]^T$ *que é mostrada na figura 2.9(b).*



(a) (b)

Figura 2.9: Rede de Petri do exemplo 2.9 antes do disparo de t_1 (a), e rede de Petri do exemplo 2.9 após o disparo de t_1 com a nova marcação alcançada.

Redes de Petri rotuladas

Para que o formalismo de redes de Petri possa ser usado para descrever SEDs, faz-se necessário realizar uma correspondência entre eventos e transições da rede de Petri. Dessa maneira, é possível usar redes de Petri para representar linguagens, desde que cada transição tenha ao menos um evento associado. Isso nos leva à seguinte definição de redes de Petri rotuladas [1, 4].

Definição 2.23 *(Redes de Petri rotuladas) Uma rede de Petri rotulada $\mathcal{N}$ é uma séptupla $\mathcal{N} = (P, T, Pre,\ Post, x_0, \Sigma, l)$, em que, $(P, T, Pre, Post, x_0)$ é, de acordo com a definição 2.21, uma rede de Petri. Σ é o conjunto de eventos que são utilizados para a rotulação das transições e $l : T \to 2^{\Sigma}$ é a função de rotulagem que associa um subconjunto de eventos de Σ a cada transição.*

No grafo de uma rede de Petri, o rótulo de uma transição é indicado próximo à transição. Em uma rede de Petri rotulada, para que uma transição t_j dispare, é necessário que t_j esteja habilitada e que o evento associado à transição ocorra.

2.3.2 Classes especiais de redes de Petri

Redes de Petri máquina de estados

A rede de Petri máquina de estados é uma classe especial de redes de Petri em que cada transição possui apenas um lugar de entrada e um lugar de saída. Se, além disso, essa rede de Petri possuir apenas uma ficha, então a rede de Petri máquina de estados se comporta exatamente como um autômato, em que cada lugar está

associado com um estado no autômato correspondente. O algoritmo de construção de uma rede de Petri máquina de estados a partir de um autômato é apresentado no próximo capítulo.

O exemplo 2.10 ilustra a equivalência entre um autômato e uma rede de Petri máquina de estados.

Exemplo 2.10 *Considere o autômato G cujo diagrama de estados está representado na figura 2.10(a). A figura 2.10(b) é a rede de Petri máquina de estados $\mathcal{N}$ equivalente ao autômato G. Para representar um autômato como uma rede de Petri máquina de estados basta substituir os estados do autômato por lugares da rede e substituir os arcos do autômato por transições, preservando a equivalência entre os lugares de entrada e saída.*

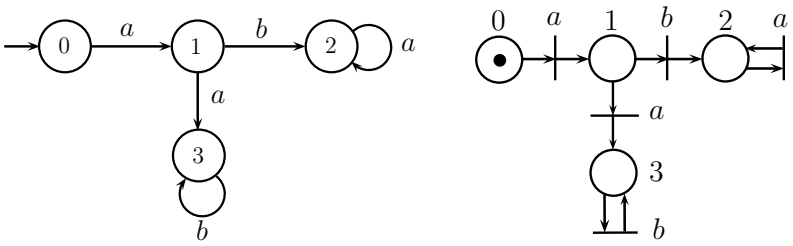


Figura 2.10: Autômato G do exemplo 2.10 (a), e rede de Petri máquina de estados equivalente ao autômato G (b).

Redes de Petri binárias

Outra classe de redes de Petri é a chamada rede de Petri binária [40]. Nesse tipo de rede de Petri, o número máximo de fichas de cada lugar é um. Dessa forma, se um lugar já possui uma ficha e, devido ao disparo de uma transição, o mesmo lugar recebe outra ficha, então o lugar continua com apenas uma ficha obrigatoriamente. Assim, cada lugar da rede de Petri é forçado a possuir um número de fichas igual a um ou zero, ou seja, todos os lugares são obrigatoriamente seguros.

Uma rede de Petri binária pode ser definida como uma rede de Petri com uma regra de evolução diferente para a marcação de lugares alcançados após o disparo

de uma transição t_j, dada por:

$$\overline{x}(p_i) = \begin{cases} 0, \text{se } x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i) = 0 \\ 1, \text{se } x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i) > 0, \end{cases} \tag{2.5}$$

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

2.4 Diagnosticabilidade de SEDs

Seja G o autômato que modela um sistema e seja $\Sigma_f \subseteq \Sigma_{uo}$ o conjunto de eventos de falha, que corresponde a todas as falhas que podem ocorrer na planta que está sendo considerada. Suponha que existam r tipos de falha no sistema, de forma que o conjunto de eventos de falha Σ_f possa ser particionado da seguinte forma:

$$\Sigma_f = \bigcup_{k=1}^{r} \Sigma_{f_k}, \tag{2.6}$$

em que Σ_{f_k} representa o conjunto de eventos de falha do mesmo tipo. Uma partição genérica de Σ_f será representada por Π_f.

Para definir a diagnosticabilidade de um sistema é preciso antes definir seu comportamento normal com relação à falha do tipo F_k, como apresentado a seguir.

Definição 2.24 *Seja $\mathcal{L}(G) = L$ a linguagem gerada pelo autômato G e seja L_{N_k} a linguagem prefixo-fechada formada por todas as sequências de L que não contém nenhum evento de falha do conjunto Σ_{f_k}. Assim, o comportamento normal do sistema G, em relação à falha do tipo F_k, é modelado pelo autômato G_{N_k} que gera a linguagem L_{N_k}.*

A definição de diagnosticabilidade é apresentada a seguir [5].

Definição 2.25 *(Diagnosticabilidade) Sejam L e $L_{N_k} \subset L$ as linguagens prefixo-fechadas geradas por G e G_{N_k}, respectivamente, e defina a operação de projeção $P_o : \Sigma^\star \to \Sigma_o^\star$. Seja também $I_r = \{1, 2, \ldots, r\}$. Então, L é dita ser diagnosticável*

com relação à projeção P_o e com relação à partição Π_f se

$$(\forall k \in \Pi_f)(\exists n_k \in \mathbb{N})(\forall s \in L \setminus L_{N_k})(\forall st \in L \setminus L_{N_k})$$
$$(|t| \geq n_k) \Rightarrow (\forall \omega \in P_o^{-1}(P_o(st)) \cap L, \omega \in L \setminus L_{N_k}),$$

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

De acordo com a definição 2.25, L é diagnosticável com relação a P_o e Π_f se e somente se para todas as sequências st de comprimento arbitrariamente longo após a ocorrência de um evento de falha do conjunto Σ_{f_k}, não existirem sequências $s_{N_k} \in L_{N_k}$, de tal forma que $P_o(s_{N_k}) = P_o(st)$ para todo $k \in I_r$. Portanto, se L é diagnosticável então sempre é possível identificar unicamente o tipo de falha que ocorreu após um número finito de observações de eventos.

Dessa forma, o primeiro passo para se implementar um diagnosticador online é verificar a diagnosticabilidade do sistema com relação a todos os tipos de falha, ou seja, verificar se é sempre possível identificar se uma falha ocorreu depois de um número finito de observações de eventos após a ocorrência da falha. Existem diversos trabalhos na literatura que propõem um método de verificação para determinar se a linguagem gerada por um sistema é diagnosticável. Por exemplo, em [41] é apresentado um algoritmo em tempo polinomial para identificar se uma linguagem L é diagnosticável.

Verificar se uma linguagem L é diagnosticável não faz parte do escopo deste trabalho. Este trabalho visa a construção de um diagnosticador online para um sistema cuja linguagem gerada é diagnosticável em relação a P_o e Π_f. Além disso, a menos que seja indicado, em relação ao diagnóstico de falhas para os autômatos apresentados neste trabalho, é considerado que o conjunto de estados marcados é igual ao conjunto de estados desses autômatos, ou seja, $Q_m = Q$.

No próximo capítulo são apresentados os passos necessários para a construção de uma rede de Petri diagnosticadora para sistemas a eventos discretos modelados por autômatos finitos.

Capítulo 3

Rede de Petri diagnosticadora

Neste capítulo, o método de diagnóstico online de falhas proposto neste trabalho é apresentado. O método é baseado na construção de uma rede de Petri diagnosticadora que é obtida a partir de um autômato, denotado por G_C, que modela o comportamento normal do sistema em relação a cada tipo de falha. Como G_C modela apenas a parte normal do sistema, o diagnosticador tem, em geral, complexidade menor do que diagnosticadores baseados no modelo completo do sistema.

Uma rede de Petri observadora de estados é construída com o objetivo de fornecer a estimativa de estados de G_C após a observação de uma sequência de eventos. A rede de Petri diagnosticadora é baseada na rede de Petri observadora de estados de forma a indicar a ocorrência da falha, quando a estimativa de estados de G_C não retornar nenhum estado que pertença ao comportamento normal do sistema.

Os passos necessários para a construção da rede de Petri diagnosticadora são apresentados nas seções seguintes.

3.1 Obtenção do autômato G_C

O diagnóstico de sistemas a eventos discretos modelados por autômatos finitos pode ser realizado com base apenas no comportamento normal do sistema, diferente do que é feito usando-se autômatos diagnosticadores tradicionais, que usam o comportamento normal e de falha do sistema para realizar o diagnóstico de falhas, gerando

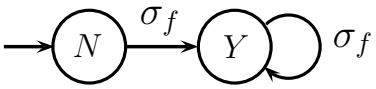


Figura 3.1: Autômato rotulador A_ℓ, em que σ_f é o evento de falha.

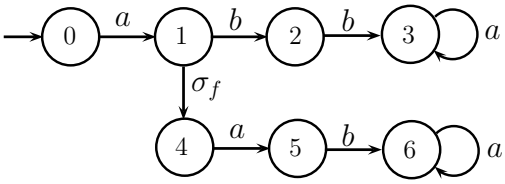


Figura 3.2: Autômato G do exemplo 3.1.

um diagnosticador mais complexo do que o necessário [1, 5, 7].

Para ilustrar o fato que não é preciso acompanhar o comportamento de falha do sistema para o diagnóstico, considere o diagnosticador proposto em [5], denotado por G_{diag}. G_{diag} é um autômato cujo conjunto de eventos é igual ao conjunto de eventos observáveis do autômato G que modela o sistema. Os estados de G_{diag} são formados pelos estados de G rotulados por Y ou N para indicar se um evento de falha ocorreu ou não, respectivamente. O autômato G_{diag} pode ser construído em dois passos: (i) calcule a composição paralela $G_\ell = G \| A_\ell$, em que $A_\ell = (\{N, Y\}, \Sigma_f, f_\ell, N)$ é um autômato rotulador de dois estados mostrado na figura 3.1 (nesse caso não é necessário considerar estados marcados); (ii) calcule $Obs(G_\ell)$ [39]. O exemplo 3.1 ilustra a construção de G_{diag} para uma planta G.

Exemplo 3.1 *Considere o autômato G mostrado na figura 3.2. O primeiro passo para a construção de G_{diag} é a composição paralela entre o autômato G e o autômato rotulador mostrado na figura 3.1. O autômato G_ℓ, resultante da composição $G \| A_\ell$, pode ser visto na figura 3.3. O último passo para a construção de G_{diag} é o cálculo do observador de $G \| A_\ell$ que é mostrado na figura 3.4.*

A falha é diagnosticada quando o diagnosticador alcança estados certos, ou seja, estados que possuem apenas rótulos Y. Suponha que a sequência $a\sigma_f a$ tenha sido executada pelo sistema. O estado alcançado em G_{diag} a partir de $P_o(a\sigma_f a) = aa$ é o estado $5Y$, indicando, assim, que a falha ocorreu. Por outro lado, considere o

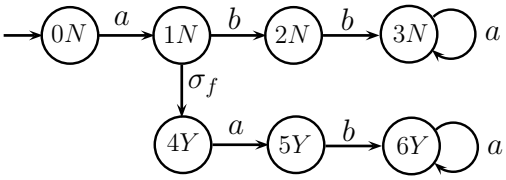


Figura 3.3: Autômato G_ℓ do exemplo 3.1.

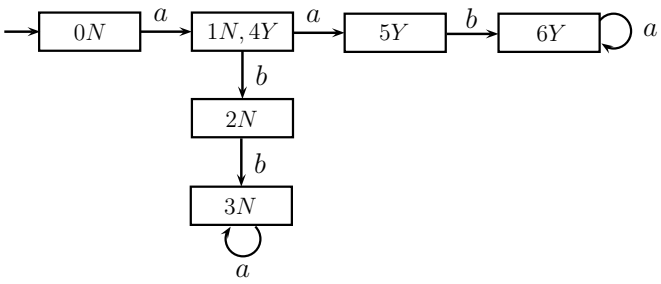


Figura 3.4: Autômato diagnosticador G_{diag} do exemplo 3.1.

autômato G_N, ilustrado na figura 3.5, que modela a parte normal do sistema. O autômato G_N pode ser obtido a partir de G_ℓ apagando-se os estados rotulados por Y e as transições relacionadas. Note que o autômato G_N não gera a sequência aa, porque como G_N modela apenas o comportamento normal, qualquer sequência que não esteja definida em $\mathcal{L}(G_N)$ é uma sequência de falha. Dessa forma, o diagnóstico de falhas pode ser feito com base apenas no comportamento normal do sistema.

Observe que, de acordo com o exemplo 3.1, os estados de G_{diag} são da forma (q, N) e (q, Y), em que $q \in Q$. O diagnóstico é realizado analisando-se os rótulos dos estados de G_{diag} alcançados após a observação de uma sequência de eventos s. Quando um estado de G_{diag} contendo apenas rótulos Y é alcançado, a falha é diagnosticada. Para simplificar a notação, é usual representar os estados de G_ℓ e de G_{diag} como qN e qY ao invés de (q, N) e (q, Y).

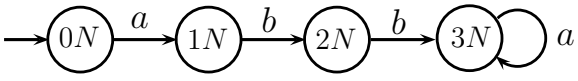


Figura 3.5: Autômato G_N que modela o comportamento normal do sistema G do exemplo 3.1.

Note que, G_{diag} representa tanto o comportamento normal quanto o comportamento de falha do autômato G que modela o sistema, mas a construção do comportamento de falha de G, ou seja, dos estados que possuem apenas rótulos Y, não é necessária para o diagnóstico. Isso ocorre porque basta que o sistema deixe seu comportamento normal para que se tenha certeza que a falha ocorreu, sem a necessidade da construção dos estados certos que fazem parte do comportamento de falha do sistema. Esse é o objetivo da construção do autômato G_C: acompanhar o sistema até que a observação de uma sequência de eventos que não pertença a $\mathcal{L}(G_N)$ ocorra e, assim, indicar a falha.

3.1.1 Cálculo de G_C

O autômato G_C é construído usando-se apenas o comportamento normal do sistema em relação a cada tipo de falha. Isso é feito através da composição produto entre G e A_{N_k}, em que A_{N_k} é um autômato com apenas um estado, que também é seu estado inicial, com um autolaço rotulado com todos os eventos normais do sistema, ou seja, rotulado com todos os eventos do conjunto $\Sigma \setminus \Sigma_{f_k}$. O objetivo dessa composição é descartar todo o comportamento de falha e rotular o comportamento normal de G em relação à falha do tipo F_k, obtendo-se o autômato G_{N_k} que representa apenas o comportamento normal de G, em relação à falha do tipo F_k, para $k = 1, \ldots, r$.

Em seguida, um estado rotulado por F_k é adicionado à G_{N_k}, com o objetivo de indicar a ocorrência da falha. Para isso, são adicionadas transições rotuladas com o evento de falha σ_{f_k} dos estados de G_{N_k} que correspondem aos estados de G que possuíam o evento σ_{f_k} pertencendo à função de eventos ativos, para o estado F_k. Além disso, é adicionado ao estado F_k um autolaço rotulado por todos os eventos de G para representar que, uma vez que a falha tenha ocorrido, o sistema não retorna ao comportamento normal.

Por fim, o autômato G_C é obtido através da composição paralela dos autômatos G_{N_k}, para $k \in I_r$ e modela o comportamento normal em relação a cada tipo de falha do sistema, de tal forma que os rótulos dos estados de G_C indicam se cada tipo de

falha ocorreu ou não.

O algoritmo de construção do autômato G_C para o caso de múltiplas falhas é apresentado a seguir.

Algoritmo 3.1

- *Passo 1: Calcule o autômato G_{N_k}, para cada $k \in I_r$, que modela o comportamento normal de G com relação ao conjunto de eventos de falha Σ_{f_k}, da seguinte forma:*
  - *Passo 1.1: Defina $\Sigma_{N_k} = \Sigma \setminus \Sigma_{f_k}$.*
  - *Passo 1.2: Construa o autômato A_{N_k} composto de um único estado N_k (também seu estado inicial) com um autolaço rotulado com todos os eventos de Σ_{N_k}.*
  - *Passo 1.3: Construa o autômato que modela o comportamento normal de G, $G_{N_k} = G \times A_{N_k} = (Q_{N_k}, \Sigma, f_{N_k}, \Gamma_{N_k}, q_{0,N_k})$.*
- *Passo 2: Construa o autômato estendido $G^a_{N_k}$, para cada $k \in I_r$, da seguinte forma:*
  - *Passo 2.1: Adicione um novo estado F_k para indicar que um evento de falha do conjunto Σ_{f_k} ocorreu.*
  - *Passo 2.2: Adicione um autolaço em F_k rotulado com todos os eventos $\sigma \in \Sigma$.*
  - *Passo 2.3: Defina $f(x_{N_k}, \sigma_{f_k}) = F_k$ para todos os estados $x_{N_k} = (q, N_k) \in Q_{N_k}$ tais que $\sigma_{f_k} \in \Gamma(q)$.*
- *Passo 3: Calcule o autômato $G_C = (Q_C, \Sigma, f_C, \Gamma_C, q_{0,C}) = G^a_{N_1} \| G^a_{N_2} \| \ldots \| G^a_{N_r}$.*

De acordo com o algoritmo 3.1, o autômato G_C é construído a partir da composição paralela dos autômatos $G^a_{N_k}$, para $k = 1, \ldots, r$. Isso é feito com o objetivo

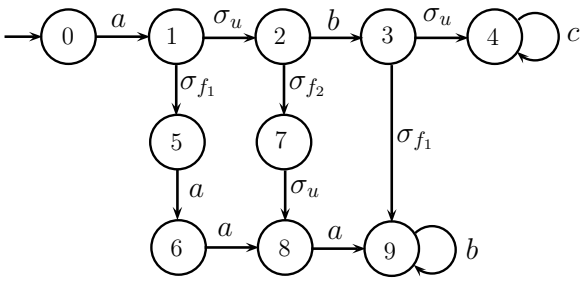


Figura 3.6: Autômato G do Exemplo 3.2.

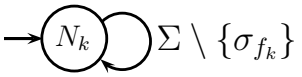


Figura 3.7: Autômato A_{N_k} do Exemplo 3.2.

de construir um autômato que continue no mesmo caminho de G enquanto o sistema está no comportamento normal e, uma vez que uma falha pertencendo a Σ_{f_k} ocorra, G_C evolui para um estado de falha e permanece nele, indicando que uma falha ocorreu. O exemplo 3.2 ilustra a construção do autômato G_C.

Exemplo 3.2 *Considere o sistema modelado pelo autômato G apresentado na figura 3.6, em que $\Sigma = \{a, b, c, \sigma_u, \sigma_{f_1}, \sigma_{f_2}\}$, $\Sigma_o = \{a, b, c\}$, $\Sigma_{uo} = \{\sigma_u, \sigma_{f_1}, \sigma_{f_2}\}$, e $\Sigma_f = \{\sigma_{f_1}, \sigma_{f_2}\}$. Suponha que o conjunto de eventos de falha possa ser particionado em $\Sigma_f = \Sigma_{f_1} \dot{\cup} \Sigma_{f_2}$ com $\Sigma_{f_1} = \{\sigma_{f_1}\}$ e $\Sigma_{f_2} = \{\sigma_{f_2}\}$. De acordo com o algoritmo 3.1, o primeiro passo é obter os autômatos A_{N_k}, para $k = 1, 2$, mostrado na figura 3.7, e os autômatos que modelam os comportamentos normais $G_{N_k} = G \times A_{N_k}$. O próximo passo é a construção dos autômatos aumentados $G^a_{N_1}$ e $G^a_{N_2}$, mostrados nas figuras 3.8 e 3.9, respectivamente, obtidos adicionando-se os estados de falha F_1 e F_2 aos autômatos G_{N_1} e G_{N_2}. O passo final do algoritmo 3.1 é o cálculo do autômato $G_C = G^a_{N_1} \| G^a_{N_2}$, ilustrado na figura 3.10.*

É importante ressaltar que para cada $G^a_{N_k}$, o comportamento do sistema com relação ao conjunto de eventos de falha Σ_{f_k} é representado pelo estado de falha F_k, adicionado ao autômato G_{N_k} para indicar que uma falha do conjunto Σ_{f_k} ocorreu, com um autolaço rotulado com todos os eventos do conjunto Σ. Como consequência,

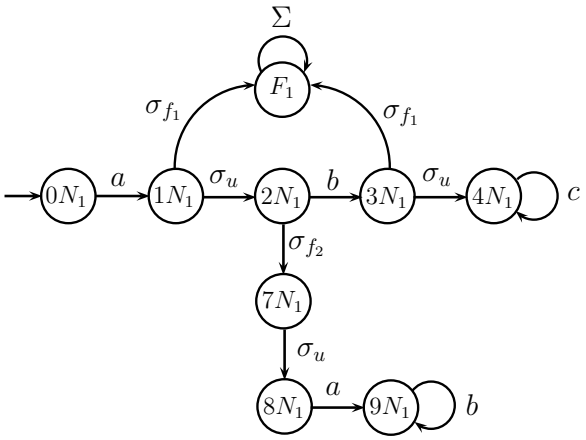


Figura 3.8: Autômato aumentado $G^a_{N_1}$ do Exemplo 3.2.

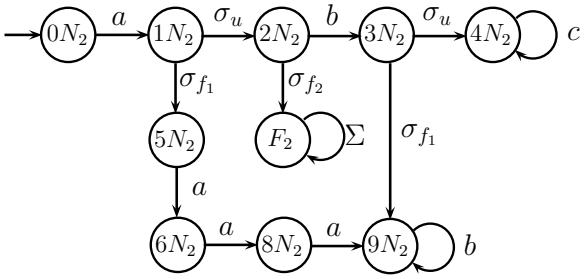


Figura 3.9: Autômato aumentado $G^a_{N_2}$ do Exemplo 3.2.

essa representação não preserva a linguagem gerada pelo sistema após a ocorrência do evento de falha. Entretanto, como o diagnosticador é um dispositivo passivo, essa representação não altera a observação dos eventos do sistema e, portanto, não interfere no diagnóstico de falhas.

3.1.2 Emprego de G_C para o diagnóstico online

Para mostrar como o autômato G_C pode ser usado no diagnóstico online de falhas é necessário primeiro definir uma função que fornece os possíveis estados atuais de G_C após a ocorrência de um evento observável, ou seja, uma função que forneça uma estimativa dos estados de G_C após a observação de uma determinada sequência de eventos. Essa estimativa é denotada neste trabalho por $Reach(\nu)$, em que $\nu = v\sigma_o = P_o(s)$ é a sequência observada pelo diagnosticador após a execução de uma sequência

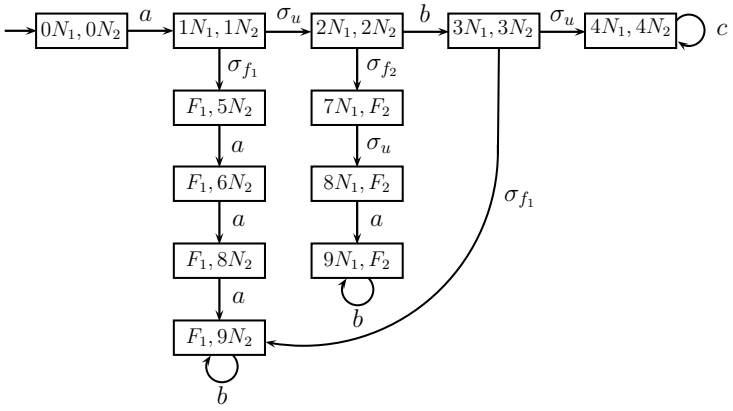


Figura 3.10: Autômato $G_C = G^a_{N_1} \| G^a_{N_2}$ do Exemplo 3.2.

$s \in L$ cujo último evento observável é σ_o, e pode ser calculada recursivamente como

$$Reach(\varepsilon) = UR(q_{0,C}), \tag{3.1}$$

$$Reach(v\sigma_o) = UR(\delta(Reach(v), \sigma_o)), \tag{3.2}$$

em que $\delta(Reach(v), \sigma_o) = \bigcup_{i=1}^{\kappa} \delta_C(q_{C_i}, \sigma_o)$, com $q_{C_i} \in Reach(v)$, $\kappa = |Reach(v)|$, e $\delta_C(q_{C_i}, \sigma_o) = f_C(q_{C_i}, \sigma_o)$ se $f_C(q_{C_i}, \sigma_o)$ é definida e $\delta_C(q_{C_i}, \sigma_o) = \emptyset$, caso contrário.

Após a observação de uma sequência de eventos ν, o conjunto dos possíveis estados atuais de G_C, $Reach(\nu)$, pode ser calculado e esses estados podem ser usados para identificar a ocorrência de um evento de falha. O teorema a seguir apresenta a base para o método de diagnóstico proposto neste trabalho [38].

Teorema 3.1 *Sejam L e L_{N_k} as linguagens geradas por G e por G_{N_k}, respectivamente, e suponha que L é diagnosticável com relação a P_o e Π_f. Seja $s \in L \setminus L_{N_k}$ tal que $\forall \omega \in L$ que satisfaz $P_o(\omega) = P_o(s)$, tem-se que $\omega \in L \setminus L_{N_k}$. Então, a k-ésima coordenada de todos os possíveis estados de G_C alcançados após a ocorrência de s, dados por $Reach(P_o(s))$, é igual a F_k.*

Prova: De acordo com a construção do autômato G_C, é possível notar que se $s \in L \setminus L_{N_k}$, então a k-ésima coordenada do estado alcançado de G_C após a ocorrência de s, $f_C(q_{0,C}, s)$, é igual a F_k. Uma vez que L é diagnosticável com relação a P_o e Π_f, então, se s é uma sequência arbitrariamente longa de eventos após a ocorrência

de um evento de falha do conjunto Σ_{f_k}, então não existe nenhuma sequência normal $\omega \in L_{N_k}$, tal que $P_o(\omega) = P_o(s)$. Isso implica que todos os estados dados pela estimativa $Reach(P_o(s))$ possuem F_k como sua k-ésima coordenada. ■

De acordo com o teorema 3.1, se L é diagnosticável com relação a P_o e Π_f, então sempre é possível identificar a ocorrência de uma falha do tipo F_k com um número limitado de observações de eventos verificando os possíveis estados atuais de G_C. Em outras palavras, se após a ocorrência de uma sequência s que contém um evento de falha $\sigma_{f_k} \in \Sigma_{f_k}$, todos os estados de $Reach(\nu)$, em que $\nu = P_o(s)$, não possuem uma coordenada (q, N_k), então não é possível que uma sequência de eventos normal com relação ao conjunto de eventos de falha Σ_{f_k}, com a mesma projeção que ν tenha sido executada, o que implica que uma falha do tipo F_k ocorreu.

Dessa forma, o diagnóstico de uma falha do tipo F_k pode ser feita verificando-se se um estado do comportamento normal descrito por G_{N_k} é uma coordenada de um possível estado atual de G_C, ou seja, basta verificar se um estado de G_{N_k} pertence à estimativa de estado de G_C após a observação de uma sequência de eventos.

O exemplo 3.3 mostra como o autômato G_C pode ser usado para o diagnóstico online.

Exemplo 3.3 *Considere novamente o autômato composto G_C da figura 3.10. Suponha que uma sequência de falha $s = a\sigma_{f_1}aa \in L \setminus L_{N_1}$ tenha sido executada pelo sistema. Então, a sequência observada é $\nu = P_o(s) = aaa$. De acordo com o teorema 3.1, se não existir uma sequência $\omega \in L_{N_1}$ tal que $P_o(\omega) = \nu$ então todos os estados no conjunto de estados alcançáveis $Reach(\nu)$ possuem a primeira coordenada igual a F_1. O conjunto de estados alcançáveis $Reach(\nu)$ pode ser obtido recursivamente de acordo com as equações (3.1) e (3.2), da seguinte forma:*

$$Reach(\varepsilon) = \{(0N_1, 0N_2)\},$$

$$Reach(a) = \{(1N_1, 1N_2), (2N_1, 2N_2), (F_1, 5N_2), (7N_1, F_2), (8N_1, F_2)\},$$

$$Reach(aa) = \{(F_1, 6N_2), (9N_1, F_2)\},$$

$$Reach(aaa) = \{(F_1, 8N_2)\}.$$

Uma vez que o único estado alcançado após a observação de $\nu = aaa$ possui a primeira coordenada igual a F_1, então é possível garantir que o evento de falha σ_{f_1} ocorreu.

3.1.3 Análise da complexidade computacional

De acordo com o algoritmo 3.1, o autômato G_C é obtido através da composição paralela dos autômatos $G^a_{N_k}$, para $k = 1, \ldots, r$, em que r é o número de tipos de falha do sistema. Então, um estado $q_C \in Q_C$ é composto pelos estados de $G^a_{N_k}$, para $k = 1, \ldots, r$, cujos estados podem ser iguais a (q, N_k) ou F_k, em que $q \in Q$ é exatamente o mesmo para todos os componentes de q_C diferentes de F_k. O número máximo de estados de G_C associados com o mesmo estado $q \in Q$ e que tem pelo menos um componente (q, N_k) é igual a $(2^r - 1)$. Isso implica que o número de estados de G_C é, no pior caso, igual a $[(2^r - 1) \times |Q|] + 1$ e, como G_C é um autômato determinístico, o número máximo de transições de G_C é $\{[(2^r - 1) \times |Q|] + 1\} \times |\Sigma|$. Portanto, a complexidade computacional da construção do autômato G_C é $O(2^r \times |Q| \times |\Sigma|)$, o que mostra que a complexidade é linear no número de estados e eventos do modelo em autômato do sistema e exponencial no número de tipos de falha.

A complexidade computacional pode ser linear em relação ao número de tipos de falha se cada comportamento normal com relação a um único tipo de falha for considerado separadamente. Nesse caso, ao invés de um único autômato G_C, seriam considerados r autômatos $G^a_{N_k}$, em que cada um leva em consideração apenas o tipo de falha F_k, e, portanto, a complexidade computacional é $O(r \times |Q| \times |\Sigma|)$.

Embora a análise de pior caso sugira que é mais vantajoso considerar os autômatos $G^a_{N_k}$, $k = 1, \ldots, r$, ao invés de um único autômato G_C, é importante ressaltar que o número de estados de G_C pode ser menor que a soma do número de estados de $G^a_{N_k}$ para $k = 1, \ldots, r$, levando a um código de programação menor para a implementação do diagnosticador online.

Exemplo 3.4 *A partir da construção do autômato G_C do exemplo 3.2 é possível notar que G_C tem 12 estados e $G^a_{N_1}$ e $G^a_{N_2}$ têm 9 e 10 estados, respectivamente. Portanto, G_C tem menos estados do que a soma dos estados de $G^a_{N_1}$ e $G^a_{N_2}$. Assim, o diagnóstico online pode ser feito, nesse caso, com menor custo computacional usando o autômato G_C ao invés de $G^a_{N_k}$, $k = 1, 2$.*

Como mostrado nessa seção, uma falha do tipo F_k pode ser diagnosticada analisando-se a estimativa de estado de G_C após a observação de uma sequência $\nu \in \Sigma_o^\star$. Para tanto, observadores podem ser usados para realizar a estimação de estados de G_C. Entretanto, como mostrado no capítulo 1, o cálculo de observadores tem, no pior caso, complexidade exponencial. Na Seção 3.2, uma rede de Petri é usada para fornecer um diagnosticador online capaz de encontrar os estados alcançáveis de G_C após a observação de uma sequência de eventos ν, ou seja, capaz de representar o resultado da função $Reach(\nu)$, para a identificação da ocorrência de um evento de falha.

3.2 Construção da rede de Petri diagnosticadora

Uma vez calculado o autômato G_C, é preciso obter uma estrutura capaz de solucionar o problema de encontrar os possíveis estados de G_C após a observação de uma sequência de eventos $\nu \in \Sigma_o^\star$, ou seja, é necessário um observador online que armazene os estados estimados de G_C após a ocorrência de um evento observável. Esse observador online pode ser construído usando o formalismo de redes de Petri, explorando a natureza distribuída do estado de uma rede de Petri. Esse observador é denominado neste trabalho de rede de Petri observadora de estados.

O primeiro passo para a construção de uma rede de Petri observadora de estados é a obtenção de uma rede de Petri máquina de estados, chamada de $\mathcal{N}_C$, a partir do autômato G_C. Assim como apresentado na subseção 2.3.2, a construção da rede de Petri máquina de estados, $\mathcal{N}_C$, é realizada associando-se para cada estado q_{C_i} de G_C um lugar p_{C_i} em $\mathcal{N}_C$ e para cada arco direcionado em G_C, rotulado por $\sigma \in \Gamma_C(q_{C_i})$, uma transição t_{C_j}, rotulada por σ, em $\mathcal{N}_C$ [1]. O estado inicial de $\mathcal{N}_C$ é definido atribuindo-se uma ficha ao lugar de $\mathcal{N}_C$ associado ao estado inicial de G_C e não atribuindo-se fichas aos demais lugares. Esse procedimento é formalizado de acordo com o algoritmo 3.2.

Algoritmo 3.2 *(Rede de Petri máquina de estados) Seja* $G_C = (Q_C, \Sigma, f_C, \Gamma_C, q_{0,C})$ *o autômato composto do sistema. Então, uma rede de Petri máquina de estados* $\mathcal{N}_C = (P_C, T_C, Pre_C, Post_C, x_{0,C}, \Sigma, l_C)$ *pode ser obtida da seguinte forma.*

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

Uma vez obtido $\mathcal{N}_C$, o próximo passo para o cálculo da rede de Petri observadora de estados de G_C é a criação de novos arcos, conectando cada transição rotulada por um evento observável a lugares específicos que correspondem ao alcance não observável de lugares após o disparo de uma transição observável. Para isso, seja $T_{C_o} \subseteq T_C$ o conjunto de todas as transições de $\mathcal{N}_C$ rotuladas com eventos observáveis e defina a função $Reach_T : T_{C_o} \to 2^{P_C}$. O conjunto de lugares $Reach_T(t_{C_j})$, em que $t_{C_j} \in T_{C_o}$, pode ser calculado de acordo com o algoritmo 3.3.

Algoritmo 3.3 *(Cálculo de $Reach_T(t_{C_j}), t_{C_j} \in T_{C_o}$) Sejam $O(t)$ e $O(p)$ o conjunto de todos os lugares de saída de t e o conjunto de todas as transições de saída de p, respectivamente. Seja também $O(P) = \bigcup_{p \in P} O(p)$ e $O(T) = \bigcup_{t \in T} O(t)$.*

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

Para implementar o alcance não observável após o disparo de cada transição observável, um arco de peso um, conectando cada transição $t_{C_j} \in T_{C_o}$, a cada lugar $p_{C_i} \in Reach_T(t_{C_j})$, precisa ser adicionado a $\mathcal{N}_C$ gerando uma nova rede de Petri, $\mathcal{N}'_C$. Após isso, todas as transições de $\mathcal{N}'_C$ que sejam rotuladas com eventos não observáveis, e seus arcos relacionados, precisam ser removidas gerando uma nova rede de Petri, $\mathcal{N}_{C_o}$, cujas transições são rotuladas apenas com eventos observáveis pertencentes a Σ_o. O cálculo da rede de Petri $\mathcal{N}_{C_o}$ pode ser formalizado de acordo com o algoritmo a seguir.

Algoritmo 3.4 *(Cálculo de $\mathcal{N}_{C_o}$) Seja $\mathcal{N}_C = (P_C, T_C, Pre_C, Post_C, x_{0,C}, \Sigma, l_C)$ a rede de Petri máquina de estados obtida a partir de G_C usando o algoritmo 3.2. A rede de Petri $\mathcal{N}_{C_o} = (P_C, T_{C_o}, Pre_{C_o}, Post_{C_o}, x_{0,C}, \Sigma_o, l_{C_o})$ pode ser calculada da seguinte forma.*

- *Passo 1: Seja $T_{C_o} \subseteq T_C$ o conjunto de todas as transições de $\mathcal{N}_C$ rotuladas com eventos observáveis. Defina uma nova função $Post'_C : T_C \times P_C \to \mathbb{N}$ tal que $Post'_C(t_{C_j}, p_{C_i}) = 1$, se $t_{C_j} \in T_{C_o}$ e $p_{C_i} \in Reach_T(t_{C_j})$, e $Post'_C(t_{C_j}, p_{C_i}) = Post_C(t_{C_j}, p_{C_i})$, caso contrário.*
- *Passo 2: Defina as funções $Pre_{C_o} : P_C \times T_{C_o} \to \mathbb{N}$ e $Post_{C_o} : T_{C_o} \times P_C \to \mathbb{N}$ em que $Pre_{C_o}(p_{C_i}, t_{C_j}) = Pre_C(p_{C_i}, t_{C_j})$ e $Post_{C_o}(t_{C_j}, p_{C_\ell}) = Post'_C(t_{C_j}, p_{C_\ell})$ para todo $t_{C_j} \in T_{C_o}$ e $p_{C_i}, p_{C_\ell} \in P_C$.*

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

A função da rede de Petri $\mathcal{N}_{C_o}$ é calcular a estimativa de estado de G_C a cada evento observado na evolução do sistema, de tal forma que cada lugar da rede de Petri representa um possível estado atual de G_C a partir da estimativa. Dessa forma, apenas os lugares que são associados aos possíveis estados atuais de G_C devem ter fichas e, após a ocorrência de um evento observável, o número de fichas nos lugares que não são mais possíveis, ou seja, lugares que representam estados que não fazem mais parte desse alcance, deve ser igual a zero. Com isso, o número de fichas em cada lugar da rede de Petri observadora de estados deve ser sempre igual a um ou zero, mesmo que o disparo de uma transição $t_{C_j} \in T_{C_o}$ resulte, de acordo com a equação (2.4), em uma marcação com duas ou mais fichas. Assim, é preciso que os lugares sejam forçados a ter marcações binárias e a equação (2.4) não é mais válida. Esse requisito pode ser satisfeito usando redes de Petri binárias [40]. Como mostrado na subseção 2.3.2, uma rede de Petri binária pode ser definida como uma rede de Petri com uma regra de evolução diferente para a marcação de lugares alcançados após o disparo de uma transição t_j dada pela equação (2.5).

É importante observar que definir $\mathcal{N}_{C_o}$ como uma rede de Petri binária não é suficiente para garantir que $\mathcal{N}_{C_o}$ possa ser usada como um observador de estados. Suponha, por exemplo, que p_{C_i} é um lugar de $\mathcal{N}_{C_o}$ que possui uma ficha e não tem uma transição de saída rotulada com um evento observável $\sigma_o \in \Sigma_o$. Suponha ainda que p_{C_i} não possui uma transição de entrada habilitada rotulada com σ_o. Então, se σ_o ocorrer, p_{C_i} permanece com uma ficha. Considerando que um lugar p_{C_i} com uma ficha representa um possível estado atual q_{C_i} de G_C, pode-se verificar que, neste exemplo, p_{C_i} não deveria ter permanecido com uma ficha, o que mostra que o estado da rede de Petri binária $\mathcal{N}_{C_o}$ não corresponde aos possíveis estados atuais de G_C após a ocorrência de σ_o.

Para corrigir esse problema é necessário adicionar um arco conectando cada lugar p_{C_i} de $\mathcal{N}_{C_o}$ a uma nova transição, rotulada com os eventos observáveis de Σ_o que

não estão no conjunto de eventos ativos do estado q_{C_i} de G_C associados a p_{C_i}, gerando a rede de Petri observadora de estados, $\mathcal{N}_{SO}$. O conjunto formado com as novas transições criadas para eliminar as fichas dos lugares que não são associados com a estimativa de estados de G_C será denotado neste trabalho por conjunto de transições observáveis complementares, T'_{C_o}. Essa modificação e o fato da rede de Petri observadora de estados ser uma rede de Petri binária garantem que se o lugar p_{C_i} não está associado a um possível estado atual de G_C após o disparo de uma transição observável, então o número de fichas de p_{C_i} será igual a zero.

Para definir o estado inicial de $\mathcal{N}_{SO}$, uma ficha é atribuída a cada lugar associado a um estado de $UR(q_{0,C})$ e o número de fichas dos demais lugares é feito igual a zero. Essa definição garante que o conjunto de lugares de $\mathcal{N}_{SO}$ que têm inicialmente uma ficha corresponde ao conjunto de possíveis estados iniciais de G_C, dados por $UR(q_{0,C})$. Finalmente, as transições de autolaço e seus arcos associados foram removidas da rede de Petri, já que o disparo de uma transição de autolaço não altera a estimativa de estados. O seguinte algoritmo mostra como a rede de Petri observadora de estados $\mathcal{N}_{SO}$ pode ser obtida a partir de $\mathcal{N}_{C_o}$.

Algoritmo 3.5 *(Rede de Petri observadora de estados) Seja* $\mathcal{N}_{C_o} = (P_C, T_{C_o}, Pre_{C_o}, Post_{C_o}, x_{0,C}, \Sigma_o, l_{C_o})$ *obtida a partir dos passos do algoritmo 3.4. Então, a rede de Petri observadora de estados binária* $\mathcal{N}_{SO} = (P_C, T_{SO}, Pre_{SO}, Post_{SO}, x_{0,SO}, \Sigma_o, l_{SO})$ *pode ser calculada da seguinte forma.*

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

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

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

- *Passo 4: Defina* $Pre_{SO} : P_C \times T_{SO} \rightarrow \mathbb{N}$ *e* $Post_{SO} : T_{SO} \times P_C \rightarrow \mathbb{N}$, *em que* $Pre_{SO}(p_{C_i}, t_{C_j}) = Pre_{C_o}(p_{C_i}, t_{C_j})$ *e* $Post_{SO}(t_{C_j}, p_{C_\ell}) = Post_{C_o}(t_{C_j}, p_{C_\ell})$, *para*

todo $p_{C_i}, p_{C_\ell} \in P_C$ *e* $t_{C_j} \in T_{C_o}$, *e* $Pre_{SO}(p_{C_i}, t^i_C) = 1$, $Pre_{SO}(p_{C_\ell}, t^i_C) = 0$ *e* $Post_{SO}(t^i_C, p_{C_\ell}) = Post_{SO}(t^i_C, p_{C_i}) = 0$, *para todo* $t^i_C \in T'_{C_o}$ *e* $p_{C_i}, p_{C_\ell} \in P_C$, *em que* $i \neq \ell$.

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

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

Após $\mathcal{N}_{SO}$ ter sido obtida, a rede de Petri diagnosticadora $\mathcal{N}_D$ pode ser calculada adicionando-se a $\mathcal{N}_{SO}$ transições t_{f_k}, para $k = 1, \ldots, r$, em que também são adicionados a cada transição t_{f_k} um lugar de entrada p_{N_k}, inicialmente com uma ficha, e um lugar de saída p_{F_k} sem fichas, ambos conectados a t_{f_k} por arcos ordinários. Cada transição t_{f_k} está associada à verificação da ocorrência de um tipo de falha. Arcos inibidores [4] de peso igual a um são usados para conectar cada lugar associado a um estado de G_C que tem uma coordenada (q, N_k) à transição t_{f_k}. Como o arco inibidor de peso um habilita a transição apenas quando o número de fichas do lugar de entrada é igual a zero, então t_{f_k} será habilitada apenas quando o comportamento normal do sistema com relação à falha do tipo F_k não for possível, ou seja, apenas quando todos os lugares com coordenadas (q, N_k) não possuírem fichas, o que implica que uma falha do conjunto Σ_{f_k} ocorreu. Um arco inibidor será representado por um arco cuja extremidade final possui um pequeno círculo.

A transição t_{f_k} é rotulada com o evento sempre ocorrente λ [4] para representar que t_{f_k} dispara imediatamente após ter sido habilitada, removendo a ficha do lugar p_{N_k} e adicionando uma ficha ao lugar p_{F_k}, o que indica que uma falha do tipo F_k ocorreu.

O algoritmo 3.6 resume os passos para o cálculo da rede de Petri diagnosticadora $\mathcal{N}_D$ a partir da rede de Petri $\mathcal{N}_{SO}$.

Algoritmo 3.6 *(Rede de Petri diagnosticadora) Seja* $\mathcal{N}_{SO} = (P_C, T_{SO}, Pre_{SO}, Post_{SO}, x_{0,SO}, \Sigma_o, l_{SO})$ *obtida de acordo com o algoritmo 3.5.*

Então a rede de Petri diagnosticadora $\mathcal{N}_D = (P_D, T_D, Pre_D, Post_D, In_D, x_{0,D}, \Sigma_o \cup \{\lambda\}, l_D)$ *pode ser calculada da seguinte forma.*

- *Passo 1: Seja* $T_f = \bigcup_{k=1}^{r} \{t_{f_k}\}$*, em que* t_{f_k} *é uma transição criada para identificar a ocorrência de um evento de falha do conjunto* Σ_{f_k}*.* $T_D = T_{SO} \cup T_f$*.*

- *Passo 2: Defina a função de marcação* $l_D : T_D \rightarrow 2^{\Sigma_o \cup \{\lambda\}}$*, em que* λ *denota o evento sempre ocorrente, tal que* $l_D(t_D) = l_{SO}(t_D)$ *para todo* $t_D \in T_{SO}$*, e* $l_D(t_D) = \{\lambda\}$ *para todo* $t_D \in T_f$*.*

- *Passo 3: Seja* $P_{NF} = \bigcup_{k=1}^{r} \{p_{N_k}, p_{F_k}\}$*, em que* p_{N_k} *e* p_{F_k} *são, respectivamente, os lugares de entrada e saída da transição* t_{f_k}*.* $P_D = P_C \bigcup P_{NF}$*.*

- *Passo 4: Defina* $Pre_D : P_D \times T_D \rightarrow \mathbb{N}$ *e* $Post_D : T_D \times P_D \rightarrow \mathbb{N}$ *em que* $Pre_D(p_{C_i}, t_D) = Pre_{SO}(p_{C_i}, t_D)$ *e* $Post_D(t_D, p_{C_i}) = Post_{SO}(t_D, p_{C_i})$ *para todo* $t_D \in T_{SO}$ *e* $p_{C_i} \in P_C$*,* $Pre_D(p_{N_k}, t_{f_k}) = Post_D(t_{f_k}, p_{F_k}) = 1$*, para todo* $t_{f_k} \in T_f$*, e* $Pre_D(p_{C_i}, t_{f_k}) = Post_D(t_{f_k}, p_{C_i}) = 0$*, para todo* $t_{f_k} \in T_f$ *e* $p_{C_i} \in P_C$*.*

- *Passo 5: Defina a função dos arcos inibidores* $In_D : P_D \times T_D \rightarrow \{0, 1\}$*, em que* $In_D(p_D, t_{f_k}) = 1$ *para todo lugar* $p_D \in P_C$ *associado a um estado de* G_C *que tem uma coordenada* (q, N_k)*, e* $In_D(p_D, t_D) = 0$*, para todos os outros lugares* $p_D \in P_D$ *e transições* $t_D \in T_D$*.*

- *Passo 6: O estado inicial dos lugares* p_{N_k} *é um e dos lugares* p_{F_k} *é zero. Os demais lugares possuem a mesma condição inicial definida por* $x_{0,SO}$*.*

O algoritmo 3.7 resume os passos necessários para a obtenção da rede de Petri diagnosticadora $\mathcal{N}_D$ a partir do autômato G_C.

Algoritmo 3.7

- *Passo 1: Calcule a rede de Petri máquina de estados rotulada* $\mathcal{N}_C = (P_C, T_C, Pre_C, Post_C, x_{0,C}, \Sigma, l_C)$ *a partir de* G_C *usando o algoritmo 3.2.*

- *Passo 2: Calcule a rede de Petri* $\mathcal{N}_{C_o} = (P_C, T_{C_o}, Pre_{C_o}, Post_{C_o}, x_{0,C}, \Sigma_o, l_{C_o})$ *de acordo com o algoritmo 3.4.*

- *Passo 3: Calcule a rede de Petri observadora de estados binária* $\mathcal{N}_{SO} = (P_C, T_{SO}, Pre_{SO}, Post_{SO}, x_{0,SO}, \Sigma_o, l_{SO})$ *seguindo os passos do algoritmo 3.5.*

- *Passo 4: Calcule a rede de Petri diagnosticadora* $\mathcal{N}_D = (P_D, T_D, Pre_D, Post_D, In_D, x_{0,D}, \Sigma_o \cup \{\lambda\}, l_D)$*, usando o algoritmo 3.6.*

Para provar que a rede de Petri diagnosticadora $\mathcal{N}_D$, obtida a partir do algoritmo 3.7, pode ser usada para o diagnóstico online, são apresentados os seguintes lemas que mostram que o estado da rede de Petri observadora de estados $\mathcal{N}_{SO}$, alcançado após a observação de uma sequência de eventos $\nu \in \Sigma_o^\star$, representa corretamente a estimativa de estados de G_C.

Lema 3.1 *Seja* $q_C = f_C(q'_C, \sigma_o)$ *um estado de* G_C*, em que* σ_o *é um evento observável, e seja* $t_{C_j} \in T_{C_o}$ *a transição de* $\mathcal{N}_{SO}$ *correspondente à transição* $q_C = f_C(q'_C, \sigma_o)$ *de* G_C*. Então, cada lugar pertencente a* $Reach_T(t_{C_j})$ *está associado a um estado de* $UR(q_C)$*.*

Prova: A prova é imediata a partir da equação (2.1) e do algoritmo 3.3. ■

Note, de acordo com o lema 3.1, que como os arcos de saída da transição $t_{C_j} \in T_{C_o}$ são definidos a partir de $Reach_T(t_{C_j})$, então o disparo de t_{C_j} faz com que todos os lugares associados ao alcance não observável de $q_C = f_C(q'_C, \sigma_o)$ recebam fichas.

Lema 3.2 *Seja* $\underline{x}_{SO}$ *o estado de* $\mathcal{N}_{SO}$ *alcançado após a observação de uma sequência de eventos* ν*, e seja* q_{obs} *o estado de* $Obs(G_C)$ *alcançado após a observação de* ν*. Então, existe um lugar* p_{C_i} *tal que* $x_{SO}(p_{C_i}) = 1$ *se e somente se* p_{C_i} *está associado a um estado* q_{C_i} *pertencente a* q_{obs}*.*

Prova: Seja q_{obs} um estado de $Obs(G_C)$ e $\underline{x}_{SO}$ o estado de $\mathcal{N}_{SO}$ tal que $x_{SO}(p_{C_i}) = 1$ se e somente se p_{C_i} está associado a um estado $q_{C_i} \in q_{obs}$. Suponha que um evento $\sigma_o \in \Sigma_o$ seja observado. Então, de acordo com o algoritmo 2.1, o próximo estado de $Obs(G_C)$, q'_{obs}, é obtido definindo-se inicialmente os estados de q_{obs} que possuem σ_o como evento ativo e fazendo o alcance não observável dos estados alcançados após a ocorrência de σ_o.

Note que esse procedimento para obtenção da estimativa de estado do sistema é análogo ao utilizado na rede de Petri $\mathcal{N}_{SO}$. Cada estado de q_{obs} está associado a um lugar com uma ficha em $\mathcal{N}_{SO}$, habilitando transições rotuladas por eventos observáveis. Na ocorrência de σ_o as transições habilitadas rotuladas por σ_o disparam. Essas transições correspondem aos eventos ativos dos estados de q_{obs}. Assim, de acordo com o lema 3.1, o disparo dessas transições leva a uma nova marcação em que todos os lugares associados a q'_{obs} receberão uma ficha.

Para que apenas os lugares que permanecem com fichas sejam os associados a q'_{obs}, é necessário retirar as fichas dos lugares de $\underline{x}_{SO}$ associados aos estados de q_{obs} que não possuem σ_o como evento ativo. Isso é feito através das transições observáveis complementares T'_{C_o} introduzidas na rede de Petri.

Finalmente, como a rede de Petri $\mathcal{N}_{SO}$ é binária, cada lugar que já possui uma ficha e recebe outra, permanece com uma ficha indicando que o lugar pertence à nova estimativa de estado.

Para concluir a prova, basta notar, de acordo com o passo 5 do algoritmo 3.5, que os lugares de $\mathcal{N}_{SO}$ que possuem fichas inicialmente são os associados ao estado inicial de $Obs(G_C)$. ■

Os resultados apresentados no teorema 3.1 e nos lemas 3.1 e 3.2 levam ao seguinte teorema [38].

Teorema 3.2 *Seja L a linguagem gerada pelo sistema G e suponha que L é diagnosticável em relação a P_o e Π_f. Seja $s \in L \setminus L_{N_k}$ tal que $\forall \omega \in L$ satisfazendo $P_o(\omega) = P_o(s)$, $\omega \in L \setminus L_{N_k}$. Então, o número de fichas no lugar p_{F_k} de $\mathcal{N}_D$, após a observação de uma sequência $P_o(s)$ é igual a um.*

Prova: Uma vez que $s \in L \setminus L_{N_k}$ não é uma sequência ambígua, então, de acordo com o teorema 3.1, a *k-ésima* coordenada de todos os possíveis estados de G_C, alcançados após a ocorrência de s, dados por $Reach(P_o(s))$, é igual a F_k. Portanto, de acordo com os lemas 3.1 e 3.2, todos os lugares p_{D_i}, associados a um estado q_{C_i} da forma (q, N_k), não possuem fichas. Como esses lugares estão conectados à transição

t_{f_k} através de arcos inibidores e t_{f_k} é uma transição rotulada pelo evento sempre ocorrente, então t_{f_k} dispara tão logo se torna habilitada e p_{F_k} recebe uma ficha. ■

Observação 3.1 *Note que a rede de Petri diagnosticadora tem* $|T_{C_o}| + |T'_{C_o}| + |T_f|$ *transições e* $|P_C| + |P_{NF}|$ *lugares. Como* $|P_{NF}| = 2 \times |T_f|$ *e, no pior caso,* $|T'_{C_o}| = |P_C|$*, então o número de transições da rede de Petri diagnosticadora depende linearmente do número de estados e transições de* G_C *e do número de tipos de falha. O número de arcos ordinários da rede de Petri diagnosticadora é limitado por* $(|P_C| + 1) \times |T_{C_o}| + 2 \times |T_f| + |P_C|$ *e o número de arcos inibidores é limitado por* $|P_C| \times |T_f|$*. Portanto, a complexidade computacional da construção da rede de Petri diagnosticadora* $\mathcal{N}_D$ *é polinomial no número de estados e transições de* G_C *e no número de tipos de falha. Essa é uma consequência direta de como o alcance não observável é implementado em* $\mathcal{N}_D$*, i.e., usando o alcance não observável de lugares,* $Reach_T$*, ao invés do usual alcance não observável de um estado* q*,* $UR(q)$*. O alcance não observável de lugares é aplicado às transições da rede de Petri rotulada com eventos observáveis, definindo os lugares que recebem uma ficha após seus disparos. A implementação do alcance não observável de lugares é, portanto, estrutural e independente do estado da rede de Petri.*

O exemplo a seguir ilustra os passos para a obtenção da rede de Petri diagnosticadora $\mathcal{N}_D$ e o processo de diagnóstico online a partir do autômato G_C obtido no exemplo 3.2.

Exemplo 3.5 *A partir do autômato* G_C *mostrado na figura 3.10, deseja-se obter a rede de Petri diagnosticadora* $\mathcal{N}_D$ *para* G_C*. De acordo com o algoritmo 3.7, o primeiro passo é calcular a rede de Petri máquina de estados* $\mathcal{N}_C$ *a partir de* G_C*, que pode ser vista na figura 3.11. Como resultado do segundo passo do algoritmo 3.7, a rede de Petri binária* $\mathcal{N}_{C_o}$*, ilustrada na figura 3.12, é obtida.*

Em seguida, realizando o Passo 3 do algoritmo 3.7, a rede Petri observadora de estados $\mathcal{N}_{SO}$*, mostrada na figura 3.13, é obtida a partir de* $\mathcal{N}_{C_o}$*. Por fim, ao executar o Passo 4 do algoritmo 3.7, obtém-se a rede de Petri diagnosticadora* $\mathcal{N}_D$*, apresentada na figura 3.14.*

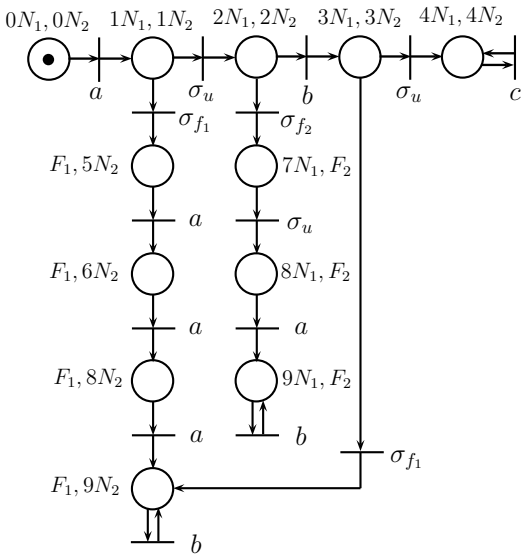


Figura 3.11: Rede de Petri máquina de estados $\mathcal{N}_C$ do exemplo 3.5.

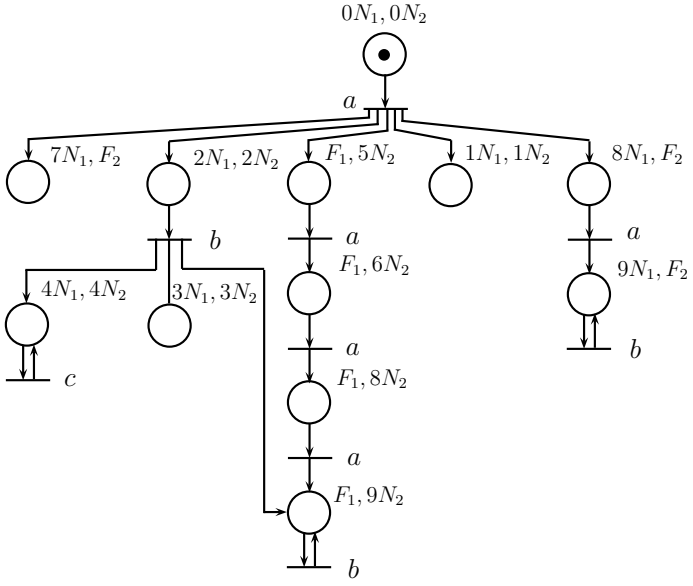


Figura 3.12: Rede de Petri binária $\mathcal{N}_{C_O}$ do exemplo 3.5.

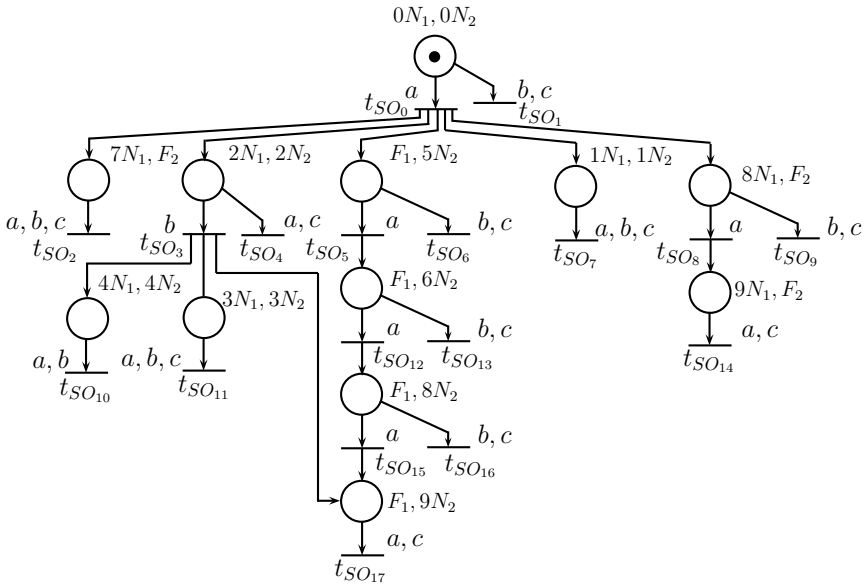


Figura 3.13: Rede de Petri observadora de estados $\mathcal{N}_{SO}$ do exemplo 3.5.

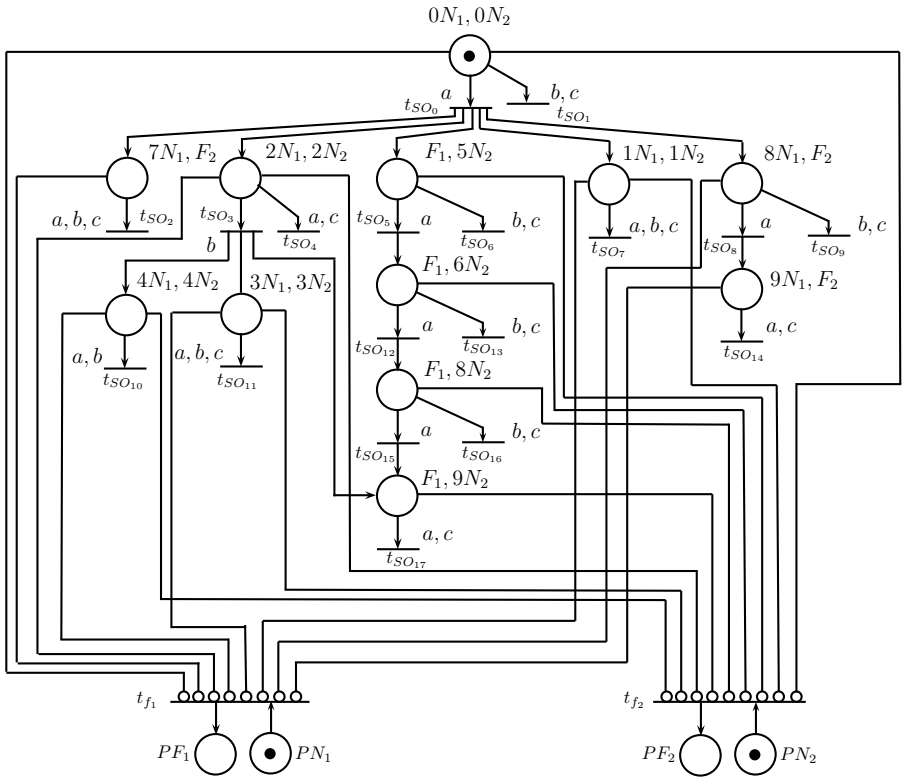


Figura 3.14: Rede de Petri diagnosticadora $\mathcal{N}_D$ do exemplo 3.5.

Tabela 3.1: Tabela que ilustra os lugares com ficha da rede de Petri $\mathcal{N}_D$ do exemplo 3.5 para cada sequência de eventos observada.

Sequência	Lugares com ficha
ε	$\{(0N_1, 0N_2)\}$
a	$\{(7N_1, F_2), (2N_1, 2N_2), (F_1, 5N_2), (1N_1, 1N_2), (8N_1, F_2)\}$
aa	$\{(F_1, 6N_2), (9N_1, F_2)\}$
aaa	$\{(F_1, 8N_2)\}$

Após $\mathcal{N}_D$ ter sido calculada, o processo de diagnóstico online pode ser iniciado. Esse processo será exemplificado da seguinte forma: suponha que uma sequência de falha $s = a\sigma_{f_1}aa \in L \setminus L_{N_1}$ tenha sido executada pelo sistema. Então, a sequência observada é $\nu = P_o(s) = aaa$. Como o estado inicial de $\mathcal{N}_D$ possui uma ficha apenas no lugar $(0N_1, 0N_2)$, associado ao estado inicial de G_C, então, após a ocorrência do primeiro evento a, a transição t_{SO_0} dispara e o conjunto de lugares associados com os possíveis estados de G_C que possuem uma ficha é dado por $\{(7N_1, F_2), (2N_1, 2N_2), (F_1, 5N_2), (1N_1, 1N_2), (8N_1, F_2)\}$. Quando o segundo evento a é observado, as transições t_{SO_2}, t_{SO_4}, t_{SO_5}, t_{SO_7}, t_{SO_8} disparam simultaneamente e o conjunto de lugares com uma ficha é dado por $\{(F_1, 6N_2), (9N_1, F_2)\}$. Note que as transições t_{SO_2}, t_{SO_4} e t_{SO_7} foram criadas de acordo com o Passo 1 do algoritmo 3.5 para remover as fichas dos lugares que não estão associados aos possíveis estados atuais de G_C. Após a ocorrência do terceiro evento a, as transições $t_{SO_{12}}, t_{SO_{14}}$ disparam e o único lugar de $\mathcal{N}_D$ que continua com uma ficha é dado por $(F_1, 8N_2)$.

Essa evolução é resumida na tabela 3.1, que ilustra os lugares com ficha após o disparo das transições rotuladas pelos eventos observados da sequência considerada, ou, do ponto de vista do sistema, ilustra a estimativa dos estados alcançados após a observação de uma sequência de eventos realizada. Por fim, como todos os lugares associados a um estado de G_C com uma coordenada (q, N_1) não possuem fichas, então a transição t_{f_1}, rotulada com o evento λ, é habilitada e dispara, removendo a ficha do lugar p_{N_1} e adicionando uma ficha ao lugar p_{F_1}, indicando a ocorrência do evento de falha σ_{f_1}.

Observação 3.2 *Note que o cálculo da estimativa de estado do diagnosticador, após*

a ocorrência de um evento observável $\sigma_o \in \Sigma_o$, pode ser realizado em dois passos: (i) identificação das transições habilitadas de $\mathcal{N}_D$ rotuladas com σ_o; (ii) disparo dessas transições e cálculo da nova marcação da rede de Petri diagnosticadora. Esse procedimento tem complexidade computacional linear em relação ao tamanho da rede de Petri diagnosticadora. Como, de acordo com a observação 3.1, $\mathcal{N}_D$ pode ser obtida em tempo polinomial em relação ao número de estados e transições de G_C e em relação ao número de tipos de falha, então a complexidade computacional de cada passo do procedimento de diagnóstico é também polinomial no número de estados e transições de G_C e no número de tipos de falha.

Neste capítulo foram apresentados todos os fundamentos para a criação de uma rede de Petri diagnosticadora que é capaz de realizar o alcance não observável de um sistema modelado por um autômato finito. Além disso, a rede de Petri diagnosticadora realiza o processo de diagnóstico online, indicando, após uma sequência arbitrariamente longa de eventos realizada pelo sistema, se uma falha ocorreu e qual o tipo de falha ocorreu.

No próximo capítulo serão apresentados os fundamentos básicos de controladores lógicos programáveis e de duas linguagens de programação definidas pela norma IEC61131-3 [22]: diagrama Ladder e SFC. Esses fundamentos servem de base para os métodos de conversão da rede de Petri diagnosticadora apresentados no capítulo 5.

Capítulo 4

Controladores Lógicos Programáveis

Um controlador lógico programável (CLP) é um computador projetado para funcionar em um ambiente industrial. Trata-se de um sistema eletrônico que usa memória programável capaz de armazenar um conjunto de instruções que são usadas para executar uma série de funções específicas. Essas funções são usadas para controlar vários tipos de processo através de entradas e saídas digitais ou analógicas [42].

O CLP interage com uma planta de automação através de sensores e atuadores. Sensores são dispositivos capazes de converter uma condição física de um elemento em um sinal elétrico que pode ser usado por um CLP através de uma conexão de entrada. Atuadores são dispositivos que convertem um sinal elétrico emitido pelo CLP em uma condição ou ação física. O usuário interage com o CLP através da programação de códigos de controle.

Por exemplo, um sensor de presença digital fornece o sinal lógico 1 quando um determinado objeto está posicionado em frente ao sensor e fornece o sinal 0 caso contrário. Dessa forma, o sensor converte a condição física de existir um objeto em frente ao sensor para um sinal elétrico que é interpretado como sinal lógico 1. De forma semelhante, um atuador, por exemplo, uma esteira, recebe o sinal lógico da saída do CLP (0 ou 1) e então executa a ação correspondente ao valor

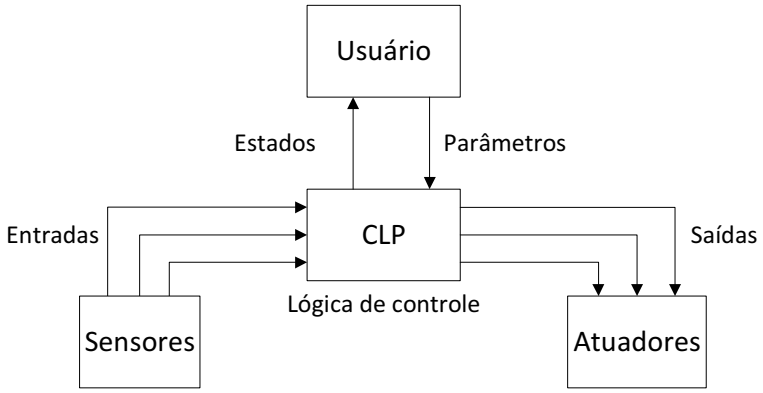


Figura 4.1: Esquema que ilustra a relação entre o CLP e os componentes de um sistema de automação.

lógico considerado. Nesse caso, a esteira se move se o sinal lógico tiver o valor 1 e interrompe seu movimento se o sinal possuir o valor 0, em outras palavras, o valor lógico foi convertido em uma condição física (esteira parada ou em movimento).

O código de controle é programado de tal forma que o CLP controla o sistema recebendo valores dos sensores e gerando as saídas para os atuadores, fazendo com que a planta realize um comportamento desejado. O usuário pode interagir com o controlador através de mudanças nos parâmetros de controle. A figura 4.1 ilustra como o CLP interage com o usuário e o sistema através de sensores e atuadores.

De forma geral, o controlador pode operar em dois modos chamados de programação e execução [43]. No modo de programação, o CLP fica aguardando ser configurado, programado, ou receber modificações de programas previamente configurados. Nesse modo, o CLP não executa ações, sendo apenas possível configurá-lo. No modo de execução, o CLP executa o código programado pelo usuário. Nesse modo, o CLP funciona realizando ciclos de varredura. Um ciclo de varredura é constituído de três etapas: (*i*) a leitura das entradas é realizada; (*ii*) o código de controle programado é executado; (*iii*) as variáveis de saída são atualizadas. A figura 4.2 representa o ciclo de varredura, mostrando a ordem em que as três etapas são realizadas.

Na primeira etapa do ciclo de varredura, o CLP lê e armazena os valores lógicos das variáveis de entrada em sua memória interna, e em seguida, inicia-se a etapa de

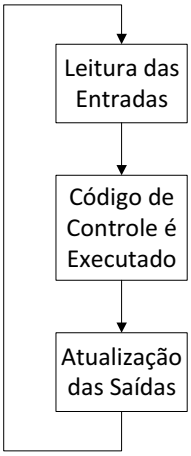


Figura 4.2: Esquema que ilustra a ordem de execução das etapas do ciclo de varredura.

execução do código de controle, em que os valores de entrada armazenados são utilizados para determinar os estados dos dispositivos. Conforme o código é executado, os valores de saída, resultantes da execução das lógicas de controle, são armazenados internamente, e, então, na terceira etapa, esses valores são usados para efetivamente atualizar os estados dos dispositivos de saída. Além disso, na terceira etapa também é realizada a atualização de valores de outras variáveis que representam resultados aritméticos, de contagem e temporizadores. Após essas etapas, o ciclo de varredura é encerrado e então um novo ciclo de varredura é iniciado. O tempo gasto para o CLP executar cada ciclo de varredura é chamado de tempo de varredura.

As linguagens de programação em que o CLP pode ser programado, definidas pela norma internacional IEC61131-3 [22], são: (i) diagrama bloco de funções; (ii) diagrama ladder; (iii) sequenciamento gráfico de funções (em inglês, *sequential function chart* - SFC); (iv) lista de instruções e (v) texto estruturado.

Entre as cinco linguagens, o presente trabalho aborda a conversão de uma rede de Petri diagnosticadora em SFC e em diagrama ladder. A linguagem SFC foi escolhida porque foi desenvolvida para atender processos com várias etapas simultâneas, o que torna a conversão de uma rede de Petri binária em um diagrama SFC um processo quase direto. Por outro lado, o diagrama ladder é o mais utilizado pela indústria e está disponível em quase todos os CLPs.

4.1 SFC

O SFC é uma linguagem baseada no Grafcet, que é usado para representação de sistemas sequenciais [43], tendo sido desenvolvido a partir das redes de Petri. Dessa forma, a linguagem SFC é ideal para modelagem de sistemas sequenciais que possuam processos que ocorrem em paralelo. A representação gráfica do SFC é apresentada a seguir.

4.1.1 Representação gráfica dos elementos

Etapas

No SFC, um sistema evolui a partir da ativação sequencial de etapas. Etapas são representadas por quadrados como pode ser visto na figura 4.3. A identificação da etapa é inserida dentro do quadrado. Quando o sistema entra em funcionamento, apenas as etapas iniciais estão ativas. Etapas iniciais são representadas por quadrados duplos, como pode ser visto na figura 4.4 [43].

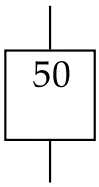


Figura 4.3: Ilustração de uma etapa simples de um código SFC.

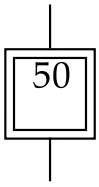


Figura 4.4: Ilustração de uma etapa inicial de um código SFC.

É possível associar ações às etapas, bastando, para isso, adicionar à direita da etapa um retângulo dividido em duas partes, em que, na primeira parte é colocado o qualificador da ação e, na segunda, a ação associada. Os qualificadores são letras

que indicam como a ação deve ser executada. A figura 4.5 mostra uma etapa inicial com uma ação associada [44].

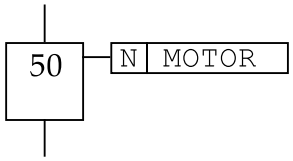


Figura 4.5: Exemplo de uma etapa com uma ação associada.

A tabela 4.1 mostra os qualificadores para cada ação que pode ser executada sobre o sistema automatizado [44].

Tabela 4.1: Correspondência entre os qualificadores do SFC e as ações a serem executadas no sistema automatizado.

Qualificador	**Ação**
N	Ação Simples
S	Set
R	Reset
L	Ação com tempo limitado
D	Ação de entrada retardada
SD	Ação de entrada com retardo prefixado
P	Ação pulsada
DS	Ação retardada e setada
SL	Ação retardada com tempo limitado

Transições

Cada etapa é ligada a outra através de uma transição. Arcos orientados fazem a ligação entre etapas e transições. O SFC evolui de cima para baixo, então, a orientação do arco é representada no diagrama SFC apenas quando o sentido é inverso. As transições são representadas por barras perpendiculares aos arcos orientados. O sistema progride através da transposição das transições, o que permite a desativação e ativação de etapas. Para que uma transposição ocorra é preciso que todas as etapas de entrada da transição estejam ativas e que a receptividade da transição seja verdadeira. Quando todas as etapas de entrada de uma transição estão ativas, diz-se que a transição está habilitada.

A receptividade de uma transição é uma expressão lógica associada à transição. Quando essa expressão se torna verdadeira, a transição pode ser transposta, tão logo as etapas de entrada estejam ativas. A figura 4.6 mostra um trecho de um código SFC em que a etapa 50 está ativa e a transição 1 será transposta tão logo a variável $SENSOR$ possua valor lógico verdadeiro. Neste trabalho, a ativação das etapas é representada por um pequeno ponto preto logo abaixo da identificação da etapa, mas na linguagem SFC não há representação gráfica para a ativação das etapas.

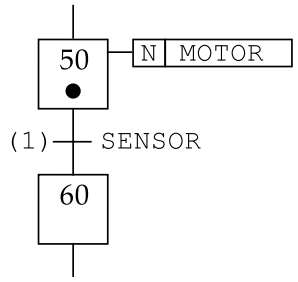


Figura 4.6: Ilustração de um código SFC composto de duas etapas e uma transição.

Regra de sintaxe e regra de evolução

A regra de sintaxe do SFC determina que cada etapa do SFC seja ligada a uma transição e cada transição seja ligada a uma ou mais etapas. Dessa forma, duas etapas nunca podem ser conectadas diretamente e a conexão direta deve conectar apenas uma etapa a uma transição ou uma transição a uma etapa.

Além da regra de sintaxe, existem cinco regras que determinam a evolução do SFC [43, 44]:

- Regra 1: A situação inicial, escolhida pelo projetista, é descrita pelas etapas ativas no tempo inicial.
- Regra 2: Uma transição é dita estar habilitada quando todas as etapas que imediatamente precedem e estão ligadas à transição estão ativas. A transição é transposta quando ela está habilitada e quando sua condição associada é verdadeira.

- Regra 3: A transposição de uma transição faz com que todas as etapas que imediatamente a sucedem se tornem ativas e todas as etapas que imediatamente a precedem sejam desativadas.
- Regra 4: Todas as transições que podem ser transpostas simultaneamente são transpostas simultaneamente.
- Regra 5: Se, durante a operação, uma etapa é simultaneamente ativada e desativada, ela permanece ativa.

4.1.2 Representação gráfica de estruturas sequenciais

Sequência

Uma sequência é uma sucessão de etapas de tal forma que [43]:

- cada etapa, com exceção da última etapa, tem apenas uma transição de saída,
- cada etapa, com exceção da primeira etapa, tem apenas uma transição de entrada habilitada por uma única etapa da sequência.

Além disso, uma sequência pode ter um número arbitrário de etapas. Na figura 4.7 é mostrado um exemplo de uma sequência de etapas genérica.

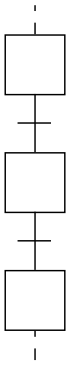


Figura 4.7: Representação gráfica de uma sequência de etapas.

Ciclo de uma única sequência

Um ciclo de uma única sequência possui as seguintes características [43]:

- cada etapa possui apenas uma transição de saída,
- cada etapa possui apenas uma transição de entrada habilitada por uma única etapa da sequência.

A representação gráfica de um ciclo de uma única sequência de etapas pode ser vista na figura 4.8.

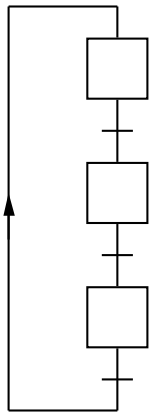


Figura 4.8: Representação gráfica de um ciclo de uma única sequência de etapas.

Seleção de sequências

A seleção de sequências mostra uma escolha de evolução entre diversas sequências iniciando de uma ou várias etapas [43]. A representação gráfica da seleção de sequências pode ser vista na figura 4.9. Essa estrutura é representada por todas as transições que são simultaneamente habilitadas pela mesma etapa e as possibilidades de evolução do sistema são iguais ao número de transições habilitadas.

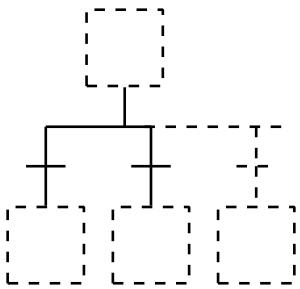


Figura 4.9: Representação gráfica da estrutura de seleção de sequências.

A ativação exclusiva de uma determinada sequência não é garantida apenas pela estrutura. O projetista deve garantir que o tempo, a lógica ou aspectos mecânicos das condições das transições sejam mutuamente excludentes. Os exemplos seguintes demonstram como é possível criar condições mutuamente excludentes.

Exemplo 4.1 *Considere o SFC parcial mostrado na figura 4.10. A escolha do caminho de evolução é garantida pela relação mutuamente excludente entre as receptividades das duas transições da figura 4.10. Se a e b são ambos verdadeiros quando a etapa 2 se tornar ativa, então nenhuma transição será transposta.*

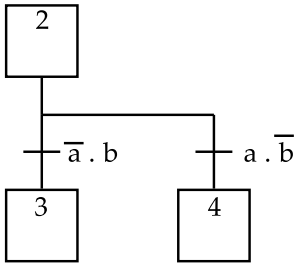


Figura 4.10: Representação gráfica de uma seleção de sequências com transições mutuamente excludentes.

Exemplo 4.2 *Considere o SFC parcial mostrado na figura 4.11. Nesse caso existe uma relação de prioridade que é dada à transição que conecta a etapa 2 à etapa 3. Quando a variável b for verdadeira e a etapa 2 estiver ativa, então a transição é transposta e a etapa 3 se torna ativa.*

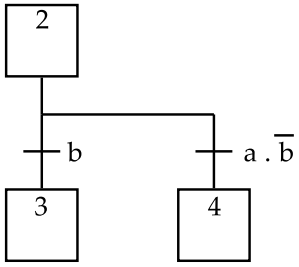


Figura 4.11: Representação gráfica de uma seleção de sequências com prioridade de sequência.

Salto de etapa

Ao programar em SFC é possível pular uma etapa ou uma sequência de etapas. Isso é feito ao colocar uma transição em paralelo à sequência de etapas que se deseja pular [43]. Um exemplo dessa estrutura pode ser vista na figura 4.12.

Figura 4.12: Representação gráfica de uma estrutura que permite saltar uma sequência de etapas.

Repetição de sequência

Ao programar em SFC há a possibilidade de criar um ciclo, ou de repetir uma sequência até que, por exemplo, uma determinada condição seja satisfeita [43]. A configuração da estrutura em SFC que permite isso pode ser vista na figura 4.13.

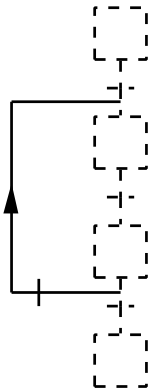


Figura 4.13: Representação gráfica de uma estrutura que permite a repetição de sequências de etapas até que uma determinada condição seja satisfeita.

Ativação de sequências paralelas

A estrutura em SFC que permite a ativação simultânea de diversas sequências a partir de uma ou mais etapas está ilustrada na figura 4.14 [43]. Note o símbolo de barras duplas horizontais indicando a sincronização das sequências a partir de uma ou mais etapas.

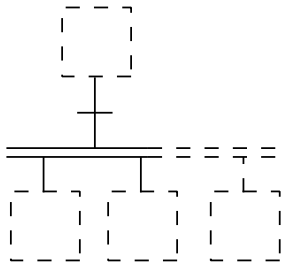


Figura 4.14: Representação gráfica de uma estrutura que permite a ativação de sequências paralelas.

Sincronização de sequências

Nessa estrutura, a transição só pode ser transposta quando todas as etapas precedentes a ela estiverem ativas e a receptividade da transição for verdadeira. Se essa sincronização for resultado de processos que estavam sendo realizados em paralelo, a evolução da etapa de saída da transição só continuará quando todos os processos que estiverem em paralelo terminarem. Essa estrutura está mostrada na figura 4.15, note que o símbolo de barras duplas novamente é usado para indicar a sincronização de sequências [43].

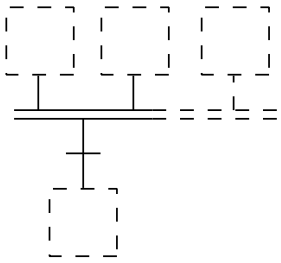


Figura 4.15: Representação gráfica de uma estrutura que sincroniza sequências em paralelo.

Sincronização e ativação de sequências em paralelo

A figura 4.16 ilustra a estrutura que permite a sincronização de sequências de etapas e a ativação de sequências de etapas em paralelo. Note a presença do símbolo de sincronização, indicando que, para a transição ser transposta, todas as etapas precedentes a ela devem estar ativas. Além disso, quando a transição é transposta, todas as etapas que sucedem a transição são ativadas simultaneamente, iniciando diversos processos em paralelo [43].

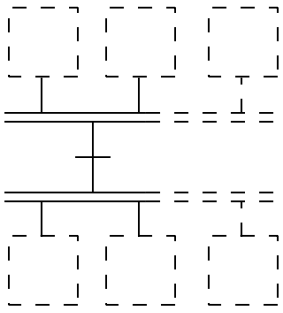


Figura 4.16: Representação gráfica de uma estrutura que sincroniza e ativa sequências em paralelo.

Etapa fonte

Uma etapa fonte é uma etapa que não possui nenhuma transição de entrada, como pode ser visto na figura 4.17 [45].

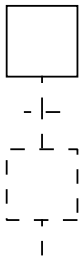


Figura 4.17: Representação gráfica de uma etapa fonte.

Fim de uma sequência por uma etapa dreno

Uma etapa dreno é uma etapa que não possui nenhuma transição de saída, como pode ser visto na figura 4.18 [45].

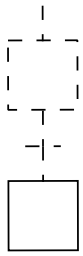


Figura 4.18: Representação gráfica de uma etapa dreno.

Transição fonte

Uma transição fonte é uma transição que não possui nenhuma etapa de entrada, como mostrado na figura 4.19 [45]. Por convenção, uma transição fonte é sempre habilitada, e é transposta tão logo a sua receptividade passa a ser verdadeira. Note que a etapa de saída da transição fonte permanece ativa enquanto a receptividade da transição for verdadeira, por isso, aconselha-se que a receptividade da transição fonte esteja associada a um evento de entrada ou a um evento interno.

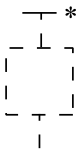


Figura 4.19: Representação gráfica de uma transição fonte.

Transição dreno

Uma transição dreno é uma transição que não possui etapas de saída, como é possível ver na figura 4.20 [45]. Note que, para que uma transição dreno seja transposta, é preciso que a etapa de entrada esteja ativa e que a receptividade da transição seja verdadeira. Conforme a transição é transposta, o único efeito é a desativação da etapa de entrada, ou das etapas de entrada, caso a transição dreno também seja uma transição de sincronização.

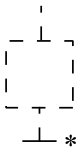


Figura 4.20: Representação gráfica de uma transição dreno.

Na próxima seção são apresentados alguns fundamentos do diagrama ladder, a segunda linguagem de programação em CLPs escolhida neste trabalho para a conversão da rede de Petri diagnosticadora.

4.2 Diagrama ladder

O diagrama ladder é uma das cinco linguagens definidas pela norma IEC61131-3 [22] para a programação de CLPs e é uma das linguagens mais usadas na indústria. Trata-se de uma linguagem simbólica que utiliza diversos componentes como contatos, bobinas, temporizadores, contadores, instruções de comparação, instruções de cálculos matemáticos elementares e instruções de cálculos matemáticos complexos.

Neste trabalho, são considerados apenas contatos e bobinas no código de controle em diagrama ladder.

4.2.1 Contatos

Contatos são componentes fundamentais no diagrama ladder. Os contatos mais usados são os contatos normalmente abertos (NA), os contatos normalmente fechados (NF) e os contatos tipo P e tipo N.

Os contatos NA funcionam verificando o estado lógico do bit endereçado ao contato. Se o valor lógico do bit for igual a 0, o contato retorna o valor lógico falso e não dá continuidade lógica no trecho do código ladder em que o contato está inserido. Se o bit endereçado possuir o valor lógico 1, o contato retorna o valor verdadeiro e dá continuidade lógica ao trecho em que está inserido. A figura 4.21 representa um contato NA associado à variável S.

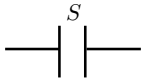


Figura 4.21: Contato NA associado à variável S.

O contato NF, ilustrado na figura 4.22, funciona de maneira inversa em relação ao contato NA. Quando o estado lógico do bit endereçado ao contato NF for igual a 1, o contato retorna o valor lógico falso, interrompendo a continuidade do trecho em que está inserido e, se o valor lógico do bit for 0, o contato retorna o valor verdadeiro, dando continuidade lógica ao trecho em que está inserido.

Figura 4.22: Contato NF associado à variável S.

Além dos contatos NA e NF, os contatos "positive signal edge" (tipo P) e "negative signal edge" (tipo N) são também muito utilizados na programação de con-

troladores de sistemas a eventos discretos. O contato do tipo P fica aberto e fecha por apenas um ciclo de varredura quando o valor lógico da variável associada muda de 0 para 1. Em outras palavras, o contato tipo P é usado para detectar a borda de subida da variável associada a ele.

Um exemplo de contato tipo P pode ser visto na figura 4.23. O exemplo 4.3 ilustra o funcionamento do contato tipo P associado à variável S.

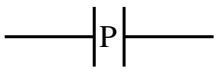


Figura 4.23: Contato "positive signal edge" (tipo P).

Exemplo 4.3 *Considere o trecho de código ladder exibido na figura 4.24. Suponha que o valor lógico da variável S seja 0 e, portanto, o contato tipo P está aberto. Caso a variável S mude seu valor lógico para 1, o contato tipo P detecta essa mudança e fecha durante um único ciclo de varredura. No próximo ciclo de varredura, a variável S continua com valor lógico 1, portanto, não há mudança positiva no valor lógico de S e o contato P abre novamente até que haja alguma mudança positiva no valor lógico de S.*

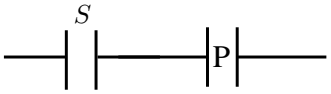


Figura 4.24: Contato tipo P associado a uma variável S.

O contato tipo N fica aberto até que uma mudança negativa no valor lógico da variável associada a ele seja detectada. Quando isso ocorre, o contato fecha por um ciclo de varredura e volta a abrir no próximo ciclo, até identificar novamente uma mudança negativa na variável associada, ou seja, o contato tipo N detecta a borda de descida da variável associada a ele. A representação do contato tipo N pode ser vista na figura 4.25.

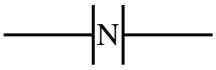


Figura 4.25: Contato “negative signal edge” (tipo N).

É importante notar que a característica dos contatos tipo P e N faz com que esses contatos sejam muito utilizados na implementação de diagramas ladder relacionados a SEDs. Nessas situações é necessário registrar a ocorrência de eventos no sistema que esteja sendo considerado. Os eventos são detectados com auxílio de sensores que produzem um sinal elétrico enviado ao CLP para que seja tomada a ação necessária. A detecção do evento é normalmente realizada utilizando-se a técnica de detecção de borda do sinal do sensor.

A técnica de detecção de borda consiste em detectar o instante em que houve uma transição de um valor para outro de uma determinada variável. Quando, por exemplo, um sensor de presença identifica uma peça que está sendo transportada por uma esteira, ele envia um sinal lógico ao CLP similar ao mostrado na figura 4.26. É possível determinar a borda de subida (instante em que o nível lógico de um sinal muda de 0 para 1) ou descida (instante em que o nível lógico de um sinal muda de 1 para 0) de um sinal. Uma seta para cima é usada como representação para a borda de subida e uma seta para baixo é usada como representação para a borda de descida, como pode ser visto na figura 4.26.

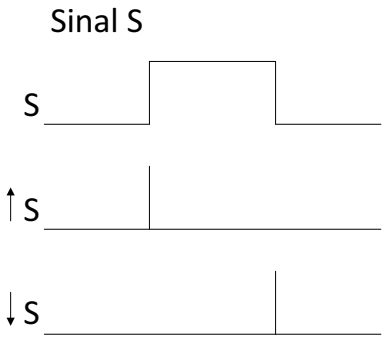


Figura 4.26: Exemplo de um sinal lógico S e a detecção da borda de subida e da borda de descida.

Suponha que a variável S do exemplo 4.3 seja relacionada ao sinal da figura 4.26. Então, conforme mostrado no exemplo 4.3, o contato tipo P vai fechar ao identificar a borda de subida de S, ou, em outras palavras, o contato tipo P fechará quando o sensor identificar o evento, indicando a ocorrência do evento no diagrama ladder.

4.2.2 Bobinas

Assim como os contatos, as bobinas são componentes básicos do diagrama ladder e funcionam atualizando as informações de saída, modificando o estado lógico de variáveis booleanas. Os principais tipos de bobinas são bobinas simples, bobinas SET e bobinas RESET.

Em bobinas simples, caso a lógica que as antecede seja verdadeira, diz-se que a bobina é energizada, isto é, muda seu valor lógico de 0 para 1. Caso a lógica anterior à bobina se torne falsa, a bobina então é desenergizada, retornando ao valor lógico 0. Esses valores lógicos são atribuídos à variável associada à bobina. Um exemplo de bobina simples pode ser visto na figura 4.27.

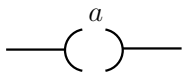


Figura 4.27: Representação de uma bobina simples com a variável a associada.

As bobinas SET e RESET funcionam de maneira diferente das bobinas simples. Se a lógica que antecede a bobina SET for verdadeira, o valor da variável relacionada à bobina se tornará igual a 1, ainda que a lógica que antecede a bobina se torne falsa, o valor lógico da variável continuará sendo igual a 1.

A bobina RESET funciona de forma inversa, ou seja, quando a bobina é energizada, o valor da variável associada a ela se torna igual a 0. A variável associada à bobina RESET permanece igual a zero até que uma bobina SET associada à mesma variável seja energizada.

As figuras 4.28 e 4.29 ilustram uma bobina SET e uma bobina RESET, respectivamente.

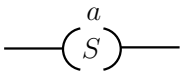


Figura 4.28: Representação de uma bobina SET com a variável a associada.

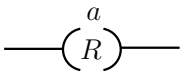


Figura 4.29: Representação de uma bobina RESET com a variável a associada.

Neste capítulo foram apresentados fundamentos básicos de CLP e das linguagens de programação SFC e diagrama ladder definidas pela norma internacional IEC61131-3 [22]. Métodos de conversão de redes de Petri diagnosticadoras em SFC e diagramas ladder, bem como aspectos práticos sobre a implementação de redes de Petri diagnosticadoras em CLPs, são apresentados no próximo capítulo.

Capítulo 5

Implementação em CLP de redes de Petri diagnosticadoras

Como apresentado no capítulo 4, um CLP opera realizando ciclos de varredura que consistem de três passos: (*i*) leitura e armazenagem das entradas do CLP; (*ii*) execução do código de programação e; (*iii*) atualização das saídas. Em geral, eventos de entrada são associados à borda de subida ou de descida de sinais de sensores e as saídas são comandos enviados do controlador para a planta em resposta a mudanças nos valores dos sinais dos sensores.

Para que o diagnosticador online seja implementado no mesmo CLP usado para controlar o sistema, o código do diagnosticador não pode ser inserido após o código do controlador, do contrário, eventos associados a mudanças nos sinais de sensores, e eventos de comando associados à resposta do controlador a essas mudanças seriam vistos pelo diagnosticador como eventos que estariam ocorrendo ao mesmo tempo. Para evitar esse problema, o código do diagnosticador deve ser implementado antes do código de controle, como é mostrado na figura 5.1.

Nas seções seguintes são apresentados métodos de conversão de redes de Petri diagnosticadoras para as linguagens de programação SFC e ladder, definidas pela norma internacional IEC61131-3 [22].

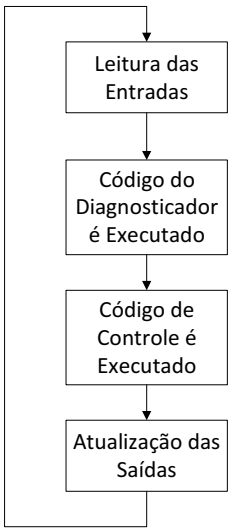


Figura 5.1: Ciclo de varredura do CLP com o código do diagnosticador implementado antes do código do controlador do sistema.

5.1 Conversão da rede de Petri diagnosticadora em SFC

O método de conversão da rede de Petri diagnosticadora em um SFC se dá de forma praticamente direta, uma vez que se trata de uma rede de Petri binária. O código do diagnosticador pode ser dividido em $r+1$ SFCs parciais em que um SFC parcial corresponde à rede de Petri observadora de estados $\mathcal{N}_{SO}$, e os demais r SFCs parciais representam testes de verificação da ocorrência de eventos de falha para cada um dos tipos de falha.

Na figura 5.2 o SFC da rede de Petri observadora de estados $\mathcal{N}_{SO}$ da figura 3.13 é apresentado. Cada lugar de $\mathcal{N}_{SO}$ é transformado em uma etapa do SFC, e as transições não são alteradas. Eventos são associados à borda de subida ou à borda de descida de sinais de sensores ou com comandos enviados à planta pelo controlador. Na rede de Petri da figura 3.13 existem três eventos, a, b e c, rotulando as transições. Assim, os sinais S_a, S_b e S_c, correspondendo, respectivamente, aos eventos a, b e c, são adicionadas às transições do SFC da figura 5.2 para representar as condições de disparo associadas aos eventos externos. Nesse exemplo, cada evento corresponde à borda de subida de um sinal de sensor.

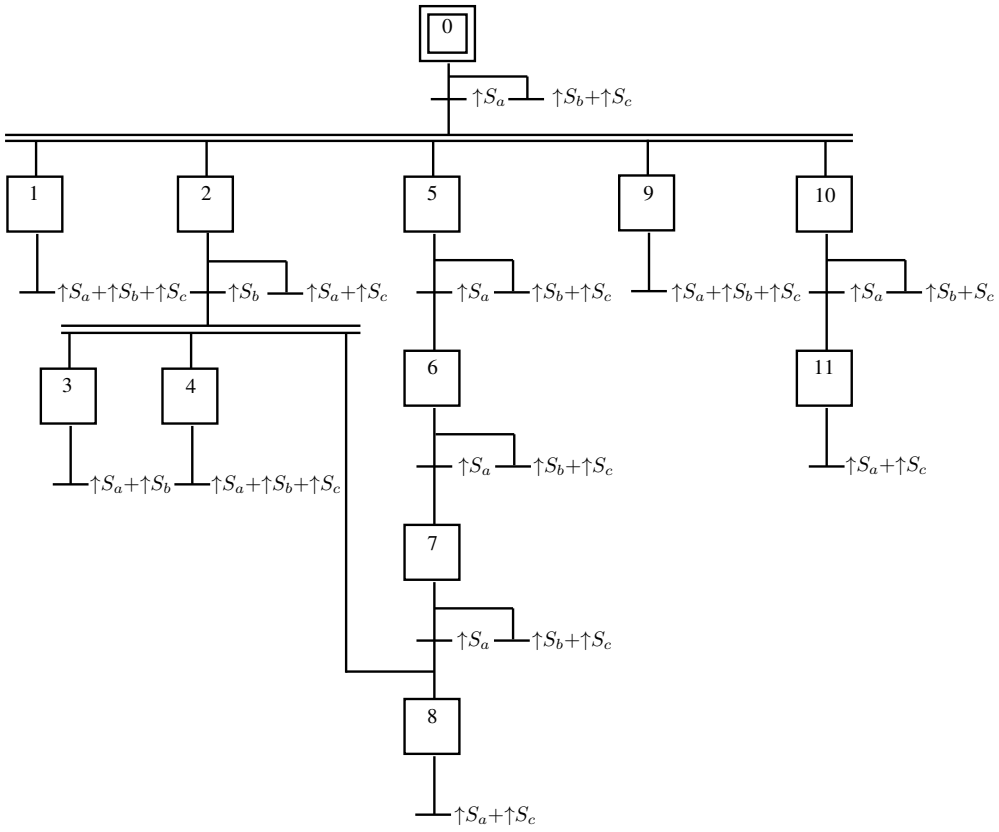


Figura 5.2: SFC da rede de Petri observadora de estados $\mathcal{N}_{SO}$ da figura 3.13.

A tabela 5.1 apresenta a correspondência entre cada lugar da rede de Petri observadora de estados e a etapa associada do SFC. Nas figuras 5.3 e 5.4 a verificação de falha é realizada para dois tipos de falha. Os SFCs parciais que realizam a verificação de falha possuem apenas duas etapas associadas aos lugares p_{N_k} e p_{F_k} e a única transição t_{f_k} possui a receptividade rotulada com uma expressão booleana que simula o efeito dos arcos inibidores da rede de Petri diagnosticadora $\mathcal{N}_D$. Quando essa expressão se torna verdadeira, a etapa associada ao lugar p_{F_k} é ativada e um conjunto de ações pode ser alocado a essa etapa para informar a ocorrência da falha.

Tabela 5.1: Correspondência entre os lugares da rede de Petri observadora de estados $\mathcal{N}_{SO}$ e as etapas associadas da implementação em SFC.

Lugares	**Etapas**
$0N_1 0N_2$	0
$7N_1 F_2$	1
$2N_1 2N_2$	2
$4N_1 4N_2$	3
$3N_1 3N_2$	4
$F_1 5N_2$	5
$F_1 6N_2$	6
$F_1 8N_2$	7
$F_1 9N_2$	8
$1N_1 1N_2$	9
$8N_1 F_2$	10
$9N_1 F_2$	11

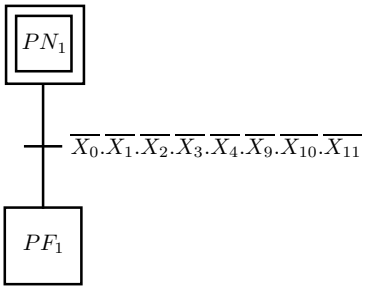


Figura 5.3: SFC da verificação da ocorrência do evento de falha σ_{f_1}.

5.2 Conversão da rede de Petri diagnosticadora em diagrama ladder

A linguagem ladder é a linguagem de programação em CLP mais usada pela indústria e está presente na maioria dos CLPs. Além disso, nesta seção é proposto um método para a conversão da rede de Petri diagnosticadora em diagrama ladder. O método, baseado em [46], consiste em dividir o código em módulos com o objetivo de evitar problemas de implementação. Na próxima seção são apresentados problemas relacionados à implementação de códigos de controle em diagramas ladder.

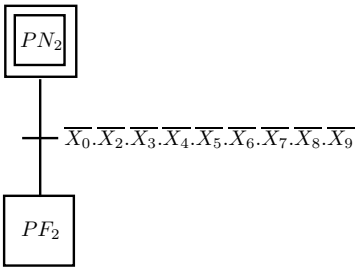


Figura 5.4: SFC da verificação da ocorrência do evento de falha σ_{f_2}.

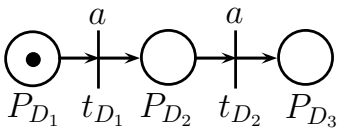


Figura 5.5: Rede de Petri parcial do exemplo 5.1.

5.2.1 Problemas de implementação de códigos de controle em diagrama ladder

Um problema importante relacionado à implementação de controladores em diagramas ladder é o chamado efeito avalanche. O efeito avalanche ocorre no código ladder quando as condições associadas a duas ou mais transições consecutivas são satisfeitas no mesmo ciclo de varredura, e uma transição que não estava habilitada é transposta. Esse efeito faz com que o programa pule um número arbitrário de estados durante um único ciclo de varredura, como é ilustrado no seguinte exemplo.

Exemplo 5.1 *Considere a rede de Petri da figura 5.5, a implementação em ladder dessa rede de Petri é mostrada na figura 5.6. Nesse exemplo, se o evento a ocorrer, uma ficha será retirada do lugar p_{D_1} e uma ficha será alocada no lugar p_{D_2}. Logo em seguida, no mesmo ciclo de varredura, uma ficha será retirada do lugar p_{D_2} e uma ficha será colocada no lugar p_{D_3}. Assim, devido a apenas uma ocorrência do evento a, duas transições consecutivas, t_{D_1} e t_{D_2}, são satisfeitas no mesmo ciclo de varredura e a transição t_{D_2}, que não estava habilitada, dispara.*

Um outro problema relacionado à implementação em ladder de redes de Petri, particularmente de redes de Petri binárias, ocorre quando um lugar simultaneamente

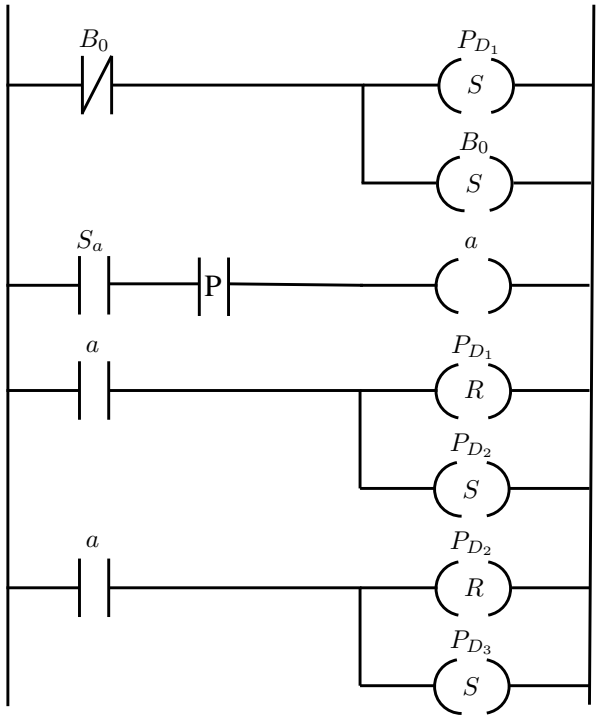


Figura 5.6: Diagrama ladder da rede de Petri parcial da figura 5.5 do exemplo 5.1.

recebe e perde uma ficha após o disparo de duas transições distintas. Dependendo da implementação da rede de Petri em ladder, a marcação dos lugares resultante pode estar errada, levando a uma implementação incorreta da dinâmica da rede de Petri. Esse problema é melhor apresentado na seção 5.2.5, junto com um exemplo e uma solução para evitá-lo. Nenhuma das técnicas de conversão de redes de Petri em diagramas ladder apresentadas na literatura pode solucionar esse problema.

Em [46], uma técnica de conversão que estabelece regras de transformação de redes de Petri interpretadas para controle em diagramas ladder que preserva a estrutura da rede de Petri e evita o efeito avalanche é apresentada. Neste trabalho, um método de conversão da rede de Petri diagnosticadora em um diagrama ladder é proposto baseado no método proposto por MOREIRA *et al.* [46]. O método foi alterado para considerar redes de Petri binárias e o problema de um lugar simultaneamente receber e perder uma ficha após o disparo de duas transições distintas. O método proposto consiste em dividir o diagrama ladder em cinco módulos da seguinte forma:

- Módulo 1, que representa a inicialização da rede de Petri, ou seja, define a marcação inicial;
- Módulo 2, associado à identificação de ocorrência de eventos externos;
- Módulo 3, associado às condições para os disparos das transições;
- Módulo 4, que descreve a evolução das fichas na rede de Petri;
- Módulo 5, que define os alarmes que serão acionados caso uma falha seja identificada e isolada;

Os módulos devem ser implementados na ordem apresentada para garantir o correto funcionamento da rede de Petri diagnosticadora. Nas próximas seções são apresentados cada um dos cinco módulos com mais detalhes e o método de conversão é ilustrado com a conversão da rede de Petri diagnosticadora da figura 3.14 em um diagrama ladder.

5.2.2 Módulo de inicialização

O módulo de inicialização contém apenas uma linha formada por um contato NF associado a uma variável binária interna B_0 que, no primeiro ciclo de varredura, energiza bobinas SET associadas aos lugares que contém uma ficha na marcação inicial. Após o ciclo de varredura inicial, o contato NF é aberto. É importante observar que não é preciso alocar o valor zero às variáveis associadas aos lugares sem marcação inicial já que as variáveis são automaticamente iniciadas com o valor zero.

A figura 5.7 ilustra o módulo inicial em ladder para a rede de Petri diagnosticadora da figura 3.14.

5.2.3 Módulo de eventos externos

Eventos externos são associados às bordas de subida ou descida de sinais de sensores na rede de Petri diagnosticadora. A mudança do valor lógico do sinal de um sensor

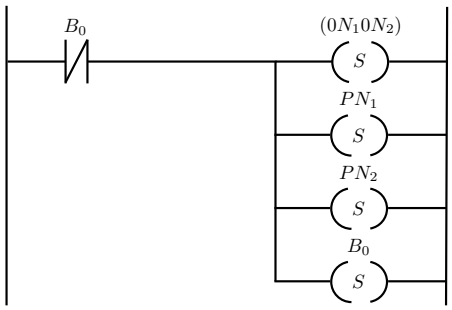


Figura 5.7: Módulo de inicialização da rede de Petri diagnosticadora da figura 3.14.

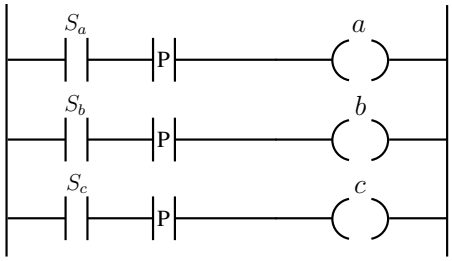


Figura 5.8: Módulo de eventos externos para a rede de Petri diagnosticadora da figura 3.14.

pode ser detectada usando um contato "positive signal edge" (tipo P) ou um contato "negative signal edge" (tipo N). O contato tipo P (ou tipo N) é normalmente aberto e então fecha, por apenas um ciclo de varredura, quando a condição booleana da mesma linha mudar seu valor lógico de zero para um (ou de um para zero).

Na rede de Petri da figura 3.14 existem três eventos: a, b e c, rotulando as transições. Neste trabalho será considerado que esses eventos são identificados pela borda de subida dos sinais dos sensores S_a, S_b e S_c, respectivamente. Portanto, o módulo de eventos para essa rede de Petri diagnosticadora deve ter três linhas, como pode ser visto na figura 5.8. Quando, por exemplo, S_a muda seu valor de zero para um, o contato tipo P fecha por um único ciclo de varredura, energizando a bobina denotada por a, que representa a borda de subida de S_a.

5.2.4 Módulo das condições para o disparo das transições

O módulo das condições para o disparo das transições possui $|T_D|$ linhas, em que $|.|$ denota a cardinalidade, e cada linha descreve as condições para o disparo da

transição $t_{D_j} \in T_D$. O conjunto de transições T_D pode ser particionado em $T_D = T_{SO} \dot{\cup} T_f$. Uma transição $t_{SO_j} \in T_{SO}$ está habilitada se e somente se seu único lugar de entrada possui uma ficha, e t_{SO_j} dispara quando um evento associado a t_{SO_j} ocorre. Por outro lado, uma transição $t_{f_k} \in T_f$, associada ao tipo de falha F_k, está habilitada quando todos os lugares de entrada conectados a t_{f_k}, por meio de arcos inibidores, não possuem fichas e apenas o lugar de entrada p_{N_k} possui uma ficha. Como a transição t_{f_k} está associada com o evento λ, ela dispara tão logo seja habilitada.

As condições de habilitação de uma transição $t_{SO_j} \in T_{SO}$ podem ser facilmente representadas em um diagrama ladder usando-se um contato normalmente aberto associado ao lugar de entrada de t_{SO_j}, em série com uma associação de contatos normalmente abertos em paralelo associados aos eventos de t_{SO_j}.

As condições de disparo de uma transição $t_{f_k} \in T_f$ podem ser representadas por uma associação em série de contatos normalmente fechados usados para simular o efeito dos arcos inibidores conectando os lugares p_{D_i} à transição t_{f_k}, em que $In(p_{D_i}, t_{f_k}) = 1$, e um contato NA, em série com os contatos NF, que representam o arco ordinário do lugar p_{N_k} a t_{f_k}. Cada linha tem uma bobina associada com uma variável binária que representa a habilitação de uma transição da rede de Petri.

Na figura 5.9, apenas seis linhas do diagrama ladder do módulo das condições para o disparo das transições da rede de Petri diagnosticadora da figura 3.14 estão representadas. A primeira, a segunda e a terceira linhas estão associadas com o disparo das transições $t_{SO_0}, t_{SO_1}, t_{SO_2} \in T_{SO}$ e as três últimas linhas estão associadas com o disparo das transições $t_{SO_{17}} \in T_{SO}$ e $t_{f_1}, t_{f_2} \in T_f$.

5.2.5 Módulo da dinâmica da rede de Petri

Após a ocorrência de um evento observável, o número de fichas nos lugares da rede de Petri deve ser atualizado para representar a estimativa de estado correta do sistema. Esse processo é realizado através do módulo da dinâmica da rede de Petri. Uma vez que todos os lugares da rede de Petri diagnosticadora devem ser seguros, então uma

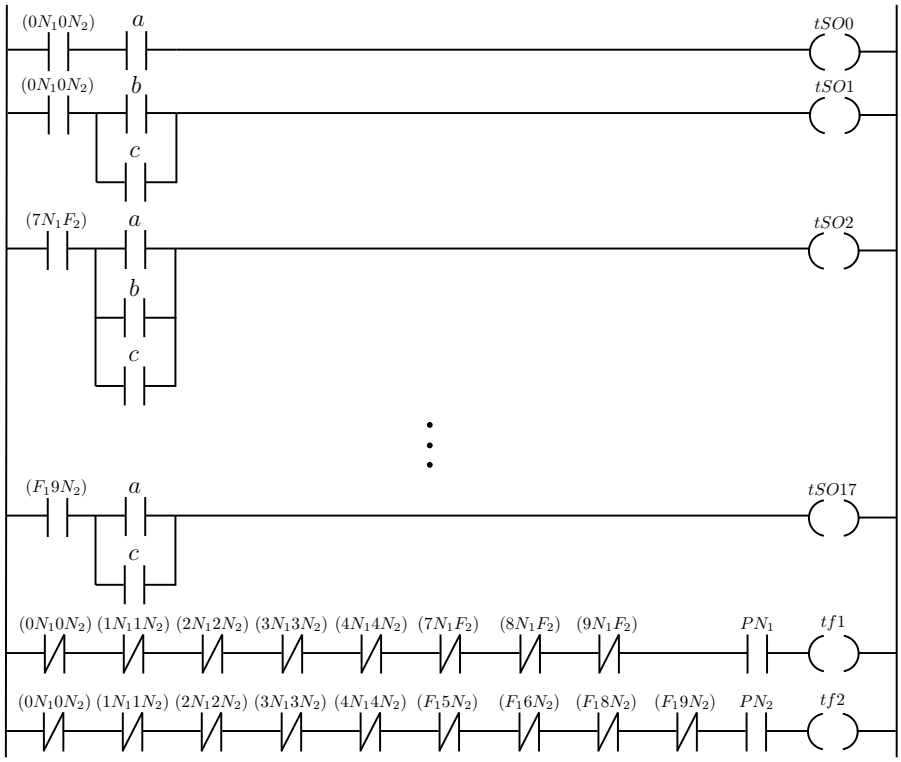


Figura 5.9: Módulo das condições de disparo das transições para a rede de Petri diagnosticadora da figura 3.14.

bobina SET ou RESET deve ser usada para alocar o valor um ou zero à variável binária que representa o número de fichas de um lugar da rede de Petri.

No módulo da dinâmica da rede de Petri, cada linha do diagrama ladder representa o disparo de uma transição. Isso é feito utilizando-se um contato NA associado a uma dada transição em série com bobinas SET e RESET que representam os lugares que recebem e perdem ficha, respectivamente, após o disparo dessa transição. Para tanto, as bobinas SET e RESET estão associadas às variáveis que representam os lugares de saída e entrada de cada transição, respectivamente. Dessa forma, o módulo da dinâmica da rede de Petri implementa a evolução das fichas na rede de Petri diagnosticadora em consequência do disparo das transições.

O disparo simultâneo de diversas transições rotuladas com o mesmo evento σ_o da rede de Petri diagnosticadora pode levar à situação em que a transição de saída de um lugar p_{D_i}, que tem uma ficha, dispara ao mesmo tempo em que uma transição de entrada de p_{D_i} também dispara. Nesse caso, p_{D_i} precisa continuar com uma

ficha após a ocorrência do evento observável σ_o. Dependendo da implementação em ladder da dinâmica da rede de Petri, a marcação do lugar p_{D_i} pode, de forma incorreta, ser igual a zero após a ocorrência de σ_o. Para ilustrar esse fato, considere uma parte de uma rede de Petri diagnosticadora mostrada na figura 5.10. Nesse exemplo, se as linhas são implementadas na ordem apresentada na figura 5.11(a), então a marcação de p_{D_3} será igual a zero após a ocorrência do evento a, uma vez que as transições t_{D_2} e t_{D_3} estão habilitadas de acordo com a marcação atual e são rotuladas com o mesmo evento.

Esse comportamento incorreto pode ser evitado mudando a ordem das linhas do módulo da dinâmica da rede de Petri. Entretanto, definir a ordem correta das linhas pode ser difícil se a rede de Petri for complexa. Uma maneira simples de contornar esse problema é considerar duas linhas ao invés de uma para representar a mudança de marcação dos lugares após o disparo de uma transição t_{D_j}. Na primeira linha, uma associação em série de contatos NF é adicionada para verificar se uma transição de entrada do único lugar de entrada de t_{D_j} satisfaz as condições de disparo. Se a resposta for sim, então o lugar de entrada de t_{D_j} deve permanecer com uma ficha, o que implica que a bobina de RESET associada com o lugar de entrada de t_{D_j} não pode ser energizada. A segunda linha garante que as bobinas SET dos lugares de saída de t_{D_j} são energizadas. O módulo correto da dinâmica da rede de Petri da figura 5.10 é apresentada na figura 5.11(b). Note que, após a ocorrência do evento a, na implementação em diagrama ladder da figura 5.11(b), as variáveis binárias que terão valor igual a um são as que estão associadas com os lugares p_{D_3} e p_{D_4}, como esperado.

O módulo da dinâmica da rede de Petri possui, no pior caso, $2 \times |T_D|$ linhas. O diagrama ladder do módulo da dinâmica da rede de Petri diagnosticadora da figura 3.14 é apresentado na figura 5.12.

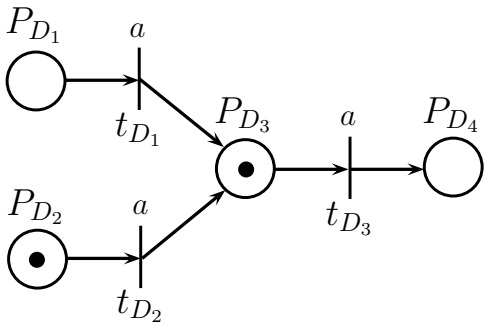


Figura 5.10: Fração de uma rede de Petri com duas transições consecutivas habilitadas sincronizadas com o mesmo evento.

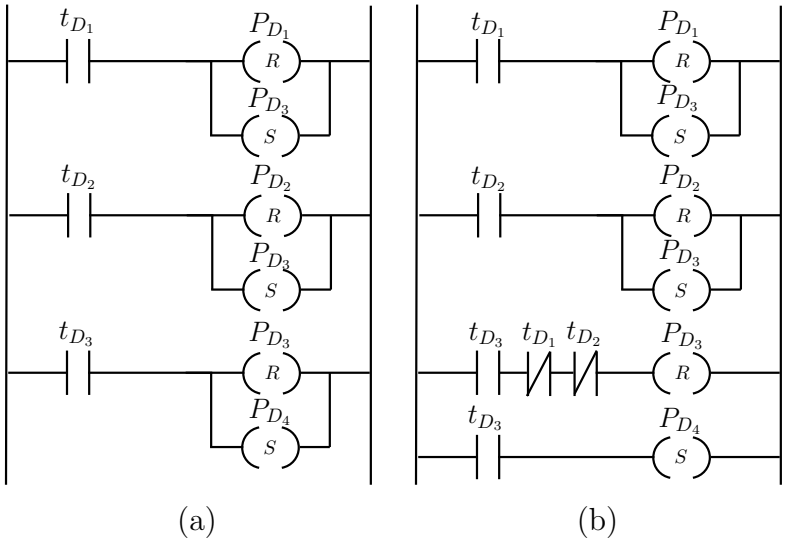


Figura 5.11: Módulo incorreto da dinâmica da rede de Petri para a rede de Petri da figura 5.10 (a), e módulo correto da dinâmica da rede de Petri usando uma associação em série de contatos NF para o *reset* da variável binária associada com o lugar de entrada de t_{D_3}, p_{D_3} (b).

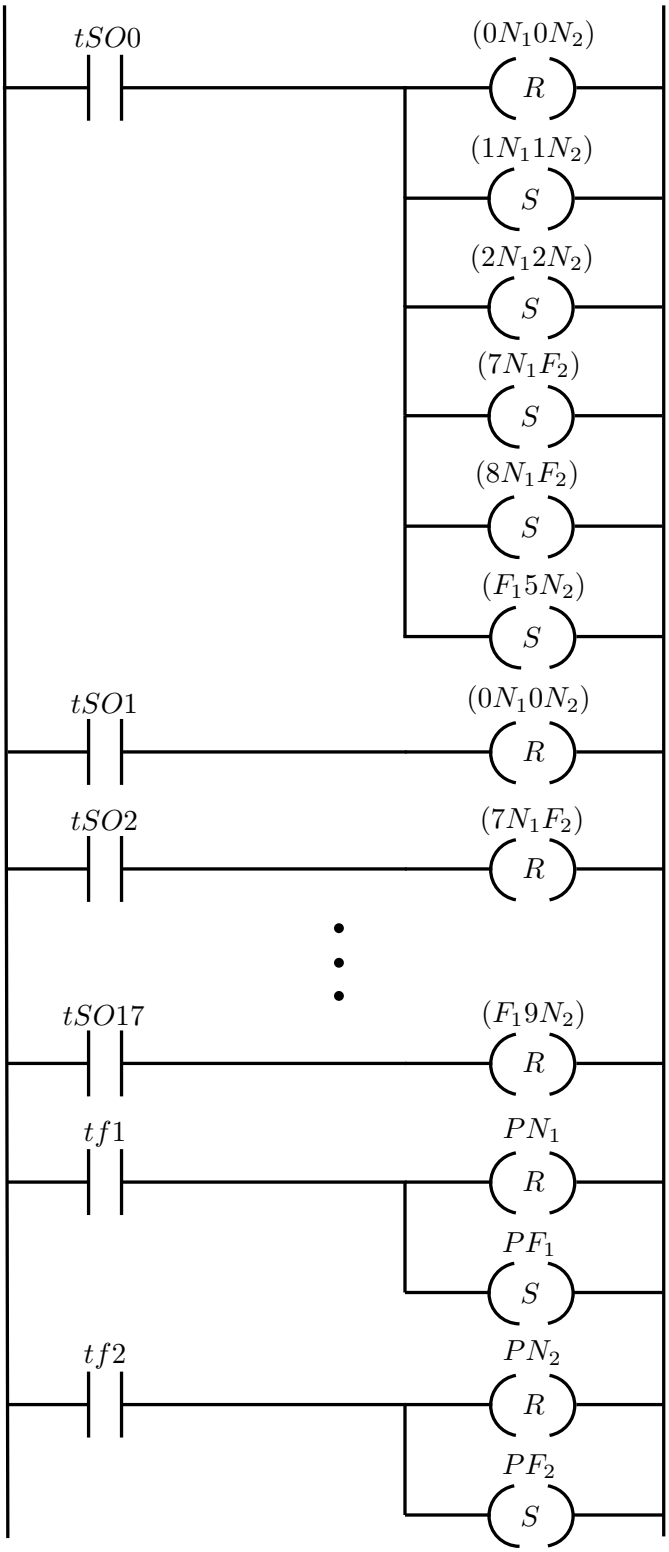


Figura 5.12: Módulo da dinâmica para a rede de Petri diagnosticadora da figura 3.14.

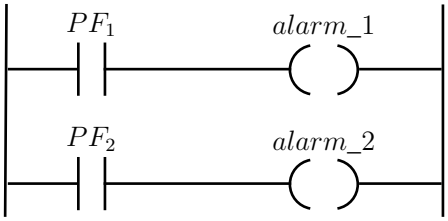


Figura 5.13: Módulo dos alarmes para a rede de Petri diagnosticadora da figura 3.14.

5.2.6 Módulo dos alarmes

O número de linhas no módulo dos alarmes é igual ao número de tipos de falha na rede de Petri diagnosticadora. Cada linha do módulo de alarmes é formada por um contato NA associado ao lugar PF_k, que indica que uma falha do tipo F_k ocorreu, em série com uma bobina simples associada a uma variável usada para o alarme de cada tipo de falha. Um conjunto de ações pode ser definido para cada tipo de falha dependendo do seu grau de importância. O módulo dos alarmes para o exemplo da figura 3.14 é apresentado na figura 5.13. Note que o diagrama ladder do módulo dos alarmes tem apenas duas linhas, já que a rede de Petri diagnosticadora tem apenas dois tipos de falha.

5.3 Organização do diagrama ladder

Os cinco módulos devem ser implementados na mesma ordem em que foram apresentados neste trabalho, ou seja: (i) módulo de inicialização; (ii) módulo de eventos externos; (iii) módulo das condições para o disparo das transições; (iv) módulo da dinâmica da rede de Petri; (v) módulo dos alarmes.

A ordem dos módulos no diagrama ladder evita o efeito avalanche porque as condições para o disparo de todas as transições são verificadas primeiro no módulo das condições para o disparo das transições e só então a evolução das fichas é realizada no módulo da dinâmica da rede de Petri. Essa organização da implementação garante que cada marcação da rede de Petri diagnosticadora se mantenha sem alterações por pelo menos um ciclo de varredura em sua implementação ladder.

Portanto, apenas transições habilitadas podem disparar quando o evento associado ocorrer.

5.4 Complexidade do diagrama ladder

De acordo com o método de conversão da rede de Petri diagnosticadora para diagrama ladder proposto neste trabalho, a análise da complexidade do diagrama ladder pode ser divida em cinco partes, cada uma referente a um módulo apresentado.

- Módulo de inicialização: o diagrama ladder do módulo de inicialização possui apenas uma linha usada para realizar a marcação inicial.

- Módulo de eventos externos: Supondo que existam l eventos externos distintos associados à borda de subida ou descida de sinais de sensores, o módulo de eventos possui l linhas, já que cada linha é usada para detectar a ocorrência de um evento externo.

- Módulo das condições para o disparo das transições: como apresentado na seção 5.2.4, o módulo das condições para o disparo das transições possui $|T_D|$ linhas, já que cada linha descreve as condições para o disparo de cada transição da rede de Petri diagnosticadora.

- Módulo da dinâmica da rede de Petri: Se o problema de implementação que ocorre quando um lugar recebe e perde uma ficha simultaneamente após o disparo de duas transições diferentes não for considerado, o módulo possui $|T_D|$ linhas que realizam a nova marcação da rede de Petri diagnosticadora em consequência do disparo das transições. Mas, devido à possibilidade de, no pior caso, esse problema existir para todos os lugares da rede de Petri diagnosticadora, uma linha para cada transição deve ser adicionada para garantir o correto funcionamento do módulo. Dessa forma, o módulo da dinâmica da rede de Petri possui, no pior caso, $2 \times |T_D|$ linhas.

Assim, o número máximo de linhas no diagrama ladder obtido a partir desse método é $(1 + l + 3|T_D| + r)$.

O método de conversão proposto neste trabalho pode levar a um diagrama ladder mais complexo do que outros métodos propostos na literatura. Entretanto, o método leva a um diagrama ladder que simula o comportamento da rede de Petri diagnosticadora e evita problemas de implementação, como o efeito avalanche e o problema de um lugar receber e perder uma ficha simultaneamente.

Capítulo 6

Conclusão

Neste trabalho, uma abordagem utilizando-se redes de Petri para o diagnóstico online de falhas em sistemas a eventos discretos modelados por autômatos finitos foi apresentada. Essa abordagem leva à construção de uma rede de Petri diagnosticadora que pode ser usada para detecção e isolamento de falhas online e requer, em geral, menos memória computacional do que outros métodos de diagnóstico de falhas propostos na literatura.

Além disso, foram propostos métodos para conversão da rede de Petri diagnosticadora em SFC e em diagrama ladder para implementação em um CLP. A implementação pode ser realizada no mesmo CLP usado para o controle do sistema, permitindo a redução do equipamento usado para o diagnóstico. As técnicas de conversão aplicadas levam a códigos que simulam o comportamento da rede de Petri e permitem a implementação da dinâmica da rede de Petri, evitando o efeito avalanche e o problema da remoção e adição simultânea de uma ficha a um lugar após o disparo de duas transições diferentes.

Como trabalhos futuros são propostos: (i) o estudo da implementação da rede de Petri diagnosticadora em sistemas sujeitos à observação simultânea de eventos; (ii) o estudo e a implementação de uma rede de Petri diagnosticadora online modular.

O fenômeno de observação simultânea de eventos ocorre quando mais de um evento é observado durante um único ciclo de varredura do CLP e pode ocasionar a perda de diagnosticabilidade do sistema. Além disso, mesmo que o sistema continue

diagnosticável, mudanças nos métodos de conversão da rede de Petri diagnosticadora para as linguagens de programação apresentadas devem ser feitas para garantir o funcionamento correto do diagnosticador em sistemas sujeitos a esse fenômeno.

O estudo e a implementação de uma rede de Petri diagnosticadora online modular são motivados pelo fato de que os autômatos que são usados para modelar sistemas reais são formados por uma composição paralela dos autômatos que modelam seus componentes. Ao realizar essa composição paralela, a cardinalidade do espaço de estados do autômato resultante, no pior caso, cresce exponencialmente com o número de componentes do sistema. Para sistemas com um grande número de componentes, isso pode levar a um modelo muito complexo e, consequentemente, a um diagnosticador computacionalmente complexo. Trabalhando-se de maneira modular, o diagnosticador não é construído a partir do modelo completo do sistema, mas sim a partir dos modelos dos componentes, reduzindo, assim, sua complexidade computacional.

Referências Bibliográficas

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

[2] HOPCROFT, J., MOTWANI, R., ULLMAN, J. *Introduction to Automata Theory, Languages, and Computation*. Prentice Hall, 2006.

[3] LAWSON, M. *Finite Automata*. Chapman and Hall/CRC, 2003.

[4] DAVID, R., ALLA, H. *Discrete, Continuous and Hybrid Petri Nets*. Springer, 2005.

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

[6] 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.

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

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

[9] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C. "Generalized robust diagnosability of discrete event systems". In: *18th IFAC World Congress*, pp. 8737–8742, Milano, Italy, 2011.

[10] CARVALHO, L. K., BASILIO, J. C., MOREIRA, M. V. "Robust diagnosis of discrete-event systems against intermittent loss of observations", *Automatica*, v. 48, n. 9, pp. 2068–2078, 2012.

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

[12] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C., et al. "Robust diagnosis of discrete-event systems against permanent loss of observations", *Automatica*, v. 49, n. 1, pp. 223–231, 2013.

[13] 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*, 2012. Aceito para publicação.

[14] CABASINO, M., GIUA, A., LAFORTUNE, S., et al. "A New Approach for Diagnosability Analysis of Petri Nets using Verifiers Nets", *IEEE Transactions on Automatic Control*, v. 57, n. 12, pp. 3104–3117, 2012.

[15] ZAD, S., KWONG, R., WONHAM, W. "Fault diagnosis in discrete-event systems: framework and model reduction", *IEEE Trans. on Automatic Control*, v. 48, n. 7, pp. 1199–1212, 2003.

[16] GIUA, A., SEATZU, C., CORONA, D. "Marking estimation of Petri nets with silent transitions", *IEEE Transactions on Automatic Control*, v. 52, n. 9, pp. 1695–1699, 2007.

[17] CABASINO, M. P., GIUA, A., SEATZU, C. "Fault detection for discrete event systems using Petri nets with unobservable transitions", *Automatica*, v. 46, pp. 1531–1539, 2010.

[18] CABASINO, M. P., GIUA, A., SEATZU, C. "Diagnosis using labeled Petri nets with silent or undistinguishable fault events", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 43, n. 2, pp. 345–355, 2013.

[19] CABASINO, M. P., GIUA, A., PAOLI, A., et al. "Decentralized Diagnosis of Discrete Event Systems using labeled Petri nets", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 43, n. 6, pp. 1477–1485, 2013.

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

[21] RAMIREZ-TREVINO, A., RUIZ-BELTRAN, E., RIVERA-RANGEL, I., et al. "Online fault diagnosis of discrete event systems. A Petri net-based appro-

ach", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 1, pp. 31–39, 2007.

[22] ISO/IEC. *International standard IEC 61131-3*. ISO/IEC, 2001.

[23] LUCA, F., MASSIMO, A., ALESSIO, D. "A methodology for fault isolation and identification in automated equipments". In: *9th IEEE International Conference on Industrial Informatics*, pp. 157–162, Lisbon, Portugal, 2011.

[24] UZAM, M., JONES, A. H., AJLOUNI, N. "Conversion of Petri Nets Controllers for Manufacturing Systems into Ladder Logic Diagrams". In: *IEEE Conference on Emerging Technologies and Factory Automation*, pp. 649–655, 1996.

[25] JONES, A. H., UZAM, M., AJLOUNI, N. "Design of discrete event control systems for programmable logic controllers using T-timed Petri nets". In: *IEEE Int. Symp. Computer-Aided Control System Design*, pp. 212–217, 1996.

[26] UZAM, M., JONES, A. H. "Discrete Event Control System Design Using Automation Petri Nets and their Ladder Diagram Implementation", *Int J Adv Manuf Technol*, v. 14, pp. 716–728, 1998.

[27] JIMENEZ, I., LOPEZ, E., RAMIREZ, A. "Synthesis of Ladder diagrams from Petri nets controller models". In: *2001 IEEE International Symposium on Intelligent Control*, pp. 225–230, Mexico City, Mexico, 2001.

[28] PENG, S. S., ZHOU, M. C. "Ladder diagram and Petri-net-based discrete-event control design methods", *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, v. 34, pp. 523–531, 2004.

[29] UZAM, M. "A general technique for the PLC-based implementation of RW supervisors with time delay functions", *Int J Adv Manuf Technol*, v. 62, n. , pp. 687–704, 2012.

[30] MOREIRA, M. V., BASILIO, J. C. "Bridging the Gap Between Design and Implementation of Discrete-Event Controllers", *IEEE Transactions on Automation Science and Engineering*, v. 11, n. 1, pp. 48–65, 2014.

[31] FABIAN, M., HELLGREN, A. "PLC-based implementation of supervisory control for discrete event systems". In: *37th IEEE Conference on Decision and Control*, pp. 3305–3310, Tampa, Florida USA, 1998.

[32] HELLGREN, A., FABIAN, M., LENNARTSON, B. "On the execution of sequential fucntion charts", *Control Engineering Practice*, v. 13, pp. 1283–1293, 2005.

[33] BASILE, F., CHIACCHIO, P. "On the implementation of supervised control of discrete event systems", *IEEE Transactions on Control Systems Technology*, v. 15, n. 4, pp. 725–739, 2007.

[34] BASILE, F., CHIACCHIO, P., GERBASIO, D. "On the implementation of industrial automation systems based on PLC", *IEEE Transactions on Automation Science and Engineering*, v. 10, n. 4, pp. 990–1003, 2013.

[35] MOREIRA, M. V., CABRAL, F. G., DIENE, O. "Diagnosticador rede de Petri para um SED modelado por um autômato finito". In: *XIX Congresso Brasileiro de Automática, CBA 2012*, pp. 3723–3730, Campina Grande - PB, Brasil, 2012. ISBN: 978-85-8001-069-5.

[36] MOREIRA, M. V., CABRAL, F. G., DIENE, O. "Petri net diagnoser for DES modeled by finite state automata". In: *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pp. 6742–6748, Maui, HI, USA, 2012. doi: 10.1109/CDC.2012.6426235.

[37] MOREIRA, M. V., CABRAL, F. G., DIENE, O., et al. "Implementação de uma rede de Petri para diagnose online de falhas em controladores lógicos programáveis". In: *XI Simpósio Brasileiro de Automação Inteligente, SBAI 2013*, Fortaleza - CE, Brasil, 2013.

[38] CABRAL, F. G., MOREIRA, M. V., DIENE, O., et al. "A Petri net diagnoser for discrete event systems modeled by finite state automata", *IEEE Transactions on Automatic Control*, 2014. Submetido para publicação.

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

[40] ALAYAN, H., NEWCOMB, R. W. "Binary Petri-Net Relationships", *IEEE transactions on circuits and systems*, v. CAS-34, pp. 565–568, 1987.

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

[42] MIYAGI, P. *Controle Programável - Fundamentos do Controle de Sistemas a Eventos Discretos*. São Paulo, SP, Edgard Blücher, 2007.

[43] FRANCHI, C. M., CAMARGO, V. L. A. *Controladores Lógicos Programáveis - Sistemas Discretos*. 1 ed. São Paulo, Editora Érica, 2008.

[44] MORAES, C. C., CASTRUCCI, P. L. *Engenharia de Automação Industrial*. 2 ed. Rio de Janeiro, Livros Técnicos e Científicos Editora S.A., 2007.

[45] ISO/IEC. *International standard IEC 60848*. ISO/IEC, 2002.

[46] MOREIRA, M. V., BOTELHO, D. S., BASILIO, J. C. "Ladder Diagram Implementation of Control Interpreted Petri Nets: a State Equation Approach". In: *4th IFAC Workshop on Discrete-Event System Design*, pp. 85–90, Gandia Beach, Spain, 2009.