Universidade Federal
do Rio de Janeiro
Escola Politécnica

UMA REDE DE PETRI DIAGNOSTICADORA PARA SISTEMAS A EVENTOS DISCRETOS MODELADOS POR AUTÔMATOS FINITOS

Felipe Gomes de Oliveira Cabral

Projeto de Graduação apresentado ao Curso de Engenharia Elétrica da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Marcos Vicente de Brito Moreira

Rio de Janeiro
Março de 2013

UMA REDE DE PETRI DIAGNOSTICADORA PARA SISTEMAS A EVENTOS DISCRETOS MODELADOS POR AUTÔMATOS FINITOS

Felipe Gomes de Oliveira Cabral

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELÉTRICA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRICISTA.

Examinado por:

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

Prof. João Carlos dos Santos Basílio, Ph.D.

Prof. Oumar Diene, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2013

Cabral, Felipe Gomes de Oliveira

Uma Rede de Petri Diagnosticadora para Sistemas a Eventos Discretos Modelados por Autômatos Finitos/Felipe Gomes de Oliveira Cabral. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2013.

XII, 77 p.: il.; 29, 7cm.

Orientador: Marcos Vicente de Brito Moreira

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia Elétrica, 2013.

Referências Bibliográficas: p. 75 – 77.

1. Sistemas a eventos discretos. 2. Redes de Petri. 3. Autômatos. 4. Diagnose de falha. 5. Controladores lógicos programáveis. I. Moreira, Marcos Vicente de Brito. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso 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 namorada Julia Rodrigues Chagas pelo companheirismo e paciência durante a elaboração deste trabalho.

Finalmente, agradeço ao meu professor e orientador, Marcos Vicente de Brito Moreira, por todas as horas gastas de aconselhamento e orientação.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

Uma Rede de Petri Diagnosticadora para Sistemas a Eventos Discretos Modelados por Autômatos Finitos

Felipe Gomes de Oliveira Cabral

Março/2013

Orientador: Marcos Vicente de Brito Moreira

Curso: Engenharia Elétrica

Este trabalho consiste no desenvolvimento de uma rede de Petri diagnosticadora para sistemas a eventos discretos modelados por autômatos. O método de diagnose proposto requer, em geral, menos memória do que outros métodos existentes na literatura. Além disso, métodos para a conversão da rede de Petri diagnosticadora em SFC e em diagrama ladder para a implementação em um controlador lógico programável (CLP) são apresentados. Os métodos de conversão geram códigos de programação que preservam a estrutura e representam a evolução das fichas na rede de Petri diagnosticadora.

Palavras-chave: Sistemas a eventos discretos, Redes de Petri, Autômatos, Diagnose de falha, Controladores lógicos programáveis.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

PETRI NET DIAGNOSER FOR DISCRETE EVENT SYSTEMS MODELED BY FINITE STATE AUTOMATA

Felipe Gomes de Oliveira Cabral

March/2013

Advisor: Marcos Vicente de Brito Moreira

Course: Electrical Engineering

In this work, we propose a Petri net diagnoser for online diagnosis of discrete event systems modeled by finite state automata. The diagnosis method 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. The conversion methods lead to PLC programming codes that preserve the structure and represent the evolution of the tokens of the Petri net diagnoser.

Keywords: Discrete event systems, Petri nets, Automata, Fault diagnosis, Programmable Logic Controllers.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Cada vez mais sistemas capazes de realizar tarefas automaticamente são desenvolvidos. Esses sistemas 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. Sistemas desse tipo são regidos, em geral, por eventos. Eventos são alterações do sistema ou do ambiente externo que podem causar alguma mudança no estado do sistema. 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 funcionário.

Sistemas que são regidos por eventos são denominados sistemas a eventos discretos (SED) [1], em que os eventos são modelados como uma ocorrência instantânea. A natureza discreta de SEDs 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 levar em consideração a natureza discreta desses sistemas.

Por essa razão existe um grande esforço de pesquisa voltado para a criação de modelos matemáticos adequados para representar SEDs. Dentre esses destacam-se os autômatos e as redes de Petri [1, 2]. O primeiro representa SEDs como um grafo orientado em que os vértices representam os estados e os arcos representam as transições ocasionadas pela ocorrência de eventos. O segundo representa SEDs de forma diferente, baseado em um grafo bipartido ponderado, cujos estados possuem

uma natureza distribuída no grafo. Dessa forma, redes de Petri são, em geral, uma maneira de representação mais vantajosa do que autômatos para sistemas com grande número de estados.

Como todos os sistemas, os SEDs 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, é necessário um mecanismo capaz de detectar e isolar falhas em sistemas de automação, chamado de diagnosticador. Muitos trabalhos têm sido publicados na literatura com esse objetivo [3–10].

Em [3, 4] é apresentada uma abordagem para diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos. O método de diagnose proposto por SAMPATH *et al.* [3, 4] consiste dos seguintes passos: (i) Cálculo do autômato G_ℓ, obtido a partir da composição paralela entre o autômato do sistema G e o autômato rotulador A_ℓ, cujos estados são dados por (x, ℓ) em que x é um estado de G e $\ell \in \{Y, N\}$; (ii) obtenção de um autômato diagnosticador G_{diag} através do cálculo do observador de estados do autômato rotulado G_ℓ; (iii) identificação dos eventos de falha baseados no estado de G_{diag} alcançados após a observação de uma sequência de eventos executada pelo sistema.

O diagnosticador proposto por SAMPATH *et al.* [3, 4] pode ser usado para detecção e isolamento de eventos de falha online e para a verificação off-line 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 G_{diag} cresce exponencialmente com a cardinalidade do espaço de estados do modelo do sistema G [3–5, 11].

Em [5], um método para a diagnose online que evita a construção e o armazenamento completo de G_{diag} é proposto. Para isso, um autômato não determinístico G_ℓ^{nd} é calculado ao substituir cada transição de G_ℓ associada com um evento não observável por uma transição ε. Nesse método, apenas o estado atual do diagnos-

ticador G_{diag} e do autômato G_{ℓ}^{nd} precisam ser armazenados para a diagnose online. Após a ocorrência de um evento observável, o próximo estado de G_{diag} pode ser obtido online a partir do estado atual de G_{diag} e a partir do autômato G_{ℓ}^{nd} em tempo polinomial.

Diversas outras técnicas de diagnose que usam autômatos finitos ou redes de Petri para modelar tanto o sistema quanto o diagnosticador foram propostas na literatura [11–15]. Apesar de diversos trabalhos abordarem métodos de obtenção de diagnosticadores, 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 [16]: (*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 diagnose 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 do equipamento usado para a diagnose. Note que, nesse caso, todos os eventos de comando se tornam observáveis para o diagnosticador, sem a necessidade de sensores adicionais ou barramentos de comunicação.

Em [17], uma plataforma de CLP particular, chamada *softPLC Orchestra*, é usada para diagnose. 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.* [17] 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 [18–23]. Embora esses métodos de conversão tenham sido aplicados com sucesso ao controle de sistemas automatizados, alguns problemas de implementação de controladores não foram abordados. Em [24, 25], um importante problema associado à implementação de códigos de controle modelados por autômatos finitos e SFCs, chamado de efeito avalanche, é abordado e um método que evita o efeito avalanche é proposto. A principal desvantagem da solução proposta por FABIAN e HELLGREN [24] é a falta de métodos formais para lidar com redes de Petri complexas. Em [26], um método geral para a conversão de redes de Petri interpretadas para controle em diagramas ladder é proposto. O método leva a um diagrama ladder que simula o comportamento da rede de Petri e evita o efeito avalanche.

Além do efeito avalanche, um problema diferente, também associado com a implementação de redes de Petri em diagramas ladder, ocorre quando um lugar recebe e perde uma ficha após o disparo de duas transições distintas. Dependendo da forma com que a rede de Petri é implementada em um diagrama ladder, a marcação resultante dos lugares pode estar errada, levando a uma representação incorreta da dinâmica da rede de Petri.

Este trabalho apresenta uma abordagem por redes de Petri para a diagnose 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 dos autômatos G_{N_k}, para $k = 1, \ldots, r$, em que o 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 da diagnose online em comparação com o método proposto por QIU e KUMAR [5], que usa o comportamento normal e de falha do sistema.

A técnica de diagnose 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. 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 [27, 28]. A rede de Petri diagnosticadora provê uma estrutura para o procedimento de diagnose online que facilita a implementação do código do diagnosticador em um computador.

Neste trabalho, 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, são apresentados. 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 é necessária para a implementação em CLPs que não suportam a programação na linguagem SFC. O método de conversão, baseado em [26], evita o efeito avalanche e gera um diagrama ladder bem estruturado. 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 também é abordado e uma solução simples para esse problema é apresentada.

Este trabalho está organizado da seguinte forma: no capítulo 2 são apresentados os fundamentos de sistemas a eventos discretos. Os fundamentos de CLP são apresentados no capítulo 3. No capítulo 4 a rede de Petri diagnosticadora é proposta e, no capítulo 5, as técnicas de conversão da rede de Petri diagnosticadora em diagramas SFC e ladder são apresentadas. Finalmente, as conclusões e trabalhos futuros são mostrados 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 sobre 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 várias condições que são satisfeitas (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. Então, sequências de eventos formam palavras e o conjunto de

todas as sequências possíveis de um sistema é chamado de linguagem. 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.

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 serão considerados dois tipos de formalismos: autômatos e redes de Petri.

2.1 Linguagens

Uma linguagem é um conjunto de sequências de eventos 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 conjunto do *alfabeto*. ε representa a sequência vazia. O símbolo e será usado para representar um evento genérico. Se s é uma sequência, seu comprimento será denotado por $|s|$. Por convenção, o comprimento da sequência vazia ε é zero.

Definição 1 *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.* □

Será denotado por Σ^* o conjunto formado por todas as sequências finitas de elementos de Σ, incluindo a sequência vazia ε. Uma linguagem definida em um

conjunto de eventos Σ é um subconjunto de Σ^* e, em particular, $\emptyset$, Σ e Σ^* são linguagens.

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

2.1.2 Operações com linguagens

Como linguagens são conjuntos, todas as operações usualmente usadas em conjunto são também definidas para linguagens. Além dessas operações, são definidas também as seguintes:

- Concatenação: Seja L_a, $L_b \subseteq \Sigma^*$, então

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

- Prefixo fechamento: Seja $L \subseteq \Sigma^*$, então

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

- Fecho de Kleene: Seja $L \subseteq \Sigma^*$, então

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

- Projeções:

$$P : \Sigma_l^* \rightarrow \Sigma_s^*, \Sigma_s \subset \Sigma_l,$$

em que:

$$P(\varepsilon) := \varepsilon,$$

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

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

De maneira semelhante, é possível definir a operação inversa da seguinte forma:

$$P^{-1} : \Sigma_s^* \to 2^{\Sigma_l^*},$$

em que:

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

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 e é definido formalmente como uma sêxtupla, como pode ser visto na definição 2.

Definição 2 *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. □

A maneira mais simples de representar um autômato é na forma 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 pelos arcos, são as transições entre os estados. As transições são rotuladas por eventos em Σ para representar o evento responsável pela transição de estados.

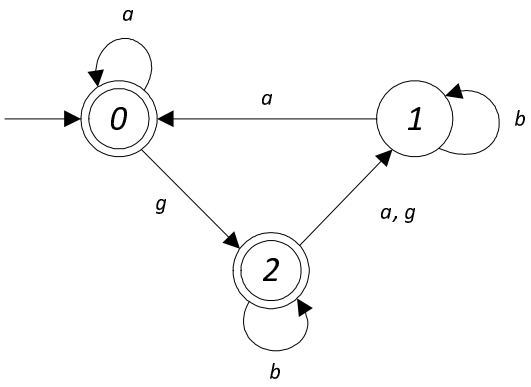


Figura 2.1: Diagrama de transição de estados do autômato G do exemplo 1.

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 1 *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 $\Sigma = \{a, b, g\}$. A função de transição de estados de G é definida da seguinte forma: $f(0, a) = 0$; $f(0, g) = 2$; $f(1, a) = 0$; $f(1, b) = 1$; $f(2, b) = 2$; $f(2, a) = f(2, g) = 1$. Assim, a função de eventos ativos de cada estado possui os seguintes resultados: $\Gamma(0) = \{a, g\}$; $\Gamma(1) = \{a, b\}$; $\Gamma(2) = \{a, b, g\}$. Por fim, o estado inicial de G é $q_0 = 0$ e o conjunto de estados marcados é $Q_m = \{0, 2\}$.*

As linguagens gerada e marcada por um autômato são descritas de acordo com a definição 3.

Definição 3 *A linguagem gerada por um autômato G é dada por:*

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

e a linguagem marcada por G é dada por:

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

□

É importante ressaltar que na definição 3 é suposto que a função de transição f é estendida, ou seja, $f : Q \times \Sigma^* \to Q$. 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 todos os caminhos que podem ser seguidos 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 necessariamente é 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 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.

Parte Acessível

A parte acessível é uma operação que elimina todos os estados de G que não são alcançáveis a partir do estado inicial q_0 e suas transições relacionadas. Formalmente: $Ac(G) := (Q_{ac}, \Sigma, f_{ac}, q_0, Q_{ac,m})$ em que:

$$\begin{aligned} Q_{ac} &= \{q \in Q : (\exists s \in \Sigma^*)[f(q_0, s) = q]\}, \\ Q_{ac,m} &= Q_m \cap Q_{ac}, \\ f_{ac} &: Q_{ac} \times \Sigma^* \to Q_{ac}. \end{aligned}$$

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 $\mathcal{L}_m(G)$.

Parte Coacessível

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 correspondentes transições, que não são coacessíveis. De maneira formal: $CoAc(G) := (Q_{coac}, \Sigma, f_{coac}, q_{0,coac}, Q_m)$ em que:

$$\begin{aligned} Q_{coac} &= \{q \in Q : (\exists s \in \Sigma^*)[f(q, s) \in Q_m]\}, \\ q_{0,coac} &:= \begin{cases} q_0 & \text{se } q_0 \in Q_{coac} \\ \text{indefinido} & \text{caso contrário,} \end{cases} \\ f_{coac} &: Q_{coac} \times \Sigma^* \to Q_{coac}. \end{aligned}$$

Note que $\mathcal{L}(CoAc(G)) \subseteq \mathcal{L}(G)$, contudo $\mathcal{L}_m(CoAc(G)) = \mathcal{L}_m(G)$.

Operação Trim

Um autômato que é tanto acessível quanto coacessível é chamado de Trim. A operação Trim pode ser definida da seguinte forma:

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

Operação produto

A operação produto é chamada de composição completamente síncrona e é denotada por $\times$. O produto entre dois autômatos G_1 e G_2 resulta no seguinte autômato:

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

em que:

$$\begin{aligned} f((q_1, q_2), e) &:= \begin{cases} (f_1(q_1, e), f_2(q_2, e)), & \text{se } e \in \Gamma_1(q_1) \cap \Gamma_2(q_2) \\ \text{indefinido}, & \text{caso contrário}, \end{cases} \\ \Gamma_{1\times 2}(q_1, q_2) &= \Gamma_1(q_1) \cap \Gamma_2(q_2). \end{aligned}$$

De acordo com a definição de composição produto, 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 o estado atual de G_1 e o segundo componente é o estado atual de G_2. Além disso, a linguagem gerada e a linguagem marcada por $G_1 \times G_2$ são:

$$\begin{aligned} \mathcal{L}(G_1 \times G_2) &= \mathcal{L}(G_1) \cap \mathcal{L}(G_2), \\ \mathcal{L}_m(G_1 \times G_2) &= \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2). \end{aligned}$$

Composição paralela

A composição paralela também é chamada de composição síncrona e 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 composição paralela entre dois autômatos G_1 e G_2 resulta no seguinte autômato:

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

em que:

$$f((q_1, q_2), e) := \begin{cases} (f_1(q_1, e), f_2(q_2, e)), & \text{se } e \in \Gamma_1(q_1) \cap \Gamma_2(q_2) \\ (f_1(q_1, e), q_2), & \text{se } e \in \Gamma_1(q_1) \setminus \Sigma_2 \\ (q_1, f_2(q_2, e)), & \text{se } e \in \Gamma_2(q_2) \setminus \Sigma_1 \\ \text{indefinido}, & \text{caso contrário,} \end{cases}$$

$$\Gamma_{1||2}(q_1, q_2) = [\Gamma_1(q_1) \cap \Gamma_2(q_2)] \cup [\Gamma_1(q_1) \setminus \Sigma_2] \cup [\Gamma_2(q_2) \setminus \Sigma_1].$$

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 for possível. Assim, a composição paralela sincroniza apenas os eventos que são comuns aos dois autômatos.

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:

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

Com base nessas projeções, as linguagens resultantes da composição paralela podem ser caracterizadas como:

$$\begin{aligned} \mathcal{L}(G_1||G_2) &= P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)], \\ \mathcal{L}_m(G_1||G_2) &= P_1^{-1}[\mathcal{L}_m(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)]. \end{aligned}$$

2.2.2 Autômatos com observação parcial de eventos

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 será 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. 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 sequencia 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 preciso definir o alcance não observável, denotado por $UR(q)$, como:

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

Essa definição é estendida a um subconjunto de estados $B \subseteq Q$ da seguinte forma:

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

Assim, o alcance não observável gera um conjunto de estados que corresponde

à estimativa de estado do sistema após a observação de um evento. Em outras palavras, suponha que um sistema tenha gerado uma sequência $s \in \Sigma^*$ de eventos e alcançado um estado $v \in Q$, 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.

Com base nas equações 2.1 e 2.2 é possível construir o observador de G de acordo com o algoritmo 1 [1].

Algoritmo 1 *Seja* $G = (Q, \Sigma, f, q_0, Q_m)$ *um autômato determinístico, sendo* $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. *Então,* $Obs(G) = (Q_{obs}, \Sigma_o, f_{obs}, q_{0,obs}, Q_{m,obs})$ *e é construído da seguinte forma:*

- *Passo 1: Defina* $q_{0,obs} := UR(q_0)$. *Faça* $Q_{obs} = \{q_{0,obs}\}$.
- *Passo 2: Para cada* $B \in Q_{obs}$ *e* $e \in \Sigma_o$, *defina:*

$$f_{obs}(B, e) := UR(\{q \in Q : (\exists q_e \in B)(q \in f(q_e, e))\})$$

  sempre que $f(q_e, e)$ *é definida para algum* $Q_e \in B$. *Nesse caso, adicione o estado* $f_{obs}(B, e)$ *a* Q_{obs}. *Se* $f(q_e, e)$ *não é definida para nenhum* $q_e \in B$, *então* $f_{obs}(B, e)$ *não é definida.*

- *Passo 3: Repita o passo 2 até que toda a parte acessível de* $Obs(G)$ *tenha sido construída.*
- *Passo 4: Defina* $Q_{m,obs}$ *da seguinte forma:* $Q_{m,obs} := \{B \in Q_{obs} : B \cap Q_m \neq \emptyset\}$.

□

É importante ressaltar que as linguagens gerada e marcada pelo autômato $Obs(G)$ são: $\mathcal{L}(Obs(G)) = P[\mathcal{L}(G)]$ e $\mathcal{L}_m(Obs(G)) = P[\mathcal{L}_m(G)]$, sendo $P : \Sigma^* \to \Sigma_o^*$. O exemplo 2 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.

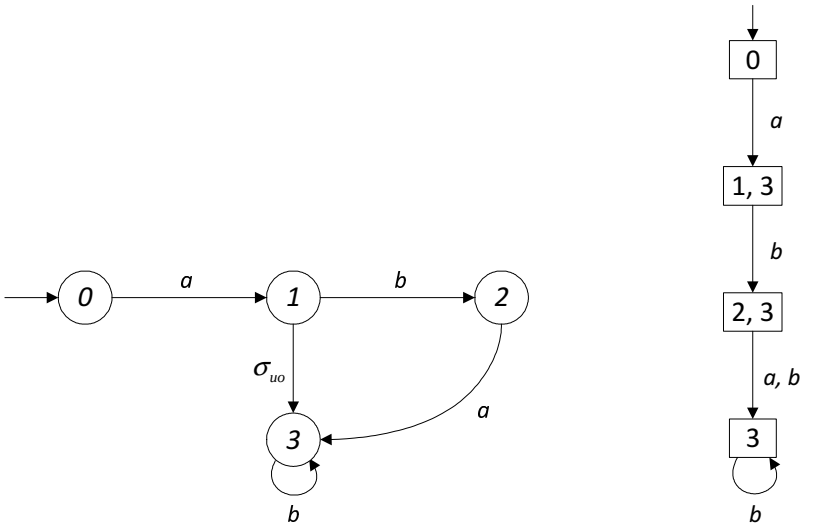


Figura 2.2: Diagrama de transição de estados do autômato G com eventos não observáveis (a), e autômato observador de G, $Obs(G)$, que fornece uma estimativa dos estados alcançados de G após a observação de uma sequência de eventos gerada pelo sistema (b).

Exemplo 2 *Seja G o autômato cujo diagrama de transição de estados pode ser visto na figura 2.2(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 2.2(b). Considere que o sistema tenha executado a sequência de eventos $t = a\sigma_{uo}b$, a sequência observada será $P(t) = ab$, para $P : \Sigma^* \rightarrow \Sigma_o^*$. Ao acompanhar a sequência $P(t)$ em $Obs(G)$, é alcançado o estado $\{2, 3\}$, que corresponde à estimativa de estado de G após a observação dessa sequência.*

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

2.3 Redes de Petri

Uma rede de Petri é um outro 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 manipular eventos de acordo com regras 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 mais complexos.

2.3.1 Fundamentos básicos das redes de Petri

No formalismo das redes de Petri, os eventos estão associados às transições e, para que determinada transição ocorra, é necessário que um conjunto de condições seja satisfeita e que o evento que a rotula ocorra. As condições estão relacionadas com fichas colocadas em determinados lugares da rede. Em relação às transições, os lugares podem ser de entrada ou de saída das transições.

Grafo de uma rede de Petri

Lugares, transições e as relações entre eles formam o conjunto de informações básicas capaz de definir um grafo de uma rede de Petri. O grafo 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, as redes de Petri possuem um grafo bipartido.

A definição formal de um grafo de uma rede de Petri é apresentada a seguir:

Definição 4 *Um grafo 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 $Pre(p_i, t_j) > 0$ ($Post(t_j, p_i) > 0$).

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 um grafo de uma rede de Petri pode ser visto a seguir.

Exemplo 3 *Seja um grafo de uma rede de Petri definido 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 2.3.*

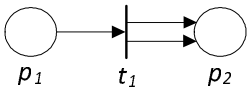


Figura 2.3: Grafo de uma rede de Petri do exemplo 3.

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.

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 $\underline{x} = [x(p_1)x(p_2)...x(p_n)]^T$, 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 nos leva à seguinte definição de uma rede de Petri.

Definição 5 *Uma rede de Petri $\mathcal{N}$ é uma quíntupla $\mathcal{N} = (P, T, Pre, Post, x_0)$, em que, de acordo com a definição 4, $(P, T, Pre, Post)$ é o grafo de uma 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 4 ilustra uma rede de Petri marcada.

Exemplo 4 *Considere a rede de Petri do exemplo 3 mostrada na figura 2.3. 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 2.4.*

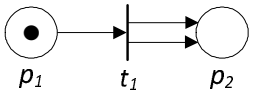


Figura 2.4: Rede de Petri com marcação inicial do exemplo 4.

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 6 *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

O mecanismo de transição de estado da rede de Petri é providenciado pela marcação das fichas ao longo da rede. Quando uma transição está habilitada, ela pode disparar. A evolução da marcação de uma rede de Petri ocorre de acordo com os disparos das transições. Se uma transição t_j, que está habilitada para uma marcação $\underline{x}$, dispara, então a rede de Petri alcança uma nova marcação $\overline{\underline{x}}$. A definição 7 formaliza a regra de marcação de uma rede de Petri.

Definição 7 *A evolução da marcação de uma rede de Petri é dada por:*

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

□

É importante ressaltar que, de acordo com a definição 7, o próximo estado de uma rede de Petri, ou seja seu próximo vetor de marcação $\overline{\underline{x}}$, que pode ser obtido pela equação 2.4, depende explicitamente dos lugares de entrada e saída de uma transição e dos pesos dos arcos que conectam esses lugares à transição.

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 de volta.

O exemplo 5 ilustra o processo de disparo de uma transição, mostrando a distribuição de fichas antes e depois do disparo.

Exemplo 5 *Considere a rede de Petri da figura 2.5(a). É possível notar que a transição t_1 está habilitada e, portanto, pode disparar. Suponha que t_1 dispare, então, 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, então duas fichas são colocadas no lugar p_2, resultando na rede de Petri com a marcação que é mostrada na figura 2.5(b).*

(a) (b)

Figura 2.5: Rede de Petri do exemplo 5 antes do disparo de t_1 (a), e rede de Petri do exemplo 5 após o disparo de t_1 com a nova marcação alcançada (b).

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 corresponda a um evento. Isso nos leva à definição 8.

Definição 8 *Uma rede de Petri rotulada* $\mathcal{N}$ *é uma sétupla* $\mathcal{N} = (P, T, Pre, Post, x_0, \Sigma, l)$, *em que,* $(P, T, Pre, Post, x_0)$ *é, de acordo com a definição 5, uma rede de Petri.* Σ *é o conjunto de eventos que são utilizados para a rotulação das transições e* $l : T \rightarrow 2^{\Sigma}$ *é a função de rotulação 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 dispare, é necessário que as condições relativas aos pesos dos arcos de entrada sejam satisfeitas e que o evento correspondente à transição ocorra.

2.3.2 Classes especiais de redes de Petri

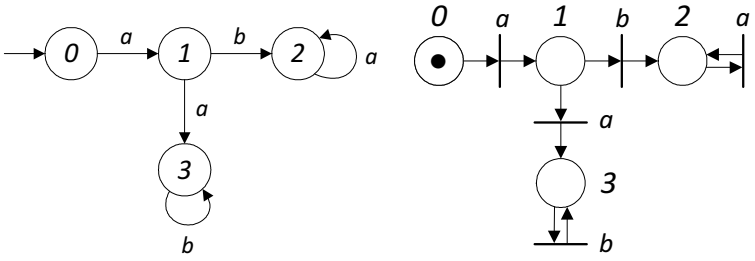
Redes de Petri máquina de estados

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. A característica mais importante de uma rede de Petri máquina de estados é que ela pode ser usada para representar um autômato de forma direta. Isso é feito substituindo

os estados do autômato por lugares e os arcos do autômato por transições rotuladas pelos mesmos eventos e com pesos dos arcos iguais a um. Além disso, adiciona-se uma ficha ao lugar referente ao estado inicial do autômato. Assim, a evolução da ficha na rede de Petri indicará a evolução dos estados do autômato correspondente.

O exemplo 6 ilustra a equivalência entre um autômato e uma rede de Petri máquina de estados.

Exemplo 6 *Considere o autômato G cujo diagrama de estados está representado na figura 2.6(a). A figura 2.6(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.*



(a) (b)

Figura 2.6: Autômato G do exemplo 6 (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 [29]. 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, por causa do 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.

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. Considere 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.

Seja $\mathcal{L}(G) = L$ a linguagem gerada pelo autômato G e G_{N_k} o subautômato de G que representa o comportamento normal do sistema com relação ao conjunto de eventos de falha Σ_{f_k}, ou seja, se $\mathcal{L}(G_{N_k}) = L_{N_k}$, então, L_{N_k} é a linguagem prefixo-fechada formada por todas as sequências de L que não contém nenhum evento de falha do conjunto Σ_{f_k}.

A definição de diagnosticabilidade é apresentada a seguir [3]:

Definição 9 *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 $P_o : \Sigma^* \to \Sigma_o^*$. 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 9, 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 observação 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 [30] é apresentado um algoritmo em tempo polinomial para identificar se uma linguagem L é diagnosticável. Esse algoritmo é baseado na construção de um autômato verificador determinístico e a verificação tem complexidade computacional inferior comparada com outros métodos propostos na literatura [5, 31].

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 Po e Π_f.

Capítulo 3

Controladores Lógicos Programáveis

Um 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 programar um série de funções específicas. Essas funções são usadas para controlar vários tipos de processo através de entradas e saídas digitais ou analógicas.

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.

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 lógico considerado. Nesse caso, a esteira se move se o sinal lógico tiver o valor 1 e

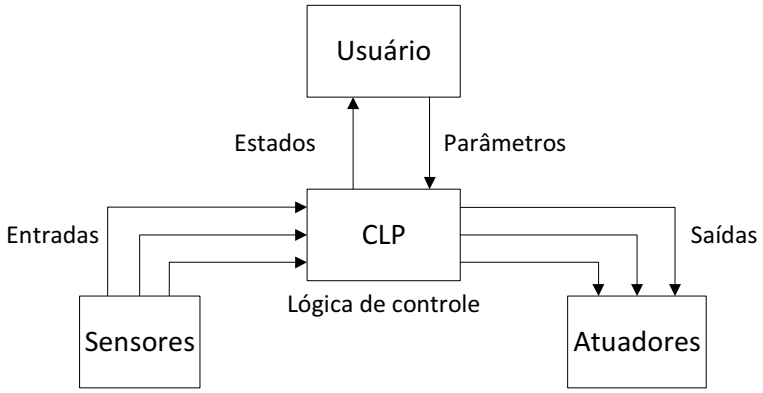


Figura 3.1: Esquema que ilustra a relação entre o CLP e os componentes de um sistema de automação.

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 controle 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 3.1 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 [32]. No modo de programação, o CLP fica aguardando ser configurado, programado, ou receber modificações de programas previamente configurados. Nesse modo, o CLP não executa nenhuma ação, apenas é 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 3.2 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 execução do código de controle, em que os valores de entrada armazenados são utili-

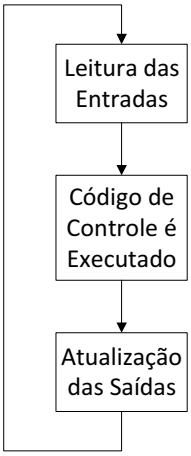


Figura 3.2: Esquema que ilustra a ordem de execução das etapas do ciclo de varredura.

zados 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 [16], 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 em um SFC e em um 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 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.

3.1 SFC

O SFC é uma linguagem baseada no Grafcet, que é uma forma de modelagem desenvolvida na França para representação de sistemas sequenciais [32], sendo 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.

3.1.1 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 3.3. 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 concêntricos, como pode ser visto na figura 3.4.

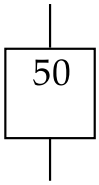


Figura 3.3: Ilustração de uma etapa simples de um código SFC.

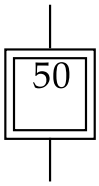


Figura 3.4: Ilustração de uma etapa inicial de um código SFC.

É possível associar ações às etapas, bastando, para isso, adicionar à direita da etapa um retângulo dividido em dois, em que, no primeiro é colocado o qualificador da ação e, no segundo, a ação associada. Os qualificadores são letras que indicam

como a ação deve ser executada. A figura 3.5 mostra uma etapa inicial com uma ação associada.

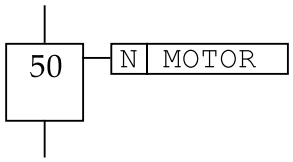


Figura 3.5: Exemplo de uma etapa com uma ação associada.

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 3.6 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.

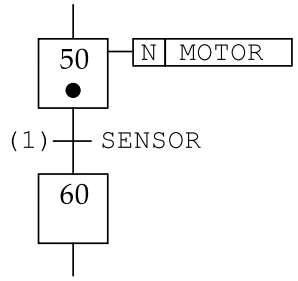


Figura 3.6: Ilustração de um código SFC simples composto de duas etapas e uma transição. A etapa 50 possui uma ação associada e a transição 1 possui uma condição associada. Como a etapa 50 está ativa, a transição será transposta assim que a variável $SENSOR$ passar para o valor lógico 1.

Sincronismo

Por fim, barras duplas horizontais são usadas para indicar a existência de uma sincronia de uma transição, como pode ser visto na figura 3.7.

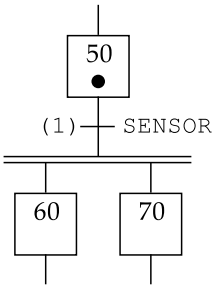


Figura 3.7: Representação de uma situação em que a transposição de uma transição ativa mais de uma etapa. Caso a transição 1 seja transposta, as etapas 60 e 70 serão ativadas simultaneamente. As barras duplas são usadas para representar a sincronia de duas ou mais sequências de etapas.

Na figura 3.7, a transição 1, quando transposta, ativa mais de uma etapa simultaneamente e isso é representado pelas barras duplas horizontais abaixo da transição. O mesmo símbolo é usado para representar a sincronia de dois ou mais trechos de um código SFC, ou seja, quando dois ou mais trechos de etapas estão em paralelo.

Quando há uma transição de sincronia, a transição só pode ser transposta quando todas as etapas de entrada estiverem ativas, indicando que a evolução de todas as sequências de etapas chegou ao fim.

3.1.2 Representação gráfica de estruturas sequenciais

Sequência

Uma sequência é uma sucessão de etapas de tal forma que:

- 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 3.8 há um exemplo de uma sequência de etapas genérica.

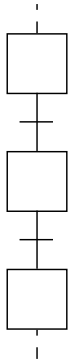


Figura 3.8: Representação gráfica de uma sequência de etapas.

Ciclo de uma única sequência

Um ciclo de uma única sequência possui as seguintes características:

- 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 3.9.

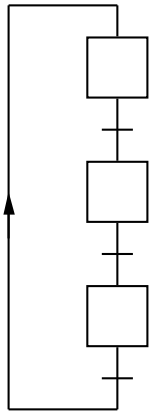


Figura 3.9: Representação gráfica de um ciclo de uma única sequência de etapas.

Seleção de sequências

A seleção de sequências mostra uma escolha de evolução entre diversas sequências iniciando de uma ou várias etapas. A representação gráfica da seleção de sequências pode ser vista na figura 3.10. 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.

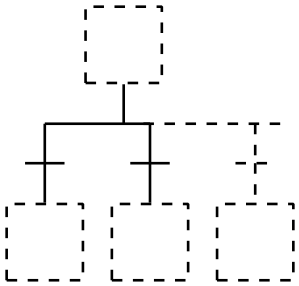


Figura 3.10: Representação gráfica da estrutura de seleção de sequências.

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 7 *Considere o SFC parcial mostrado na figura 3.11. A escolha do caminho de evolução é garantida pela relação mutuamente excludente entre as receptividades das duas transições da figura 3.11. Se a e b são ambos verdadeiros quando a etapa 2 se tornar ativa, então nenhuma transição será transposta.*

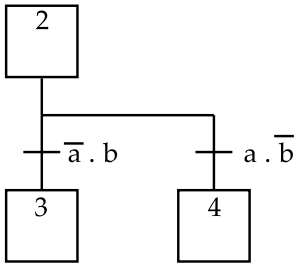


Figura 3.11: Representação gráfica de uma seleção de sequências com transições mutuamente excludentes.

Exemplo 8 *Considere o SFC parcial mostrado na figura 3.12. 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.*

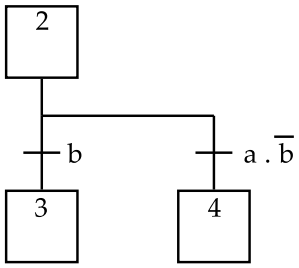


Figura 3.12: Representação gráfica de uma seleção de sequências com prioridade de sequência.

Exemplo 9 *Considere o SFC parcial mostrado na figura 3.13. A seleção das sequências que se sucedem por x e y é possível apenas quando as transições são habilitadas pela atividade simultânea das etapas 1 e 2. Nesse caso há a seleção de sequências seguidas pela sincronização de outras duas sequências precedentes.*

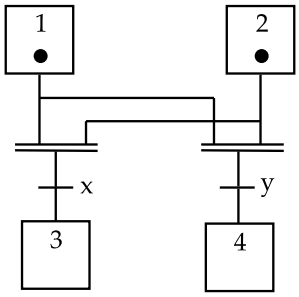


Figura 3.13: Representação gráfica de uma seleção de sequências que segue a sincronização de duas sequências precedentes.

Salto de etapa

Ao programar em SFC é possível pular uma etapa ou uma sequência de etapas. Essa funcionalidade é desejável quando, por exemplo, ações associadas às etapas que se deseja pular se tornam desnecessárias. Um exemplo dessa estrutura pode ser vista na figura 3.14.

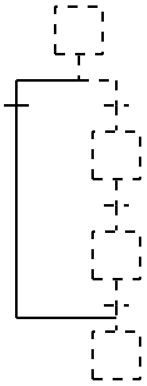


Figura 3.14: Representação gráfica de uma estrutura que permite saltar uma sequência de etapas.

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. A configuração da estrutura em SFC que permite isso pode ser vista na figura 3.15.

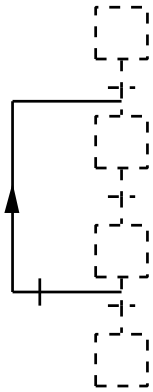


Figura 3.15: Representação gráfica de uma estrutura que permite a repetição de sequências de etapas até que uma determinada condição seja satisfeita.

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 3.16. Note o símbolo de barras duplas horizontais indicando a sincronização das sequências a partir de uma ou mais etapas. Após a ativação simultânea das sequências, a evolução das etapas ativas em cada uma das sequências em paralelo se torna independente.

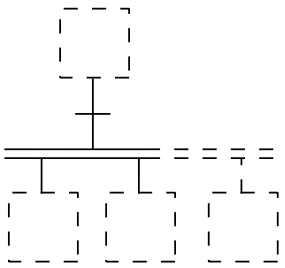


Figura 3.16: Representação gráfica de uma estrutura que permite a ativação de sequências paralelas.

Sincronização de sequências

Note, na figura 3.17, que o símbolo de barras duplas novamente é usado para indicar a 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. Note que, se essa sincronia 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.

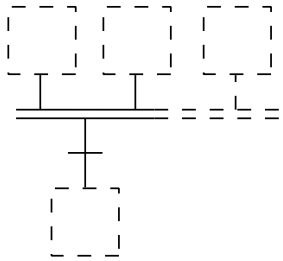


Figura 3.17: Representação gráfica de uma estrutura que sincroniza sequências em paralelo.

Sincronização e ativação de sequências em paralelo

A figura 3.18 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 sincronia, 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, dando início a um processo em paralelo.

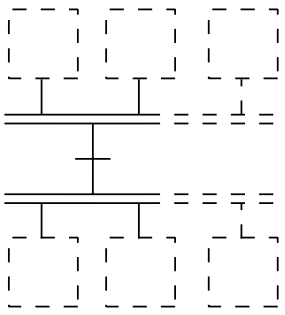


Figura 3.18: Representação gráfica de uma estrutura que sincroniza e ativa sequências em paralelo.

Etapa fonte

Uma etapa fonte é uma etapa que não possui nenhuma transição de entrada, como pode ser visto na figura 3.19.

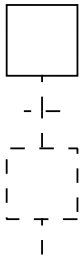


Figura 3.19: Representação gráfica de uma etapa fonte.

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 3.20.

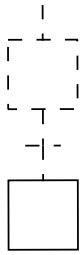


Figura 3.20: Representação gráfica de uma etapa dreno.

Transição fonte

Uma transição fonte é uma transição que não possui nenhuma etapa de entrada, como mostrado na figura 3.21. 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 habilitada enquanto a receptividade da transição for verdadeira, por isso, aconselha-se que a receptividade da transição esteja associada a um evento de entrada ou a um evento interno.

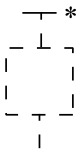


Figura 3.21: Representação gráfica de uma transição fonte.

Transição dreno

Uma transição dreno é uma transição que não possui etapas de saída, como é possível ver na figura 3.22. 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.

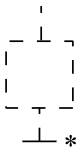


Figura 3.22: Representação gráfica de uma transição dreno.

3.2 Diagrama ladder

O diagrama ladder é uma das cinco linguagens definidas pela norma IEC61131-3 [16] para a programação de CLPs e é uma das linguagens mais usadas na indústria. No diagrama ladder, funções lógicas são representadas através de contatos e bobinas, de maneira análoga a um esquema elétrico com relés e contatores.

O diagrama ladder é 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, serão considerados apenas contatos e bobinas no código

de controle em diagrama ladder.

3.2.1 Contatos

Contatos são componentes fundamentais no diagrama ladder e, dentre eles, são muito usados os contatos normalmente abertos (NA) e os contatos normalmente fechados (NF).

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 3.23 representa um contato NA associado à variável S.

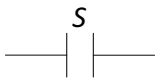


Figura 3.23: Contato NA associado à variável S. Quando o valor lógico de S for igual a 1, o contato NA fecha, dando continuidade lógica ao trecho do diagrama em que está inserido.

O contato NF, ilustrado na figura 3.24, 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.

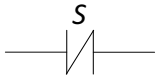


Figura 3.24: Contato NF associado à variável S. Quando o valor lógico de S for igual a 1, o contato NF abre, interrompendo a continuidade lógica do trecho do diagrama em que está inserido.

Além dos contatos NA e NF, os contatos "positive signal edge" (tipo P) e "negative signal edge" (tipo N) são também muito importantes na programação de controladores de sistemas a eventos discretos. O contato do tipo P é análogo ao contato NA, ou seja, ele fica aberto e fecha quando o valor lógico da variável associada muda de 0 para 1. A diferença entre os dois é que o contato tipo P fecha apenas durante o ciclo de varredura imediatamente após a subida da variável booleana associada, ou seja, da mudança do valor da variável booleana associada de 0 para 1. No próximo ciclo de varredura, mesmo que a variável continue com valor lógico 1, o contato abre, pois nenhuma mudança positiva de estado lógico foi detectada.

Um exemplo de contato tipo P pode ser visto na figura 3.25. O contato tipo P analisa o valor lógico da variável associada a ele e o compara com o valor lógico que a variável possuía no último ciclo de varredura. Caso haja alteração positiva da variável, o contato fecha por apenas um ciclo de varredura. O exemplo 10 ilustra o funcionamento do contato tipo P associado à variável S.

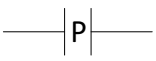


Figura 3.25: Contato "positive signal edge" (tipo P).

Exemplo 10 *Considere o trecho de código ladder exibido na figura 3.26. 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 3.26: Contato tipo P associado a uma variável S.

O contato tipo N funciona de maneira inversa ao contato tipo P, ou seja, 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. A representação do contato tipo N pode ser vista na figura 3.27.

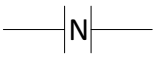


Figura 3.27: Contato "negative signal edge" (tipo N).

É importante notar que a característica dos contatos tipo P e tipo N faz com que esses contatos sejam muito importantes 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 3.28. 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 3.28.

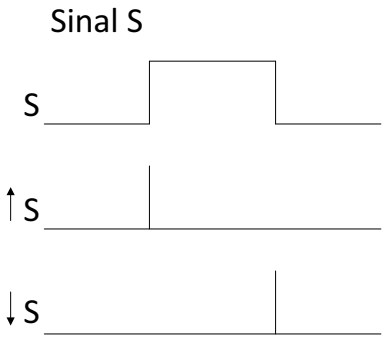


Figura 3.28: Exemplo de um sinal lógico S e a detecção da borda de subida e da borda de descida.

Suponha que a variável S do exemplo 10 seja relacionada ao sinal da figura 3.28. Então, conforme mostrado no exemplo 10, 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.

3.2.2 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 3.29.

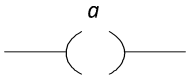


Figura 3.29: Representação de uma bobina simples com a variável a associada.

As bobinas SET e RESET funcionam de maneira um pouco diferente das bobinas

simples. Se a lógica que antecede a bobina SET for verdadeira, o valor da variável relacionada à bobina será levado para 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, a bobina leva para 0 o valor da variável que está associada a ela quando a lógica anterior à bobina for positiva e, como a bobina SET, a bobina RESET mantém o valor 0 à variável associada mesmo que a lógica anterior à bobina se torne falsa novamente.

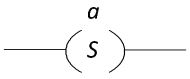


Figura 3.30: Representação de uma bobina SET com a variável a associada.

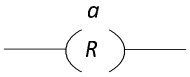


Figura 3.31: Representação de uma bobina RESET com a variável a associada.

Para que o valor da variável associada a uma bobina SET volte a ser 0, é necessário que haja uma bobina RESET associada à mesma variável ao longo do diagrama ladder e que essa bobina seja energizada. O mesmo vale para uma variável que tenha ficado com valor lógico 0 por conta de uma energização de uma bobina RESET: para que seu valor retorne ao valor lógico 1 é necessário que uma bobina SET associada à mesma variável seja energizada. As figuras 3.30 e 3.31 ilustram uma bobina SET e uma bobina RESET, respectivamente.

Capítulo 4

Rede de Petri diagnosticadora

A rede de Petri diagnosticadora tem a característica de formalizar o alcance não observável e fornecer uma estrutura capaz de realizar esse alcance de tal forma que seja possível sua implementação em um CLP, permitindo assim a diagnose online de um sistema. Neste trabalho é considerado que o sistema é modelado por um autômato finito.

Seja G o autômato que modela o comportamento controlado do sistema. O primeiro passo para a construção da rede de Petri diagnosticadora é construir o autômato G_C que modela a composição dos comportamentos normais do sistema em relação aos eventos de falha de Σ_{f_k}.

4.1 Obtenção do autômato G_C

O autômato G_C é obtido a partir dos autômatos que modelam o comportamento normal do sistema em relação à falha do tipo k, G_{N_k}, para $k = 1, \ldots, r$. Esse método é diferente do apresentado por QIU e KUMAR [5], que usa o comportamento normal e de falha do sistema, reduzindo a complexidade computacional do processo de diagnose online.

O algoritmo 2 ilustra o procedimento de construção do autômato G_C.

Algoritmo 2

- *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: Faça $G_{N_k} = G \times A_{N_k} = (Q_{N_k}, \Sigma, f_{N_k}, \Gamma_{N_k}, q_{0,N_k})$.*
- *Passo 2: Construa o autômato estendido $G^a_{N_k}$, para cada $k \in I_r$, adicionando um novo estado F_k, que indica que um evento de falha do conjunto Σ_{f_k} ocorreu. Uma nova transição rotulada com um evento $\sigma_{f_k} \in \Sigma_{f_k}$ é adicionada, conectando o estado (q, N_k) de G_{N_k} ao estado de falha F_k, se $\sigma_{f_k} \in \Gamma(q)$. Adicione um autolaço rotulado com todos os eventos $\sigma \in \Sigma$ ao estado de falha F_k.*
- *Passo 3: Calcule o autômato $G_C = (Q_C, \Sigma, f_C, \Gamma_C, q_{0,C}) = G^a_{N_1} \| G^a_{N_2} \| \ldots \| G^a_{N_r}$.*

□

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}, com um autolaço rotulado com todos os eventos do conjunto Σ. Dessa forma, se o sistema alcançar o estado F_k, então uma falha do conjunto Σ_{f_k} ocorreu. É importante mencionar que essa representação não preserva a linguagem gerada pelo sistema após a ocorrência do evento de falha. Entretanto, como o diagnosticador é um dispositivo passivo, sua representação não altera a observação dos eventos do sistema e, portanto, não interfere na diagnose de falhas.

Para mostrar como o autômato G_C pode ser usado na diagnose 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 $s \in L$ cujo último evento observável é σ_o, e pode ser calculada recursivamente como em [5]

$$Reach(\varepsilon) = UR(q_{0,C}), \tag{4.1}$$

$$Reach(v\sigma_o) = UR(\delta(Reach(v), \sigma_o)), \tag{4.2}$$

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 diagnose proposto neste trabalho.

Teorema 1 *Seja L a linguagem gerada por G 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. □

Se L é diagnosticável com relação a P_o e Π_f, então, de acordo com o teorema 1, sempre é possível identificar a ocorrência de uma falha do tipo F_k com um número limitado de observação 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, a diagnose 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.

Observação 1 *No pior caso, o número de estados de G_C é igual a $[(2^r - 1) \times |Q|] + 1$, em que r é o número de tipos de falha do sistema. Logo, a complexidade computacional da construção de um autômato G_C é $O(2^r \times |Q| \times |\Sigma|)$, o que mostra que a complexidade é linear com o número de estados e eventos do autômato do sistema e exponencial com relação ao número de tipos de falhas. A complexidade computacional pode ser linear com relação ao número de tipos de falha se cada comportamento normal com relação a um tipo de falha é considerado separadamente. Nesse caso, ao invés de um único autômato G_C, tem-se r autômatos $G^a_{N_k}$, em que cada um leva em consideração apenas a falha do tipo F_k, e a complexidade computacional é $O(r \times |Q| \times |\Sigma|)$. Embora a análise de pior caso sugira que é vantajoso considerar os autômatos $G^a_{N_k}$, para $k = 1, \ldots, r$, ao invés de G_C, é importante observar 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.* □

A construção do autômato G_C e o processo de diagnose com base na função $Reach(\nu)$, em que ν é uma sequência de eventos, para um sistema com dois tipos de falha é ilustrado no exemplo 11.

Exemplo 11 *Seja G o autômato do sistema apresentado na figura 4.1, 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 $\Sigma_{f_2} = \{\sigma_{f_2}\}$, e suponha que se deseja calcular o autômato G_C.*

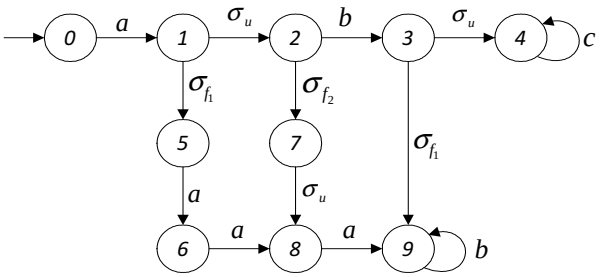


Figura 4.1: Autômato G do Exemplo 11.

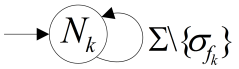


Figura 4.2: Autômato A_{N_k} do Exemplo 11.

De acordo com o algoritmo 2, o primeiro passo é obter os autômatos A_{N_k}, para $k = 1, 2$, mostrado na figura 4.2, 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 4.3 e 4.4, 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 2 é o cálculo do autômato $G_C = G^a_{N_1} \| G^a_{N_2}$, ilustrado na figura 4.5.

A partir de agora será apresentado como o autômato G_C pode ser usado no processo de diagnose online. 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 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 (4.1) e (4.2), da seguinte forma:

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

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

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

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

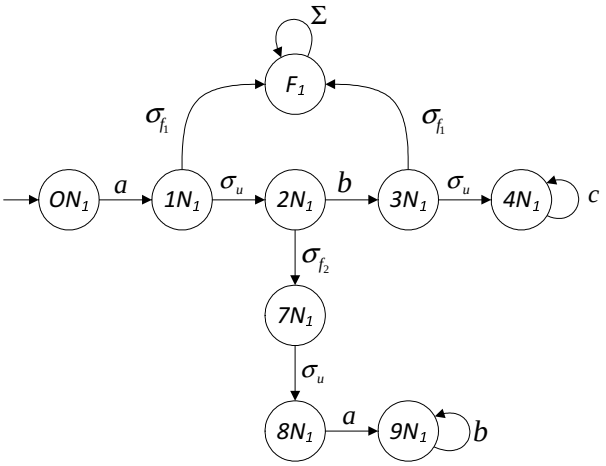


Figura 4.3: Autômato aumentado $G^a_{N_1}$ do Exemplo 11.

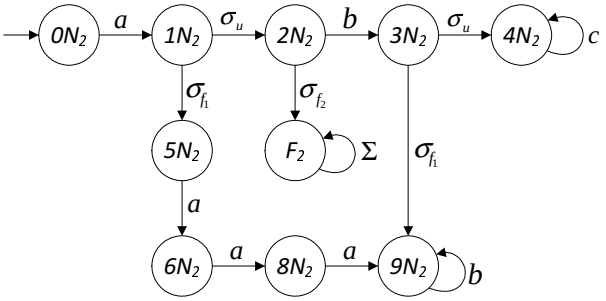


Figura 4.4: Autômato aumentado $G^a_{N_2}$ do Exemplo 11.

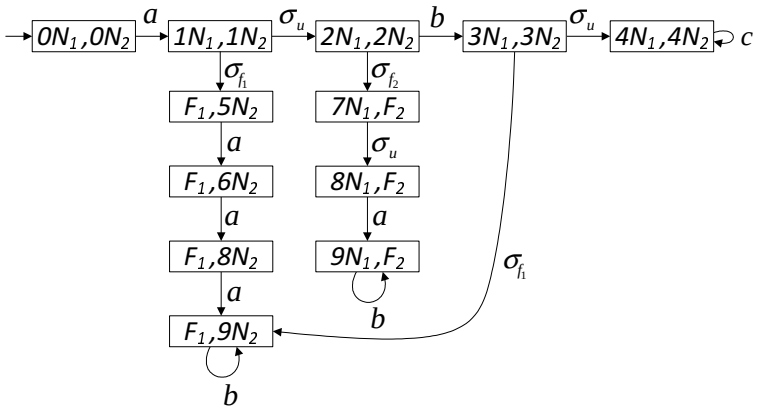


Figura 4.5: Autômato $G_C = G^a_{N_1} \| G^a_{N_2}$ do Exemplo 11.

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.

Com relação à complexidade computacional para construção de G_C, pode ser visto que G_C possui 12 estados e $G^a_{N_1}$ e $G^a_{N_2}$ possuem 9 e 10 estados, respectivamente. Logo G_C possui um número menor de estados do que a soma de estados de $G^a_{N_1}$ e $G^a_{N_2}$. Portanto, como mostrado na observação 1, a diagnose online pode ser executada, neste caso, com um custo computacional menor usando G_C ao invés de $G^a_{N_k}$, para $k = 1, 2$.

Na Seção 4.2, 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.

4.2 Construção da rede de Petri diagnosticadora

Nesta altura do trabalho 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 $\nu \in \Sigma_o^*$, ou seja, é necessário um observador online que armazena 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 da rede de Petri, levando a uma 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 de uma rede de Petri máquina de estados, $\mathcal{N}_C$, a partir de um autômato G_C, pode ser realizada associando-se um lugar p_{C_i} em $\mathcal{N}_C$ a cada estado q_{C_i} de G_C e associando-se a cada arco direcionado em G_C, $(q_{C_i}, \sigma, \bar{q}_{C_i})$, em que $\bar{q}_{C_i} = f_C(q_{C_i}, \sigma)$ e $\sigma \in \Gamma_C(q_{C_i})$, uma

transição t_{C_j}, rotulada com σ, em $\mathcal{N}_C$ [1]. Para ligar lugares e transições em $\mathcal{N}_C$, dois arcos com peso igual a um precisam ser criados para cada transição: um arco (p_{C_i}, t_{C_j}) e um arco $(t_{C_j}, \bar{p}_{C_i})$, em que $\bar{p}_{C_i}$ é o lugar de $\mathcal{N}_C$ associado ao estado $\bar{q}_{C_i}$. 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 atribuindo-se zero fichas aos outros lugares.

Uma vez que se tenha 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 que isso seja feito é necessário definir a função $Reach_T : T_{C_o} \rightarrow 2^{P_C}$, em que T_{C_o} é o conjunto de todas as transições de $\mathcal{N}_C$ rotuladas por eventos observáveis e P_C é o conjunto finito de lugares de $\mathcal{N}_C$. 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.

Algoritmo 3 *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.*

□

De acordo com o algoritmo 3 é preciso adicionar um arco com peso 1 a $\mathcal{N}_C$ conectando cada transição $t_{C_j} \in T_{C_o}$ a cada lugar $p_{C_i} \in Reach_T(t_{C_j})$, gerando uma nova rede de Petri, $\mathcal{N}'_C$. Para implementar o alcance não observável após o disparo de cada transição observável é necessário remover todas as transições de $\mathcal{N}'_C$ que sejam rotuladas com eventos não observáveis e seus arcos relacionados, gerando

uma nova rede de Petri, $\mathcal{N}_{C_o}$, cujas transições são rotuladas apenas com eventos observáveis pertencentes a Σ_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 $\mathcal{N}_{C_o}$ 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 [29], como mostrado na subseção 2.3.2.

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 e obter a rede de Petri observadora de estados, $\mathcal{N}_{SO}$, é necessário adicionar um arco conectando cada lugar p_{C_i} de $\mathcal{N}_{C_o}$ a uma nova transição que não possui lugares de saída, chamada de transição de descarte, rotulada com os eventos observáveis de Σ_o que não estão no conjunto de eventos ativos do estado q_{C_i}

de G_C associado a p_{C_i}. 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} é 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 auto-laço não altera a estimativa de estados.

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} e lugares p_{N_k} e p_{F_k}, para $k = 1, \ldots, r$, em que p_{N_k} são os lugares de entrada de t_{f_k}, e são adicionados com uma ficha cada, e p_{F_k} são os lugares de saída de t_{f_k}, sem nenhuma ficha. Os lugares p_{N_k} e p_{F_k} são ligados às transições t_{f_k} por arcos de peso igual a um. Cada transição t_{f_k} está associada à verificação da ocorrência de um tipo de falha. Arcos inibidores [2] 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, o que implica que uma falha do conjunto Σ_{f_k} ocorreu. Um arco inibidor é representado por um arco cuja extremidade final possui um pequeno círculo.

A transição t_{f_k} será rotulada com o evento sempre ocorrente λ [2] 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 4 resume os passos necessários para a obtenção da rede de Petri

diagnosticadora $\mathcal{N}_D$ a partir do autômato G_C.

Algoritmo 4

- *Passo 1: Calcule a rede de Petri máquina de estados* $\mathcal{N}_C = (P_C, T_C, Pre_C, Post_C, x_{0,C})$ *a partir de* G_C.
- *Passo 2: Adicione a* $\mathcal{N}_C$ *arcos conectando cada transição observável* $t_{C_j} \in T_{C_o}$ *aos lugares em* $Reach_T(t_{C_j})$*, gerando a rede de Petri* $\mathcal{N}'_C = (P_C, T_C, Pre_C, Post'_C, x_{0,C})$.
- *Passo 3: Elimine todas as transições de* $\mathcal{N}'_C$ *rotuladas com eventos não observáveis e seus arcos relacionados, gerando a rede de Petri binária* $\mathcal{N}_{C_o} = (P_C, T_{C_o}, Pre_{C_o}, Post_{C_o}, x_{0,C})$.
- *Passo 4: Calcule* $\mathcal{N}_{SO} = (P_C, T_{SO}, Pre_{SO}, Post_{SO}, x_{0,SO})$ *da seguinte forma:*
  - *Passo 4.1: Adicione à* $\mathcal{N}_{C_o}$ *transições rotuladas com eventos observáveis de* Σ_o *que não estão no conjunto de eventos ativos do estado* q_{C_i} *de* G_C *associado a* p_{C_i}.
  - *Passo 4.2: Defina o estado inicial de* $\mathcal{N}_{SO}$ *atribuindo uma ficha a cada lugar associado a um estado de* $UR(q_{0,C})$ *e nenhuma ficha aos outros lugares.*
  - *Passo 4.3: Elimine todas as transições de auto-laço e seus arcos associados.*
- *Passo 5: Calcule a rede de Petri diagnosticadora* $\mathcal{N}_D = (P_D, T_D, Pre_D, Post_D, In_D, x_{0,D})$*, em que* $T_D = T_{SO} \cup T_f$*,* $T_f = \bigcup_{k=1}^{r} \{t_{f_k}\}$ *e* $In_D : P_D \times T_f \to \mathbb{N}$ *denota o conjunto de arcos inibidores, como segue:*
  - *Passo 5.1: Adicione à* $\mathcal{N}_{SO}$ *transições* t_{f_k}*, para* $k = 1, \ldots, r$*, rotuladas com o evento sempre ocorrente* λ.
  - *Passo 5.2: Adicione a cada transição* t_{f_k} *um lugar de entrada* p_{N_k} *e um lugar de saída* p_{F_k}*, ambos conectados à* t_{f_k} *por arcos com peso igual a um.*

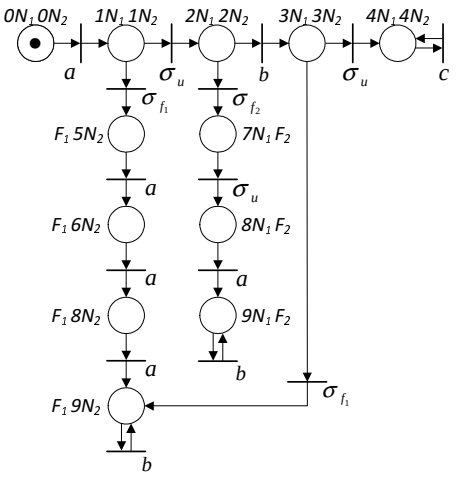


Figura 4.6: Rede de Petri máquina de estados $\mathcal{N}_C$ do exemplo 12.

- *Passo 5.3: Conecte cada lugar associado a um estado de G_C que tem uma coordenada (q, N_k) à transição t_{f_k} com um arco inibidor.*
- *Passo 5.4: A marcação inicial dos lugares p_{N_k} é igual a um e dos lugares p_{F_k} é igual a zero. Os outros lugares possuem a mesma condição inicial definida por $x_{0,SO}$.*

□

O exemplo 12 ilustra os passos para a obtenção da rede de Petri $\mathcal{N}_D$ e o processo de diagnose online a partir do autômato G_C obtido no exemplo 11.

Exemplo 12 *A partir do autômato G_C mostrado na figura 4.5, deseja-se obter a rede de Petri diagnosticadora $\mathcal{N}_D$ para G_C. De acordo com o algoritmo 4, 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 4.6. Como resultado dos passos 2 e 3 do algoritmo 4, a rede de Petri binária $\mathcal{N}_{C_o}$, ilustrada na figura 4.7, é obtida.*

Em seguida, realizando o Passo 4 do algoritmo 4, a rede Petri observadora de estados $\mathcal{N}_{SO}$, mostrada na figura 4.8, é obtida a partir de $\mathcal{N}_{C_o}$. Por fim, ao executar o Passo 5 do algoritmo 4, obtém-se a rede de Petri diagnosticadora $\mathcal{N}_D$ que está ilustrada na figura 4.9.

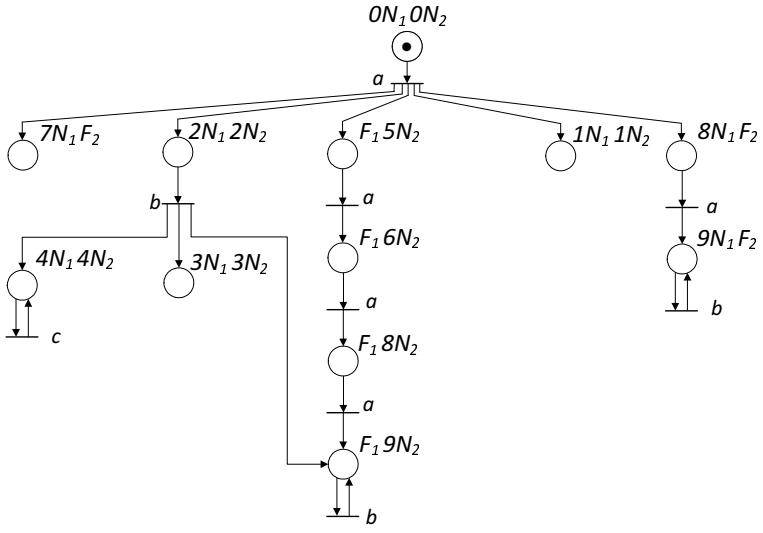


Figura 4.7: Rede de Petri binária $\mathcal{N}_{CO}$ do exemplo 12.

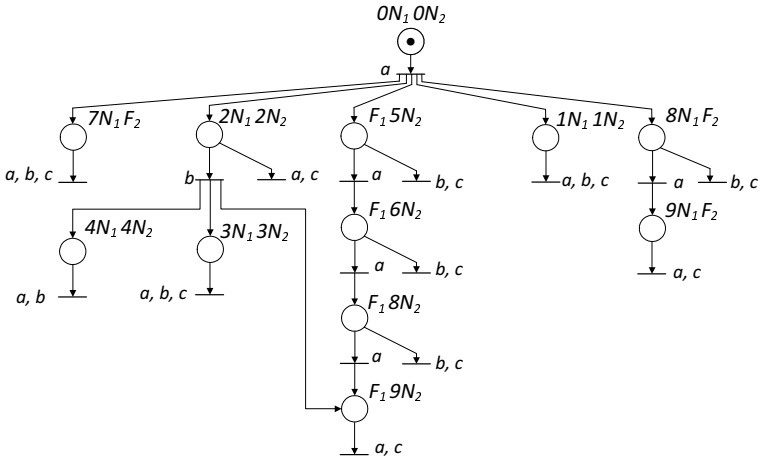


Figura 4.8: Rede de Petri observadora de estados $\mathcal{N}_{SO}$ do exemplo 12.

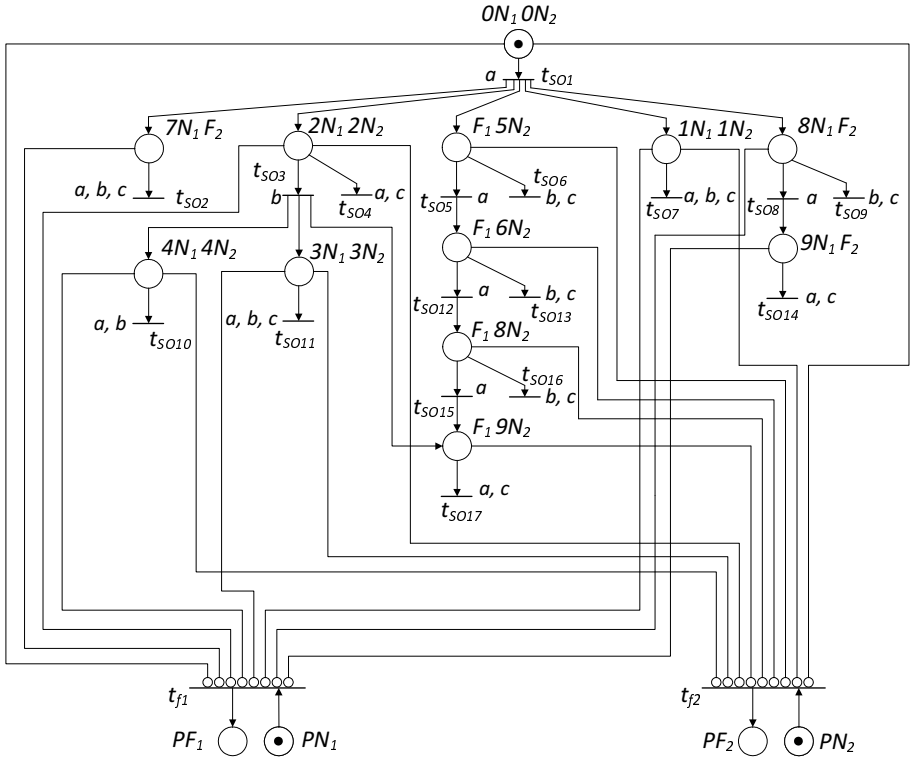


Figura 4.9: Rede de Petri diagnosticadora $\mathcal{N}_D$ do exemplo 12.

Após $\mathcal{N}_D$ ter sido calculada, o processo de diagnose 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_1} dispara e o conjunto de lugares associados com os possíveis estados de G_C que possuem uma ficha é dado por $\{(1N_1, 1N_2)$, $(2N_1, 2N_2)$, $(7N_1, F_2)$, $(8N_1, F_2)$, $(F_1, 5N_2)\}$. 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 4.1 do algoritmo 4 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 4.1, que ilustra os lugares com ficha após o

Tabela 4.1: Tabela que ilustra os lugares com ficha da rede de Petri $\mathcal{N}_D$ do exemplo 12 para cada sequência de eventos observada.

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

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 sempre ocorrente 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}.

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 diagnose online, indicando, após uma sequência arbitrariamente longa de eventos realizada pelo sistema, se uma falha ocorreu e qual o tipo de falha ocorreu. O diagnosticador construído pode ser implementado em um CLP, uma vez que usa o formalismo de redes de Petri. No próximo capítulo serão apresentados dois métodos de conversão de uma rede de Petri em duas linguagens de programação de CLP definidas pela norma IEC61131-3 [16]: diagrama Ladder e SFC.

Capítulo 5

Conversão da rede de Petri diagnosticadora para linguagens de programação em CLP

Como apresentado no capítulo 3, 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 do usuário e; (*iii*) atualização das saídas. Em geral, eventos de entrada são associados com a 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 com mudanças nos sinais de sensores, e eventos de comando associados com a resposta do controlador a essas mudanças seriam vistos pelo diagnosticador como eventos que estariam ocorrendo ao mesmo tempo. Então, com o objetivo de simular o real comportamento do sistema, o código do diagnosticador deve ser implementado antes do código de controle, como é mostrado na figura 5.1. Além disso, neste trabalho, é suposto que dois eventos não ocorrem simultaneamente.

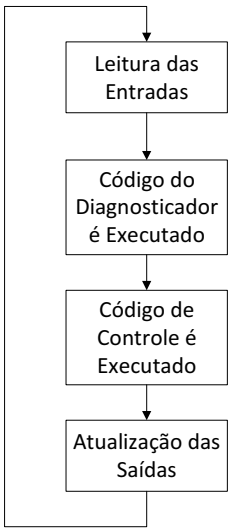


Figura 5.1: Ciclo de varredura do CLP com o código do diagnosticador implementado antes do código do controlador do sistema.

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 outros r SFCs parciais representam testes de verificação da ocorrência de eventos de falha para cada um dos r tipos de falha.

Na figura 5.2 o SFC da rede de Petri observadora de estados $\mathcal{N}_{SO}$ da figura 4.8 é apresentado. Cada lugar de $\mathcal{N}_{SO}$ é simplesmente transformado em uma etapa do SFC, e as transições não são alteradas. 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 do teste de falha é realizado para dois tipos de falha. Os SFCs parciais que realizam a verificação do teste 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

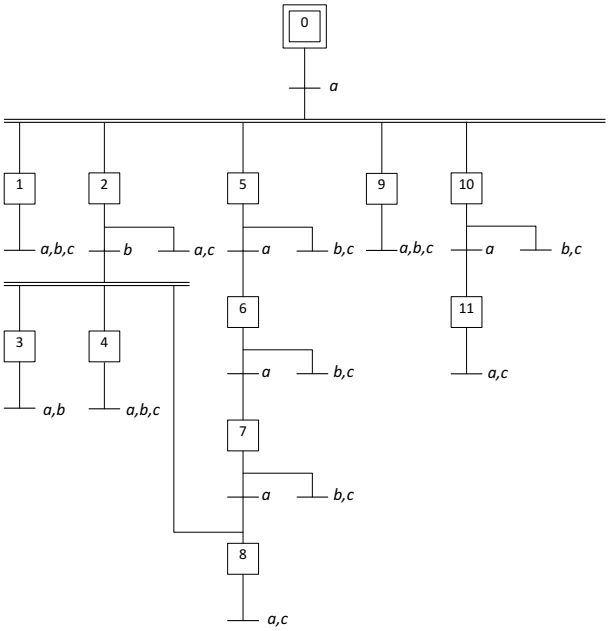


Figura 5.2: SFC da rede de Petri observadora de estados $\mathcal{N}_{SO}$ da figura 4.8.

verdadeira, a etapa associada ao lugar p_{F_k} é ativada e um conjunto de ações podem ser alocadas 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

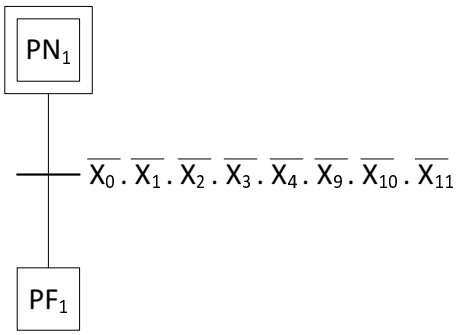


Figura 5.3: SFC da verificação da ocorrência do evento de falha σ_{f_1}.

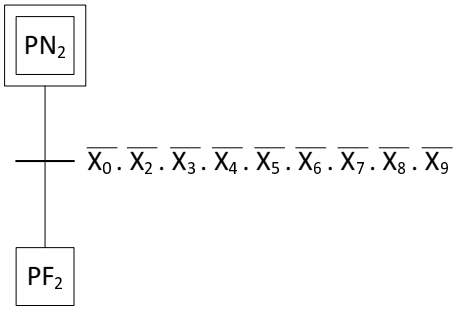


Figura 5.4: SFC da verificação da ocorrência do evento de falha σ_{f_2}.

5.2 Conversão da rede de Petri diagnosticadora 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.

O efeito avalanche foi tratado inicialmente em [24], que propôs um procedimento sistemático para evitar o efeito avalanche em implementações em ladder de sistemas de controle supervisório modelados por autômatos. Entretanto, FABIAN e HELLGREN [24] não abordam um método formal para a implementação de redes de Petri complexas. Diversos outros métodos para a conversão de controladores modelados por redes de Petri em diagramas Ladder para a implementação em CLP podem ser

encontrados na literatura [18–20, 22, 23]. Embora esse métodos tenham sido implementados com sucesso ao controle de sistemas automatizados, eles não levam em consideração o efeito avalanche.

Um outro problema relacionado com a implementação em ladder de redes de Petri ocorre quando um lugar instantaneamente recebe e perde uma ficha após o disparo de duas transições diferentes. 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 não pode ser solucionado com nenhuma das técnicas apresentadas na literatura.

Em [26], 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.* [26]. O método foi alterado para considerar redes de Petri binárias e o problema de simultaneamente alterar o valor de uma variável binária de 0 para 1 e de 1 para 0 que seja associada com a marcação de um lugar da rede de Petri diagnosticadora. O método proposto consiste em dividir o diagrama ladder em cinco módulos da seguinte forma:

- Módulo M1, que representa a inicialização da rede de Petri, ou seja, define a marcação inicial;
- Módulo M2, associado à identificação de ocorrência de eventos externos;
- Módulo M3, associado às condições para os disparos das transições;
- Módulo M4, que descreve a evolução das fichas na rede de Petri;
- Módulo M5, que define os alarmes que serão acionados caso uma falha seja identificada e isolada;

Nas próximas seções serão apresentados cada um dos cinco módulos com mais detalhes e o método de conversão será ilustrado com a conversão da rede de Petri diagnosticadora da figura 4.9 em um diagrama ladder.

5.2.1 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 de 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 5.5 ilustra o módulo inicial em ladder para a rede de Petri diagnosticadora da figura 4.9.

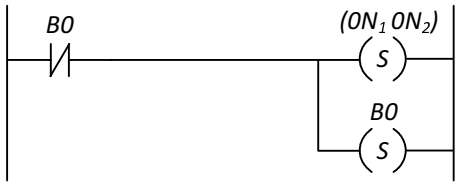


Figura 5.5: Módulo de inicialização da rede de Petri diagnosticadora da figura 4.9.

5.2.2 Módulo de eventos externos

Eventos externos são associados às bordas de subida ou descida de sinais de sensores na rede de Petri diagnosticadora. O sinal de disparo de uma transição pode ser detectado 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 4.9 existem três eventos: a, b e c, sincronizando 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 5.6. Quando, por exemplo, S_a muda seu valor de zero para

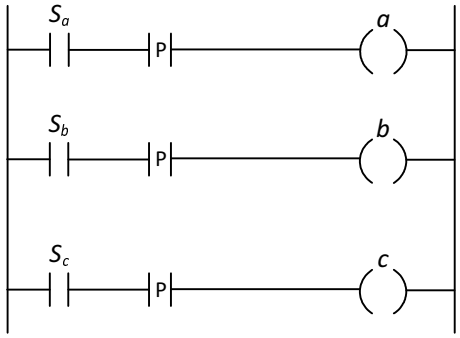


Figura 5.6: Módulo de eventos externos para a rede de Petri diagnosticadora da figura 4.9.

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.

5.2.3 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 $T_D = T_{SO} \cup T_f$. 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 a um 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 sempre ocorrente, 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 em paralelo de contatos normalmente abertos 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

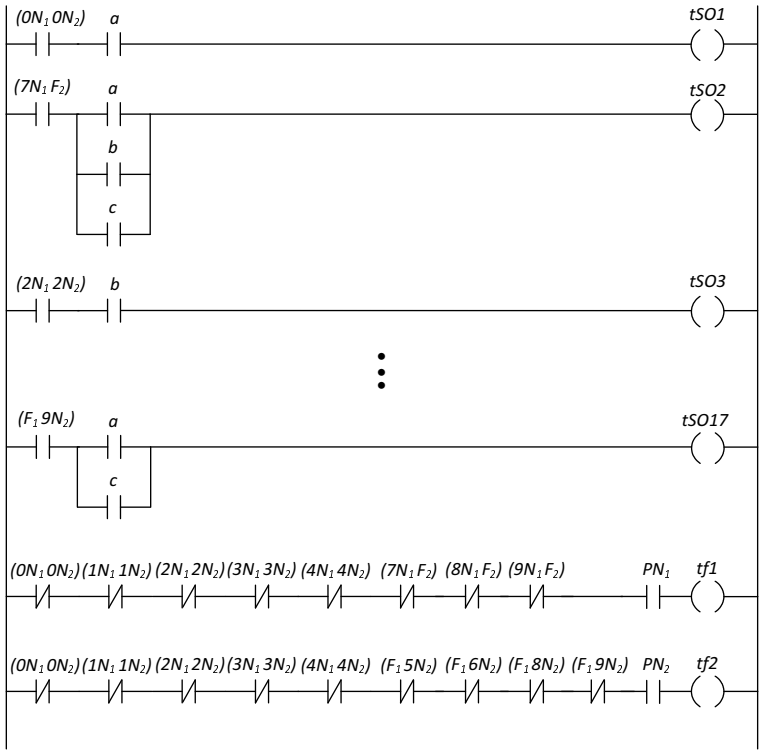


Figura 5.7: Módulo das condições de disparo das transições para a rede de Petri diagnosticadora da figura 4.9.

$In(p_{D_i}, t_{f_k}) > 0$. Um contato NA, em série com NF podem ser usados para representar o arco 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 5.7, apenas seis linhas do diagrama ladder do módulo das condições para o disparo das transições da rede de Petri diagnosticadora da figura 4.9 estão representadas. A primeira, a segunda e a terceira linhas estão associadas com o disparo das transições $t_{SO_1}, t_{SO_2}, t_{SO_3} \in T_{SO}$ e as três últimas linhas estão associadas com o disparo das transições $t_{SO_{17}} \in T_{SO}$ e $t_{f_1}, t_{f_2} \in T_f$.

5.2.4 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 bobina de SET ou RESET é usada para alocar o valor um ou zero à variável binária que representa o número de fichas de um lugar da rede de Petri. Então, após a ocorrência de um evento observável e_o, um conjunto de transições rotuladas com e_o disparam simultaneamente, levando a uma nova marcação da rede de Petri.

O disparo simultâneo de diversas transições rotuladas com o mesmo evento e_o da rede de Petri diagnosticadora pode levar à situação em que a transição de saída de um lugar que tem uma ficha, p_{D_i}, 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 e_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 e_o. Para ilustrar esse fato, considere uma parte de uma rede de Petri diagnosticadora mostrada na figura 5.8. Nesse exemplo, se as linhas são implementadas na ordem apresentada na figura 5.9(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 5.8 é apresentada na figura 5.9(b). Note que, após a ocorrência do evento a, na implementação em diagrama ladder da figura 5.9(b), as variáveis binárias que

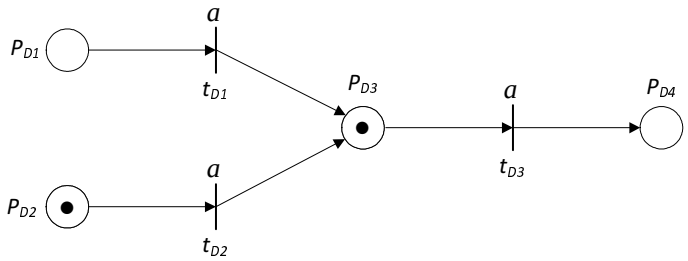


Figura 5.8: Fração de uma rede de Petri com duas transições consecutivas habilitadas sincronizadas com o mesmo evento.

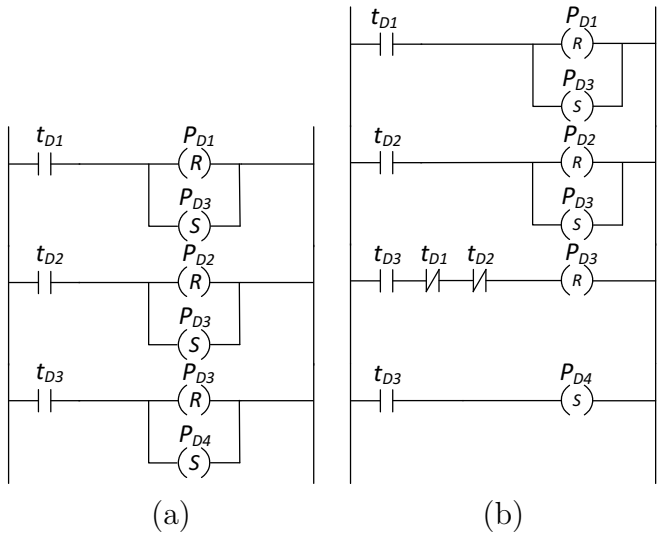


Figura 5.9: Módulo incorreto da dinâmica da rede de Petri para a rede de Petri da figura 5.8 (a), e módulo correto da dinâmica da rede de Petri usando uma associação em série de contatos NF para o *reset* da variável binária associada com o lugar de entrada de t_{D_3}, p_{D_3} (b).

terão valor igual a um são as que estão associadas com os lugares p_{D_3} e p_{D_4}, como era desejado.

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 4.9 é apresentado na figura 5.10.

5.2.5 Módulo dos alarmes

O número de linhas no módulo dos alarmes é igual ao número de tipos de falha na rede de Petri diagnosticadora. Um conjunto de ações pode ser definido para cada tipo de falha dependendo do seu grau de importância. O módulo dos alarmes para o

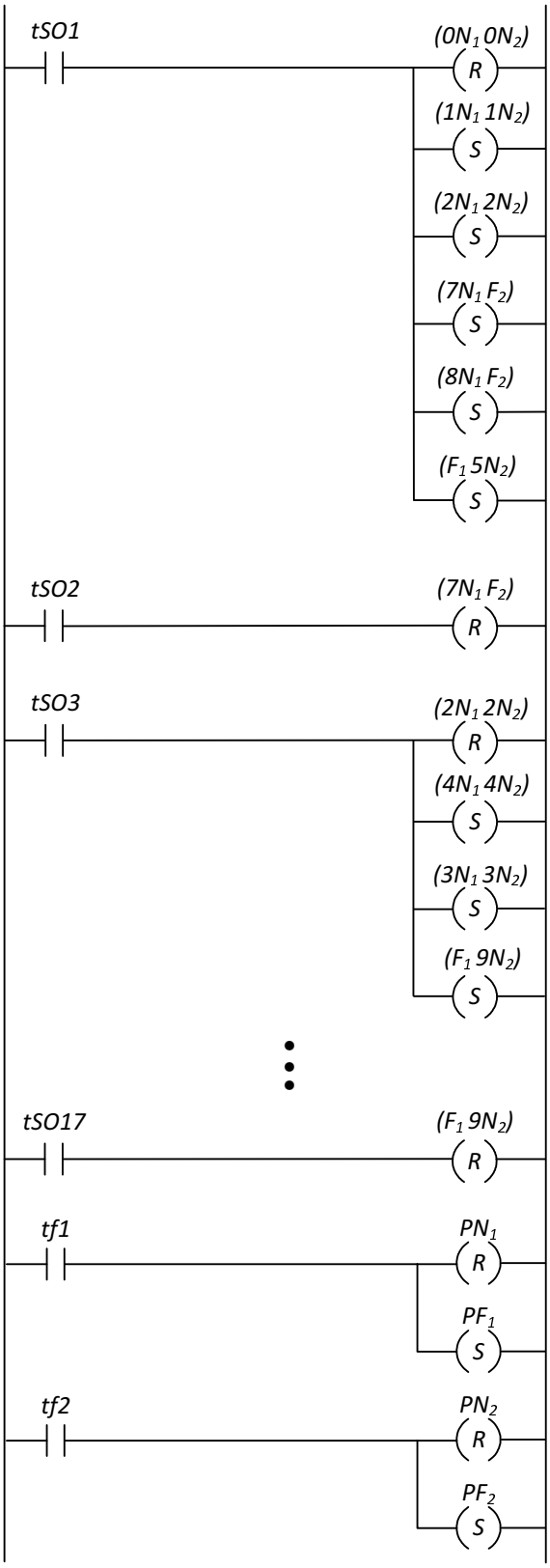


Figura 5.10: Módulo da dinâmica para a rede de Petri diagnosticadora da figura 4.9.

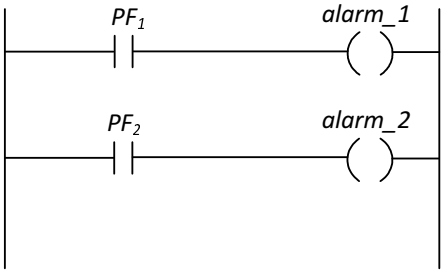


Figura 5.11: Módulo dos alarmes para a rede de Petri diagnosticadora da figura 4.9.

exemplo da figura 4.9 é apresentado na figura 5.11. 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 mantém 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

Assumindo que existam l eventos externos distintos associados com a borda de subida ou descida de sinais de sensores, então, o máximo número de linhas no

diagrama ladder obtido a partir desse método é $(1 + l + 3|T_D| + r)$.

Capítulo 6

Conclusão

Neste trabalho, uma rede de Petri diagnosticadora para sistemas a eventos discretos modelados por autômatos finitos foi apresentada. Esse diagnosticador pode ser usado para detecção e isolamento de falhas online e requer, em geral, menos memória computacional do que outros métodos 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 a diagnose. As técnicas de conversão aplicadas levam a códigos de controle que simulam o comportamento da rede de Petri e permitem a implementação correta da dinâmica da rede de Petri, evitando o efeito avalanche e o problema da remoção e adição de uma ficha a um lugar após o disparo de duas transições diferentes.

Como trabalhos futuros são propostos o estudo e a implementação de uma rede de Petri diagnosticadora online modular. O estudo é motivado 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 cresce exponencialmente com o número de componentes do sistema, levando a um modelo muito complexo e, consequentemente, a um diagnosticador computacionalmente complexo. Trabalhando-se de maneira modular, evita-se esse crescimento e

a consequente complexidade computacional de um diagnosticador único do sistema ao invés de um diagnosticador modular.

Referências Bibliográficas

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

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

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] XU, S., JIANG, S., KUMAR, R. "Diagnosis of dense-time systems under event and timing masks", *IEEE Transactions on Automation Science and Engineering*, v. 7, n. 4, pp. 870–878, 2010.

[14] 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. to appear.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

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

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