Uma Nova Abordagem para a Síntese de Supervisores de Sistemas a Eventos Discretos

Giovanni Cordeiro Barroso

Tese de Doutorado submetida à Coordenação dos Cursos de Pós-Gradução em Engenharia Elétrica da Universidade Federal da Paraíba - Campus II como parte dos requisitos necessários para obtenção do grau de Doutor em Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Antonio Marcus Nogueira Lima, Dr.
Orientador
Angelo Perkusich, D.Sc.
Orientador

Campina Grande, Paraíba, Brasil

Uma Nova Abordagem para a Síntese de Supervisores de Sistemas a Eventos Discretos

Giovanni Cordeiro Barroso

Tese de Doutorado apresentada em Agosto de 1996

Antonio Marcus Nogueira Lima, Dr.
Orientador
Angelo Perkusich, D.Sc.
Orientador

José Eduardo R. Cury, Dr. - UFSC/SC
Componente da Banca
Maurizio Tazza, Dr. - CEFET/PR
Componente da Banca
José Reinaldo Silva, Dr. - USP/SP
Componente da Banca
Jorge Cesar A. de Figueiredo, Dr. - UFPB/PB
Componente da Banca
Ulrich Schiel, Dr. - UFPB/PB
Componente da Banca

Campina Grande, Paraíba, Brasil, Agosto de 1996

B277n Barroso, Giovanni Cordeiro
Uma nova abordagem para a sintese de supervisores de sistemas a eventos discretos / Giovanni Cordeiro Barroso. - Campina Grande, 1996.
172 f.

Tese (Doutorado em Engenharia Eletrica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Redes de Petri 2. Supervisores de Sistemas a Eventos Discretos 3. Tese I. Lima, Antonio Marcus Nogueira, Dr. II. Perkusich, Angelo, Dr. III. Universidade Federal da Paraiba - Campina Grande (PB)

CDU 621.3.011.75(043)

UMA NOVA ABORDAGEM PARA A SÍNTESE DE SUPERVISORES DE SISTEMAS A EVENTOS DISCRETOS

GIOVANNI CORDEIRO BARROSO

Tese Aprovada em 28.08.1996

PROF. ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFPB
Orientador

PROF. ANGELO PERKUSICH, D.Sc., UFPB
Orientador

PROF. JOSÉ EDUARDO RIBEIRO CURY, Ph.D., UFSC
Componente da Banca

PROF. MAURIZIO TAZZA, Ph.D., CEFET-PR
Componente da Banca

JOSÉ REINALDO SILVA, Ph.D., USP-SP
Componente da Banca

JORGE CÉSAR ABRANTES DE FIGUEIREDO, D.Sc., UFPB
Componente da Banca

ULRICH SCHIEL, Dr.ver.nat., UFPB
Componente da Banca

CAMPINA GRANDE - PB
Agosto - 1996

Dedicatória

Este trabalho é dedicado à minha esposa e filhos

Marta, Joana e Ivan

pelo amor, carinho e dedicação e pela vida que compartilhamos;

A meus pais e irmãos

Seu Manuel Natalicio, Dona Albinha, Natal, Pacelli, Álcio, Natalia, Eliana, Carlinhos, Jânio, Ádson, Liliane e Paulo Henrique(in memmorium) pelo amor, carinho, dedicação e amizade;

A meus tios e primos Rubens, Izinha, Rubinho, Giovanna e Mateus que são tambêm pais e irmãos para mim;

A meus tios e primas Aldo, Nícia, Úrsula e Patrícia pelo imenso carinho e apoio durante todo o tempo em que me acolheram em sua casa e pela amizade que nos une.

Agradecimentos

Agradecimento especial aos meus orientadores Antonio Marcos e Angelo pela dedicação, atenção e apoio incondicional em todos os momentos deste trabalho conjunto feito a seis mãos.

A todos os amigos deste departamento, colegas, professores e funcionários que sempre propiciaram um ambiente de companheirismo e amizade.

Agradecimento especial aos amigos e companheiros de muitas alegrias e poucas tristezas desde os tempos da graduação Helano, Paulo e Mauro.

A Paulo e Sueli pelo apoio e amizade incondicionais por estes anos compartilhados em Campina Grande.

A todos que tenho a honra de chamá-los amigos.

Resumo

As redes de Petri têm sido amplamente utilizadas para a especificação, projeto, análise e controle de sistemas a eventos discretos. A teoria de controle supervisório, em conjunto com as redes de Petri, têm sido utilizadas para a solução do problema de síntese de supervisores para sistemas a eventos discretos. Este trabalho introduz uma nova abordagem para a síntese de supervisores, utilizando controle supervisório e redes de Petri. Para tanto, são introduzidos uma nova classe de redes de Petri, denominada *Redes de Petri com Funções de Habilitação de Transições*, e dois algoritmos para a síntese do supervisor. O objetivo é construir um supervisor que possua a mesma estrutura do modelo do sistema a ser controlado. Apesar dos poderosos mecanismos de estruturação disponíveis na teoria de redes de Petri para a modelagem de sistema complexos, existe ainda o problema da explosão de estados quando se analisam tais sistemas. Sendo assim, apresentamos também uma metodologia modular para a síntese do supervisor, usando dois tipos de redes de alto nível, *Redes de Petri Coloridas* e *G-Nets*. Mais ainda, pensando na aplicação desta abordagem para sistemas reais, introduzimos de forma sistemática, no supervisor, propriedades de tolerância a falhas. Aplicações e exemplos desta abordagem são direcionadas para sistemas de manufatura automatizados.

Abstract

Petri nets have been widely used to specifying, designing, analyzing and controlling discrete event systems. In the same way, the supervisory control theory, together with Petri nets, has been used to solve the supervisors synthesis problem of discrete event systems. This work introduces a new approach, using Petri nets and the supervisory control theory, to solve the supervisor synthesis problem. To do so, two algorithms are presented, as well as a new kind of Petri nets called *Petri nets with transition enabling function.* The objective is to synthesize a supervisor that has the same structure of the model of the system to be controlled. Despite of powerful structuring mechanisms available in the Petri net theory for the construction of the model of complex systems, we are still likely to face the problem of state explosion when analyzing large systems. Thus, we also present a modular approach to supervisors synthesis, using two kinds of high level nets called *colored Petri nets* and *G-Nets*, respectively. Moreover, having in mind the objective of applying this approach to actual systems, we use this approach to the synthesis of fault-tolerant supervisors. The application and exemplification of this presented approach is on the synthesis of supervisors for automated manufacturing systems.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Ao longo dos anos o desenvolvimento de sistemas físicos dinâmicos tem sido ancorado no uso de equações diferenciais ordinárias e parciais. Por outro lado, a moderna tecnologia tem permitido o desenvolvimento dos chamados sistemas dinâmicos feitos pelo homem (*man-made dynamic systems*). Esses sistemas não são facilmente descritos por equações diferenciais ordinárias ou parciais [Ho89]. Tais sistemas estão presentes em uma série de aplicações, incluindo por exemplo redes de computadores, sistemas de manufatura, robótica, supervisão de tráfegos aéreo e ferroviário, sistemas operacionais e otimização de processos distribuídos [CR90].

Tais sistemas têm em comum a forma pela qual os mesmos percebem as ocorrências do ambiente, denominadas eventos. A chegada de um cliente em uma fila e o recebimento de uma mensagem em um sistema de comunicação são exemplos de eventos. A ocorrência de um evento, em geral, causa uma mudança na configuração interna, ou estado, do sistema. Estes eventos são, por natureza instantâneos o que lhes confere um caráter discreto no tempo. Após a ocorrência de um evento, o sistema reage acomodando-se em uma nova configuração ou estado, onde permanecerá até que um novo evento ocorra. Sistemas com estas características são denominados *sistemas dinâmicos a eventos discretos* ou simplesmente *sistemas a eventos discretos* (SEDs), diferentemente dos *sistemas dinâmicos a variáveis contínuas* (SDVCs), tratados pela teoria de controle clássica e descritos por equações diferenciais [Ho89].

Atualmente existe uma intensa atividade de pesquisa voltada à busca de modelos

matemáticos adequados à representação dos SEDs, sem que se tenha, até agora, um modelo matematicamente conciso e computacionalmente adequado tais como o são as equações diferenciais para os SDVCs. A falta de um paradígma de modelagem para os SEDs é a razão pela qual a aplicação da teoria de controle para estes sistemas é ainda um problema desafiador.

De um modo geral, um sistema é uma parte limitada do universo que interage com o mundo externo através das fronteiras que o delimitam. Este conceito se aplica também aos SEDs, cujas características básicas são:

- Seu ciclo de funcionamento é descrito através do encadeamento de eventos (e.g.: chegada de matéria prima, acionamento de uma correia transportadora, carregamento de um laminador, programação de um laminador, laminação e retirada do rolo);
- Ocorrência de eventos em paralelo (e.g.: a programação de um laminador é executada durante o seu carregamento);
- Necessidade de sincronização (e.g.: um canal de comunicação que é utilizado por vários usuários para enviar e receber mensagens);

Assim, um SED evolui no tempo, pela ocorrência de eventos, sejam eles internos ou externos. Note que a ocorrência de eventos pode depender de fatores alheios ao sistema (por exemplo a quebra de uma máquina). Desta forma, um SED pode ser assim definido [RW88]:

Definição 1.1 *Um Sistema a Eventos Discretos (SED) é um sistema que evolui com a ocorrência abrupta de eventos físicos, a intervalos de tempo em geral irregulares e desconhecidos.*

Diz-se ainda que entre a ocorrência de dois eventos consecutivos, o sistema permanece em um determinado *estado*. A ocorrência de um evento causa então uma *transição* ou *mudança de estado* no sistema. Assim, a evolução dinâmica de um SED pode ser representada graficamente pela trajetória percorrida no seu espaço de estados, conforme mostra o exemplo apresentado na Figura 1.1.

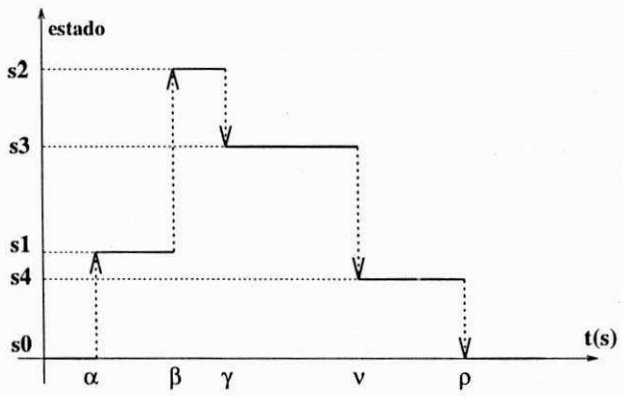


Figura 1.1: Exemplo da trajetória de um SED

Neste exemplo, os eventos são representados pelas letras $\alpha, \beta, \gamma, \nu$ e ρ. Note que um mesmo evento pode ocorrer a partir de diferentes estados, porém, assume-se que o número total de eventos diferentes que podem ocorrer é finito. Em geral, o número de estados pode ser ilimitado, embora a classe de sistemas com um número finito de estados seja um caso particular importante.

O *estado inicial* de um SED é o estado em que o mesmo se encontra antes da ocorrência do primeiro evento. Muitas vezes é desejável que o sistema, após a realização de uma tarefa especificada, sempre retorne a um determinado estado, denominado *estado de repouso* (*home state*). Neste estado, o sistema estará sempre pronto para iniciar outra tarefa. Se o estado de repouso for o estado inicial, a este processo denomina-se *reinicialização.*

No exemplo apresentado na Figura 1.1, s_0 representa o estado inicial, ou estado de repouso, por exemplo, de um sistema de comunicação de dados. Com a ocorrência do evento α (mensagem 1 recebida), o sistema muda para o estado s_1 (nó 1 transmitindo). Ocorrendo o evento β (mensagem 2 recebida), o sistema passa para o estado s_2 (nó 2 transmitindo). Após a ocorrência de γ (transmissão completada), o sistema está no estado s_3 (esperando reconhecimentos), mudando para o estado s_4 (reconhecimentos verificados) com a ocorrência de ν (reconhecimentos recebidos). Finalmente, com a ocorrência do evento ρ (sistema desativado) o sistema retorna ao seu estado de repouso.

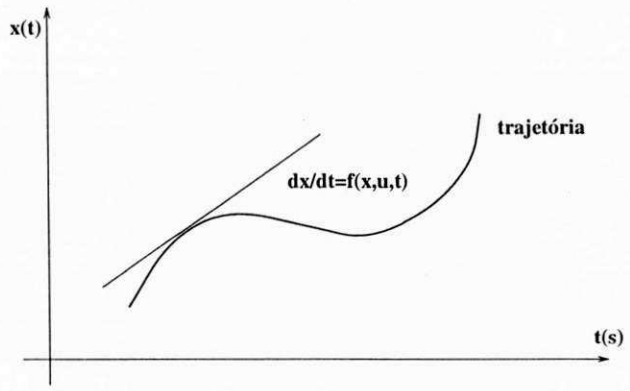


Figura 1.2: Exemplo da trajetória de um SDVC

1.1 Sistemas Discretos e Sistemas Contínuos

De acordo com a Definição 1.1, os SEDs, contrastam com os SDVCs, descritos por equações diferenciais. Assim, é ilustrativo comparar a trajetória de um SED, como aquela apresentada na Figura 1.1, com a trajetória de um SDVC, apresentada na Figura 1.2.

Observando estas duas figuras, nota-se que o espaço de estados de um SED é limitado a um conjunto enumerável, ao passo que o espaço de estados de um SDVC é contínuo. Estes, em geral, mudam de estado a cada instante, e seu comportamento é descrito por uma função que relaciona o estado (variável dependente) ao tempo (variável independente). Os SEDs, por sua vez, podem permanecer um tempo arbitrariamente longo em um mesmo estado, sendo que sua trajetória pode ser representada por uma seqüência de eventos na forma $\{\sigma_1, \sigma_2, \cdots\}$.

Para ilustrar, a trajetória apresentada na Figura 1.1 aplica-se a um sistema de manufatura cujos estados são: sistema inicialmente em repouso (s0); robô se movimentando para pegar uma peça (s1); robô se movimentando para colocar peça em um centro de processamento (s2); peça sendo processada (s3); robô se movimentando para entregar peça processada (s4); e finalmente o sistema retorna ao estado de repouso (s0). Como pode ser visto na Figura 1.1, a mudança de estados se dá pela ocorrência dos respectivos eventos. Por outro lado, a trajetória apresentada na Figura 1.2 poderia dizer respeito a um dos movimentos do robô.

De um modo geral, a teoria de SEDs trata do controle das atividades que um sistema

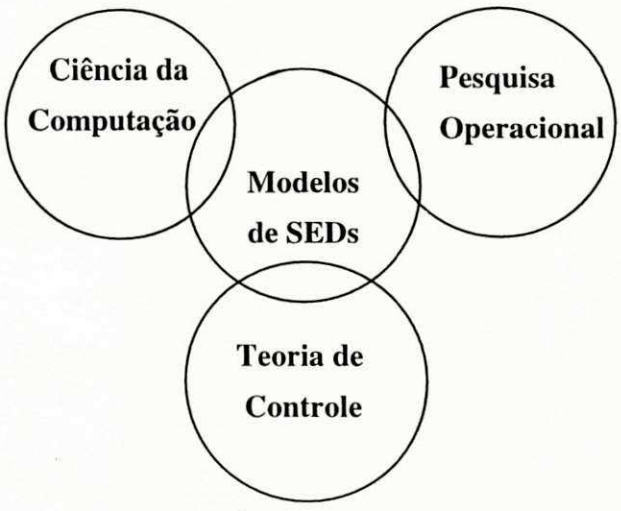


Figura 1.3: Áreas relacionadas à teoria de SEDs

pode executar, sem se preocupar com a realização física de cada atividade. Já a teoria de SDVCs se preocupa com a realização física dessas atividades.

Desta forma, podemos dizer que a tarefa de especificar o comportamento de um SED é estabelecer seqüências ordenadas de eventos que levam à realização de determinados objetivos. Comparados com os SDVCs, os SEDs são um fenômeno relativamente novo. Pode-se dizer que esse fenômeno pertence à área da Pesquisa Operacional, valendo-se de resultados da Ciência da Computação. A pesquisa operacional pode ser vista como a ciência das operações e eventos dos SEDs. Por outro lado, o estudo dos SEDs tem-se desenvolvido também com a ajuda da teoria de controle tradicional, visto que conceitos da dinâmica tais como constantes de tempo, resposta no domínio do tempo e da freqüência, controlabilidade e observabilidade têm cumprido um papel importante no desenvolvimento de modelos e ferramentas para os SEDs. Veja gráfico ilustrativo apresentado na Figura 1.3

Em resumo, como mostra o gráfico apresentado na Figura 1.3, as pesquisas nas áreas de modelagem, análise e controle de SEDs *existem* na intersecção das áreas de teoria de controle, pesquisa operacional e teoria de computação.

1.2 Modelos de SEDs

Os SEDs, por exemplo os modernos sistemas de manufatura, se caracterizam pela alta densidade de investimentos de capital e pela complexidade. Quanto maior o sistema, maior sua complexidade devido a alta integração de seus componentes de *hardware*, *software* e humano. Assim, a realização e a operação de um SED requer o emprego de metodologias de projeto sistematizadas capazes de determinar a melhor alternativa de projeto e a melhor política de gerenciamento e controle, garantindo assim, o comportamento que satisfaça às especificações lógicas e restrições temporais impostas ao SED. Mais ainda, erros cometidos na etapa de modelagem podem contribuir para prolongar o tempo de desenvolvimento, aumentar os custos e deteriorar a eficiência operacional do sistema.

As colocações acima destacam o elevado grau de importância que deve ser atribuído à etapa de modelagem de um SED.

Baseados nas várias propriedades, como aquelas mencionadas anteriormente, por exemplo, reinicialização, sincronização, concorrência, vários modelos foram propostos e desenvolvidos para SEDs sem que, no entanto, nenhum deles tivesse conseguido se afirmar como universal. Dentre eles podem ser citados [Ho89, ZD93]:

- Cadeias de Markov;
- Teoria das Filas;
- Processos Semi-Markovianos Generalizados (*GSMP*)
- Álgebra de Processos;
- Álgebra Max-Plus;
- Teoria de Linguagens Formais e Autômatos;
- Redes de Petri.

As cadeias de Markov são objeto de estudo da Teoria de Processos Markovianos, um dos capítulos mais importantes da Teoria de Processos Estocásticos [Pap65].

Dado um sistema caracterizado por um conjunto enumerável de estados (por exemplo, o conjunto dos números inteiros, ou um subconjunto deste) que pode ser observado em

vários instantes de tempo, o comportamento do sistema, a partir de um instante qualquer, independe dos estados passados. Apenas o estado atual influencia o comportamento futuro. Esta propriedade é denominada *propriedade Markov* [Pap65]. Os sistemas que apresentam tal propriedade são denominados *cadeias de Markov*. Conhecidas as probabilidades de transição de estado, para um dado sistema, o mesmo pode ser representado por uma cadeia de Markov.

As cadeias de Markov se adequam à modelagem de sistemas em que o interesse reside nas probabilidades de ocorrência dos eventos. Por exemplo quando se deseja fazer análise de desempenho. Um exemplo ilustrativo de utilização deste modelo para SEDs é apresentado por Cao e Ho [CH90].

Sistemas de filas são um caso particular da classe dos *sistemas de fluxo* [Kle75]. Num sistema de fluxos, objetos e/ou informações fluem, movem-se ou são transportados através de um ou mais canais de capacidade finita, de um ponto a outro do sistema. Sendo a capacidade dos canais do sistema limitada, podem ser formadas filas de espera pela liberação dos mesmos.

Para se descrever um sistema deste tipo, é necessário conhecer o processo de chegada das solicitações de transporte e a forma como estas solicitações são atendidas. Diferentes representações originam diversos tipos de filas, cada qual com características e aplicações próprias.

Os sistemas de filas encontram aplicação na análise de desempenho de SEDs. Um exemplo de modelagem de SEDs utilizando esta teoria, é apresentado por Cao e Ho [CH90].

A abordagem utilizando processos semi-markovianos generalizados, procura formalizar programas e linguagens para simulação de SEDs. As partes discretas do sistema, tais como o número e a localização de clientes e a quantidade de recursos disponíveis, são representados por estados. As partes contínuas, tais como o tempo restante de serviço, são chamados de tempo de vida dos eventos. O tempo de vida dos eventos é determinado da seguinte forma: para cada estado, tem-se um conjunto de eventos que podem ocorrer quando o sistema está naquele estado. Para cada evento, é gerado um tempo de vida exponencialmente distribuído. O tempo de vida de um evento especifica o tempo até a próxima ocorrência do mesmo, levando-se em conta que nenhum outro evento tenha

ocorrido durante este intervalo de tempo. Dentre os eventos associados a um estado, aquele que possuir o menor tempo de vida ocorrerá naquele estado. Após a ocorrência de um evento, o sistema se acomoda em um novo estado. Um novo conjunto de eventos e seus respectivos tempos de vida, são determinados e combinados com os tempos de vida restantes do estado anterior, para formar os novos tempos de vida dos eventos associados ao estado atual. O processo é então repetido e desta forma, o sistema evolui no tempo. Exemplos de aplicação podem ser vistos em [CH90] e [Gly89]

Dá-se o nome de processo a um conjunto de eventos que podem ocorrer em uma dada seqüência. Desta forma, uma seqüência de eventos gerada por um SED pode ser vista como um processo.

Ao se representar um SED por um conjunto de processos, é necessário definir um conjunto de operações sobre os processos que permitam representar a composição dos mesmos.

O emprego destas operações permite escrever expressões algébricas envolvendo processos, bem como demonstrar identidade entre as expressões. Um conjunto de operadores, axiomas e identidades entre tais expressões forma uma *álgebra de processos.*

Várias álgebras de processos têm sido desenvolvidas para o tratamento de SEDs, inspiradas nos trabalhos de Milner [Mil80] e Hoare [Hoa85].

Uma dessas abordagens, denominada *processos recursivos finitos* (PRFs) é apresentada, através de um exemplo ilustrativo em [CH90]. Os PRFs são baseados nos processos sequenciais de comunicação de Hoare. Dado um conjunto de eventos, um processo é constituído de três componentes: um conjunto de traços (*traces*) que o processo pode executar; uma função de eventos; e uma função de terminação. Uma das características dos PRFs é que cada processo pode ser descrito por um conjunto de equações recursivas. Assim, um sistema pode ser descrito com base nestas equações.

O modelo álgebra max-plus, proposto em [CDQV85], se baseia numa estrutura algébrica constituída de um conjunto e duas operações definidas sobre este conjunto.

As propriedades algébricas desta estrutura mostram-se úteis na descrição de SEDs, permitindo uma representação próxima à da teoria de controle para SDVCs. Em particular, o comportamento de SEDs que envolvem um conjunto de atividades repetitivas, pode ser caracterizado pela solução de uma equação matricial escrita nesta álgebra. Exemplos

de aplicação são apresentados em [CH90] e [CDQV85].

Os autômatos (máquinas de estados finitos) [HH79] e as linguagens a eles associadas têm sido amplamente utilizados para a modelagem e análise de SEDs [RW89]. Entretanto, implementar um SED, modelado por uma máquina de estados, se torna difícil por causa do aumento exponencial do número de estados do SED à medida que o mesmo cresce em sua estrutura, ou seja, aumenta a quantidade de seus componentes. Sua representação gráfica e consequentemente sua visualização se tornam difíceis.

As redes de Petri (RPs) [Pet81, BRA83, Rei85, Mur89] são uma ferramenta matemática e gráfica que apresentam um método unificado para a modelagem, análise e implementação física de sistemas. As RPs apresentam as seguintes vantagens sobre outros modelos, como aqueles citados anteriormente:

- Facilidade para modelar as características de um SED, ou seja: concorrência, sincronismo e assincronismo, conflito, exclusão mútua, relações de precedência, não determinismo e bloqueio;
- Excelente visualização de dependência de sistemas;
- Focalização na informação local;
- Abordagens de modelagem do tipo refinamento (*top-down*) e do tipo composição modular (*bottom-up*);
- Possibilidade de gerar o código de controle supervisório diretamente de sua representação gráfica;
- Possibilidade de testar propriedades indesejáveis para os sistemas, tais como bloqueio e reinicialização;
- Análise de desempenho sem simulação é possível para muitos sistemas. Taxas de produção, utilização de recursos, segurança e desempenho podem ser avaliadas;
- Simulação dos eventos discretos diretamente a partir do modelo;
- Informação do estado atual do sistema que permite monitorização em tempo real.

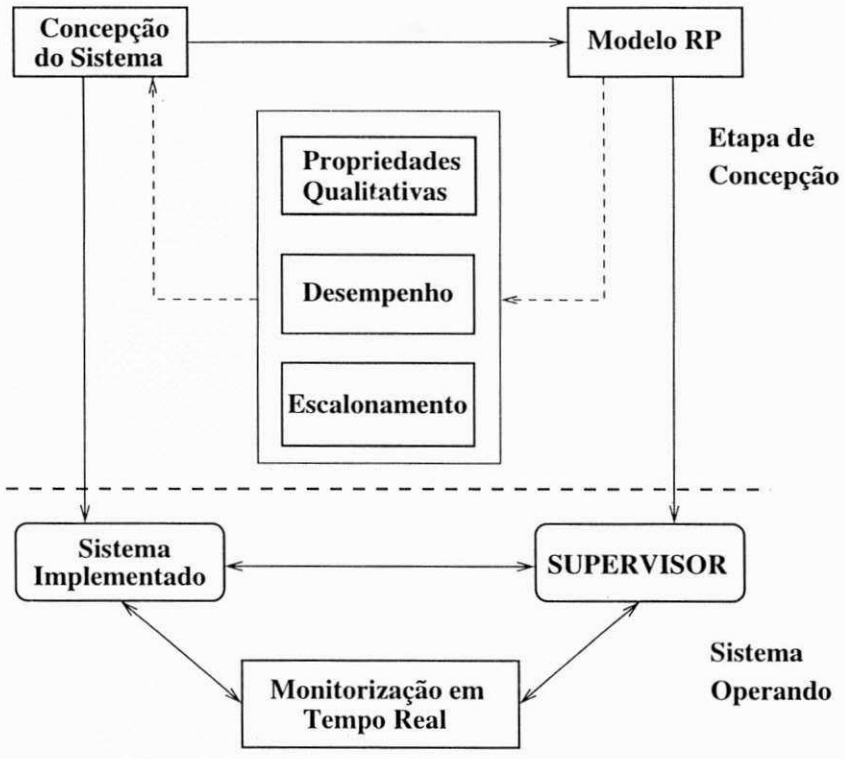


Figura 1.4: Representação de uma rede de Petri para modelagem, análise e controle de um SED.

A Figura 1.4, que reproduz o diagrama em blocos apresentado em [ZD93], mostra a utilização de redes de Petri na modelagem, análise e controle de um SED.

Quando se empregam redes de Petri (ou qualquer um dos outros modelos apresentados) na modelagem de SEDs, normalmente a lógica de controle já faz parte do modelo. Desta forma, modificações na estratégia de controle acarretam modificações na estrutura da rede que modela o sistema.

Holloway e Krogh [HK90] empregam as redes de Petri controladas para separar a lógica de controle do modelo, através do uso de transições controladas. Assim, modela-se um SED através de uma rede de Petri, livre de qualquer ação de controle. Este modelo permanece inalterado durante as etapas seguintes do projeto. A esta rede são agregados os chamados lugares de controle, os quais permitem inibir determinadas transições (ocorrência de eventos) às quais estão associados. Modificações na estratégia de controle implicam apenas em inibir ou habilitar tais transições.

As redes de Petri e as redes de Petri controladas serão apresentadas com mais detalhes no Capítulo 2.

1.2.1 Concepção de Controladores para SEDs

Existe uma diversidade de teorias para a concepção e projeto de sistemas de controle contínuos ou discretos no tempo para o controle de processos [TRA70, Kuo70, FP80, Kuo80, AW90, AW95]. Entretanto, a teoria de controle e implementação de SEDs só recentemente se tornou uma área de pesquisa. Paralelamente, as redes de Petri [Pet81, BRA83, Mur89] têm sido amplamente utilizadas para a modelagem, controle, simulação e análise de desempenho de SEDs, devido à facilidade das mesmas para representar os componentes de *hardware*, *software* e humano, bem como suas interações.

Para a concepção de controladores para os SEDs, mais especificamente para os sistemas de manufatura, estruturas de controle hierárquico têm sido discutidas. Segundo Zhou e DiCesare [ZD93], a estrutura de controle de um SED real pode ser vista como uma estrutura em dois níveis:

1. Controle supervisório ou controle relacionado a eventos e
2. Controle de processos tradicional.

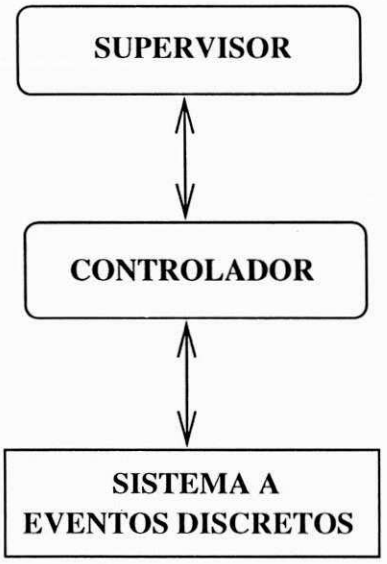


Figura 1.5: Arquitetura de controle de um SED real.

Tal estrutura de controle pode ser visualizada através de um exemplo simples da arquitetura de um sistema de manufatura, como apresentado na Figura 1.5.

Em um ambiente automatizado, um controle hierárquico pode ser implementado através de dois métodos básicos:

1. Controle centralizado, onde um computador hospedeiro é usado para realizar e sincronizar as atividades relacionadas aos eventos e controlar os processos de baixo nível;

2. Controle distribuído, onde os componentes de um sistema podem ser vistos como sub-sistemas independentes que se comunicam entre si, de modo a cooperativamente solucionar o problema de controle. Um controlador supervisório, ou simplesmente supervisor, tem que coordenar as tarefas assíncronas entre os componentes distribuídos para satisfazer os requisitos do sistema.

Mais recentemente, Ramadge e Wonham [RW88, RW89, RW87c, RW87b] têm desenvolvido uma teoria de controle para SEDs, denominada *Teoria de Controle Supervisório* (TCS) baseada em linguagens formais e autômatos, que é muito elegante e independente das ferramentas usadas para modelar o sistema. Na maioria das aplicações um SED é modelado por um autômato e seu comportamento é completamente descrito pela linguagem

gerada[1] pelo autômato. Os requisitos do sistema são também especificados através de linguagens formais. Assim, o problema da síntese do supervisor consiste em encontrar o autômato controlador tal que o autômato combinado (supervisor/sistema) gere a linguagem especificada. Um supervisor controla um SED habilitando ou desabilitando eventos controláveis.

Diferentemente de outras teorias e modelos de análise e síntese para os SEDs, a TCS tem a vantagem de separar explicitamente o sistema a ser controlado (*open loop dinamics*) do controlador em si (*feedback control*) e desta forma apresentar as condições necessárias e suficientes para a existência do supervisor.

Aproveitando as várias qualidades das redes de Petri, apresentadas anteriormente, e a elegância e simplicidade da teoria de controle supervisório, bem como sua independência em relação às ferramentas de modelagem, vários autores têm utilizado a TCS em associação com as redes de Petri como forma de contribuir com a teoria e alargar o campo das aplicações da TCS para sistemas reais mais complexos [GD91, GD92, GD94a, SK92, Sre93, BLP95c, BLP95b, BLP95a, BLP96c, BLP96b].

Para os SEDs, os problemas computacionais advindos da solução de síntese de supervisores são frequentemente complexos, pois o número de estados em um sistema real cresce exponencialmente com o número de constituintes do sistema [RW89]. Este problema pode ser minimizado com o uso de síntese modular [RW88, BLP95b, RW87a] e em alguns casos, restringindo a atenção às partes constituintes de interesse de um sistema [BLP95b, BLP96c].

Os modelos para SEDs podem ser classificados em temporizados e não temporizados, conforme permitam ou não considerar os instantes de tempo em que ocorrem os eventos, e ainda em algébricos, lógicos e de análise de desempenho, conforme sejam orientados ao uso, asserções lógicas ou à formulação probabilística, respectivamente. A tabela 1.1 baseada em [Ho89] e atualizada em [Zil93] apresenta os modelos baseada nos critérios acima citados.

[1] Ver Capítulo 2, Seção 2.1.

Modelos	Temporizados	Não temporizados
Lógicos	Autômatos Temporizados Redes de Petri Temporizadas	Autômatos Redes de Petri
Algébricos	Álgebra Max-Plus	Álgebra de Processos
Análise de Desempenho	Cadeias de Markov Teoria das Filas GSMP Redes de Petri Estocásticas	

Tabela 1.1: Classificação dos modelos de SEDs

1.3 Contribuições à TCS

A teoria de controle supervisório (TCS), desenvolvida por Ramadge e Wonham [RW89, RW87c, RW87b], utiliza autômatos e linguagens formais para a modelagem de sistemas a eventos discretos (SEDs). A TCS tinha como objetivo inicial o exame de conceitos tais como *controlabilidade*, *observabilidade* e controle hierárquico e descentralizado, de um ponto de vista qualitativo [RW89].

Provada sua utilidade na análise teórica de problemas básicos de controle supervisório, novas investigações estenderam o modelo para a solução do problema de controle para sistemas modulares [RW88, RW87a] e com restrições de tempo [LW88a, BW94].

Numa abordagem modular [RW88, RW87a], o sistema a ser controlado é composto de subsistemas modulares. O sistema global é obtido pela composição dos subsistemas. Os eventos (que podem ser controlados ou não por um agente externo ao sistema) são identificados a nível global. Assume-se que os eventos nos diferentes módulos não ocorrem concorrentemente.

A especificação de um sistema modular, nesta mesma abordagem, é dada por uma coleção de linguagens, cada uma representando uma especificação para o sistema controlado. O problema de síntese do supervisor é solucionado para cada subsistema e os supervisores resultantes são então compostos para formar a solução global para o problema especificado. Além do fato de ser mais facilmente solucionado, o supervisor modular é

mais fácil de ser modificado, atualizado e mantido. Por exemplo, se a especificação de um módulo é modificada, então é necessário modificar somente o controlador do módulo correspondente, em vez do supervisor global.

Infelizmente, nem sempre, nesta abordagem, a composição de supervisores produz a solução desejável para o sistema global, visto que algumas condições necessárias para a existência do supervisor não são, muitas vezes, preservadas. Mais ainda, cada subsistema é modelado por um autômato, assim, o número de estados do sistema global aumenta exponencialmente com o número de subsistemas do mesmo. Este fato, segundo Balemi et al. [BHG+93], é desastroso quando se trabalha com sistemas reais devido ao conhecido problema da explosão de estados. Uma aplicação de síntese de supervisor utilizando esta abordagem é apresentada em [BHG+93].

Brandin e Wonham [BW94] introduziram, na estrutura da TCS, restrições de tempo para a ocorrência de eventos, ou seja, os mesmos só podem ocorrer dentro de um certo limite de tempo. Restrições de tempo são de considerável interesse para a modelagem e controle de SEDs reais, mas introduzem um aumento na complexidade dos mesmos [BW94]. Esta abordagem associa ao autômato que modela o sistema as características dos *modelos de transição temporizada*, definidos por Ostroff [OW90], incorporando atrasos e tempo máximo para a ocorrência de cada evento. O modelo suporta ainda a composição modular de subsistemas. Segundo Ostroff e Wonham, como o sistema e o supervisor são modelados por autômatos, torna-se difícil a utilização desta abordagem para sistemas reais, embora seja de significativa importância a contribuição téorica para a teoria de controle supervisório.

Lin e Wonham [LW88b] propõem uma solução para a síntese de supervisores de sistemas que apresentam um conjunto de eventos particionados em eventos observáveis e eventos não observáveis. A síntese é baseada no conjunto de eventos observáveis. Esta abordagem utiliza autômatos finitos e linguagens regulares para modelar os SEDs e é baseada no fato de que nem todos os eventos em um sistema real são observáveis.

Como os modelos dos sistemas apresentados, pelos vários autores citados acima, são expressos por autômatos e linguagens formais, os problemas de controle tratados são de exemplos simples, visto que a análise de sistemas reais modelados por autômatos se torna difícil pelos motivos já citados anteriormente.

Vários outros autores têm investigado formalismos alternativos para a modelagem e controle de SEDs. Um dos formalismos alternativos utilizados são as redes de Petri pela sua facilidade de modelar SEDs. Por este motivo, redes de Petri têm sido utilizadas na TCS, por exemplo Inan e Varayia [IV88], Giua e DiCesare [GD91, GD92, GD94a], Sreenivas e Krogh [SK92], Sreenivas [Sre93], Holloway e Krogh [HK90], Boel et. al. [BLNB95] e Barroso et. al. [BLP95c, BLP96a]. Podem-se enumerar várias vantagens de se utilizar redes de Petri em vez de modelos mais simples tais como máquinas de estado, dentre elas:

- a estrutura de uma rede não requer uma enumeração explícita de todos os estados do sistema e em muitos casos podem ser utilizadas técnicas de análise, baseadas somente na estrutura da rede, que permitem uma boa solução para o problema de controle;
- o modelo de rede de Petri cresce linearmente com o número de componentes do sistema modelado;
- uma rede de Petri supervisora contém sua ação de controle implícita na própria estrutura. Nesse caso o modelo em malha fechada do sistema controlado pode ser construído e analizado usando-se as mesmas técnicas de análise de redes de Petri que são usadas para se analizar o sistema.

Sreenivas e Krogh [SK92] consideram uma classe de problemas de controle supervisório que requerem supervisores de estados infinitos. Utilizando redes de Petri, em vez de autômatos, o supervisor de estados infinitos pode ser então representado por uma rede de Petri, que é sempre finita. Sreenivas e Krogh [SK92] mostram que, neste caso, o ganho na modelagem é acompanhada por uma desvantagem, ou seja, em geral a propriedade de *controlabilidade*[2] de uma linguagem gerada pelo sistema em malha fechada (supervisor/sistema) não é decidível.

Sreenivas [Sre93] mostra que a controlabilidade de uma linguagem em relação a outra é decidível quando se utiliza uma classe de redes de Petri denominada *Redes de Petri*

[2]Este conceito é apresentado no Capítulo 3.

Livremente Etiquetadas[3]. Sreenivas mostra também que sistemas de estados infinitos podem ser modelados por esta classe de redes.

Giua e DiCesare [GD93] estendem a classe de problemas de controle que podem ser modelados por redes de Petri, apresentando uma noção de comportamento terminal para redes de Petri denominado *comportamento frágil* (*weak behavior*). As redes de Petri que apresentam este comportamento, geram uma classe de linguagens denominadas *linguagens frágeis* (*weak languages*). Um importante resultado quando se utiliza esta classe de linguagens é que a propriedade de controlabilidade de uma linguagem em relação a outra é decidível.

As redes de Petri, que modelam sistemas reais, são limitadas (uma subclasse de redes de Petri cujo conjunto de marcações alcançáveis é finito). A classe de linguagens gerada por esse modelo é equivalente à classe de linguagens regulares e todos os resultados obtidos pela TCS, em relação às condições de existência de supervisores, para as linguagens regulares se aplicam às linguagens geradas por redes de Petri limitadas [GD94a]. Sendo assim, é possível derivar as condições necessárias e suficientes para a existência de supervisores modelados por essas redes.

Utilizando redes de Petri limitadas como modelos de SEDs, várias abordagens têm sido desenvolvidas para a solução do problema de síntese do supervisor utilizando a teoria de controle supervisório.

Giua e DiCesare [GD91] utilizam redes de Petri com capacidade finita como modelo do sistema e do supervisor. A metodologia por eles utilizada requer que a síntese do supervisor seja realizada através da composição de subredes e de um algoritmo de refinamento do supervisor para torná-lo *não bloqueável*[4]. Esse refinamento é realizado através da inclusão de novos arcos e, eventualmente, duplicação de transições no modelo preliminar do supervisor. Nesse caso, embora exista uma sistemática de síntese do supervisor, a sua utilização para sistemas reais mais complexos se torna difícil devido à necessidade de se modificar a estrutura da rede supervisora para que a mesma não alcance estados que não satisfaçam às especificações desejáveis.

Giua e DiCesare [GD94b] utilizam uma clase de redes de Petri denominada *Máquinas*

[3]Uma rede de Petri livremente etiquetada associa eventos distintos e não nulos, de um dado alfabeto, a transições distintas [Pet81], esta classe de redes de Petri é apresentada no Capítulo 2.

[4]Este conceito é apresentado no Capítulo 3.

de Estado Elementares Compostas (Elementary Composed State Machines)[5]. Baseada na estrutura da rede, é utilizada a *Técnica de Programação de Inteiros* para a validação do supervisor.

Embora mais tratável que o método de construção do espaço de estados da rede, a técnica de programação de inteiros nem sempre tem solução em tempo polinomial. Entretanto, Giua e DiCesare mostram que é possível usar técnicas de programação linear para derivar as condições suficientes para a validação do supervisor.

Note que essa abordagem é limitada pelo tipo de rede utilizada para a modelagem do sistema, visto que são restritos os supervisores de SEDs reais que podem ser modelados por essa classe de redes. Nessa abordagem, o problema de controle não é diretamente solucionado, pois embora exista um procedimento para a validação do supervisor, se o modelo não possui as propriedades desejáveis, a metodologia não leva diretamente à construção de um supervisor próprio.

Boel et. al. [BLNB95] estudam o *problema de estados proibidos* para uma classe de SEDs modelados por *Máquinas de Estado Controladas*, uma classe de redes de Petri onde transições podem ser habilitadas ou desabilitadas por entradas externas binárias (*lugares de controle*). O problema consiste em sintetizar supervisores para evitar que um conjunto de estados não desejáveis, denominados *estados proibidos*, sejam alcançados.

A abordagem não permite a representação de certos conjuntos de estados proibidos, visto que o conjunto de estados permitidos possui propriedades que excluem especificações tais como aquelas em que o número de fichas em um subconjunto de lugares é pelo menos igual ou exatamente igual a um dado valor. Assim, as marcações proibidas são caracterizadas por conjuntos de restrições obtidas a partir de uniões ou intersecções de restrições mais simples, as quais expressam que certos lugares da rede não podem conter mais que uma determinada quantidade de fichas.

Conceitos tais como *zonas de influência, transições de influência, lugares de estado de influência* e *redução de modelos* são utilizados para a construção de algoritmos de síntese do supervisor, baseados na estrutura da rede, ou seja, na estrutura das máquinas de estado controladas.

[5]Máquinas de estado são uma classe de redes de Petri cujas transições possuem apenas um arco de entrada e um arco de saída.

Holloway e Krogh [HK90] utilizam uma classe de redes de Petri controladas, denominadas *Grafos Marcados*[6] *Cíclicos Controlados*, com uma ficha por ciclo[7], para a solução do problema de habilitar e desabilitar transições controláveis tal que lugares pertencentes a um conjunto de lugares proibidos nunca sejam marcados simultaneamente.

Devido à estrutura dos grafos marcados, a classe de problemas considerados por Holloway e Krogh [HK90] é relativa à exclusão mútua e utiliza a metodologia de construção modular do controle supervisório considerada por Ramadge e Wonham [RW88].

A metodologia utiliza a estrutura do grafo marcado que modela o sistema para construir o supervisor sem a necessidade da geração e análise do espaço de estados do sistema. Nesse sentido, o algoritmo de construção do supervisor é eficiente em relação àqueles que utilizam os métodos de análise do espaço de estados para esta classe de redes de Petri.

A restrição do método está no fato de que o mesmo se aplica aos grafos marcados cíclicos com restrições quanto ao seu estado inicial.

Assim, a solução para o problema de estados proibidos, baseado na estrutura da rede, é particularmente baseado na forma como os padrões de controle influenciam as marcações dos lugares de estado contidos nos *caminhos de lugares* que começam nos lugares que sao saída de transições controladas.

1.4 Apresentação da Tese

Este trabalho apresenta uma nova abordagem para a solução do problema de síntese de supervisores, usando a Teoria de Controle Supervisório e as Redes de Petri. A técnica é baseada em uma extensão de uma classe de redes de Petri denominada *Redes de Petri com Funções de Habilitação de Transições* (*RPFHT*)[PC92]. Dois algoritmos são apresentados. O primeiro é baseado no conhecido *Algoritmo da Árvore de Alcançabilidade* [Pet81, Mur89]. O segundo, é baseado nos algoritmos apresentados por Ramadge e Wonham [RW87b] e Ziller [ZC94]. A síntese do supervisor é realizada executando-se os algoritmos tendo como dados o modelo do sistema e seu comportamento desejável.

[6]Grafos marcados são uma classe de redes de Petri cujos lugares possuem apenas um arco de entrada e um arco de saída.

[7]Um ciclo em um grafo marcado é um caminho começando e terminando em um mesmo nó, com todos os outros nós no caminho ocorrendo uma única vez.

A utilização do *Algoritmo da Árvore de Alcançabilidade* [Pet81, Mur89] para a análise de sistemas a eventos discretos é impraticável para a grande maioria dos sistemas reais, devido ao grande número de estados que os mesmos podem alcançar, inviabilizando sua análise. Para melhor gerenciar este problema, apresentamos também uma abordagem modular baseada em *Sistemas de G-Nets* [PdFC94, Per94] e *Redes de Petri Coloridas* [Jen92].

Utilizamos ainda essa abordagem para a síntese de supervisores com propriedades de segurança e tolerância a falhas dependentes e não dependentes do tempo. Baseados no fato de que é necessário antecipar, ainda na etapa de projeto, as possíveis falhas e erros mais comuns ao sistema, é possível providenciar as ações de controle necessárias para que o sistema possa evitar estas falhas ou, em última análise, se recuperar automaticamente caso as mesmas ocorram.

Para a introdução de características de tolerância a falhas dependentes do tempo, são impostas restrições de tempo à abordagem modular, através da utilização de uma classe de redes de Petri denominada *Redes de Petri com Temporização Nebulosa* (*RPTN*) (*Fuzzy Time Petri Nets*), desenvolvidas por Figueiredo [dF94, dF95]. Ao modelo de sistemas de *G-Nets* acrescentam-se parâmetros de tempo para que se possa fazer análise de tempos e que falhas dependentes do tempo possam ser toleradas. Assim, a estrutura interna de cada *G-Net* será modelada por uma *RPTN*. O supervisor, sendo executado sincronamente com o sistema, poderá detectar falhas no comportamento do mesmo e se possível, através de ações de controle previamente estabelecidas, fazer com que o sistema volte a operar segundo as especificações desejadas.

Esta Tese está estruturada da seguinte forma: No Capítulo 2 serão apresentados os conceitos básicos de linguagens formais e autômatos, redes de Petri, redes de Petri controladas, redes de Petri coloridas, sistemas de *G-Nets* e redes de Petri com temporização nebulosa. No Capítulo 3 são introduzidos os conceitos básicos e aplicações da teoria de controle supervisório. Nos Capítulos 4 e 5 estabelecemos as direções da pesquisa de que trata este trabalho. No Capítulo 4 apresentamos uma nova abordagem para a solução do problema de síntese do supervisor, bem como uma abordagem modular para a solução de síntese do supervisor de sistemas mais complexos. No Capítulo 5 são incorporadas ao supervisor propriedades de segurança e tolerância a falhas dependentes e não dependentes

do tempo. Por fim apresentamos as conclusões no Capítulo 6.

Capítulo 2

Definições Gerais

Neste Capítulo serão apresentados os conceitos básicos de linguagens formais e autômatos [HH79], bem como a diferença entre autômatos e geradores. Serão apresentados também os conceitos básicos de redes de Petri, redes de Petri controladas, redes de Petri coloridas, *G-Nets* e sistemas de *G-Nets* e redes de Petri com temporização nebulosa.

2.1 Linguagens Formais, Autômatos e Geradores

2.1.1 Linguagens Formais

Um símbolo, designado por σ, é uma entidade não definida formalmente. Letras e dígitos são exemplos típicos de símbolos. A partir de um conjunto de símbolos, um *Alfabeto* é assim definido:

Definição 2.1 Alfabeto *é um conjunto não vazio de símbolos, representado por uma letra grega maiúscula, em geral* Σ.

$\Sigma = \{\alpha, \beta, \lambda\}$ é um exemplo de um alfabeto formado pelos símbolos, ou letras gregas minúsculas, α, β e λ.

De forma análoga à escrita habitual, a justaposição de símbolos de um alfabeto é denominada *palavra*.

Definição 2.2 Palavra *sobre o Alfabeto* Σ *é qualquer justaposição de um número finito de símbolos (com ou sem repetição) de* Σ, *na forma* $\sigma_1, \sigma_2, \cdots, \sigma_k$, *com* $\sigma_i \in \Sigma$ *para* $i = 1, 2, \cdots, k$.

Definição 2.3 *O* comprimento *de uma palavra s, representado por* $|s|$, *é igual ao número de símbolos que a compõem.*

Definição 2.4 *A* palavra nula, *representada por* ϵ, *é a única palavra de comprimento nulo.*

Vale ressaltar que $\epsilon \notin \Sigma$, pois ϵ é uma palavra e não um símbolo de Σ.

Conjuntos de palavras são estruturas fundamentais para a Teoria de Linguagens. Assim, define-se um conjunto de palavras como:

Definição 2.5 *Dado* $k \in \mathbb{N}$, *denota-se por* Σ^k *o conjunto de todas as palavras sobre* Σ *cujo comprimento é igual a* k.

Por exemplo, dado o alfabeto $\Sigma = \{\alpha, \beta\}$, tem-se:
$\Sigma^0 = \{\epsilon\}$
$\Sigma^1 = \{\alpha, \beta\}$
$\Sigma^2 = \{\alpha\alpha, \alpha\beta, \beta\alpha, \beta\beta\}$

Definição 2.6 *Dado um alfabeto* Σ, *definem-se dois conjuntos especiais,* Σ^+ *e* Σ^* *por:*

$$\Sigma^+ = \bigcup_{k=1}^{\infty} \Sigma^k = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$
$$\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

Interpreta-se Σ^+ como o conjunto de todas as palavras que podem ser formadas com os símbolos do alfabeto Σ. Σ^* difere de Σ^+ apenas por incluir a palavra nula , ou seja, $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$.

Um conjunto de palavras formadas com símbolos de um alfabeto é denominado *linguagem*, formalmente:

Definição 2.7 *Dado um alfabeto* Σ, L *é uma* linguagem *sobre* Σ *se e somente se* $L \subseteq \Sigma^*$.

Esta definição implica que tanto $\emptyset$ (linguagem vazia) quanto Σ^* são linguagens. Note que a linguagem vazia, $\emptyset = \{\}$, é diferente da linguagem formada apenas pela palavra nula $\Sigma^0 = \{\epsilon\}$. Σ^* é o conjunto de todas as palavras formadas com os símbolos de um alfabeto Σ.

Para a composição de palavras, é necessário a seguinte definição:

Definição 2.8 *Dadas duas palavras s_1 e s_2 sobre um alfabeto Σ, com $s_1 = \sigma_1\sigma_2\cdots\sigma_k$ e $s_2 = \sigma_{k+1}\sigma_{k+2}\cdots\sigma_n$, a concatenação de s_1 e s_2, denotada por s_1s_2, é dada por*

$$\sigma_1\sigma_2\cdots\sigma_k\sigma_{k+1}\sigma_{k+2}\cdots\sigma_n.$$

Quando se deseja referenciar uma parte inicial de uma palavra de comprimento arbitrário, utiliza-se a noção de *prefixo.*

Definição 2.9 Prefixo *de uma palavra* s *sobre um alfabeto Σ é qualquer palavra $u \in \Sigma^*$ que possa ser completada com outra palavra $v \in \Sigma^*$ para formar a palavra s.*

Por exemplo, seja o alfabeto $\Sigma = \{\alpha, \beta, \gamma\}$, $u = \alpha\beta$ é um prefixo de $s = \alpha\beta\alpha\gamma\beta$, porque $\exists(v = \alpha\gamma\beta) \in \Sigma^*$ tal que $uv = s$.

Definição 2.10 *Dada uma palavra s, denota-se por $Pre(s)$ o conjunto de todos os prefixos de s, incluindo a palavra vazia ϵ.*

Note que a palavra $s = \alpha\beta\alpha\gamma\beta$ é prefixo de si mesma pois a igualdade $\alpha\beta\alpha\gamma\beta\epsilon = \alpha\beta\alpha\gamma\beta$ satisfaz a condição da Definição 2.9.

Dada uma linguagem $L \subseteq \Sigma^*$, existe uma linguagem associada a L, formada pelas palavras de L e por todos os seus prefixos, formalmente:

Definição 2.11 *O* prefixo-fechamento *ou simplesmente* fechamento *de L, é dado por:*

$$\overline{L} = \{u : \exists v \in \Sigma^* \wedge \ uv \in L\}$$

Como conseqüência desta definição tem-se que $L \subseteq \overline{L}$.

Existe uma classe importante de linguagens que podem ser expressas pelas chamadas expressões regulares [HH79]. Tais linguagens são denominadas *linguagens regulares.* As expressões regulares são seqüências de símbolos obtidas pela aplicação repetitiva de um conjunto de regras de formação. A seguir são apresentadas algumas informações suficientes para a compreensão de algumas expressões utilizadas ao longo do texto.

- σ^n e s^n representam, respectivamente, a repetição do símbolo σ e da palavra s por n vezes;
- σ^* e s^* representam, respectivamente, a repetição do símbolo σ e da palavra s um número arbitrário de vezes, inclusive zero.
- o símbolo + é empregado como o operador lógico *ou*, indicando uma opção entre duas ou mais possibilidades.

Esta notação permite escrever representações finitas para linguagens formadas por um número infinito de palavras.

Por exemplo, sejam $\Sigma = \{\alpha, \beta\}$ e L uma linguagem prefixo-fechada onde os símbolos α e β ocorram alternadamente, sendo que α ocorre primeiro. A linguagem L pode ser assim representada

$$L = \{\epsilon, \alpha, \alpha\beta, \alpha\beta\alpha, \alpha\beta\alpha\beta, \ldots\} = \{(\alpha\beta)^*(\alpha + \epsilon)\}$$

Lê-se: $(\alpha\beta)$ pode ocorrer um número arbitrário de vezes, inclusive zero; logo após pode ocorrer α ou então a palavra nula ϵ, ou seja, nada. Note que L é um subconjunto próprio da linguagem $\Sigma^* = \{\alpha^*\beta^*\}^*$, a qual inclui as palavras $\beta\alpha\beta\alpha\beta$ e $\alpha\alpha\beta$, que não são palavras da linguagem L.

2.1.2 Autômatos

Um autômato é um modelo matemático de máquinas, com entradas e saídas discretas, que reconhece um conjunto de palavras sobre um dado alfabeto. Um autômato pode ser visto como uma entidade de controle que lê sequencialmente uma fita de símbolos de um dado alfabeto e aceita uma palavra sempre que a mesma pertencer ao conjunto de palavras reconhecidas.

Em um dado instante o modelo se encontra numa determinada configuração interna denominada de estado do modelo. O estado atual sumariza as informações de estados passados necessários à determinação de futuros estados. Um autômato pode ser finito ou infinito, determinístico ou não determinístico.

Um autômato finito consiste de um conjunto finito de estados, designado por Q, e um conjunto de transições de estado, designado por δ, que ocorrem a partir de símbolos de entrada escolhidos de um alfabeto Σ. O *estado inicial* do autômato, designado por q_0, é o estado em que este se encontra antes de ler o primeiro símbolo da fita. Alguns estados do autômato são designados por estados finais ou *estados marcados* Q_m. Um autômato reconhece exatamente as palavras da fita que, quando processadas, o levam a um estado marcado. De outra forma, um autômato que possui infinitos estados é designado por autômato infinito.

Se para cada símbolo de entrada existe exatamente uma transição de saída de cada estado, diz-se que o autômato é determinístico. De outra forma, se existem duas ou mais transições de saída de um estado que podem ocorrer a partir do mesmo símbolo, então o autômato é não determinístico.

Define-se formalmente um autômato da seguinte forma:

Definição 2.12 *Um* autômato determinístico finito *ou simplesmente* autômato *é uma quíntupla* $A = (Q, \Sigma, \delta, q_0, Q_m)$ *onde:*

- Q *é o conjunto finito de estados* q*;*
- Σ *é o alfabeto ou conjunto de símbolos* σ*;*
- $\delta : \Sigma \times Q \rightarrow Q$ *é a função de transição de estados;*
- $q_0 \in Q$ *é o estado inicial e*
- $Q_m \subseteq Q$ *é o conjunto de estados marcados.*

Definição 2.13 *Dado um autômato* $A = (Q, \Sigma, \delta, q_0, Q_m)$*, a* função de transição $\delta : \Sigma \times Q \rightarrow Q$ *é tal que:*

$$\delta(\epsilon, q) = q \text{ e}$$
$$\delta(\sigma, q) = q', \text{ para } q, q' \in Q \text{ e } \sigma \in \Sigma.$$

diz-se que q' *é um estado do autômato* A *se* σ *é uma entrada aceita por* A*.*

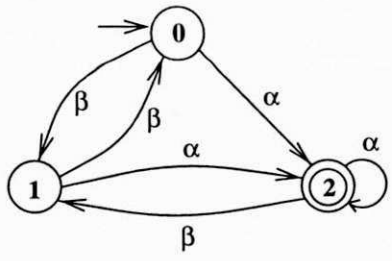


Figura 2.1: Representação gráfica de um autômato.

Graficamente, um autômato é representado por um grafo direcionado cujos vértices representam os estados e cujos arcos representam a função de transição. Além disso, estados marcados são representados por vértices desenhados com linhas duplas e o estado inicial é identificado por uma seta.

Como exemplo, seja um autômato com

$$\Sigma = \{\alpha, \beta\};$$
$$Q = \{0, 1, 2\};$$
$$\delta(\alpha, 0) = 2, \delta(\beta, 0) = 1, \delta(\alpha, 1) = 2, \delta(\beta, 1) = 0, \delta(\alpha, 2) = 2, \delta(\beta, 2) = 1;$$
$$q_0 = 0 \text{ e}$$
$$Q_m = \{2\}.$$

A representação gráfica correspondente é apresentada na Figura 2.1.

Conhecidas a função de transição e o estado atual de um autômato, é possível determinar seu estado após processar um dado símbolo. Em se desejando processar uma cadeia de símbolos, ou seja, uma palavra, pode-se simplesmente repetir este procedimento para cada um dos símbolos que a compõem. No entanto, é conveniente estender a definição da função de transição para processar palavras.

Definição 2.14 *Dado um autômato* $A = (Q, \Sigma, \delta, q_0, Q_m)$, *a* função de transição estendida, *denotada por* δ^*, *é a função* $\delta^* : \Sigma^* \times Q \to Q$ *tal que:*

$$\delta^*(\epsilon, q) = q \text{ e}$$
$$\delta^*(s\sigma, q) = \delta(\sigma, \delta^*(s, q)), \text{ para } q \in Q \text{ e } s \in \Sigma^*.$$

Como $\delta^*(\sigma, q) = \delta(\sigma, \delta^*(s, q)) = \delta(\sigma, q)$, para $s = \epsilon$ na Definição 2.14, então, por conveniência, usaremos δ no lugar de δ^* no restante deste trabalho.

Definida esta função, pode-se determinar qual será o estado do autômato após processar uma palavra sobre seu alfabeto, a partir de um dado estado. Dependendo da palavra escolhida, o estado alcançado poderá ou não pertencer ao conjunto de estados marcados. Isso nos leva à seguinte definição:

Definição 2.15 *Dado um autômato* $A = (Q, \Sigma, \delta, q_0, Q_m)$, *diz-se que* A reconhece *a palavra* $s \in \Sigma^*$ *se e somente se* $\delta(s, q_0) \in Q_m$.

O autômato apresentado na Figura 2.1 reconhece todas as palavras sobre o alfabeto $\Sigma = \{\alpha, \beta\}$ que terminam com o símbolo α.

Formalmente, o conjunto de palavras aceitas, ou reconhecidas, por um autômato A, é assim definido:

Definição 2.16 *Dado um autômato* $A = (Q, \Sigma, \delta, q_0, Q_m)$, *a* linguagem marcada *(ou* linguagem aceita *ou ainda* linguagem reconhecida*) de* A, *denotada por* $L_m(A)$, *é:*

$$L_m(A) = \{s : s \in \Sigma^* \wedge \delta(s, q_0) \in Q_m\}.$$

O autômato da Figura 2.1 reconhece a linguagem $\{(\alpha^*\beta^*)^*\alpha)$

2.1.3 Geradores

Considere a trajetória do SED apresentada na Figura 1.1 e reapresentada na figura 2.2.

Associa-se a noção de alfabeto ao conjunto de eventos e pode-se usar expressões tais como *alfabeto de eventos* ou *um evento do alfabeto* Σ. Por exemplo, o alfabeto de eventos associado ao sistema cuja trajetória está representada na Figura 2.2 é o conjunto $\Sigma = \{\alpha, \beta, \gamma, \nu, \rho\}$.

Numa descrição não temporizada, a trajetória do sistema pode ser representada pela seqüência de eventos, ou palavra, $\alpha\beta\gamma\nu\rho$. Diz-se, portanto, que um SED produz, ou *gera*, palavras de comprimento crescente à medida que evolui.

Em geral um SED não gera qualquer seqüência de eventos sobre o seu alfabeto associado Σ. Escolhendo arbitrariamente símbolos de Σ, é possível se construir palavras que

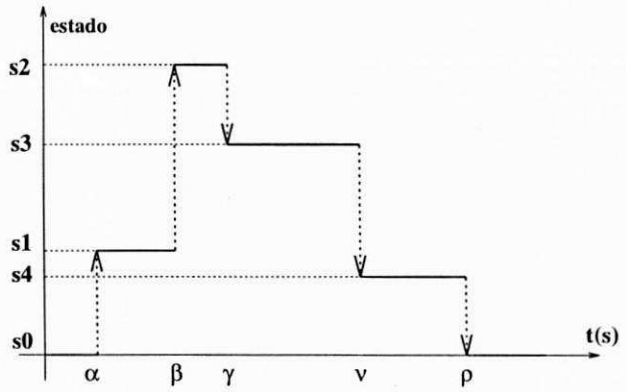


Figura 2.2: Exemplo da trajetória de um SED.

não correspondam a seqüências de eventos fisicamente possíveis no SED. Isso significa que, em geral, a linguagem gerada pelo sistema é um subconjunto próprio de Σ^*.

Considerando a evolução sequencial dos SEDs, pode-se afirmar que, se um dado sistema produziu uma palavra qualquer s, então produziu anteriormente todas as palavras do conjunto $Pre(s)$(Definição 2.10). Assim, a linguagem gerada por um SED incluirá, para cada palavra, também os seus prefixos. Isto nos leva a afirmar que o comportamento lógico de qualquer SED em que não ocorram eventos simultâneos, pode ser representado por uma linguagem prefixo-fechada.

Definição 2.17 *A linguagem prefixo-fechada que representa o comportamento lógico de um SED é denominada de* linguagem gerada *do sistema.*

É desejável que um SED, partindo de seu estado inicial e percorrendo uma determinada trajetória em seu espaço de estados, complete uma ou mais tarefas. As seqüências de eventos (palavras) que levam à realização de tarefas também formam uma linguagem.

Definição 2.18 *A linguagem que representa o conjunto de tarefas que um SED é capaz de executar é denominada* linguagem marcada *do sistema.*

Note que uma linguagem marcada de um SED não é necessariamente prefixo-fechada. Note também que, para gerar qualquer palavra dessa linguagem, o SED tem que gerar todos os seus prefixos. Assim, um SED que produza as palavras contidas numa linguagem marcada L_m, também produzirá as palavras contidas em $\overline{L}_m$.

Se L é a linguagem gerada por um sistema e L_m sua linguagem marcada, então $L_m \subseteq \overline{L_m} \subseteq L = \overline{L}$.

Conclui-se que é possível representar um SED de linguagem marcada L_m por um autômato A tal que $L_m(A) = L_m$. Também seria desejável representar a linguagem gerada pelo sistema por um autômato. No entanto, os autômatos, como definidos anteriormente, não são capazes de tal representação. A solução para este problema está em se permitir que a função de transição δ seja definida apenas para alguns pares (evento, estado) do conjunto $\Sigma \times Q$. Em geral δ é uma função parcial, pois para cada $q \in Q, \delta(\sigma, q)$ é definida somente para algum subconjunto $\Sigma(q) \subset \Sigma$ que depende de q. Utiliza-se, então, o termo *gerador* para denominar esta classe de máquinas, como definido a seguir.

Definição 2.19 *Um* gerador *é uma quíntupla* $G = (Q, \Sigma, \delta, q_0, Q_m)$, *onde* Q, Σ, q_0 *e* Q_m *já foram definidos anteriormente (veja Definição 2.12) e:*

- $\delta : \Sigma \times Q \to Q$ *é a função (em geral parcial) de transição de estados;*

A única diferença entre geradores e autômatos, é que nos primeiros a função de transição pode ser parcial, ou seja, definida apenas para um subconjunto de eventos para cada estado do gerador, enquanto que para os autômatos, a função de transição é total, ou seja, definida para todo par $(\sigma, q) \in \Sigma \times Q$.

Definição 2.20 *Dado um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$, *associa-se a cada estado* $q \in Q$ *o* conjunto de eventos, definidos $\Sigma(q)$, *dado por:*

$$\Sigma(q) = \{\sigma : \sigma \in \Sigma \wedge \delta(\sigma, q)!\}, \text{ onde}$$

$\delta(\sigma, q)!$ *significa que* δ *é definido para o par* (σ, q).

Como a função de transição δ é somente parcial, nesse caso, a função de transição estendida δ^* só é definida para aquelas palavras que, quando processadas pelo gerador, se mantiverem dentro dos limites de definição da função de transição.

Definição 2.21 *Dado um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$, *a função de transição estendida, denotada por* δ^*, *é uma função* $\delta^* : \Sigma^* \times Q \to Q$ *tal que:*

$$\delta^*(\epsilon, q) = q \text{ e}$$

$\delta^*(s\sigma, q) = \delta(\sigma, \delta^*(s, q))$, *para* $q \in Q$ *e* $s \in \Sigma^*$ *sempre que*

$q' = \delta^*(s, q)$ *e* $\delta(\sigma, q')$ *estiverem ambos definidos.*

Da mesma forma, como feito para os autômatos e pelas mesmas razões, a função de transição estendida será denotada simplesmente por δ.

Dada a limitação da função de transição estendida, o conjunto de palavras que o gerador pode processar forma uma linguagem, assim definida:

Definição 2.22 *Dado um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$, *a* linguagem gerada de G, *denotada por* $L(G)$, *é:*

$$L(G) = \{s : s \in \Sigma^* \wedge \delta(s, q_0)!\}.$$

Note que os autômatos, por possuirem uma função de transição estendida completa, podem processar qualquer palavra sobre seu alfabeto, os geradores, por sua vez, geram uma linguagem, que em geral, é um subconjunto próprio de Σ^*.

Proposição 2.1 *Dado um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$, *a linguagem gerada* $L(G)$ *é prefixo-fechada.*

Dem.: Seja uma palavra $s = (s'\sigma) : s \in L(G)$, $\sigma \in \Sigma$ e $s' \in Pre(s)$. Então $\exists q \in Q : \delta(s, q_0) = q$ e $\exists q' \in Q : \delta(s', q_0) = q'$, logo, $\delta(\sigma, q') = q$. Procedendo recursivamente desta forma, mostra-se que $Pre(s) \in L(G)$.

Portanto, para todo gerador G, tem-se:

$$L(G) = \overline{L(G)}. \tag{2.1}$$

Como no caso dos autômatos, o conjunto de estados marcados em um gerador define uma linguagem:

Definição 2.23 *Dado um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$, *a* linguagem marcada de G, *denotada por* $L_m(G)$, *é:*

$$L_m(G) = \{s : s \in \Sigma^* \wedge \delta(s, q_0) \in Q_m\}.$$

Portanto, um SED com linguagem gerada L e linguagem marcada L_m, pode ser representado por um gerador G tal que $L(G) = L$ e $L_m(G) = L_m$. Assim, os geradores conseguem representar ambas as linguagens associadas a um SED, o que não era possível no caso dos autômatos.

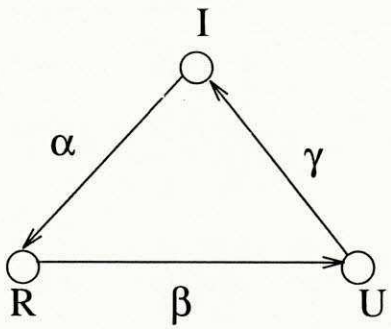


Figura 2.3: Utilização de um recurso por um usuário.

Veja o exemplo simples da utilização de um recurso por um usuário modelado pelo gerador apresentado na Figura 2.3. O SED apresenta três estados: I (*inativo*); R (*requisitado*); e U (*em uso*), e três transições de estados, representadas pelo alfabeto de eventos $\Sigma = \{\alpha, \beta, \gamma\}$.

Quando o sistema se encontra no estado inicial *inativo*, um usuário pode requisitar o recurso (evento α). A ocorrência de α leva o sistema do estado *inativo* para o estado *requisitado*. Nesse novo estado, o evento β (inicia processamento) pode ocorrer levando o sistema ao estado *em uso*. Finalmente, ao terminar a tarefa requisitada, o recurso é então liberado (ocorrência do evento γ). O sistema retorna então ao estado inicial *inativo*.

Pelas Definições 2.22 e 2.23 tem-se que

$$L_m(G) \subseteq L(G). \tag{2.2}$$

Assim, se duas linguagens K e L são tais que $K \subseteq L$, então $\overline{K} \subseteq \overline{L}$. Tomando o prefixo-fechamento de ambos os membros da equação 2.2, tem-se:

$$\overline{L_m(G)} \subseteq \overline{L(G)} \tag{2.3}$$

e pela equação 2.1

$$\overline{L_m(G)} \subseteq L(G). \tag{2.4}$$

Considerando as equações anteriores, tem-se:

$$L_m(G) \subseteq \overline{L_m(G)} \subseteq L(G) = \overline{L(G)} \tag{2.5}$$

Pela definição de gerador, nenhuma restrição é imposta à estrutura do mesmo. Portanto, as duas definições seguintes são importantes para a definição da estrutura de um gerador.

Definição 2.24 *A componente* acessível *de um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$ *é*

$$Ac(G) = (Q_{ac}, \Sigma, \delta_{ac}, q_0, Q_{ac,m}), \text{ com}$$
$$Q_{ac} = \{q : \exists s \in \Sigma^* \wedge \delta(s, q_0) = q\};$$
$$Q_{ac,m} = Q_{ac} \cap Q_m \text{ e}$$
$$\delta_{ac} = \delta \mid (\Sigma \times Q_{ac});$$

onde δ_{ac} *é a função* δ *restrita ao domínio* $\Sigma \times Q_{ac}$.

Um gerador G *é dito* acessível *se e somente se* $G = Ac(G)$.

Definição 2.25 *Um gerador* G *é dito* coacessível *se e somente se toda palavra em* $L(G)$ *for um prefixo de uma palavra em* $L_m(G)$, *ou seja, se e somente se* $L(G) \subseteq \overline{L_m(G)}$.

Assim, um gerador é coacessível se e somente se, a partir de qualquer um de seus estados, existir pelo menos uma seqüência de eventos que o leve a um estado marcado. Note que a Equação 2.4 vale para todo gerador G, então pode-se substituir a inclusão ($\subseteq$) na Definição 2.25 pela igualdade $L(G) = \overline{L_m(G)}$.

Definição 2.26 *Um gerador que é, ao mesmo tempo, acessível e coacessível, é dito* ajustado *(*trim*).*

O gerador apresentado na Figura 2.4 é acessível, pois a partir de s_0 existe sempre uma seqüência de eventos que pode levar o sistema de s_0 para qualquer estado do mesmo. O gerador não é coacessível, pois a partir do estado s_3 não existe nenhuma seqüência de eventos que possa levar o sistema a um estado marcado, no caso s_2. A componente ajustada (*trim*) do gerador, ou seja, acessível e coacessível, é apresentada dentro do retângulo pontilhado.

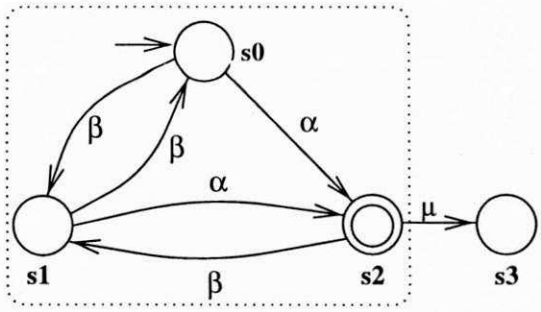


Figura 2.4: Exemplo de componente ajustada de um gerador.

2.2 Redes de Petri

Uma rede de Petri (RP) [Pet81, Rei85, Mur89] é um grafo direcionado bi-partido e ponderado que possui um estado inicial denominado marcação inicial, M_0. Uma RP possui dois tipos de nós denominados lugares e transições, respectivamente. Arcos ponderados são direcionados dos lugares para as transições e das transições para os lugares. Quando um arco é direcionado de um lugar para uma transição, este lugar é denominado de lugar de entrada da transição e a transição é denominada de transição de saída do lugar. Quando um arco é direcionado de uma transição para um lugar, este lugar é denominado de lugar de saída da transição e esta é denominada de transição de entrada do lugar. Os lugares são representados, graficamente, por círculos e as transições por barras ou retângulos. Um arco de peso k significa que k arcos paralelos ligam um lugar a uma transição ou vice-versa. Uma marcação (estado) de uma RP atribui a cada lugar um inteiro não negativo. Se uma marcação atribui a um certo lugar p um inteiro não negativo n, diz-se que p possui n fichas (*tokens*). Graficamente, as fichas são representadas por círculos pretos nos lugares. Uma marcação é designada por M (vetor m), onde m é o número de lugares da RP.

Denomina-se *rede de Petri com capacidade infinita* a rede em cujos lugares pode ser colocado um número ilimitado de fichas. De outra forma, a uma rede em cujos lugares pode ser colocado um número limitado de fichas, denomina-se *rede de Petri com capacidade finita.*

Definição 2.27 *Uma rede de Petri é definida pela tupla*

$$RP = (P, T, F, K, W, M_0),$$

onde:

- $P = \{p_1, p_2, ..., p_m\}$ *é um conjunto finito de lugares;*
- $T = \{t_1, t_2, ..., t_n\}$ *é um conjunto finito de transições;*
- $F \subseteq (P \times T) \cup (T \times P)$ *é um conjunto de arcos;*
- $K : P \rightarrow \mathbb{N} \cup \{\infty\}$ *é a função de capacidade;*
- $W : F \rightarrow \mathbb{N}^+$ *é a função de ponderação;*
- $M_0 : P \rightarrow \mathbb{N}$ *é a função de marcação inicial satisfazendo* $\forall p \in P : M_0(p) \leq K(P)$.

Definição 2.28 *Uma estrutura de rede de Petri, sem nenhuma marcação inicial, é denotada por* $N = (P, T, F, W)$. *Assim, uma rede de Petri com uma dada marcação inicial é denotada por* $RP = (N, K, M_0)$. *Se a capacidade da rede é não limitada, então a mesma é denotada por* $RP = (N, M_0)$.

Definição 2.29 *O estado, ou marcação, de uma RP varia de acordo com a seguinte* regra de disparo das transições*:*

1. *Uma transição t é dita estar habilitada (para disparar) em M se e somente se*

$$\forall p \in P \text{ que é entrada de } t : W(p,t) \leq M(p)$$
$$\forall p \in P \text{ que é saída de } t : M(p) \leq K(p) - W(t,p);$$

2. *Uma transição habilitada pode ou não disparar;*
3. *O disparo de uma transição* $t \in T$, *habilitada na marcação M, é instantâneo e resulta em uma nova marcação M' dada pela equação:*

$$M'(p) = M(p) - W(p,t) + W(t,p), \forall p \in P;$$

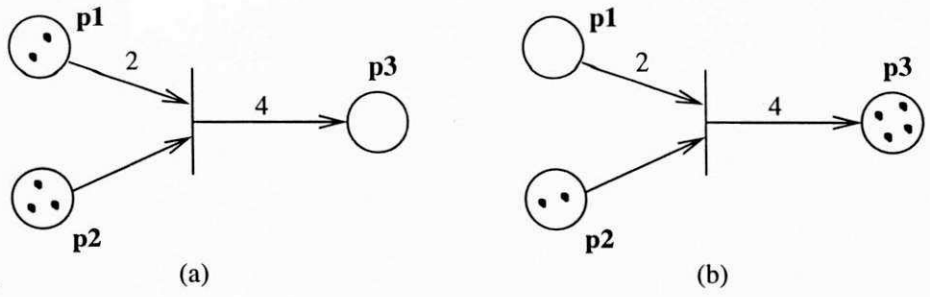


Figura 2.5: Disparo de uma transição habilitada.

4. *A ocorrência do disparo de t, que modifica a marcação M da rede para uma nova marcação M', é denotada por $M[t\rangle M'$, ou (em analogia à* função de próximo estado *δ dos autômatos) $M' = \delta(M, t)$.*

Definição 2.30 *Seja $R(N, M_0)$ o conjunto de marcações alcançáveis de uma RP a partir de M_0, tal que:*

- $M \in R(N, M_0)$ *e*
- *se $M_1 \in R(N, M_0)$ e para uma dada transição $t \in T$ $M_1[t\rangle M_2$ então $M_2 \in R(N, M_0)$.*

As Figuras 2.5(a) e 2.5(b) representam a mesma RP antes e depois do disparo da transição t, respectivamente.

Na Figura 2.5(a) $M(p_1) = 2$, $M(p_2) = 3$ e $M(p_3) = 0$; $K(p_1) = K(p_2) = K(p_3) = 5$; $W(p_1, t) = 2$, $W(p_2, t) = 1$ e $W(t, p_3) = 4$; p_1 e p_2 são lugares de entrada de t e p_3 é lugar de saída de t. Note que t está habilitada pois $M(p_1) = W(p_1, t)$, $M(p_2) > W(p_2, t)$ e $M(p_3) < K(p_3) - W(t, p_3)$. Nesta situação t pode ser disparada. Após o disparo de t, a marcação muda para aquela em 2.5(b). Note que a transição t da Figura 2.5(b) não está habilitada, pois nessa nova marcação, $M(p_1) \geq W(p_1, t)$ e $M(p_3) > K(p_3) - W(t, p_3)$.

Dada uma rede de Petri RP $= (N, M_0)$, pode-se executar a rede pelo disparo sucessivo de suas transições habilitadas em cada marcação. Disparando uma transição habilitada t_j a partir de M_0, encontra-se uma nova marcação $M_1 = \delta(M_0, t_j)$. Nesta nova marcação, pode-se disparar qualquer nova transição habilitada, por exemplo t_k, resultando em uma nova marcação $M_2 = \delta(M_1, t_k)$. Esse processo pode continuar enquanto existir pelo menos uma transição habilitada em cada marcação.

Duas seqüências resultam da execução de uma RP: a seqüência de marcações (M_0, M_1, $M_2, \cdots$) e a seqüência de transições ($t_{j0}, t_{j1}, t_{j2}, \cdots$). Estas duas seqüências são relacionadas pela relação $\delta(M_k, t_{jk}) = M_{k+1}$ para $k = 0, 1, 2, \cdots$. Assim, dada uma seqüência de transições e M_0, pode-se facilmente encontrar a seqüência de marcações pela execução da rede de Petri. Ambas as seqüências fornecem um registro da execução de uma RP.

Em analogia à função de transição de estados dos autômatos, é conveniente se estender a função de próximo estado para mapear uma marcação e uma seqüência de transições em uma nova marcação. Para uma seqüência de transições $t_{j1}, t_{j2}, \cdots, t_{jk}$ e uma marcação M, a marcação

$$M' = \delta(M, t_{j1}, t_{j2}, \cdots, t_{jk})$$

é o resultado do disparo primeiro de t_{j1}, depois de t_{j2} e assim por diante até o disparo de t_{jk}. Formalmente:

Definição 2.31 *A* função estendida de próximo estado *é definida para uma marcação M e uma seqüência de transições $s \in T^*$ por*

$$\delta(M, t_j s) = \delta(\delta(M, t_j), s)$$

onde T^* representa o conjunto de todas as seqüências de transições possíveis para uma dada rede de Petri.

As redes de Petri são uma ferramenta matemática e gráfica que se aplicam à modelagem de sistemas caracterizados como concorrentes, assíncronos, distribuídos, paralelos, não determinísticos e estocásticos. Como uma ferramenta gráfica, as RPs ajudam na visualização da estrutura do sistema modelado e seu comportamento dinâmico. Como uma ferramenta matemática, é possível utilizar equações de estado, equações algébricas, linguagens formais e outros modelos matemáticos que descrevem o comportamento dinâmico do sistema. Dentre as muitas áreas de aplicação, podem ser citadas algumas de interesse, tais como: sistemas de manufatura flexíveis [ZD93, Sil85], sistemas de controle industrial e robótica [ZD93, Sil85, SV89]. A utilização prática das RPs para modelar e analizar sistemas mais complexos, incrementou o estudo teórico das redes fazendo surgir dois tipos de extensões relevantes, as relacionadas à capacidade de modelagem funcional e as que envolvem restrições de tempo. Uma RP simula o comportamento dinâmico de um

sistema mudando sua marcação de acordo com a regra de disparo das transições, descrita anteriormente.

Para um estudo mais detalhado sobre teoria e aplicação de redes de Petri, o leitor deve se reportar a [Pet81, Rei85, Mur89, DS95, MBC+95, Res92, Uni95]

2.2.1 Redes de Petri Etiquetadas

Como descrito anteriormente, as RPs podem descrever e analizar sistemas propostos ou já existentes. Pela análise de uma RP, pode-se determinar as propriedades da rede e, consequentemente, do sistema modelado. Uma das propriedades de um sistema a eventos discretos é o conjunto de todas as seqüências de eventos possíveis que podem ocorrer. O conjunto de todas as seqüências de eventos possíveis caracteriza o sistema.

Numa RP, eventos são modelados por transições e a ocorrência de um evento é modelada pelo disparo de uma transição. Sequências de eventos são modeladas por seqüências de transições. Uma seqüência de transições é uma cadeia e um conjunto de cadeias forma uma linguagem, como visto no Capítulo 2.

Definindo-se o conjunto das possíveis seqüências de transições em uma RP como uma linguagem, é possível analizar uma rede através da análise de sua linguagem.

Sistemas a eventos discretos podem ser modelados por redes de Petri utilizando-se uma notação análoga àquela empregada na modelagem de SEDs por autômatos. Para tanto, é necessário que, em adição à estrutura de uma RP, se defina um estado inicial, um alfabeto e um conjunto de estados finais. A especificação destas características pode resultar em diferentes classes de linguagens de redes de Petri.

O estado inicial é simplesmente a marcação inicial da rede. O alfabeto, como definido pela teoria de linguagens formais, é um conjunto de símbolos os quais são associados às transições tal que uma seqüência de disparo de transições gera uma cadeia de símbolos de uma certa linguagem. Uma função de etiquetagem das transições associa a cada transição um símbolo. A definição de estados marcados, ou finais, para uma rede de Petri, tem um efeito importante na caracterização da linguagem gerada pela rede, como veremos ainda nesta seção.

Uma rede de Petri que possui um estado inicial, um conjunto de eventos associados e um conjunto de estados finais, é denominada de *Rede de Petri Etiquetada* ou *Gerador*

de Rede de Petri (*Petri net generator*). A seguir apresentamos a definição das redes de Petri etiquetadas, bem como as linguagens normalmente associadas a essas redes segundo as definições apresentadas em [Pet81, Jan87].

Definição 2.32 *Uma rede de Petri etiquetada, ou gerador de rede de Petri, é uma sêxtupla*

$$G = (N, K, \Sigma, l, M_0, Q_f), \text{ onde:}$$

- $N = (P, T, F, W)$ *é uma estrutura de rede de Petri;*
- K *é a função de capacidade;*
- Σ *é um conjunto finito (alfabeto) de eventos;*
- $l : T \rightarrow \Sigma$ *é a função de etiquetagem que associa um evento a cada transição e que pode ser estendida para o mapeamento* $T^* \rightarrow \Sigma^*$, *ou seja, associa uma seqüência de eventos a uma seqüência de transições;*
- M_0 *é a marcação inicial e;*
- Q_f *é um conjunto finito de marcações finais.*

A função de capacidade K será omitida sempre que não for necessário explicitar a capacidade da rede ou gerador.

Definição 2.33 *O* conjunto de cobertura *de* Q_f *é assim definido:*

$$C_f = \{M \in R(N, M_0) \mid (\exists M' \in Q_f)[M \geq M']\}.$$

Isto é, C_f é o conjunto de marcações maiores ou iguais a alguma marcação em Q_f.

Como dito anteriormente, uma seqüência de transições é obtida a partir de uma seqüência de disparo, omitindo-se as marcações alcançadas. Se o interesse reside em todas as seqüências de transições que modificam a marcação inicial de uma dada RP, levando a rede para uma marcação final, então existem várias formas de se especificar essa marcação final de modo a representar o conjunto de estados que o sistema deve alcançar.

Pode ser o caso em que um conjunto de marcações previamente definidas têm que ser alcançadas. Pode ser o caso em que seja suficiente que um conjunto de marcações sejam cobertas. Uma outra possibilidade seria encontrar todas as seqüências de transições que levam a um conjunto de marcações bloqueadas ou mortas. Restringindo a atenção a uma escolha não determinística de seqüências de disparo de transições de uma rede de Petri etiquetada, apresentamos a seguir a definição das linguagens geradas por estas redes [Pet81, Jan87].

Dada uma rede de Petri etiquetada $G = (N, \Sigma, l, M_0, Q_f)$, então:

Definição 2.34 *A linguagem tipo P de G é assim definida :*

$$L(G) = \{l(\beta) \in \Sigma^* \mid \beta \in T^*,\ M_0[\beta\rangle\}$$

Definição 2.35 *A linguagem tipo L de G é assim definida :*

$$L_m(G) = \{l(\beta) \in \Sigma^* \mid \beta \in T^*,\ M_0[\beta\rangle M,\ M \in Q_f\}$$

Definição 2.36 *A linguagem tipo G de G é assim definida :*

$$L_w(G) = \{l(\beta) \in \Sigma^* \mid \beta \in T^*,\ M_0[\beta\rangle M,\ M \in C_f\}$$

Definição 2.37 *A linguagem tipo T de G é assim definida :*

$$L_t(G) = \{l(\beta) \in \Sigma^* \mid \beta \in T^*,\ M_0[\beta\rangle M \ e\ \forall t \in T : M[t\rangle)\}$$

A *linguagem tipo L* representa todas as possíveis sequências de disparo de transições da rede que a levam necessariamente a um conjunto (usualmente finito) de marcações finais, previamente definidas. A *linguagem tipo G* representa todas as possíveis sequências de disparo de transições da rede que a levam a uma certa marcação em que uma marcação de um conjunto final previamente definido, seja coberta. A *linguagem tipo T* produz todas as seqüências de transições que produzem uma marcação morta, ou seja, uma marcação na qual nenhuma transição da rede é habilitada. Por fim, a *linguagem tipo P* define todas as seqüências de transições que são habilitadas na marcação inicial de uma rede de Petri etiquetada.

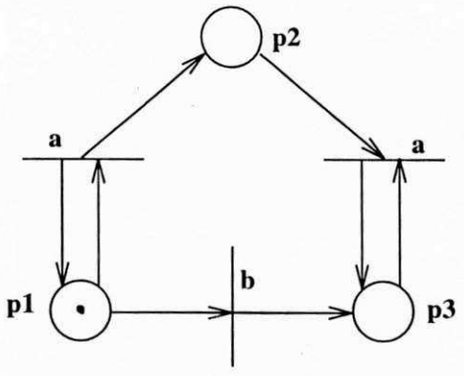


Figura 2.6: Rede de Petri etiquetada.

As duas linguagens normalmente associadas com um gerador de rede de Petri G são a *linguagem tipo P* (também denominada comportamento fechado no contexto do controle supervisório) e a *linguagem tipo L* (também denominada *comportamento marcado*). O comportamento fechado representa todas as possíveis seqüências de disparo de transições da rede etiquetada, enquanto o comportamento marcado representa todas as possíveis seqüências de disparo de transições que levam a rede a alcançar um estado marcado ou final.

Uma noção diferente de comportamento terminal (estado marcado), é definido considerando a *linguagem tipo G*, (também denominada *comportamento fraco*, ou *weak behavior*) e a *linguagem tipo T* (também denominada *linguagem bloqueada*), como visto pelas definições acima.

A figura 2.6 apresenta uma rede de Petri etiquetada. Cada transição é etiquetada com um símbolo do alfabeto $\Sigma = \{a, b\}$ e o conjunto de marcações final é $Q_f = \{(001)^T\}$. Como exemplo, as linguagens associadas à rede da Figura 2.6, para $Q_f = \{(001)^T\}$, são:

- $L(G) = \{a^m \mid m \geq 0\} \cup \{a^m b a^n \mid m \geq n \geq 0\}$;
- $L_m(G) = \{a^m b a^m \mid m \geq 0\}$ e:
- $L_w(G) = \{a^m b a^n \mid m \geq n \geq 0\}$.

Note que na definição de rede etiquetada, assume-se que l é uma função de etiquetagem $\lambda - livre$. De acordo com a terminologia usada por Peterson [Pet81] tem-se:

Definição 2.38 *Uma rede de Petri, ou RP $\lambda-$livre é aquela em que nenhuma transição é etiquetada com a cadeia vazia λ e duas ou mais transições podem possuir a mesma etiqueta.*

Outras subclasses de linguagens muito importantes quando se estudam os SEDs, são aquelas geradas pelos *geradores de redes de Petri determinísticos*, isto é, redes tais que a cadeia de eventos gerada a partir da marcação inicial, determina de forma única a marcação alcançada, ou de outra forma, a seqüência de transições disparadas. Formalmente:

Definição 2.39 *Um gerador de rede de Petri $G = (N, l, M_0, Q_f)$ é determinístico se e somente se:*

$$\forall t, t' \in T, \text{ com } t \neq t' \text{ e } \forall M \in R(N, M_0),\ M[t\rangle \wedge M[t'\rangle \Rightarrow l(t) \neq l(t').$$

Sistemas de interesse da teoria de controle supervisório são determinísticos, sendo assim, assume-se que os geradores considerados neste trabalho são sempre determinísticos.

Para maiores informações sobre linguagens de redes de Petri, inclusive comparações com outras classes de linguagens formais, o leitor deve se reportar a [Pet81, Jan87].

2.3 Redes de Petri Controladas

Redes de Petri Controladas (RPCtl) são uma classe de redes de Petri com restrições de habilitação externas denominadas *lugares de controle*. Estes lugares permitem a um controlador externo influenciar a progressão das fichas na rede. As RPCtls foram primeiro propostas por Krogh [Kro87] e Ichikawa e Hiraishi [IH88]. Usaremos neste trabalho a terminologia utilizada por Krogh [Kro87] na definição de redes de Petri controladas.

Definição 2.40 *Uma rede de Petri controlada é uma quíntupla*

$$RP^c = (P, T, P_c, F_p, F_c),$$

onde:

- *P é um conjunto finito de lugares de estado;*

- *T é um conjunto finito de transições;*
- *P_c é um conjunto finito de lugares de controle;*
- *$F_p \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos direcionados conectando lugares de estado a transições e vice-versa;*
- *$F_c \subseteq (P_c \times T)$ é um conjunto de arcos direcionados conectando lugares de controle a transições.*

Os conjuntos P, T, P_c são mutuamente disjuntos. A estrutura de uma rede de Petri controlada RP^c é definida por $N^c = (P, T, F_p)$.

Definição 2.41 *Dada uma transição $t \in T$, denota-se o conjunto de lugares de controle de entrada de t como:*

$$^{(c)}t = \{c \mid (c,t) \in F_c\}.$$

Definição 2.42 *Dado um lugar de controle $c \in P_c$, o conjunto de transições de saída de c é denotado por:*

$$c^{(t)} = \{t \mid (c,t) \in F_c\}$$

Definição 2.43 *Dada uma transição $t \in T$, denota-se o conjunto de lugares de estado de entrada de t como:*

$$^{(p)}t = \{p \mid (p,t) \in F_p\}.$$

Definição 2.44 *Dado um lugar de estado $p \in P$, o conjunto de transições de saída de p é denotado por:*

$$p^{(t)} = \{t \mid (p,t) \in F_p\}$$

Definição 2.45 *Uma transição t é dita ser uma transição controlada se seu conjunto de entradas de controle $^{(c)}t$ é não vazio. O conjunto de todas as transições controladas é denotado por $T_c \subseteq T$.*

A Figura 2.7 apresenta uma rede de Petri controlada, onde os círculos representam os lugares de estado, as barras representam as transições e o retângulo representa o lugar de controle.

O estado de uma RPCtl é representado pela sua marcação, ou seja, pela distribuição de fichas em seus lugares de estado.

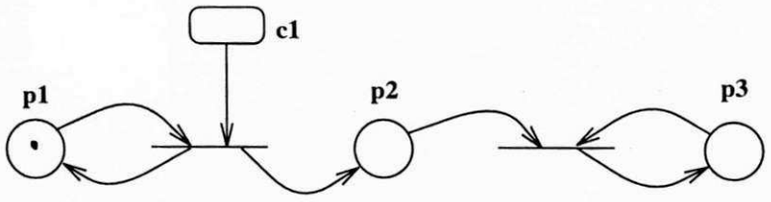


Figura 2.7: Rede de Petri controlada.

Definição 2.46 *A marcação em uma RPCtl é uma função* $M : P \rightarrow \mathbb{N}$ *que indica o número de fichas em cada lugar de estado da rede. (*$\mathbb{N}$ *indica o conjunto de inteiros não negativos).*

Definição 2.47 *Um conjunto de transições* $\tau \in T$ *é dito estar habilitado por estado (*state enabled*) em uma dada marcação* M *se e somente se:*

$$M(p) \geq \mid p^{(t)} \cap T \mid \text{ para todo } p \in {}^{(p)}t.$$

Definição 2.48 *O controle para uma RPCtl é uma função* $u : P_c \rightarrow \{0,1\}$*, associando um valor binário a cada lugar de controle. O conjunto de todos os controles é denotado por* $\mathcal{U}$*.*

Definição 2.49 *O controle* u_{zero} *é definido como* $u_{zero}(c) \equiv 0$ *para todo* $c \in P_c$*.*

Definição 2.50 *O controle* u_{um} *é definido como* $u_{um}(c) \equiv 1$ *para todo* $c \in P_c$*.*

Definição 2.51 *Dados dois controles* u *e* $u' \in \mathcal{U}$*, o controle* u' *é dito ser mais permissivo que* u *se* $u'(c) \geq u(c)$ *para todo* $c \in P_c$ *e* $u'(c) > u(c)$ *para pelo menos um* $c \in P_c$*.*

Segue que, o controle u_{um} é o controle mais permissivo em $\mathcal{U}$, enquanto que u_{zero} é o menos permissivo.

Definição 2.52 *Um conjunto de transições* $\tau \in T$ *é dito estar habilitado por controle (*control enabled*) se e somente se:*

$$\forall t \in \tau,\ u(c) = 1 \text{ para todo } c \in {}^{(c)}t.$$

Seja $T_e(M,u) \subseteq 2^T$ a coleção de conjuntos de transições que são habilitadas tanto por estado, em uma marcação M, quanto por controle $u \in \mathcal{U}$. Assim, qualquer conjunto de transições $\tau \in T_e(M,u)$ pode ser disparado, levando o sistema da marcação M para a marcação M', conforme a seguinte definição:

Definição 2.53 *Em uma RPCtl, uma variação de estado, ou marcação, ocorre, quando um conjunto de transições habilitadas $\tau \in T_e(M,u)$ é disparado. O disparo de τ resulta em uma nova marcação M' definida pela seguinte equação:*

$$M'(p) = M(p) - \mid p^{(t)} \cap T \mid + \mid {}^{(t)}p \cap T \mid$$

Assim, o disparo de uma transição de saída de um lugar de estado decrementa a marcação nesse lugar, enquanto a marcação é incrementada para os lugares de saída da transição disparada.

Definição 2.54 *Dada uma marcação $M \in R(N^c, M_0)$ e um controle $u \in \mathcal{U}$, uma marcação M' é dita ser imediatamente alcançável se existe um conjunto de transições $\tau \in T_e$ para aquela marcação M e aquele controle u, e o disparo de τ resulta em M'.*

O conjunto de marcações imediatamente alcançáveis, a partir de M e u é denotado por $R_1(M,u)$.

Definição 2.55 *Uma marcação M_k é dita ser alcançável a partir de M_0, com um controle constante $u \in \mathcal{U}$, se existe uma seqüência de marcações $(M_0, M_1, \cdots, M_k)$ tal que $M_i \in R_1(M_{i-1}, u)$ para $0 < i \leq k$.*

O conjunto de todas as marcações alcançáveis a partir de M e um controle constante u, é denotado por $R_\infty(M,u)$.

O modelo de redes de Petri controladas desenvolvido por Holloway et. al.[HKG95], difere daquele desenvolvido por Ichikawa e Hiraishi [IH88]. No modelo proposto por Ichikawa e Hiraishi, as fichas, nos lugares de controle, são consumidas pelo disparo das transições controladas. Assim, esses *lugares de controle* são similares aos *lugares de estado*, podendo possuir marcações não binárias. Nesse contexto, os lugares de controle são inicialmente *carregados* com um certo número de fichas e são considerados os seguintes

problemas: uma dada seqüência de transições; um dado vetor de disparo de transições; e uma dada marcação final. Para todos esses problemas, o controle é feito em malha aberta. No modelo desenvolvido por Holloway et. al., são focalizados problemas de controle em malha fechada, como na *abordagem de controle por realimentação de estado* que veremos a seguir.

2.3.1 Abordagem de Controle por Realimentação de Estado e Especificações de Estado

Nesta abordagem, as especificações de estado são dadas como um conjunto de *marcações legais* para um sistema a ser controlado. O objetivo do controle é restringir o comportamento de um sistema tal que somente marcações consideradas legais possam ser alcançadas. A abordagem de controle correspondente é denominada *realimentação de estado* (*state feedback*). A seguir é mostrado como esse controle pode ser computado para o modelo de RPCtl, desenvolvido por Hclloway et. al, apresentado nesta Seção.

A abordagem de controle por realimentação de estado é, em geral, não determinística porque ela identifica um conjunto de controles possíveis. A abordagem é determinística se $U(M)$ é única para todas as marcações da rede.

Estendendo a notação de *marcações imediatamente alcançáveis* a partir de uma marcação $M \in R(M_0, N^c)$ para a abordagem de controle U, tem-se:

Definição 2.56 *O conjunto de marcações imediatamente alcançáveis a partir de* $M \in R(M_0, N^c)$ *para o controle de realimentação de estado* U*, é assim definido:*

$$R_1(M, U) = \cup_{u \in U(M)} R_1(M, u).$$

Definição 2.57 *O conjunto de todas as marcações alcançáveis a partir de* $M \in R(M_0, N^c)$ *para o controle de realimentação de estado* U*, é assim definido:*

$$R_\infty(M, u) = \begin{cases} 1.\ M \in R_\infty(M, U); \\ 2.\ \text{se } M' \in R_\infty(M, U),\ \text{então } R_1(M', U) \subseteq R_\infty(M, U) \text{ e} \\ 3.\ \forall M' \in R_\infty(M, U),\ M' \text{ é definido por 1 e 2.} \end{cases}$$

Estendendo o conceito de *permisividade* (Definição 2.51) para esta abordagem de controle, tem-se:

Definição 2.58 *O controle por realimentação de estado U_1 é mais permissivo que o controle U_2, denotado por $U_1 \geq U_2$, se para cada $M \in R(M_0, N^c)$, $U_1(M) \supseteq U_2(M)$ e para algum $M \in R(M_0, N^c), U_1(M) \supset U_2(M)$.*

Segue que $U_1 \geq U_2$ implica $R_\infty(M, U_1) \supseteq R_\infty(M, U_2)$ para qualquer marcação $M \in R(M_0, N^c)$. A abordagem por realimentação de estado associada às RPCtls tem sido investigada por vários pesquisadores [HK90, BLNB95, Ush89, HG93]. Na maioria dos casos, o problema fundamental é sintetizar um controlador que garanta ao sistema sua permanência em um conjunto especificado de estados permitidos, ou de forma equivalente, que a marcação de uma RPCtl nunca pertença a um conjunto de marcações denominadas *marcações proibidas.*

Obsevando o exposto acima, e considerando uma dada RPCtl RP^c com marcação inicial M_0, seja $\mathcal{M}_F$ o conjunto de marcações proibidas. O objetivo da abordagem é encontrar um controle por realimentação de estado $U_F \rightarrow 2^u$ para o qual:

1. $R_\infty(M_0, U_F) \cap \mathcal{M}_F = \emptyset$;
2. para qualquer controle U' tal que $U' \geq U_F$, se U' satisfaz 1., então $U' = U_F$.

Em outras palavras, a condição 1 garante que o sistema nunca alcançará um estado proibido sob o controle U_F, enquanto que a condição 2 garante controle com permissividade máxima, ou seja, controle minimamente restritivo.

O controle por realimentação de estado que satisfaz as condições acima, para uma dada especificação de estados proibidos $\mathcal{M}_F$, é denominado de *controle por realimentação de estado de permissividade máxima* (*maximally permissive state feedback policy*).

A condição necessária e suficiente para a existência de tal controle é determinada pela análise do comportamento da RPCtl sob o controle u_{zero}. Especificamente, define-se um conjunto de *marcações admissíveis*, ou seja:

Definição 2.59 *Um conjunto de marcações admissíveis para uma RPCtl RP^c em relação a um conjunto de marcações proibidas $\mathcal{M}_F$ é assim definido:*

$$\mathcal{A}(\mathcal{M}_F) = \{M \in R(N^c, M_0) \mid R_\infty(M, u_{zero}) \cap \mathcal{M}_F = \emptyset\}.$$

As condições necessárias e suficientes para a existência do controle por realimentação de estado que mantém os estados de uma RPCtl dentro do conjunto de estados admissíveis, são dadas pelo seguinte teorema [HK92]:

Teorema 2.1 *Dada uma RPCtl RP^c com uma marcação inicial M_0 e um conjunto especificado de marcações proibidas $\mathcal{M}_F$, um único controle por realimentação de estado de permissividade máxima existe se e somente se $M_0 \in \mathcal{A}(\mathcal{M}_F)$.*

O único controle por realimentação de estado de permissividade máxima, no Teorema 2.1, é não determinístico porque, em uma RPCtl, é permitido o disparo simultâneo de múltiplas transições. No caso em que o sistema é não concorrente, ou seja, apenas uma transição pode disparar de cada vez, então existe um único controle por realimentação de estado com permissividade máxima.

Quando uma marcação inicial, para uma RPCtl, satisfaz à condição $M_0 \in \mathcal{A}(\mathcal{M}_F)$, no Teorema 2.1, o controle por realimentação de estado de permissividade máxima pode ser descrito simplesmente como um controle que não permite nenhuma transição de estado para uma marcação que não pertença a $\mathcal{A}(\mathcal{M}_F)$ [HK92]. Como, nesse caso, a ação de controle é atualizada a cada transição de estado, somente as marcações imediatamente alcançáveis precisam ser consideradas na determinação do conjunto de controles admissíveis a partir de uma dada marcação. Esse raciocínio leva ao seguinte teorema [HK92]:

Teorema 2.2 *Dada uma RPCtl RP^c com um conjunto especificado de marcações proibidas $\mathcal{M}_F$ e uma marcação inicial $M_0 \in \mathcal{A}(\mathcal{M}_F)$, se $M \in R_\infty(M_0, U_F)$, então:*

$$U_F(M) = \{u \in \mathcal{U} \mid R_1(M,u) - \mathcal{A}(\mathcal{M}_F) = \emptyset\}.$$

Ao se caracterizar a estratégia de máxima permissividade para o controle por realimentação de estado, dada uma especificação de estados proibidos, resta saber como computar o conjunto de ações de controle admissíveis para uma dada marcação. Uma abordagem é simplesmente gerar o autômato controlado equivalente para a RPCtl, ou seja, gerar o grafo de alcançabilidade[1] [Pet81, Mur89] para a estrutura de rede de Petri levando em consideração a informação de controle associada. Depois, é só aplicar os algoritmos desenvolvidos por Ramadge e Wonham [RW87b] para computar o controle.

[1] a definição de grafo de alcançabilidade é apresentada no Capítulo 4

O autômato controlado é obtido a partir da árvore de alcançabilidade da RPCtl quando se admite que somente uma transição pode disparar por vez, e quando cada transição $t \in T_c$ é controlada independentemente. Este é o caso considerado em [LW93].

Como o grafo de alcançabilidade pode crescer exponencialmente, em relação ao tamanho de uma Rede de Petri [Pet81, Mur89], portanto também em relação ao tamanho de uma RPCtl, métodos alternativos têm sido considerados para a computação do controle por realimentação de estado. Nesse sentido, as RPCtls têm sido utilizadas em abordagens que usam a estrutura da rede diretamente para definir as ações de controle, sem a necessidade de se gerar o autômato controlado equivalente [BLNB95, HK92].

2.4 Redes de Petri Coloridas

Sistema reais frequentemente apresentam subsistemas similares, mas não idênticos. Utilizando redes de Petri para modelar tais sistemas, estes subsistemas têm que ser representados por subredes disjuntas que possuem estruturas quase idênticas. Isso significa que a rede de Petri que modela o sistema total pode se tornar muito grande. Desta forma, se torna difícil a visualização das similaridades e diferenças entre as subredes, bem como o entendimento global do modelo.

A utilização de Redes de Petri para descrever sistemas reais mais complexos, torna clara a necessidade de tipos de redes mais poderosas para descrever tais sistemas, facilitando a visualização do modelo e, consequentemente o seu entendimento. O desenvolvimento das *Redes de Petri de Alto Nível*, tais como as *Redes de Petri Coloridas* (RPCs) [Jen92, Jen80] e as *Redes Predicado/Transição* (redes Pr/T) [GL79], constitui um avanço significativo nessa direção. As redes de Petri de alto nível, mais especificamente as redes de Petri coloridas, possuem o mesmo poder de descrição e análise das redes de Petri. Assim, todos os sistemas modelados por redes de Petri podem ser modelados por redes de Petri coloridas, mas de forma muito mais compacta, onde as similaridades e diferenças das subredes podem ser observadas com maior facilidade.

A representação mais compacta das redes de Petri coloridas é conseguida ampliando o conceito de ficha, ou seja, numa RPC cada ficha tem uma cor associada que a diferencia das demais, indicando assim sua identidade. No caso das redes de Petri, não é possível se

distinguir entre duas fichas quando as mesmas se encontram em um mesmo lugar. Além disso, cada lugar e cada transição, em uma RPC, possui um conjunto de cores associado. Uma transição pode disparar em relação a cada uma das cores de seu conjunto de cores associado. O disparo de uma transição depende de uma função entre a cor envolvida no disparo e as cores das fichas nos lugares de entrada da transição. Estas funções estão associadas aos arcos que conectam lugares e transições.

A seguir apresentamos as principais características e a definição das redes de Petri coloridas desenvolvidas por Jensen [Jen92].

As principais características das RPCs são:

- Combinação de texto e gráfico;
- Declarações e inscrições são especificadas por meio de uma linguagem formal de programação;
  - tipos, funções, operações, variáveis e expressões são definidos;
- A estrutura da rede consiste de lugares, transições e arcos;
- RPCs possuem o mesmo poder de modelagem e análise das redes de Petri.

Numa RPC, cada lugar possui as seguintes inscrições:

- Nome ou identificação;
- Conjunto de cores associado especificando o tipo das fichas que podem residir no lugar;
- Marcação inicial (multiconjunto[2] de fichas coloridas).

Cada transição possui as seguintes inscrições:

- Nome ou identificação;
- Guarda ou expressão lógica contendo variáveis;

[2]Multiconjunto é definido como uma coleção de elementos que não são necessariamente distintos

Cada arco possui a seguinte inscrição:

- Expressão associada contendo variáveis que, quando avaliadas, produzem um multiconjunto de cores para fichas.

Formalmente, define-se uma rede de Petri colorida da seguinte forma:

Definição 2.60 *Uma rede de Petri colorida é uma tupla*

$$RPC = (C, P, T, A, F_n, F_c, F_g, E_a, I), \text{ onde:}$$

- C *é um conjunto finito de tipos não vazios, denominado conjunto de cores;*
- P *é um conjunto finito de lugares;*
- T *é um conjunto finito de transições;*
- A *é um conjunto finito de arcos tal que:*

$$P \cap T = P \cap A = T \cap A = \emptyset.$$

- $F_n : A \rightarrow (P \times T) \cup (T \times P)$ *é a função de nós;*
- $F_c : P \rightarrow C$ *é a função de cores;*
- F_g *é a função de guarda e é definida de* T *em expressões tais como:*

$$\forall t \in T : [Tipo(F_g(t)) = \mathbf{B} \wedge Tipo(Var(F_g(t))) \subseteq C]$$

onde $\mathbf{B}$ *denota o tipo lógico (contém os elementos* $\{falso, verdadeiro\}$*);*

- E_a *é a função das expressões nos arcos e é definida de* A *em expressões tais como:*

$$\forall a \in A : [Tipo(E_a(a)) = C(p(a))_{MS} \wedge Tipo(Var(E_a(a))) \subseteq C]$$

onde $p(a)$ *é o lugar de* $F_n(a)$ *e* $C(p(a))_{MS}$ *é o multiconjunto ou bolsa de cores do conjunto de cores associado a* p*;*

- *I é a função de inicialização, definida de P em expressões fechadas tais como:*

$$\forall p \in P : [Tipo(I(p)) = C(p)_{MS}].$$

Em uma RPC, uma *ligação* (*binding*) atribui uma cor (valor) a cada variável da expressão associada à transição. Um *elemento de ligação* (*binding element*) é um par (t, b), onde t é uma transição e b é uma ligação para as variáveis da expressão associada a t. Por exemplo, $(t_2, < x = c, i = 2 >)$ é um elemento de ligação, onde t_2 é uma transição, x e i são variáveis e c é um tipo ou cor.

A habilitação e disparo de uma transição são realizados através da habilitação e ocorrência de um elemento de ligação, da seguinte forma:

- Um elemento de ligação estará habilitado se e somente se:
  - Existe uma quantidade suficiente de fichas (da cor correta) em cada um dos lugares de entrada da transição;
  - A expressão de guarda é verdadeira;
- Quando um elemento de ligação ocorre:
  - Um multiconjunto de fichas é removido dos lugares de entrada da transição, de acordo com as expressões nos arcos de entrada da transição;
  - Um multiconjunto de fichas é adicionado aos lugares de saída da transição, de acordo com as expressões nos arcos de saída da transição;

As redes de Petri coloridas possuem mecanismos de estruturação que possibilitam tornar determinados modelos, tais como modelos de protocolos de comunicação, bancos de dados e sistemas de manufatura, que apresentam recursos compartilhados, mais sucintos e de mais fácil visualização.

O problema dos cinco filósofos chineses [Pet81, Jen80] sentados ao redor de uma mesa que alternadamente pensam e comem é apresentado como exemplo de um sistema modelado por RPCs. Para cada filósofo comer é necessário que ele pegue o palito que está à sua direita e o que está à sua esquerda. Com isso os vizinhos, da direita e da esquerda,

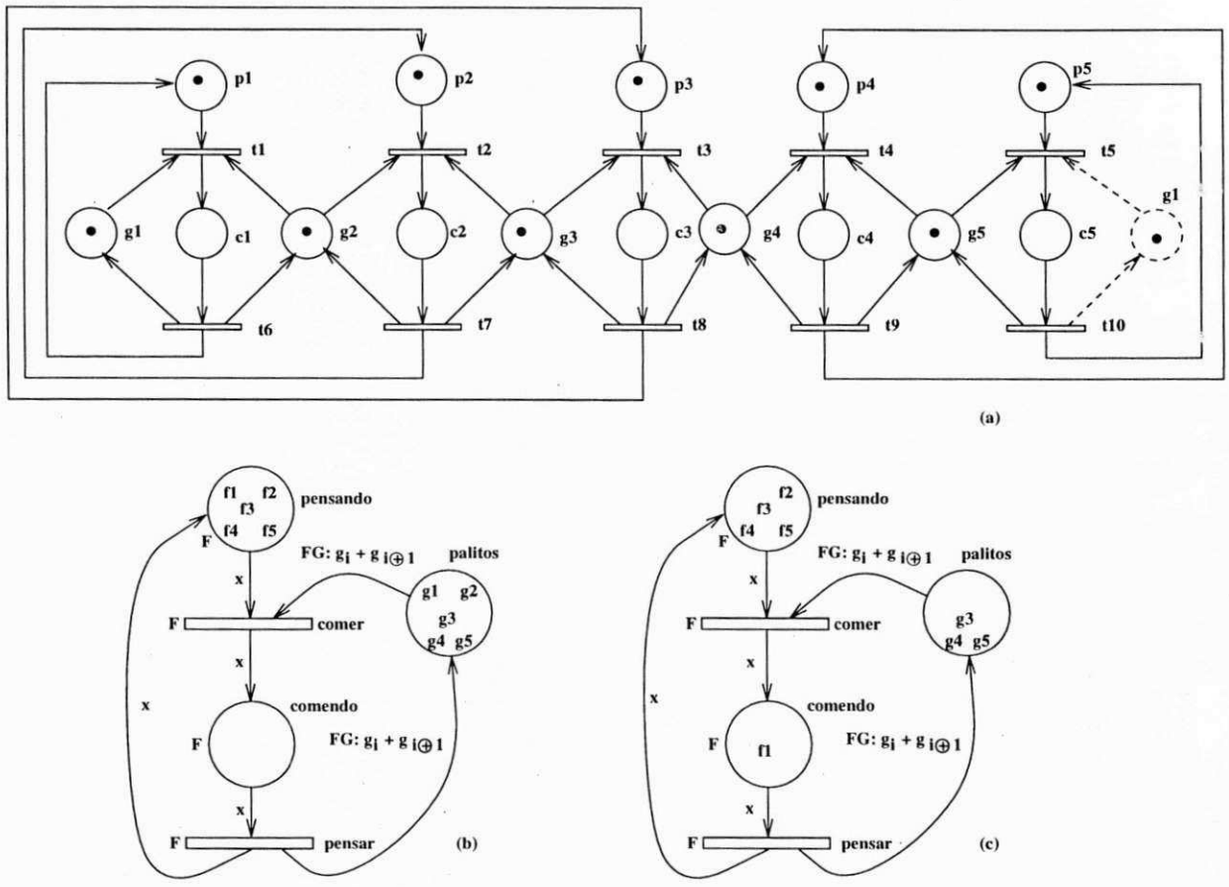


Figura 2.8: Problema dos filósofos modelado por (a) uma RP e (b) uma CPN.

ficam impossibilitados de comer até que o filósofo que está comendo volte a pensar, ou seja, até que os palitos removidos sejam repostos sobre a mesa. Nesse instante, qualquer filósofo que tenha os palitos à disposição pode pegá-los e começar a comer.

A rede de Petri que modela o problema dos cinco filósofos é apresentada na Figura 2.8(a). A rede de Petri colorida que modela o mesmo problema é apresentada na Figura 2.8(b).

Veja que o modelo RP, Figura 2.8(a), possui quinze lugares e dez transições e para cada filósofo que se quisesse acrescentar ao problema, seria necessário acrescentar ao modelo mais dois lugares e duas transições.

O modelo RPC, Figura 2.8(b), apresenta apenas três lugares e duas transições, o que reduz drasticamente a complexidade visual do modelo. Acrescentando mais um filósofo à mesa, ao modelo seriam necessárias somente mais uma ficha de cor f_6 no lugar *pensando* e mais uma ficha de cor g_6 no lugar *palitos*. Na Figura 2.8(b) a transição *comer* está habilitada em relação a todas as cores f_i, $i = 1, ..., 5$, de seu conjunto de cores associado F. Disparando-se *comer* em relação a f_1, retira-se uma ficha f_1 do lugar *pensando*, de acordo com a variável **x** associada ao arco que liga *pensando* a *comer*, e duas fichas de cores g_1 e g_2 respectivamente do lugar *palitos*, de acordo com a expressão associada ao arco que liga *palitos* a *comer*, e coloca-se uma ficha f_1 no lugar *comendo*. Note que agora a transição comer está habilitada somente em relação a f_3 e f_4 pois não existe ficha f_1 em *pensando* e não existem fichas g_1 e g_2 em *palitos*. Para a transição *comer* estar habilitada em relação à cor f_2, seria necessário pelo menos uma ficha g_2 e uma g_3 no lugar *palitos*. Para *comer* estar habilitada em relação a f_5 seria necessário pelo menos uma ficha g_5 e uma g_1 em *palitos*. A Figura 2.8(c) apresenta o novo estado da rede após o disparo de *comer* em relação a f_1. Nesse caso, a transição *pensar* está habilitada em relação a f_1. Disparando *pensar*, a rede volta à marcação inicial.

É possível mostrar, através do *algoritmo da árvore de alcançabilidade*, [Pet81, Mur89] que as redes da Figura 2.8 possuem as mesmas propriedades, o que não poderia ser diferente pois elas modelam o mesmo problema.

Mais exemplos e aplicações de redes de Petri coloridas podem ser encontrados em [Jen92, CP92, SB93, BS92, Che91, BC92].

2.5 Redes de Petri com Temporização Nebulosa

O modelo de *Redes de Petri com Temporização Nebulosa* (*RPTN*s), introduzido por Figueiredo [dF94, dF95], é uma extensão às redes de Petri que utiliza uma abordagem nebulosa para introduzir tempo em redes de Petri. As *RPTN*s são uma ferramenta adequada tanto para a modelagem de sistemas em tempo real, quanto para a avaliação de desempenho de sistemas, isto é, esse modelo combina a capacidade de modelar sistemas em tempo real das redes de Petri temporizadas determinísticas, com a habilidade de fazer avaliação de desempenho das redes de Petri estocásticas. Isso é possível através da combinação entre intervalos de tempo e a teoria de conjuntos nebulosos [Zad65, DHR93, KG88]. Os intervalos de tempo permitem a representação dos limites de tempo que são necessários para modelar sistemas em tempo real, por exemplo o tempo de processamento de uma peça em uma determinada máquina, enquanto os valores nebulosos permitem a estimação de alguns índices de desempenho, por exemplo o tempo médio de processamento de peças em uma determinada máquina. Figueiredo [dF94] demonstra que o modelo *RPTN* é mais geral que as extensões temporais determinísticas pois a partir dele, é possível derivar as extensões determinísticas ajustando os valores das funções de pertencimento dos intervalos nebulosos.

As fichas em uma *RPTN* carregam uma função de temporização nebulosa representando a possibilidade de se ter uma ficha em um determinado lugar. Além da função de temporização nebulosa, dois intervalos nebulosos $= E[e_{min}, e_{max}]$ (intervalo de sensibilização) e $D = [d_{min}, d_{max}]$ (intervalo de disparo) são associados a cada transição. O intervalo E representa o menor e o maior tempo que pode se passar entre a habilitação e o disparo temporizado da transição. O intervalo D representa o tempo de duração do disparo (atrazo) da transição. Abaixo são apresentados alguns conceitos importantes para a definição formal de uma *RPTN*, que é apresentado logo a seguir.

Definição 2.61 *Seja X uma coleção de objetos, onde um objeto genuino é designado por x, então um conjunto nebuloso $\mathcal{C}$ em X é o conjunto de pares ordenados:*

$$\mathcal{C} = \{(x, \mu_{\mathcal{C}}(x)) \mid x \in X\}$$

$\mu_{\mathcal{C}}(x)$ é dita função de pertencimento ou grau de pertencimento de x em $\mathcal{C}$, como definido a seguir.

Definição 2.62 *A função de pertencimento μ_c de um conjunto nebuloso C é uma função*

$$\mu_c : X \rightarrow [0,1]$$

Assim, todo elemento $x \in X$ possui um grau de pertencimento $\mu_c(x) \in [0,1]$.

Definição 2.63 *O conjunto de escala de tempo $TS = \{x \in R \mid x \geq 0\}$ é o conjunto de todos os números reais não negativos.*

Para a definição de intervalo nebuloso, consideram-se duas datas nebulosas na escala de tempo TS. As datas nebulosas são números nebulosos. Um intervalo nebuloso é definido como um conjunto de pontos de tempo que se encontram entre duas datas nebulosas.

Definição 2.64 *Considere y e z duas datas nebulosas na escala de tempo TS, isto é, $y, z \in TS, \forall \mu_y(t), \mu_z(t) \in [0,1]$. Um intervalo nebuloso de tempo $FINT = \{[y,z] \in TS \times TS \mid z \geq y\}$, de pontos de tempo que são maiores que y e menores que z, é definido pela função de pertencimento $\mu(FINT) = \mu_{[y,z]}(t) = \sup \min(\pi_y(s), \pi_z(s'))$ onde $t, s, s' \in TS$ e $\pi_y(s), \pi_z(s')$ delimita o conjunto nebuloso dos possíveis valores de y e z, respectivamente.*

Uma *função de temporização nebulosa* (*fuzzy time function* (FTF)) é também definida como uma data mal conhecida que é atribuída para cada ficha em um lugar. As FTFs das fichas são combinadas para determinar ou atualizar as FTFs das fichas subsequentes.

Definição 2.65 *Suponha que p é um lugar e x é uma data mal conhecida representando o instante de tempo que define a existência de uma ficha em p. Então, a função de temporização nebulosa associada à ficha no lugar p, é $FTF_x(p) = \pi_x(TS)$, em que $\pi_x(TS)$ delimita o conjunto nebuloso dos possíveis valores de x.*

Definição 2.66 *Uma ficha nebulosa é uma ficha que carrega uma função de temporização nebulosa.*

Definição 2.67 *Uma rede de Petri com temporização nebulosa é definida pela tupla*

$$RPTN = (P, T, F, K, W, \mathcal{E}, \mathcal{D}, M_0), \text{ onde:}$$

- $P = \{p_1, p_2, ..., p_m\}$ *é um conjunto finito de lugares;*
- $T = \{t_1, t_2, ..., t_n\}$ *é um conjunto finito de transições;*
- $F \subseteq (P \times T) \cup (T \times P)$ *é um conjunto de arcos;*
- $K : P \rightarrow IN \cup \{\infty\}$ *é a função de capacidade;*
- $W : F \rightarrow IN^+$ *é a função de ponderação;*
- $\mathcal{E} : T \rightarrow E([e_1, e_2])$ *é um conjunto de intervalos nebulosos, chamados de intervalos de habilitação, representando os tempos mínimo e máximo entre a habilitação e o disparo temporizado das transições;*
- $\mathcal{D} : T \rightarrow D([d_1, d_2])$ *é um conjunto de intervalos nebulosos que representam a duração dos disparos das transições;*
- $M_0 : P \rightarrow FTF_\tau(p)$ *é a marcação inicial da rede onde,* $FTF_\tau(p)$ *é a função de temporização nebulosa que caracteriza a distribuição de possibilidades de existir uma ficha em um lugar p em um dado instante* $\tau \in TS$.

Uma vez que nas *RPTN*s cada ficha carrega uma *FTF*, que é definida após o disparo de uma transição, o comportamento dinâmico deste modelo depende das *FTF*s das fichas e dos intervalos nebulosos, os quais modificam as conhecidas regras de disparo de transições.

Definição 2.68 *O comportamento dinâmico de uma RPTN é descrito pelas seguintes regras de disparo das transições:*

1. *Uma transição t é dita estar habilitada, para disparar, se existe pelo menos uma ficha em cada um de seus lugares de entrada (assumindo peso 1 para cada um de seus arcos) no mesmo instante da escala de tempo, isto é, dada uma transição t e o conjunto de seus lugares de entrada* $(p_1, p_2, \cdots, p_n)$, *com as respectivas funções de temporização nebulosa* $FTF(p_1), FTF(p_2), \cdots, FTF(p_n)$, *a transição t é habilitada se e somente se:*

$$FTF(p_1) \cap FTF(p_2) \cap ... \cap FTF(p_n) \neq \emptyset$$

2. *Se, em um dado momento, mais de uma transição está habilitada, uma escolha não determinística determina qual a próxima transição a disparar. Entretanto, é importante lembrar que, em alguns casos dependendo de seus limites, os intervalos de habilitação ou sensibilização, podem contribuir para a escolha da próxima transição a disparar;*

3. *A transição permanece habilitada por um período E antes de poder disparar;*

4. *A transição começa e continua o disparo por um período D;*

5. *Após o disparo, as fichas são removidas dos lugares de entrada e uma ficha com uma nova função de temporização nebulosa é depositada em cada lugar de saída da transição. Essa nova função de temporização nebulosa é obtida a partir das funções de temporização nebulosas carregadas pelas fichas dos lugares de entrada e dos intervalos de temporização nebulosos associados à transição.*

Os intervalos de habilitação E podem modelar estouro de tempo (tempo máximo de realização de uma tarefa, por exemplo), enquanto os intervalos de retardo D modelam retardos de processamento, ou duração de um evento.

Como resultado do disparo de uma transição, tem-se que computar a nova função de temporização nebulosa, a qual depende da combinação das funções de temporização nebulosas anteriores e dos intervalos de temporização nebulosos associados às transições. Essa combinação é obtida computando-se a função de temporização nebulosa resultante das fichas nebulosas de cada lugar de entrada e combinando a função resultante com os intervalos de temporização nebulosos associados às transições. Em geral, essa computação pode ser efetuada em dois passos. No primeiro passo, as FTFs das fichas de entrada são combinadas para verificar a possibilidade de existência de fichas nos lugares de entrada em um mesmo instante de tempo. No segundo passo, a FTF resultante da combinação efetuada no primeiro passo é composta com os intervalos de temporização nebulosos da transição disparada. A seguir é detalhada a execução destes passos.

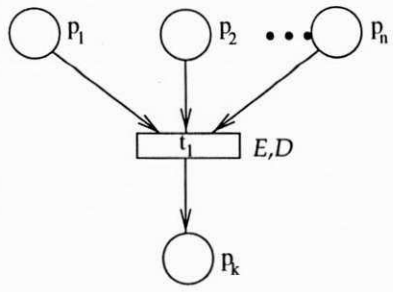


Figura 2.9: Transição com n lugares de entrada.

2.5.1 Passo 1 - Computação das Fichas Nebulosas

Considere a escala de tempo TS, definida anteriormente. A possibilidade da disponibilidade de uma ficha em um lugar p em um instante de tempo $\tau \in TS$, é dado pela função de pertencimento μ:

$$\forall \tau \in TS : \mu_p(\tau) \in [0,1].$$

Considere a transição da Figura 2.9. Nesse caso, para disparar a transição, a FTF resultante, $\mu_R(\tau)$, é a intersecção das FTFs das fichas de entrada. Na Figura 2.9, $FTF(p_1), FTF(p_2), \ldots, FTF(p_n)$ são as função nebulosas carregadas pelas fichas nos lugares $p_1, p_2, ..., p_n$, respectivamente. Sejam $FTF(p_1), FTF(p_2), \ldots, FTF(p_n)$ representadas pelas funções de pertencimento $\mu_{p_1}(\tau), \mu_{p_2}(\tau), \ldots, \mu_{p_n}(\tau)$, respectivamente. A FTF resultante $\mu_R(\tau)$, é dada por:

$$\mu_R(\tau) = \bigcap_{i=1}^{n} \mu_{p_i}(\tau) = \min(\mu_{p_1}(\tau), \mu_{p_2}(\tau), \ldots, \mu_{p_n}(\tau)).$$

Definição 2.69 *O operador $\diamond$ é definido como a intersecção das funções de pertencimento de seus operandos (executa o* ***Passo 1****).*

Considerando o exemplo da rede apresentada na Figura 2.9, a intersecção das FTFs das fichas nos lugares de entrada de t_1 é representada por

$$FTF(p_1) \diamond FTF(p_2) \diamond \cdots \diamond FTF(p_n)$$

2.5.2 Passo 2 - Função de Temporização Nebulosa Final

Nesse passo, a FTF resultante, obtida a partir das fichas nebulosas de entrada, é combinada com os intervalos de temporização nebulosos, associados à transição disparada.

Suponha que $\mu_R(\tau)$ representa a *FTF* resultante, considerando as fichas nebulosas de entrada para a transição da Figura 2.9. Suponha também que o intervalo de disparo D, associado à transição é $[0,0]$ (disparo instantâneo). Nesse caso, quando uma ficha alcança o lugar p_k, a função de pertencimento, representando a *FTF* associada à ficha no lugar p_k, $\mu_{p_k}(\tau)$, é a combinação entre a *FTF* resultante $\mu_R(\tau)$ e o intervalo de sensibilização E associado à transição. Essa combinação é determinada pela adição de variáveis nebulosas utilizando as funções de pertencimento [KG88]:

$$\mu_{(A(+)B)}(z) = \max_{z=x+y} \min(\mu_A(x), \mu_B(y)).$$

em que z, x e $y \in TS$ e $\mu_A(x), \mu_B(y)$ e $\mu_{(A(+)B)}(z)$ representam respectivamente a *FTF* resultante, o intervalo de sensibilização e a *FTF* final associada à ficha no lugar p_k.

Se as variáveis nebulosas são descritas pelos seus intervalos de confidência, então a soma de duas variáveis nebulosas é dada por [KG88]:

$$\forall a_1, a_3, b_1, b_3 \in I\!R : A = [a_1, a_3], \quad B = [b_1, b_3],$$
$$A(+)B = [a_1, a_3](+)[b_1, b_3] = [a_1 + b_1, a_3 + b_3].$$

O mesmo método de adição de variáveis nebulosas é utilizado quando se considera o intervalo de disparo D. Nesse caso, a *FTF* final é a combinação entre a *FTF* resultante e os intervalos nebulosos de sensibilização E e disparo D.

Definição 2.70 *O operador $\oslash$ representa a adição de variáveis nebulosas (executa o **Passo 2**).*

Considerando o exemplo da rede apresentada na Figura 2.9, a *FTF* da ficha em p_k, após o disparo de t_1, é representada por

$$FTF(p_k) = (FTF(p_1) \diamond FTF(p_2) \diamond \cdots \diamond FTF(p_n)) \oslash E \oslash D$$

2.5.3 Grafo de Alcançabilidade Nebuloso

Uma das mais conhecidas técnicas para analisar redes de Petri é a técnica do grafo de alcançabilidade, o qual é uma ferramenta muito útil para muitas propriedades das redes de Petri. O grafo de alcançabilidade [Pet81, Mur89] é um grafo dirigido que enumera

todos os estados, ou marcações, alcançáveis de um sistema modelado por uma rede de Petri, descrevendo todas as possíveis transições entre estados.

A técnica de análise baseada no grafo de alcançabilidade é muito utilizada para provar diferentes tipos de propriedades. No caso das redes de Petri que não consideram aspectos temporais, propriedades relacionadas com a correção (*correctness*) podem ser facilmente provadas [Pet81, Mur89]. Propriedades temporais e avaliação de desempenho de sistemas podem também ser efetuadas pela análise de um grafo de alcançabilidade modificado, que é uma extensão do grafo de alcançabilidade incorporando-se o tempo.

O grafo de alcançabilidade nebuloso [dF94] é um grafo de alcançabilidade modificado que incorpora os conceitos temporais definidos no modelo *RPTN*. A ideia básica é ampliar o conceito de grafo de alcançabilidade para incluir as *FTF*s carregadas pelas fichas, que são computadas depois de cada disparo de transição, a partir da marcação inicial. O grafo de alcançabilidade nebuloso ainda considera os intervalos nebulosos de tempo associados às transições. Assim, o mesmo é caracterizado por:

1. Assinalar uma *FTF* para cada ficha nos lugares em um dado estado. Isso segue a definição do modelo *RPTN* em que as fichas carregam funções nebulosas características;

2. Associar a cada arco do grafo os intervalos de tempo associados às respectivas transições disparadas, representando as restrições de tempo quando do disparo de uma transição no modelo *RPTN*.

O grafo de alcançabilidade nebuloso é construído recursivamente, calculando os sucessores de todos os estados alcançados, começando do estado inicial. O procedimento para a computação dos sucessores de um estado é apresentado a seguir:

1. Dado um estado S
2. Determinar o conjunto das transições habilitadas;
3. Para toda transição t habilitada em S faça
4. Gerar um estado sucessor S' a partir de S após disparar t;
5. Associar ao arco que conecta S a S' os intervalos de tempo correspondentes;
6. Associar aos nodos do grafo o conjunto de *FTF*s correspondentes;

7. fim_faça.

Para ilustrar a utilização das *RPTN*s e o correspondente grafo de alcançabilidade nebuloso, vejamos o exemplo de uma simples célula de manufatura modelada por uma *RPTN*, como apresentado na Figura 2.10(a). Neste exemplo, as *FTF*s iniciais e os intervalos nebulosos de tempo são definidos como números nebulosos trapezoidais. Os intervalos de sensibilização são definidos pela quádrupla $(0,0,0,0)$ e representam a sensibilização instantânea das transições.

Neste exemplo, partes de peças são colocadas em duas filas de espera para processamento (lugares p_1 e p_4 modelam, respectivamente, as filas). Partes depositadas em p_1 (p_4) são processadas pela máquina representada por p_2 (p_5) e, após processadas, depositadas em p_3 (p_6). Uma peça é o resultado da montagem de uma parte processada em p_2 e outra processada em p_5 (disparo da transição t_5).

Seja a marcação inicial $M_0 = [\ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ ]^T$, significando que o sistema é inicializado com uma ficha em p_1 e uma ficha em p_4. Os outros lugares não contêm fichas quando o sistema é inicializado. Executando o algoritmo do *grafo de alcançabilidade nebuloso*, obtém-se o grafo apresentado na Figura 2.10(b). Dados as funções de temporização nebulosas (*FTF*) das fichas em p_1 e p_4, os intervalos nebulosos D e E e utilizando as regras de disparo de transições definidas por Figueiredo [dF94], pode-se encontrar as *FTF*s das fichas criadas pelo disparo das transições, ou seja:

$FTF(p_2) = FTF(p_1) \oslash D_1$; $FTF(p_3) = FTF(p_2) \oslash D_2$;

$FTF(p_5) = FTF(p_4) \oslash D_3$; $FTF(p_6) = FTF(p_5) \oslash D_4$;

$FTF(p_7) = (FTF(p_3) \diamond FTF(p_6)) \oslash D_5$.

Utilizando-se os valores da Figura 2.10(a), tem-se que $FTF(p_7) = (5, 9, 12, 16)$ na marcação (estado) $M_9 = [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ ]^T$. A função $FTF(p_7)$ define a possibilidade de existir uma ficha em p_7 no estado M_9.

Mais exemplos e aplicações de redes de Petri com temporização nebulosa podem ser vistos em [dF94, dFP95, PF95a, PF95b].

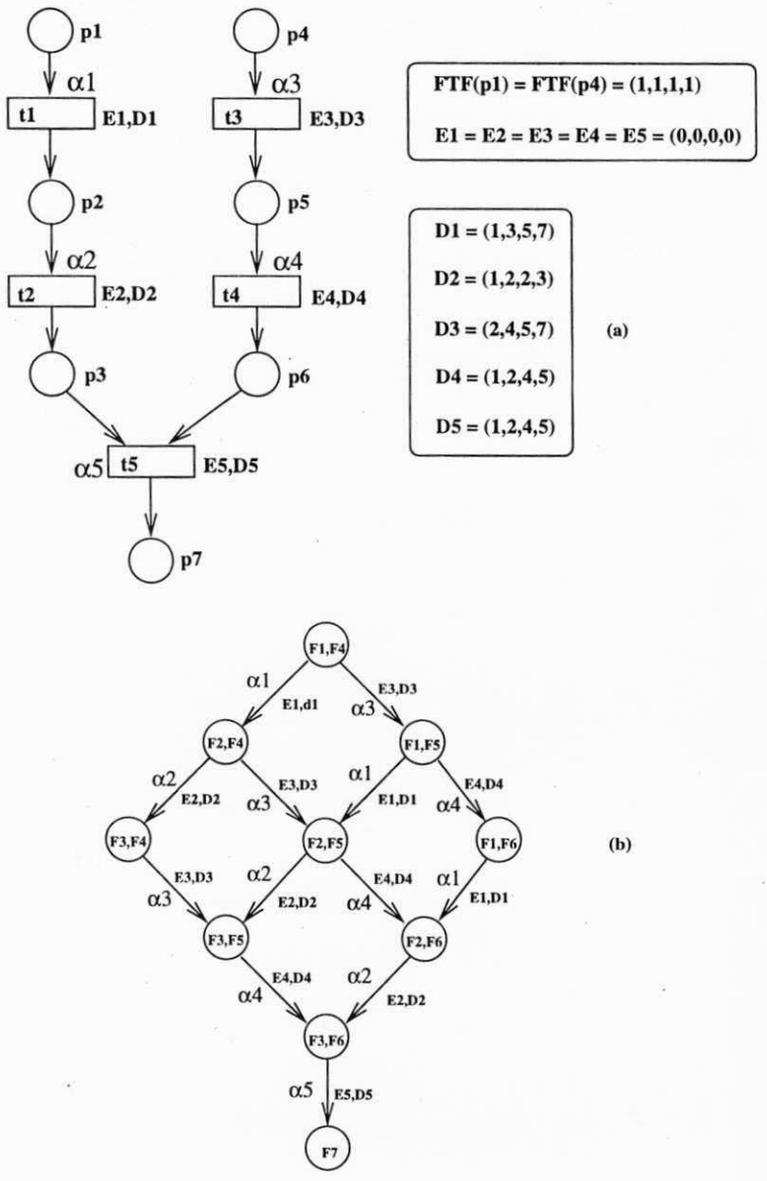


Figura 2.10: (a) Modelo *RPTN* de uma célula de manufatura; (b) Grafo de alcançabilidade nebuloso.

2.6 *G-Nets* e Sistemas de *G-Nets*

A utilização de redes de Petri (RPs) para a modelagem e análise de sistemas a eventos discretos (SEDs) reais pode se tornar intratável devido ao problema do crescimento exponencial dos estados da rede com o crescimento do sistema modelado. Logo, para projetar, analisar ou avaliar o desempenho de SEDs reais, é necessário uma ferramenta de estruturação que permita o gerenciamento da complexidade da construção do modelo e de sua análise.

As Redes de Petri de Alto Nível, como por exemplo o modelo de rede de Petri Coloridada de Jensen [Jen92] e as redes Predicado/Transição de Genrich [GL79, Gen87], são uma classe de extensões de redes de Petri propostas com o objetivo de reduzir a complexidade de modelagem e análise de SEDs reais. Estas extensões podem representar de forma simples, elegante e detalhada os aspectos funcionais de um sistema. Entretanto, sua avaliação permanece difícil uma vez que o conceito de modularidade não é inerente a estes tipos de rede, isto é, não foi definido como dividir o sistema em partes, estudá-las em separado e depois combinar os resultados parciais para chegar a uma solução global. Para superar estas dificuldades foi introduzido o conceito de sistemas de *G-Nets* [DC91, DC92].

G-Net é um modelo de especificação executável multi-nível baseado em redes de Petri que incorpora os conceitos de módulo e estrutura de sistema às redes de Petri para promover abstração, encapsulamento e fraco acoplamento entre módulos [Per94]. Um sistema de *G-Nets* é uma composição de uma especificação de sistemas baseados em *G-Nets*. Uma vez que a estrutura de *G-Nets* utiliza os conceitos do paradigma orientado a objetos [Boo91, JCJO92], a interação entre *G-Nets*, em um sistema de *G-Nets*, é realizada através de interfaces bem definidas, como será visto a seguir.

2.6.1 Conceitos Básicos

Os sistemas de *G-Nets* [PdFC94, Per94, DC91, DC92] são uma estrutura baseada em redes de Petri para a modelagem e especificação de sistemas. Essa estrutura é o resultado da integração da teoria de redes de Petri com uma abordagem de engenharia de software orientada a objeto para a especificação e análise de sistemas complexos. Os sistemas de *G-Nets* são compostos por um certo número de módulos denominados de *G-Nets*. Cada

G-Net pode ser vista como um módulo funcional ou um objeto e suas operações associadas são denominadas métodos.

Uma *G-Net, GN*, em um sistema de *G-Nets* é composta de duas partes: um lugar especial denominado *Lugar Genérico de Chaveamento* (*Generic Switching Place - GSP*) e uma *Estrutura Interna* (*Internal Structure IS*). O *GSP* fornece uma abstração da estrutura interna de um módulo e serve como interface entre uma *G-Net* e outros módulos. A estrutura interna é uma rede de Petri modificada, que representa a realização interna detalhada da aplicação modelada. A notação para *G-Nets* é apresentada na Figura 2.11 e explicada como a seguir:

1. A estrutura interna da rede (*IS*) é definida por um retângulo com as bordas arredondadas, definindo os limites da estrutura interna;

2. O lugar genérico de chaveamento (*GSP*) é indicado por uma elipse no canto superior esquerdo do retângulo, definindo os limites da *IS*. A inscrição *GSP(nome_da_rede)* define o nome da rede, para ser referida por outras *G-Nets*;

3. O retângulo com as bordas arredondadas no canto superior direito dos limites da *IS*, é utilizado para identificar os métodos e atributos da rede, onde:

   - $\langle nome\_do\_atributo \rangle = \{\langle tipo \rangle\}$ define um atributo para a rede, onde $\langle nome\_do\_atributo \rangle$ é o nome do atributo e $\langle tipo \rangle$ é o tipo para o atributo, o qual está restrito ao conjunto de inteiros não negativos;
   - $\langle nome\_do\_método \rangle$ é o nome para um método, $\langle descrição \rangle$ é uma descrição para o método. $\langle p1{:}descrição, \cdots, pn{:}descrição \rangle$ é uma lista de argumentos para o método.

4. Um círculo representa um lugar normal;

5. Uma elipse, na estrutura interna, representa um *lugar de chaveamento para instanciação* (*instantiated switching place* (*isp*)). O *isp* é utilizado para prover comunicação *inter-G-Net*. A inscrição *isp(GN'.mi)* indica a invocação da rede *GN'* com o método *mi*;

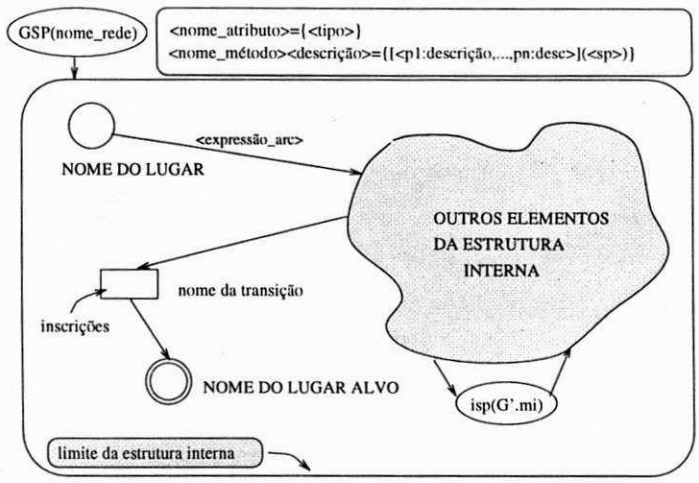


Figura 2.11: Notações usadas para representar as *G-Nets*.

6. Um retângulo representa uma transição, a qual pode ter uma inscrição associada. Esta inscrição pode tanto ser uma atribuição como uma restrição de disparo;

7. Um círculo duplo representa um lugar de terminação ou *lugar alvo*;

8. Lugares e transições são conectados por arcos direcionados, os quais possuem expressões associadas.

O *GSP* em uma *G-Net GN*, denotado por *GSP(nome_da_rede)* na elípse da Figura 2.11, identifica de forma única o módulo. O *GSP* contém a declaração para um ou mais métodos executáveis (descritos no retângulo com as bordas arredondadas na Figura 2.11), especificando as funções, operações ou serviços definidos para a rede, e um conjunto de atributos, especificando propriedades passivas do módulo (quando definidas). A estrutura detalhada e o fluxo de informação de cada método são definidos, na estrutura interna, por uma rede de alto-nível modificada. Mais especificamente, um método define os parâmetros de entrada, a marcação inicial correspondente à rede de alto-nível interna (o estado inicial da execução). A coleção de métodos e os atributos (se definidos) proveem a abstração ou visão externa do módulo.

Na estrutura interna, lugares representam primitivas e transições, juntamente com os arcos, representam conexões ou relações entre as primitivas. Estas primitivas podem ser predicados ou lugares genéricos para chaveamento. Um conjunto de lugares especiais

denominados *Lugares Alvo* representa o estado final de uma execução e os resultados a serem retornados (se algum). Uma transição, juntamente com os arcos, define uma sincronização e coordena a transferência de informação entre seus lugares de entrada e saída. Nesta mesma Seção estes conceitos serão introduzidos formalmente.

Dada uma *G-Net GN*, um *isp* em *GN* é denotado por $isp(GN_{nome}.mtd)$ (ou simplesmente *isp(GN)*, se nenhuma ambiguidade ocorrer), onde GN_{nome} é uma identificação única para *GN* e *mtd* é um método definido para *GN*. Um $isp(GN_{nome}.mtd)$ denota uma instanciação da *G-Net GN*, isto é, uma instância de invocação de *GN* baseada no método *mtd*. Portanto, executando a primitiva *isp* implica na invocação de *GN* (através do envio de fichas para *GN*) utilizando um método especificado. As fichas contêm os parâmetros necessários para a definição das fichas na marcação inicial da rede invocada. O *isp* serve como mecanismo primário para a conexão ou relacionamento entre diferentes *G-Net*s (módulos).

A seguir são apresentadas as definições formais relacionadas a *G-Net*s e sistemas de *G-Net*s.

2.6.2 Definições Formais

Formalmente, um sistema de *G-Net*s é assim definido [Per94]:

Definição 2.71 *Um sistema de* G-Net*s é uma tripla* $GNS = (TS, GS, AS)$*, onde:*

- TS *é uma coleção de fichas geradas dinamicamente durante a execução do sistema;*
- GS *é um conjunto de* G-Net*s e;*
- AS *é um conjunto de agentes de computação concorrentes e descentralizados, executando um sistema de* G-Nets.

Definição 2.72 *Uma* G-Net *é uma dupla* $GN = (GSP, IS)$*, onde:*

- GSP *é um* lugar genérico de chaveamento *que fornece uma abstração para a* G-Net*;*
- IS *é a* estrutura interna*, a qual é uma rede de Petri modificada.*

Definição 2.73 *Um GSP* $\in$ *GN é um lugar genérico de chaveamento definido por (*NID,MS,AS*), onde:*

- NID *é a identificação (nome) da* G-Net GN*;*
- MS *é um conjunto de métodos e*

  $\forall mtd_i \in MS, mtd_i = (m\_nome_i, m\_argumentos_i, m\_iniciador_i)$*, onde:*

  - m_nome_i *é um nome para o método* mtd_i*;*
  - $m\_argumentos_i$ *é um conjunto de variáveis especificando a ficha de entrada para o método* mtd_i*;*
  - $m\_iniciador_i$ *é um mapeamento dos* $m\_argumentos_i$*, associados ao método* mtd_i*, na marcação inicial da rede da estrutura interna.*

- AS *é um conjunto de atributos e* $\forall as_i \in AS, as_i \in I\!N$.

Pela Definição 2.73, um *GSP* é definido de forma única por um nome (*NID*) que abstrai um conjunto de métodos, (*MS*). Os métodos definem como a estrutura interna de uma *G-Net* pode ser executada. Em outras palavras, o conjunto de métodos define todos os possíveis conjuntos de marcações iniciais, as quais definem diferentes comportamentos para a estrutura interna. Além disso, o *m_iniciador* define como a informação, nos *m_argumentos*, é transformada em fichas do tipo correspondente. Essas fichas serão depositadas no lugar inicial para o método.

Definição 2.74 *A estrutura interna de uma* G-Net, GN.IS, *é uma rede de Petri modificada.*

Como mostrado nesta seção, existem três diferentes tipos de lugares na estrutura interna de uma *G-Net.* A seguir apresentamos as definições associadas aos diferentes tipos de lugares.

Definição 2.75 *Dado que* GN.IS *é uma estrutura interna de uma* G-Net, *o conjunto de lugares* $P \in GN.IS$ *é definido por:*

$$P = (\mathcal{ISP}, \mathcal{NP}, \mathcal{GP}), \text{ onde}$$

- $\mathcal{ISP}$ *é um subconjunto de lugares de chaveamento instanciados, denotados por elípses;*
- $\mathcal{NP}$ *é um subconjunto de lugares normais, denotados por círculos;*
- $\mathcal{GP}$ *é um subconjunto de lugares alvo (*goal places*), denotados por círculos duplos.*

Os $\mathcal{ISP}$ e $\mathcal{GP}$ são muito importantes para a decomposição, análise e verificação dos sistemas de *G-Nets*.

Um $isp \in \mathcal{ISP}$ é o mecanismo usado pela *G-Net* para implementar a interconexão entre *G-Nets*. Um $isp \in \mathcal{ISP}$ definido em uma *G-Net GN* e invocando uma rede *GN'* é denotado por *isp(GN'm)* e é definido pela quádrupla:

$$isp(GN'm) = (NID, mtd, ação\_antes, ação\_após)$$

NID é o identificador único de *GN'*, $mtd \in GN'.GSP.MS$, *ação_após* e *ação_antes* são ações primitivas cuja função é atualizar a seqüência de propagação de ficha, introduzida a seguir. Mais especificamente, um método $mtd \in GN.MS$ define os parâmetros de entrada e a marcação inicial da rede de Petri interna (estado inicial da execução). O conjunto de *Lugares Alvo* representa o estado final da execução para cada método, bem como os resultados a serem retornados. A coleção de métodos e atributos (se algum for definido) provê a abstração ou visão externa do módulo. Impõe-se a restrição de que o conjunto de lugares alvo deve ser não vazio. Essa restrição garante que um método sempre alcançará um estado final.

Definição 2.76 *Dado* GN *ser uma* G-Net *e* tkn *ser uma ficha, então* tkn *é uma tupla de itens na forma*

$$tkn = \langle seq, scor, d_1 \cdots, d_q \rangle, \text{ onde:}$$

- $tkn.seq$ *é a* seqüência de propagação *da ficha;*
- $tkn.scor \in (antes, após)$ *é a* cor de estado *e;*
- $tkn.d_i$, $i \in \mathbb{N}$ *é a mensagem associada à ficha.*

Definição 2.77 *O item tkn.seq é uma seqüência de* $\langle NID, isp, PID \rangle$*, onde* NID *é a identificação da* G-Net*,* $isp \in \mathcal{ISP}$ *e* PID *é um identificador único de processo.*

A ficha estará disponível para habilitação e disparo de transições quando *scor* = *após*.

A seqüência de propagação de uma ficha somente é alterada em um *isp* ou *GSP*. Quando uma *G-Net GN* é invocada por um isp_i em outra *G-Net GN'* (quando isp_i recebe uma ficha), a tripla $\langle GN, isp_i, Pid_{G'} \rangle$, onde $Pid_{G'}$ é o identificador do processo executando *GN'*, é associada à seqüência de propagação da ficha antes dela ser enviada a *G-Net GN*. Essa tripla indica que quando a execução de *GN* termina, a ficha resultante deverá retornar ao lugar identificado por isp_i na *G-Net GN'*. O identificador do processo é necessário para distinguir para qual instância de execução de *GN'* a ficha sendo retornada pertence. Quando a ficha de entrada é recebida pelo *GSP GN*, a tripla $\langle 0, 0, Pid_G \rangle$ é associada à seqüência de propagação da ficha, indicando que o agente responsável pela execução da invocação é identificado por Pid_G. Como Pid_G é único, a seqüência de propagação é também única. A estrutura da ficha em *G-Nets* não somente garante que todas as fichas pertencendo a uma instância de execução de uma *G-Net* têm a mesma seqüência de propagação, mas também contém a história completa de propagação das fichas, a qual governa as interações entre os processos executando um sistema de *G-Nets*.

Devido ao campo de seqüência de propagação, associado à ficha, mais de uma execução de uma *G-Net* pode ocorrer simultaneamente, já que diferentes seqüências de execução (invocações) podem ser unicamente identificadas.

Como mencionado anteriormente, a cor de estado da ficha pode assumir dois valores possíveis, tanto *antes* como *após*. Uma ficha é dita estar *disponível* se sua *scor* = *após*. Caso contrário, ela é dita estar *indisponível*. Quando uma ficha é depositada em um lugar, sua cor de estado é *antes*. Após a ação primitiva (se definida) ser executada, a cor da ficha é alterada para *após*, indicando que a ficha está pronta para ser utilizada para disparo de transições.

Um lugar normal $NP \in \mathcal{NP}$ possui as regras para manipular a ficha. Quando uma ficha é depositada em um *NP*, uma ação associada a este lugar é executada baseada nos parâmetros de entrada especificados pelo campo *msg*. O resultado da ação é associado ao campo *msg* e a *cor de desvio* da ficha é definida. A cor de desvio da ficha serve para definir para qual transição de saída a ficha está disponível. Portanto, a ação associada aos

lugares normais executa a mesma função das funções associadas aos arcos e das expressões associadas às transições, em uma *CPN*. Finalmente, o campo *scor* da ficha é atualizado para *scor* $\leftarrow$ *após*, e a ficha está disponível para habilitação e disparo de transições.

Um lugar alvo $GP_i \in \mathcal{GP}$ deve pertencer à marcação final de *GN.IS*. Para cada método *mtd*, definido para uma *G-Net*, os lugares alvo devem ser alcançáveis e devem ser únicos para cada método. Desta forma, a informação resultante da execução pode retornar para a *G-Net* invocadora.

O mecanismo de disparo de uma transição em uma *G-Net* é definido pelas seguintes regras de disparo:

Definição 2.78 *Dado* GN *ser uma* G-Net *e* GN.IS *ser uma rede de Petri colorida, as regras de disparo de transição são definidas por:*

1. *Uma transição* t *é dita habilitada se e somente se o elemento de ligação* (t,b) *estiver habilitado e,*

   (a) *todas as fichas envolvidas possuem a mesma seqüência de propagação;*

   (b) *todas as fichas possuem suas* scor = após*;*

   (c) *o número de fichas nos lugares de saída de* t *é menor que a capacidade dos lugares.*

2. *Uma transição* t *habilitada pode disparar. Quando* t *dispara:*

   (a) *Um multiconjunto de fichas é removido dos lugares de entrada da transição, de acordo com as expressões nos arcos de entrada da transição;*

   (b) *Um multiconjunto de fichas, cuja seqüência de propagação é a mesma que a das fichas removidas e cujas* scor=após*, é adicionado aos lugares de saída da transição, de acordo com as expressões nos arcos de saída da transição;*

A invocação de uma *G-Net* define os mecanismos para iniciar a execução das estruturas internas, baseando-se na mensagem associada à ficha.

Definição 2.79 *Dado* GN *ser uma* G-Net*, a invocação da rede* GN *baseada no método* $mtd \in GN.MS$ *é conduzida da seguinte forma:*

1. *Determine a marcação inicial de* GN *baseado na definição para o método (o conteúdo das fichas depende da ficha de entrada);*

2. *Dispare as transições habilitadas, se existirem;*

3. *Invoque as primitivas habilitadas, se existirem;*

4. *Repita (2)-(3) até alcançar um lugar alvo;*

5. *Envie o resultado da execução, se definido pelo mtd, para a rede invocadora.*

A seguir apresentamos o exemplo de um sistema produtor/consumidor. Este sistema consiste da sincronização entre um ou mais produtores e um ou mais consumidores. Para este exemplo, assume-se que inicialmente um produtor é capaz de produzir n items, e que o consumidor é capaz de consumir um item por vez. Este exemplo é apresentado por Perkusich [Per94].

Na Figura 2.12 é apresentada a *G-Net* $GN(P)$ para o produtor. Para esta rede um método é definido e é denominado *mp*. A função do método *mp* é produzir n itens para serem consumidos. Quando $GN(P)$ é invocada através de *GSP(P)* uma ficha, juntamente com o campo n, expressando o número de itens a produzir é depositada no lugar *PR*. A ação associada ao lugar simplesmente decrementa o campo n. Se $n < 0$, a transição *pr* dispara e a invocação de $GN(P)$ termina quando o *lugar alvo GP* é alcançado. Contrariamente, isto é $n \geq 0$, a transição *rs* dispara, a ficha alcança o lugar *isp(C.ms)*, e a rede C é invocada utilizando o método *ms*. Esta invocação serve para consultar o estado de $GN(C)$, de modo a garantir se ela está pronta ou não para consumir um item. Se a rede $GN(C)$ não está pronta, a transição *nr* dispara, e a rede $GN(C)$ é consultada novamente. De outro modo, o consumidor está pronto e a transição *cr* dispara. Após o disparo de *cr* uma ficha é depositada no lugar *isp(C.mc)* e a rede C é invocada com o método *mc*, juntamente com o item a ser consumido. Quando a ficha é retornada pela rede C a transição *co* dispara e uma ficha é novamente depositada no lugar *PR*.

A *G-Net* modelando o consumidor é mostrada na Figura 2.13. Dois métodos são definidos para $GN(C)$: método estado (ms), e método consome (mc). Quando $GN(C)$ é invocada através *GSP(C)*, se o método é *ms* uma ficha é depositada no lugar *IN*. Dependendo do estado de $GN(C)$, o qual pode ser tanto consumindo como pronta para

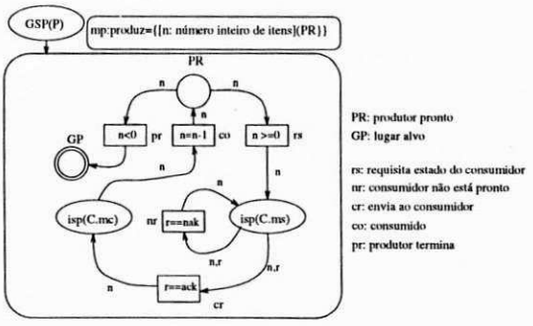


Figura 2.12: *G-Net* $GN(P)$ modelando o produtor

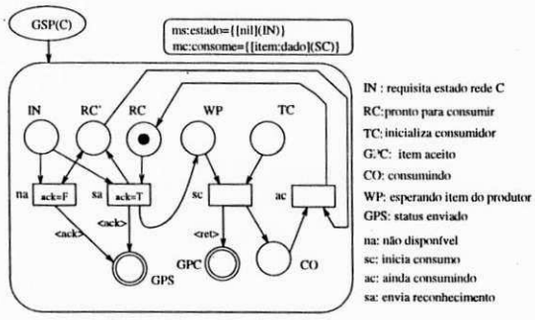


Figura 2.13: *G-Net* $GN(C)$ modelando o consumidor

consumir, a transição *na* ou *sa* disparará. A escolha entre *na* e *sa* baseia-se nos estados representados pelo lugar *RC* (pronto para consumir) e lugar *CO* (consumindo). Se *RC* está marcado, a transição sa dispara e uma ficha alcança o lugar alvo *GPS* com um campo reconhecido (*ack*) associado a ele. Contrariamente, um campo não reconhecido (*nak*) é associado à ficha. Após o disparo de *sa*, uma ficha é depositada no lugar de entrada *WP*, e o consumidor está pronto para consumir. Quando $GN(C)$ é invocada com o método consome, *mc*, uma ficha é depositada no lugar *TC*. Uma vez que *WP* foi previamente marcado, após uma execução com sucesso do método *ms*, a transição *sc* dispara e o lugar *CO* é marcado. Neste ponto $GN(C)$ indica ao produtor que a ação requerida está sendo processada ou ainda está em processamento. Isto é porque quando a transição *sc* dispara uma ficha é depositada no lugar *CO* e outra no lugar alvo *GPC*. Deve ser notado que a transição *ac* pode ou não disparar antes que o produtor receba a ficha de retorno de *GPC*. Após o disparo de *ac*, uma ficha é removida de *CO* e uma outra é removida de *RC'*, e uma ficha é depositada em *RC*. Logo $GN(C)$ está pronta para consumir um novo item.

2.6.3 Decomposição e Análise de Sistemas de *G-Nets*

Como apresentado anteriormente, os sistemas de *G-Nets* possuem as seguintes características:

- A visão externa de um módulo é fracamente acoplada, assim, a independência entre componentes é a mais alta possível e somente alguns relacionamentos bem definidos são permitidos;
- Externamente, um módulo apresenta um alto nível de coesão funcional, de forma que sua contribuição e participação no sistema como um todo é claramente definida.

Utilizando-se destas características, define-se um protocolo de alto nível, que determina como as *G-Nets* são conectadas (interagem). Sistemas de redes de Petri podem ser conectados tanto pela fusão de transições, fusão de lugares, ou por arcos.

A fusão de transições facilita a análise do modelo, dado que muitas propriedades são preservadas. No entanto, as redes ficarão fortemente acopladas e, neste caso, não se pode considerar uma rede como um sistema autônomo, visto que não se pode decidir se uma transição estará ou não localmente habilitada com base apenas no estado local da rede em questão.

A composição de redes pela fusão de lugares corresponde à comunicação pelo compartilhamento de variáveis. Este método não apresenta as mesmas desvantagens da fusão por transições, mas é necessário prover sincronização adicional às redes fundidas. Isto é feito através da adição de lugares ou transições, o que pode levar à necessidade de analizar c sistema como um todo novamente.

A composição de redes através de arcos corresponde ao mecanismo de *mensagem passante.*

Uma outra abordagem para a composição de redes é a utilização de uma outra rede como meio de comunicação. Através desta abordagem, restrições podem ser impostas à rede de comunicação, ou parte das redes comunicantes, representando o meio de comunicação, ao invés de impor restrições às redes compostas. A abordagem adotada por Perkusich [Per94] é bastante semelhante a esta.

A abordagem de decomposição de *G-Nets*, desenvolvida por Perkusich [Per94], evita problemas estruturais entre *G-Nets*, possibilita análise formal e permite a composição de

G-Nets de forma estruturada. Para tanto, diferentes partes de um modelo são independentemente consideradas e a análise e correção são executadas a nível de componentes, dado que a interface entre esses componentes é imutável.

Quando *G-Nets* são decompostas, obtém-se um claro acoplamento estrutural entre as mesmas. Este acoplamento impõe restrições na comunicação entre *G-Nets*. Além disso, a decomposição dos elementos de interface de uma *G-Net* (*GSP, isp* e *lugares alvo*) permite que um sistema seja concebido e analizado de forma incremental. Para tanto, a composição de duas *G-Nets* deve preservar as propriedades de cada uma, de modo que as propriedades do sistema global possam ser deduzidas a partir das propriedades de seus componentes.

Ao se analizar um sistema de *G-Nets*, o objetivo é prover mecanismos para gerenciar o problema da explosão de estados. Assim, considera-se um sistema de *G-Nets* como sendo a composição de um certo número de *G-Nets*. Analizam-se as propriedades dos componentes individuais, verificando se estas propriedades são válidas para o sistema global.

A metodologia de análise desenvolvida em [Per94] combina análise lógica e comportamental. A análise comportamental é aplicada para verificar o comportamento local. A análise lógica baseia-se no paradigma *assume/garante* para sistemas de transição de estados. Como uma *G-Net* interage em um sistema de *G-Nets*, o comportamento de uma *G-Net*, ou seja, sua visão decomposta, depende da reação de outras *G-Nets* que formam seu ambiente. Logo, a interface deve conter restrições a respeito das reações esperadas do ambiente. Portanto, as interfaces descrevem as propriedades dos serviços que serão garantidos pela *G-Net*. Como a interface é o comportamento visível de uma dada *G-Net*, a realização interna da mesma deve satisfazer o comportamento definido pela interface.

Para maiores informações sobre *G-Nets* e sistemas de *G-Net*, o leitor deve se referir a [Per94, DC91, DC92, DCdFP93].

Capítulo 3

Teoria de Controle Supervisório

Neste Capítulo apresentamos a *Teoria de Controle Supervisório* (TCS), desenvolvida em sua forma básica por Ramadge e Wonham [RW89, RW87c, RW87b].

Como apresentado no Capítulo 2, um sistema a eventos discretos (SED) pode ser representado por um gerador G tal que a linguagem gerada $L(G) = L$ e a linguagem marcada $L_m(G) = L_m$. No entanto, sabe-se da teoria de linguagens [HH79] que nem todas as linguagens podem ser representadas por autômatos ou geradores finitos. Como citado no Capítulo 2 é provado que a classe de linguagens representada por autômatos, ou geradores finitos é a classe de linguagens regulares.

Toda a teoria e resultados da TCS, desenvolvidos por Ramadge e Wonham, são baseados na teoria de linguagens e autômatos, portanto, os modelos de SEDs apresentados não se limitam àqueles representados por geradores finitos, embora os resultados computacionais mais precisos e definitivos se limitem àqueles representados por geradores finitos.

Tomando como exemplo a utilização de um recurso (por exempo uma máquina) por um usúario, apresentado na Figura 2.3, e supondo que o recurso pode se danificar quando em utilização, tem-se o novo modelo apresentado na Figura 3.1. Nesse caso o gerador G modela uma máquina com três estados possíveis, são eles **I** (*inativo*), **U** (*em uso* ou ativa) e **M** (*em manutenção*). As transições entre os estados são identificadas pelos eventos do alfabeto $\Sigma = \{\alpha, \beta, \lambda, \mu\}$. A máquina inicialmente está em repouso e quando é requisitada passa para o estado *em uso* através do evento α. Quando em uso, a máquina

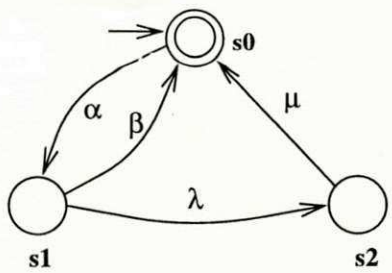


Figura 3.1: Gerador representando a utilização de um recurso por um usuário.

pode retornar ao seu estado inativo, após o término de sua tarefa, através do evento β, ou pode se danificar, passando ao estado *em manutenção* através do evento γ. Após a manutenção, a máquina retorna ao estado *inativo* através do evento μ.

A linguagem gerada, $L(G)$, é o conjunto de todas as palavras geradas a partir do estado inicial **I**. Assim, a linguagem $L(G)$ é representada da seguinte forma:

$$L(G) = (\alpha\beta + \alpha\lambda\mu)^*(\epsilon + \alpha + \alpha\lambda).$$

Como o único estado marcado de G é o inicial, então a liguagem marcada $L_m(G)$ é formada por todas as palavras que representam um ciclo completo no grafo, ou seja:

$$L_m(G) = (\alpha\beta + \alpha\lambda\mu)^*.$$

Note que $L(G)$ representa o comportamento fisicamente possível do sistema modelado, enquanto $L_m(G)$ representa as tarefas a serem realizadas pelo sistema. Nesse caso, pode-se utilizar a definição de coacessibilidade de um gerador como critério para a existência ou não de bloqueio no sistema. Como visto na Definição 2.25, um gerador é coacessível se $L(G) = \overline{L_m(G)}$. Isso significa que toda palavra gerada é prefixo de alguma palavra marcada, ou seja, toda seqüência de eventos fisicamente possível tem pelo menos uma outra seqüência de eventos concatenada que leva a uma tarefa completa. O gerador representado na Figura 3.1 é coacessível, também dito *não bloqueável.*

3.1 Geradores Controlados

O modelo de um SED como descrito até aqui, por exemplo o gerador da Figura 3.1, é simplesmente um gerador espontâneo de cadeias de eventos, sem um controle externo.

No entanto, muitas vezes é desejável que o SED realize uma tarefa específica para a qual a ocorrência de algumas seqüências de eventos fisicamente possíveis, deve ser inibida. Para tanto, é necessário uma ação de controle externa sobre o SED. Para modelar a ação de controle de um SED, é necessário admitir que alguns eventos do sistema podem ser desabilitados quando desejado. Assim, particiona-se o alfabeto Σ em eventos *controláveis* Σ_c e eventos *não controláveis* Σ_u, de modo que:

$$\Sigma = \Sigma_c \cup \Sigma_u \text{ e } \Sigma_c \cap \Sigma_u = \emptyset$$

ou seja, os eventos em Σ_c podem ser desabilitados em qualquer instante de tempo, enquanto aqueles em Σ_u não sofrem influência da ação de controle. Essa partição do alfabeto de eventos reflete características de sistemas reais, como por exemplo a quebra de uma máquina em um sistema de manufatura é um evento não controlável, enquanto o início de operação de uma máquina é um evento controlável.

Dado o alfabeto de eventos $\Sigma = \Sigma_c \cup \Sigma_u$, denomina-se *entrada de controle* o conjunto de eventos habilitados em um determinado estado. Formalmente, tem-se:

Definição 3.1 *Dados um gerador* $G = (\Sigma, Q, \delta, q_0, Q_m)$, *cujo alfabeto é particionado em* $\Sigma = \Sigma_c \cup \Sigma_u$, *o* conjunto de entradas de controle *associado a* G *é dado por:*

$$\Gamma = \{\gamma : \Sigma_u \subseteq \gamma \subseteq \Sigma\}.$$

A condição $\Sigma_u \subseteq \gamma$ significa que os eventos em Σ_u estarão sempre habilitados, pois os mesmos não sofrem influência da ação de controle.

Definição 3.2 *Seja* $\Gamma \subseteq 2^{\Sigma}$ *o conjunto de entradas de controle. Um* gerador controlado G_c *é um par* (G, Γ), *onde* G *é um gerador com alfabeto particionado em eventos controláveis e não controláveis, equipado com um conjunto de entradas de controle* Γ.

Como na teoria de controle clássica, denomina-se *planta* o modelo do sistema a ser controlado. A linguagem gerada é então denominada *linguagem da planta* e representa o comportamento do sistema na ausência de qualquer ação de controle.

Quando se aplica uma entrada de controle γ a uma planta, esta se comporta como se os eventos inibidos fossem momentaneamente apagados de sua estrutura de transição.

Assim, a função de transição δ de um gerador, cujo alfabeto é particionado em eventos controláveis e não controláveis, não está definida para os eventos inibidos por uma dada entrada de controle aplicada ao gerador em um dado instante. Nesse sentido, o mecanismo de controle adotado pela TCS consiste exatamente em chavear as entradas de controle em resposta à seqüência de eventos gerada pelo sistema, de modo que a linguagem gerada sob a ação de controle especifique um conjunto de tarefas a serem realizadas.

Considerando o gerador da Figura 3.1, suponha que o início de cada ciclo de operação (evento α) e o término da manutenção (evento μ) sejam eventos controláveis, ou seja: $\Sigma_c = \{\alpha, \mu\}$ e $\Sigma_u = \{\beta, \lambda\}$. Nesse caso, o conjunto de entradas de controle válidas são $\Gamma = \{(\beta, \lambda), (\alpha, \beta, \lambda), (\beta, \lambda, \mu), (\alpha, \beta, \lambda, \mu)\}$.

Embora esse mecanismo de controle seja intuitivamente simples, na prática é necessário que se encontrem as condições para a existência do controlador que faça o chaveamento das entradas de controle da planta, de modo que uma dada especificação seja satisfeita. Nesse sentido, apresentaremos a seguir a noção de supervisor e as condições para a existência do mesmo.

3.2 Supervisores

Como dito anteriormente, a ação de controle de um SED G consiste em chavear a entrada de controle através das entradas de controle γ, γ', γ'', ... $\in \Gamma$, em resposta à cadeia de eventos previamente gerada por G. Essa ação de controle é realizada por um agente externo denominado de supervisor. Formalmente, tem-se:

Definição 3.3 *Um supervisor para o gerador controlado $G_c = (G, \Gamma)$ é um par $\mathbf{S} = (S, \Theta)$, composto de um gerador $S = (\Sigma, X, \xi, x_0, X_m)$ e de um mapa de controle Θ, onde:*

Σ *é o mesmo alfabeto do gerador* G;

X *é um conjunto de estados;*

$\xi : \Sigma^* \times X \rightarrow X$ *é uma função de transição parcial estendida;*

$x_0 \in X$ *é o estado inicial;*

$X_m \subseteq X$ *é o conjunto de estados marcados e;*

$\Theta : X \rightarrow \Gamma$ *é uma função que associa a cada* $x \in X$ *uma entrada de controle* $\gamma \in \Gamma$.

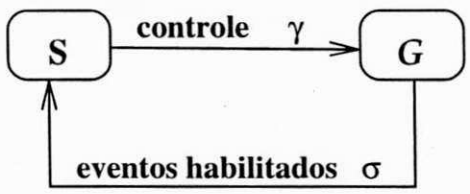


Figura 3.2: Supervisão de um SED.

A Figura 3.2 apresenta o modelo do sistema composto em malha fechada, do gerador controlado G_c supervisionado pelo supervisor **S**.

A função Θ, também denominada *mapa de controle*, associa a cada estado $x \in X$ uma entrada de controle $\gamma \in \Gamma$ para ser aplicada a G.

Pelo exposto, nota-se que a ação de controle modifica as linguagens associadas ao gerador, visto que algumas seqüências de eventos, antes possíveis de ocorrer, agora podem estar inibidas pela ação de controle. Assim, o comportamento do sistema supervisionado pode ser formalizado pelas linguagens definidas a seguir:

Definição 3.4 *Dados um gerador G_c e um supervisor **S**, a linguagem gerada pelo sistema supervisionado, denotada $L(\mathbf{S}/G)$, é assim definida:*

$$\epsilon \in L(\mathbf{S}/G) \text{ e}$$

$$s\sigma \in L(\mathbf{S}/G) \text{ se e somente se } s \in L(\mathbf{S}/G) \wedge s\sigma \in L(G) \wedge \sigma \in \Theta(\xi(s, x_0)).$$

Note que pela definição 3.4 somente os eventos habilitados em cada estado ocorrem sob supervisão.

Dado que uma palavra s pertence a $L(\mathbf{S}/G)$ se e somente se pertencer a $L(G)$, então

$$L(\mathbf{S}/G) \subseteq L(G) \tag{3.1}$$

Dado que uma palavra $s\sigma \in L(\mathbf{S}/G)$ somente se $s \in L(\mathbf{S}/G)$, então diz-se que $L(\mathbf{S}/G)$ é prefixo-fechada, isto é:

$$L(\mathbf{S}/G) = \overline{L(\mathbf{S}/G)}. \tag{3.2}$$

Definição 3.5 *Dados um gerador G_c e um supervisor **S**, a linguagem controlada do sistema supervisionado, denotada $L_c(\mathbf{S}/G)$, é assim definida:*

$$L_c(\mathbf{S}/G) = L(\mathbf{S}/G) \cap L_m(G).$$

Assim, pode-se dizer que a linguagem controlada é simplesmente a parte da linguagem marcada original que *sobrevive* sob a ação do controle, ou seja, $L_c(\mathbf{S}/G)$ representa as tarefas que podem ser completadas sob supervisão. Em conseqüência, tem-se:

$$L_c(\mathbf{S}/G) \subseteq L(\mathbf{S}/G) \tag{3.3}$$

e

$$L_c(\mathbf{S}/G) \subseteq L_m(G) \tag{3.4}$$

Definição 3.6 *Dados um gerador G_c e um supervisor $\boldsymbol{S}$, a linguagem marcada do sistema supervisionado é assim definida:*

$$L_m(\mathbf{S}/G) = L_c(\mathbf{S}/G) \cap L_m(G) \tag{3.5}$$

As Equações 3.1, 3.3 e 3.5 nos levam à seguinte conclusão:

$$L_m(\mathbf{S}/G) \subseteq L_c(\mathbf{S}/G) \subseteq L(\mathbf{S}/G) \subseteq L(G). \tag{3.6}$$

Pelo exposto, interpreta-se a linguagem $L(\mathbf{S}/G)$, gerada pelo sistema composto $\mathbf{S}/G_c$, como o conjunto de todas as possíveis seqüências finitas de eventos que podem occorrer no sistema. Para tanto, é necessário garantir que os eventos em $\mathbf{S}$ só ocorram quando eles também ocorram em G_c e estão habilitados por Θ. Esta relação é formalizada da seguinte forma:

Definição 3.7 *Um supervisor $\boldsymbol{S}$ é dito ser* completo*, em relação a um gerador G_c, quando o seguinte é verdadeiro: para todo $s \in \Sigma^*, \sigma \in \Sigma$ as três condições*

$$1)\ s \in L(S/G),$$
$$2)\ s\sigma \in L(G)\ e$$
$$3)\ \sigma \in \Theta(\xi(s, x_0))$$

juntas implicam em

$$4)\ \xi(s\sigma, x_0)!,\ isto\ é,\ \xi(s\sigma, x_0)\ é\ definido.$$

Pela Definição 3.7, se s é uma palavra que pode ocorrer no sistema supervisionado e o evento σ é uma continuação fisicamente possível da palavra s e está habilitado, então a palavra $s\sigma$ deve estar definida na função de transição do supervisor. Assume-se que todos os supervisores mencionados neste trabalho, salvo observação em contrário, são completos.

Duas restrições adicionais devem ser satisfeitas pelas linguagens $L(\mathbf{S}/G)$, $L_c(\mathbf{S}/G)$ e $L_m(\mathbf{S}/G)$. As mesmas são assim definidas:

Definição 3.8 *Um supervisor* $\boldsymbol{S}$ *é dito* não bloqueável *se e somente se*

$$\overline{L_c(\mathbf{S}/G)} = L(\mathbf{S}/G)$$

Definição 3.9 *Um supervisor* $\boldsymbol{S}$ *é dito* não rejeitável *se e somente se*

$$\overline{L_m(\mathbf{S}/G)} = \overline{L_c(\mathbf{S}/G)}$$

Se um supervisor é bloqueável, então existe pelo menos uma palavra fisicamente possível em $L(\mathbf{S}/G)$ que não é prefixo de qualquer palavra em $L_c(\mathbf{S}/G)$ e portanto o sistema pode nunca completar uma tarefa especificada. Por outro lado, se um supervisor é rejeitável, então existe pelo menos uma palavra em $\overline{L_c(\mathbf{S}/G)}$ que representa uma tarefa completada, mas que não pertence a $\overline{L_m(\mathbf{S}/G)}$. Isso significa que o sistema pode atingir um estado a partir do qual nenhuma tarefa completada é reconhecida como tal.

Assim, para a TCS, é sempre desejável construir supervisores não bloqueáveis e não rejeitáveis. Aliando-se estes conceitos ao de supervisor completo (Definição 3.7), tem-se a seguinte definição:

Definição 3.10 *Um supervisor completo, não bloqueável e não rejeitável, é dito ser um* supervisor próprio. *Em conseqüência, um supervisor próprio é um supervisor completo tal que*

$$\overline{L_m(\mathbf{S}/G)} = \overline{L_c(\mathbf{S}/G)} = L(\mathbf{S}/G).$$

3.2.1 Condições de Existência de Supervisores

Os conceitos de *fechamento* e *controlabilidade* de uma linguagem em relação a outra, que são de fundamental importância para o estabelecimento das condições de existência de supervisores, são apresentados a seguir.

O problema básico em controle supervisório é modificar o comportamento de um dado SED G, tal que o mesmo se comporte segundo uma dada especificação desejável.

Formalmente, considera-se o seguinte problema: dado um SED G com comportamento $L(G)$, que comportamento em malha fechada $K \subseteq L(G)$ pode ser encontrado por supervisão? A solução para este problema requer a definição dos conceitos *fechamento* e *controlabilidade.*

Definição 3.11 *Dadas duas linguagens $K, L \subseteq \Sigma^*$, K é dita fechada em relação a L, ou* L-fechada *se e somente se*

$$K = \overline{K} \cap L.$$

Definição 3.12 *Dadas duas linguagens arbitrárias L, $K \subseteq \Sigma^*$ e um alfabeto $\Sigma = \Sigma_c \cup \Sigma_u$, diz-se que K é* L-controlável *se e somente se:*

$$\overline{K}\Sigma_u \cap L \subseteq \overline{K}$$

A linguagem $\overline{K}\Sigma_u \cap L$ consiste de todas as seqüências $s' = s\sigma$, tal que $s' \in L$, $s \in \overline{K}$ e $\sigma \in \Sigma_u$. Se L representa o comportamento fisicamente possível e $\overline{K}$ o comportamento desejável, então a seqüência $s\sigma$ é formada por uma seqüência desejável s concatenada com um evento não controlável σ, tal que $s\sigma$ é fisicamente possível.

Dadas as definições acima, resta determinar sob que condições existe um supervisor para a realização de uma tarefa especificada. Essa condição de existência é dada através das seguintes proposições e teoremas:

Proposição 3.1 *Seja $\mathbf{S}$ um supervisor completo para G_c. Então $L(\mathbf{S}/G)$ é prefixo-fechada e $L(G)$-controlável.*

A Proposição 3.1 pode ser assim demonstrada:

Seja $\mathbf{S} = (S, \Theta)$ um supervisor completo para o gerador $G_c = (G, \Gamma)$. O prefixo-fechamento de $L(\mathbf{S}/G)$ decorre de sua definição, ou seja, $L(\mathbf{S}/G) = \overline{L(\mathbf{S}/G)}$.

Para mostrar que $L(\mathbf{S}/G)$ é $L(G)$-controlável, seja $s\sigma$ uma palavra tal que:

$s\sigma \in L(\mathbf{S}/G)\Sigma_u \cap L(G)$, isto é,

$s\sigma : s \in L(\mathbf{S}/G) \wedge \sigma \in \Sigma_u \wedge s\sigma \in L(G)$. Por ser σ não controlável,

$s\sigma : s \in L(\mathbf{S}/G) \wedge \sigma \in \Theta(\xi(s, x_0)) \wedge s\sigma \in L(G)$ e sendo $\mathbf{S}$ completo,
$s\sigma \in L(\mathbf{S}/G)$, para toda palavra $s\sigma$, ou seja,
$L(\mathbf{S}/G)\Sigma_u \cap L(G) \subseteq L(\mathbf{S}/G)$.

Os dois próximos teoremas estabelecem as condições de existência de supervisores para problemas formulados em termos de linguagens geradas e linguagens marcadas, respectivamente.

Teorema 3.1 *Dados um gerador G tal que $L(G)$ represente seu comportamento fisicamente possível e uma linguagem especificada $K \subseteq L(G)$, existe um supervisor completo $\boldsymbol{S}$ tal que $L(\mathbf{S}/G) = K$ se e somente se K for prefixo-fechada e $L(G)$-controlável.*

Teorema 3.2 *Dados um gerador G tal que $L_m(G)$ represente as tarefas que podem ser completadas pelo sistema na ausência de qualquer ação de controle e uma linguagem especificada $K \subseteq L_m(G)$ então*

1. *Existe um supervisor não bloqueável $\boldsymbol{S}$ tal que $L_c(\mathbf{S}/G) = K$ se e somente se K for $L_m(G)$-fechada e $L(G)$-controlável.*
2. *O supervisor $\boldsymbol{S}$ será próprio se e somente se o gerador $S = (\Sigma, X, \xi, x_0, X_m)$ for tal que $X_m = X$.*

As demonstrações para estes teoremas podem ser encontradas em [RW87c, Zil93].

Para problemas formulados em termos de linguagens geradas, o Teorema 3.1 estabelece as condições necessárias e suficientes para a existência do supervisor $\mathbf{S}$, o gerador S e o mapa Θ podem ser encontrados a partir das Equações

$$L(S) = K \text{ e} \tag{3.7}$$

$$\Theta(x) = \gamma : \gamma \in \Gamma \wedge \gamma \supseteq \Sigma_x^1 \wedge \gamma \cap \Sigma_x^0 = \emptyset \tag{3.8}$$

onde Σ_x^0 é o conjunto de eventos σ que, por serem fisicamente possíveis após a ocorrência de $s \in K$ e $s\sigma \notin K$, precisam ser desabilitados quando S se encontrar no estado x, de modo que não pertencem a $\Theta(x)$. Σ_x^1 é o conjunto de eventos σ que são continuações da palavra s tais que $s\sigma \in K$ e precisam estar habilitados quando S se encontrar no estado x, ou seja $\sigma \in \Theta(x)$.

Para problemas formulados em termos de linguagens marcadas, se K satifaz às condições do Teorema 3.2, então o supervisor é tal que $L(\mathbf{S}/G) = \overline{K}$, visto que nesse caso, $L_c(\mathbf{S}/G) = \overline{K} \cap L_m(G) = K$.

Os resultados acima só podem ser empregados quando a linguagem especificada K satisfizer as condições exigidas. Quando, em algumas situações, a linguagem especificada não satisfaz às condições estabelecidas para a existência do supervisor, ou seja, a linguagem especificada K não é nem $L_m(G)$-fechada nem $L(G)$-controlável, é sempre possível encontrar uma sublinguagem $K^{\uparrow} \subseteq K$ que satisfaz às referidas condições de maneira minimamente restritiva. Esta linguagem é denominada *suprema linguagem controlável* $K^{\uparrow}$ ou sup $C(L)$[RW87b].

Desta forma, se $K^{\uparrow}$ solucionar de forma satisfatória o problema, isto é, se $K \supseteq A$, onde A é uma linguagem que representa o comportamento mais restrito que pode ser tolerado, $K^{\uparrow}$ pode ser utilizada em substituição à linguagem anteriormente especificada. A solução para este problema é então um supervisor que implementa $K^{\uparrow}$. Para o caso dos geradores de estado finitos, $K^{\uparrow}$ é sempre computável [RW87b].

Seja $K \subseteq \Sigma^*$ uma dada linguagem, onde Σ^* representa o conjunto de todas as linguagens definidas a partir do alfabeto Σ. Seja $C(K)$ a família das linguagens controláveis de K. $C(K)$ é sempre não vazia pois a linguagem vazia é controlável.

Um importante resultado em relação à controlabilidade das linguagens é que a família $C(K)$ é fechada em relação à união de linguagens, ou seja, existe uma única linguagem controlável máxima $K^{\uparrow}$ tal que $K^{\uparrow} \subseteq K$. Note que $K^{\uparrow}$ pode ser a linguagem vazia.

Para o caso em que um gerador de estado finito é descrito por L (comportamento em malha aberta) e K (comportamento desejado), existe um algoritmo para a computação de $K^{\uparrow}$ descrito em [RW87b].

Pode-se então enunciar o seguinte problema de síntese de supervisores:

Dados um gerador G e uma linguagem-alvo marcada $E \subseteq L_m(G)$ e uma linguagem mínima admissível $A \subseteq E$, encontrar um supervisor próprio $\mathbf{S}$ *tal que*

$$A \subseteq L_c(S/G) \subseteq E$$

Quando E é $L_m(G)$-*fechada* e $L(G)$-*controlável*, existe um supervisor tal que $L_c(S/G) = E$, o que significa que o problema tem uma solução não restritiva. Para os casos em que E

não satisfaz estas condições, é possível uma solução minimamente restritiva, como veremos a seguir.

Considere que o conjunto de todas as sublinguagens $L(G)$-*controláveis* de E, é dado por:

$$C(E) = K' : K' \subseteq E \wedge K'\Sigma_u \cap L(G) = K' \tag{3.9}$$

e que o conjunto de todas as sublinguagens $L_m(G)$-*fechadas* de E, é dado por:

$$F(E) = K' : K' \subseteq E \wedge \overline{K'} \cap L_m(G) = K'. \tag{3.10}$$

Definição 3.13 *Dados um conjunto W qualquer e uma operação Z sobre seus elementos, W é dito* fechado sob a operação Z *se e somente se para todo par (w_1, w_2)*

$$w_1, w_2 \in W \Rightarrow w_1 Z w_2 \in W$$

Teorema 3.3 *Seja $H(K) = \{K' : K' \subseteq K\}$ um conjunto não vazio de subconjuntos de K, fechado sob a operação de união. Então existe em $H(K)$ o elemento supremo*

$$\sup H(K) = \bigcup \{K : K \in H(K)\}.$$

Para demonstrar este teorema, é necessário mostrar que $\sup H(K)$ contém qualquer elemento de $H(K)$ e que $\sup H(K) \in H(K)$. O primeiro fato decorre diretamente da definição de união, enquanto o segundo decorre da hipótese de ser $H(K)$ fechado em relação a esta operação. Além dissso, $\sup H(K)$ é sempre definido, pois $H(K)$ é, por hipótese, não vazio.

Considerando que dadas duas linguagens quaisquer $L, M \subseteq \Sigma^*$, as operações de *união* de conjuntos e de *prefixo-fechamento* comutam, isto é, $\overline{L \cup M} = \overline{L} \cup \overline{M}$, e que este raciocínio pode ser estendido a um número arbitrário de linguagens sobre Σ^*, então, a seguinte proposição pode ser apresentada:

Proposição 3.2 *$C(E)$ e $F(E)$ são não vazios e fechados sob a operação de união.*

Dem.: Dado que a linguagem vazia $\emptyset$ é *$L(G)$-controlável* e *$L_m(G)$-fechada*, pode-se afirmar que $\emptyset \in C(E)$ e $\emptyset \in F(E)$, de modo que estes são conjuntos não vazios.

Para verificar o fechamento de $C(E)$ sob união, considere um conjunto de índices $\mathbf{A}$ para enumerar os elementos deste conjunto, de modo que

$$K_\alpha \in C(E), \ \alpha \in \mathbf{A}.$$

Sendo os K_α controláveis:

$$K_\alpha \Sigma_u \cap L(G) \subseteq K_\alpha, \ \alpha \in \mathbf{A},$$

então:

$$\begin{aligned}
\overline{\left(\bigcup_\alpha K_\alpha\right)}\Sigma_u \bigcap L(G) &= \left(\bigcup_\alpha \overline{K_\alpha}\right)\Sigma_u \bigcap L(G) \\
&= \bigcup_\alpha \overline{K_\alpha}\Sigma_u \bigcap L(G) \\
&= \bigcup_\alpha \left[\overline{K_\alpha}\Sigma_u \bigcap L(G)\right] \\
&\subseteq \bigcup_\alpha \overline{K_\alpha}.
\end{aligned}$$

Logo, a união de quaisquer elementos de $C(E)$ é *L(G)-controlável* e pertence a $C(E)$.

Para verificar o fechamento de $F(E)$ sob união, considere um conjunto de índices $\mathbf{B}$ para enumerar os elementos desse conjunto, de modo que

$$K_\beta \in F(E), \ \beta \in \mathbf{B}.$$

Sendo os K_β $L_m(G)$-*fechados*, tem-se:

$$K_\beta = \overline{K_\beta} \cap L_m(G),$$

então:

$$\overline{\left(\bigcup_\beta K_\beta\right)} \bigcap L_m(G) = \left(\bigcup_\beta \overline{K_\beta}\right) \bigcap L_m(G) = \bigcup_\beta \left[\overline{K_\beta} \bigcap L_m(G)\right] = \bigcup_\beta K_\beta.$$

Logo, a união de quisquer elementos de $F(E)$ é $L_m(G)$-*fechada* e pertence a $F(E)$.♦

Corolário 3.1 *Existem em $C(E)$ e $F(E)$, respectivamente, os elementos supremos*

$$\sup C(E) = \bigcup\{K : K \in C(E)\} \ e$$
$$\sup F(E) = \bigcup\{K : K \in F(E).$$

Dem.: Demonstrado pelo Teorema 3.3. ♦

Os elementos $\sup C(E)$ e $\sup F(E)$ são denominados respectivamente de *Máxima Sublinguagem L(G)-controlável* e *Máxima Sublinguagem $L_m(G)$-fechada de E.*

Vale também a seguinte proposição:

Proposição 3.3 *O conjunto $C(E) \cap F(E)$ é não vazio e fechado sob a operação de união.*

Dem.: $C(E) \cap F(E)$ é não vazio, pois $\emptyset \in C(E) \cap F(E)$. Para o fechamento sob união, considere duas linguagens quaisquer $K_1, K_2 \in C(E) \cap F(E)$. Então

$$K_1 \in C(E) \wedge K_1 \in F(E) \wedge K_2 \in C(E) \wedge K_2 \in F(E).$$

Sendo $C(E)$ e $F(E)$ fechados sob união, tem-se

$K_1 \cup K_2 \in C(E) \wedge K_1 \cup K_2 \in F(E)$, assim

$K_1 \cup K_2 \in C(E) \cap F(E)$♦

Corolário 3.2 *Existe em $CF(E) = C(E) \cap F(E)$ o elemento supremo*

$$\sup CF(E) = \bigcup \{K : K \in C(E) \cap F(E)\}$$

Dem.: Pelo Teorema 3.3. ♦

O elemento $\sup CF(E)$ é denominado de *Máxima Sublinguagem L(G)-controlável e $L_m(G)$-fechada de E.*

Pelo Corolário 3.2, o teorema seguinte é demonstrado.

Teorema 3.4 *O problema de controle supervisório tem solução se e somente se* $\sup CF(E) \supseteq A$, *sendo* $\sup CF(E)$ *a solução minimamente restritiva.*

Considerando o caso particular em que todos os estados do gerador são marcados, tem-se:

$$L_m(G) = \overline{L_m(G)} = L(G)$$

e, conseqüentemente, pela Equação 3.1 e pela definição 3.5

$$L_c(S/G) = L(S/G).$$

Além disso, toda linguagem prefixo-fechada $E \subseteq L_m(G)$ é $L_m(G)$-$fechada$, assim:

$$\forall E : E \subseteq L_m(G), \quad E = \sup F(E) \text{ e}$$
$$\sup CF(E) = \sup C(E).$$

Com isso, o problema de controle supervisório pode ser assim explicitado:

Dados um gerador G tal que $L_m(G) = L(G)$, uma linguagem alvo prefixo-fechada $E \subseteq L(G)$ e uma mínima linguagem admissível $A \subseteq E$, encontrar um supervisor próprio **S** tal que

$$A \subseteq L(S, G) \subseteq E$$

A solução para este problema é conseqüência imediata do Teorema 3.2 dado que $L(S/G) \subseteq K$.

Teorema 3.5 *O problema de controle supervisório para linguagens geradas tem solução se e somente se* $\sup C(E) \supseteq A$, *sendo* $\sup C(E)$ *a solução minimamente restritiva.*

Vale ressaltar que caso todos os estados do gerador sejam marcados, significa que qualquer supervisor obtido será *não-bloqueável.*

Capítulo 4

Uma Nova Abordagem para a Síntese de Supervisores

4.1 Introdução

Como visto no Capítulo 3, dado o modelo de um sistema a eventos discretos (SED), representado pelo gerador G, a teoria de controle supervisório (TCS) estabelece as condições necessárias para a existência de um supervisor para o SED, baseado em um comportamento desejável, representado pela linguagem K.

Vimos também que o gerador G e o supervisor $\mathbf{S}$ podem ser modelados tanto por autômatos quanto por redes de Petri (RPs), dentre outros formalismos. Nos dois tipos de modelagem citados, o sistema em malha aberta (sistema não controlado) é representado pelo gerador G, enquanto que o sistema em malha fechada (sistema controlado) é representado pela composição do supervisor com o gerador, denotado por $\mathbf{S}/G$.

Um diagrama simplificado da utilização da teoria de controle supervisório e das redes de Petri em conjunto, para o projeto, análise e controle de um SED, é apresentado na Figura 4.1. Este diagrama deriva daquele apresentado na Figura 1.4 (apresentada no Capítulo 1). Neste, foi introduzida a etapa de síntese do supervisor e as diferenças serão explicitadas a seguir.

Note que na abordagem por redes de Petri, utilizada por Zhou e DiCesare [ZD93], Figura 1.4, o modelo do sistema, para uma dada especificação, é utilizado como super-

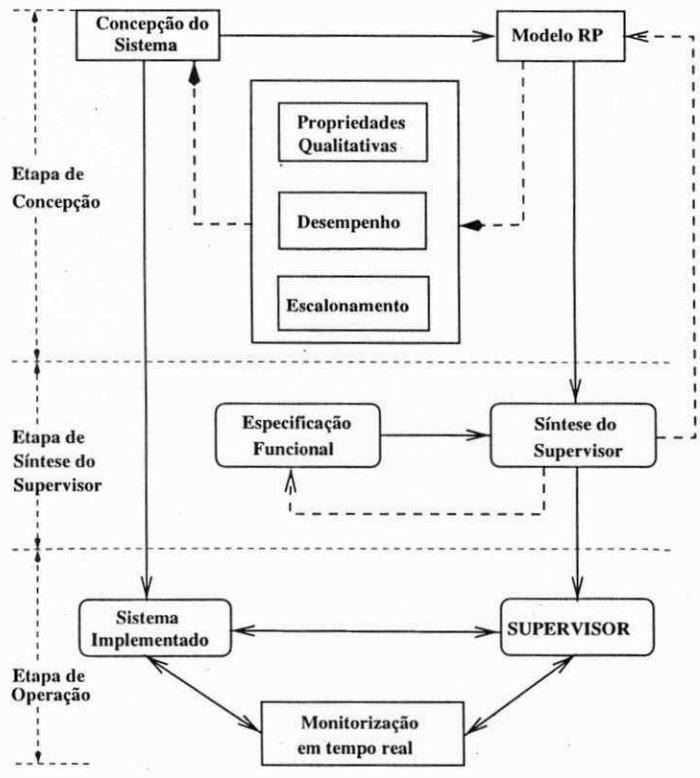


Figura 4.1: Utilização de redes de Petri e teoria de controle supervisório para a concepção, análise e controle de um sistema a eventos discretos.

visor em tempo real, ou seja, as ações de controle são embutidas no próprio modelo do sistema durante a fase de projeto. Vale lembrar que utilizando esta abordagem, qualquer modificação na ação de controle acarreta uma modificação no modelo do sistema.

Uma outra abordagem seria a concepção de um modelo de redes de Petri para um sistema não controlado. A partir do modelo do sistema sintetiza-se o controlador baseado em uma ou mais especificações de tarefas a serem realizadas pelo sistema sob supervisão. Veja diagrama representativo na Figura 4.1. Essa abordagem é mais flexível, visto que dado o modelo de redes de Petri do sistema, várias tarefas podem ser especificadas em diferentes tempos, provocando mudanças somente no supervisor. Desta forma, modelos mais simples podem, muitas vezes, ser construídos, visto que as ações de controle não mais estarão embutidas no modelo do sistema e tão somente no supervisor.

Utilizando uma abordagem por autômatos para a síntese do supervisor, como a utilizada pela teoria de controle supervisório, desenvolvida por Ramadge e Wonham [RW87b] e depois estendida por eles e outros autores, é possível se examinar conceitos da teoria de controle tais como controlabilidade, observabilidade, agregação e controle modular, distribuído ou descentralizado e hierárquico. A teoria de controle supervisório tem provado ser útil na análise teórica de problemas básicos de controle supervisório. Como citado no Capítulo 1, a maior vantagem deste modelo é o de separar explicitamente o conceito de sistema em malha aberta (planta) do conceito de controle em malha fechada e assim, permitir a formulação e solução de uma variedade de problemas de síntese de controle. Infelizmente essa abordagem, como todas as outras, não suporta todas as características que se poderia desejar de uma teoria para SEDs, mas apesar disso, a TCS tem contribuído profundamente para o entendimento dos problemas fundamentais envolvidos na análise e controle dos sistemas a eventos discretos.

Por outro lado, as redes de Petri têm sido eficientemente utilizadas juntamente com a teoria de controle supervisório, com amplas vantagens (já citadas no Capítulo 1) sobre os autômatos, dentre elas, as redes de Petri permitem uma descrição mais sucinta do sistema bem como uma síntese modular de sistemas compostos de subsistemas que interagem entre si. Algumas classes de redes de Petri têm sido utilizadas, tais como redes de Petri não limitadas, redes de Petri controladas, grafos marcados e máquinas de estado. Entretanto, a questão fundamental no estudo do controle de sistemas a eventos discretos automatizados

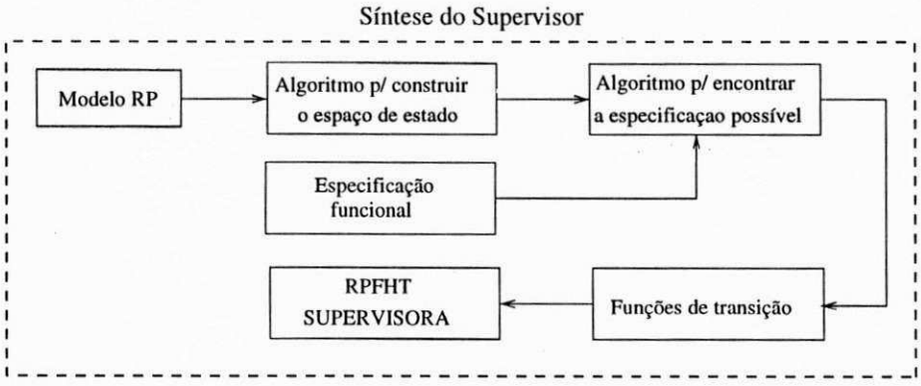


Figura 4.2: Diagrama em blocos do procedimento de síntese do supervisor.

são os problemas que aparecem na complexidade da computação envolvendo a solução de problemas de síntese de controladores. Embora tenha-se mostrado que estes problemas possuem solução de complexidade polinomial com o número de estados de um sistema, em um sistema real o número de estados pode crescer exponencialmente com o número de constituintes do mesmo. Muitas vezes este problema pode ser gerenciado através da síntese modular e em certos casos restringindo a atenção a sistemas com estruturas especiais, como por exemplo, restringindo a solução do problema de síntese do supervisor para alguns sistemas modelados por certas classes de redes de Petri.

Tendo como base os problemas e avanços na teoria de controle de sistemas a eventos discretos, citados anteriormente, esta Tese apresenta uma metodologia de síntese de supervisores, baseada em controle supervisório e redes de Petri, para sistemas a eventos discretos. A síntese do supervisor é realizada tendo como dados o modelo do sistema e seu comportamento desejável, como mostrado pelo diagrama de blocos da Figura 4.2.

O objetivo é introduzir uma sistemática de síntese do supervisor sem modificações *ad-hoc* na estrutura do modelo do sistema, ou seja, dado que um SED é modelado por uma rede de Petri, então a rede de Petri supervisora (que será definida neste Capítulo e denominada *RPFHT* supervisora) deve possuir a mesma estrutura do modelo. Desta forma, objetiva-se simplificar a construção da rede supervisora, dado que sua estrutura é previamente definida pela estrutura da rede que modela o sistema. Com isso, evita-se a complexidade da composição de redes, exclusão, ou inclusão de arcos, transições ou lugares na síntese do supervisor. Ainda mais, mantendo a estrutura das redes iguais, quando o

sistema supervisionado estiver em operação, os estados em que o supervisor e o sistema se encontram são sempre equivalentes, pois os mesmos são executados sincronamente.

Baseados nos sistemas de *G-Nets*, introduzimos uma sistemática modular de síntese do supervisor, visando a utilização da metodologia de síntese para sistemas reais mais complexos. Utilizando esta abordagem, a contrução do supervisor é facilitada pois o modelo do sistema a ser controlado é dividido em subsistemas, ou módulos, independentes e fracamente acoplados. Desta forma, o supervisor, que possui a mesma estrutura do modelo, é formado por controladores independentes para cada módulo do sistema. Esses controladores podem então ser analizados ou modificados individualmente sem a necessidade de modificar os outros controladores.

Utilizando esta sistemática de síntese, é possível sintetizar supervisores com propriedades de segurança e tolerância a falhas não dependentes do tempo. Propriedades de tolerância a falhas dependentes do tempo são também introduzidas no supervisor pela utilização de redes de Petri com temporização nebulosa. Esta abordagem é apresentada no Capítulo 5.

Nas Seções seguintes apresentamos o trabalho desenvolvido, destacando os principais resultados. Na Seção 4.2 é apresentada a metodologia de síntese de supervisores para sistemas a eventos discretos, que envolve o desenvolvimento de dois algoritmos de síntese e o desenvolvimento de uma extensão à classe de redes de Petri com funções de habilitação de transições, bem como sua extensão temporizada denominada redes de Petri com temporização nebulosa e funções de habilitação de transições. Por fim, para ilustrar a metodologia, são apresentados os exemplos de um sistema produtor/consumidor e de um sistema de manufatura simples com recursos compartilhados. A Seção 4.3 apresenta o desenvolvimento de uma metodologia de síntese modular utilizando os sistemas de *G-Nets* e um exemplo de uma célula de manufatura cujo supervisor é sintetizado utilizando esta abordagem.

4.2 Metodologia de Síntese de Supervisores

Como apresentado no Capítulo 3, o conceito de *controlabilidade* é condição necessária e suficiente para a existência de um supervisor de um sistema a eventos discretos (SED).

Pela definição de controlabilidade, um gerador controlado G_c sempre gera uma linguagem que é um prefixo da linguagem do gerador não controlado G. Sendo assim, se G e G_c são modelados por redes de Petri, o conjunto de todas as seqüências de transições disparáveis de G_c será sempre um subconjunto do conjunto das seqüências de transições disparáveis de G. Assim, uma rede de Petri supervisora que possua a mesma estrutura da rede que modela o sistema, poderá ser executada sincronamente com o sistema e garantir pelo menos que todos os eventos do sistema possam ocorrer, caso não haja restrições ao comportamento dinâmico do mesmo. Essas restrições, se existirem, serão explicitadas através de restrições impostas ao disparo das transições da rede supervisora. Os disparos das transições do gerador de rede de Petri G (rede que modela o sistema) modelam os eventos que podem ocorrer no sistema. Os disparos das transições da rede supervisora modelam os eventos que poderão ocorrer no sistema sob supervisão.

Em nossa abordagem de síntese de supervisores utilizamos uma classe de redes de Petri denominadas *Redes de Petri com Funções de Habilitação de Transições (RPFHT*s). Estas redes são utilizadas como supervisoras de sistemas a eventos discretos controlados.

As *RPFHT*s possuem a mesma estrutura das redes de Petri, mas se distinguem pelas funções de marcação que podem ser associadas às suas transições. Essas funções modificam a regra de disparo das transições.

Para definir que restrições serão impostas à ocorrência de eventos no sistema, para a realização de uma tarefa especificada, dois algoritmos foram desenvolvidos: *Algoritmo Modificado da Árvore de Alcançabilidade* (*AMArA*) e *Algoritmo para a Construção do Gerador da Suprema Linguagem Controlável* (*ACGS*). A Figura 4.2 apresenta os passos a serem seguidos para a síntese de supervisores utilizando a abordagem proposta, ou seja, dado o modelo de rede de Petri do sistema, encontra-se o espaço de estados fisicamente possível do mesmo executando-se o *AMArA*. De posse do espaço de estados e da especificação desejável, executa-se o *ACGS* para encontrar a especificação possível e a seguir as funções que restringirão a ocorrência de eventos (representada pelo disparo das transições na *RPFHT*) para que a especificação possível seja executada pelo sistema sob supervisão.

4.2.1 Redes de Petri com Funções de Habilitação de Transições

Definição 4.1 *Uma rede de Petri com funções de habilitação de transições é uma quíntupla*

$$RPFHT = (N, K, l, M_0, \Phi)$$

- $N = (P, T, F, W)$ *é uma estrutura de rede de Petri;*
- $K : P \rightarrow \mathbb{N} \cup \{\infty\}$ *é a função de capacidade;*
- $l : T \rightarrow \Sigma$, *é a função que etiqueta as transições;*
- M_0 *é a marcação inicial, como definido para as redes de Petri;*
- $\Phi = \{\varphi_1, ..., \varphi_m\} : R(N, M_0) \rightarrow \{0, 1\}$ *é a função de habilitação das transições, que mapeia o conjunto de marcações alcançáveis em 0 ou 1.*

Devido à introdução das funções de habilitação das transições, uma transição t_j com função de habilitação φ_j estará desabilitada em uma marcação $M_i \in R(N, M_0)$ se $\varphi_j(M_i) = 0$. De outra forma, se $\varphi_j(M_i) = 1$, então t_j estará habilitada sujeita às condições impostas pela regra de disparo das transições como em uma rede de Petri.

Definição 4.2 *O estado, ou marcação, de uma rede de Petri varia de acordo com a seguinte* regra de disparo das transições*:*

1. *Uma transição t_j é dita habilitada (para disparar) em M se e somente se*

$$\forall p \in P \text{ que é entrada de } t_j : W(p, t_j) \leq M(p)$$
$$\forall p \in P \text{ que é saída de } t_j : M(p) \leq K(p) - W(t_j, p);$$
$$\varphi_j = 1.$$

2. *Uma transição habilitada pode ou não disparar;*
3. *O disparo de uma transição $t_j \in T$, habilitada na marcação M, é instantâneo e resulta em uma nova marcação M' dada pela equação:*

$$M'(p) = M(p) - W(p, t_j) + W(t_j, p), \forall p \in P;$$

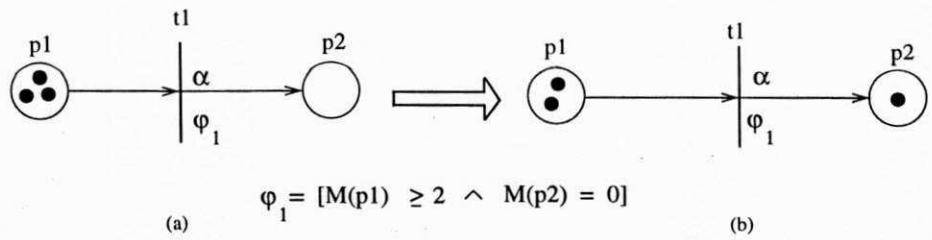


Figura 4.3: Exemplo da regra de disparo de uma transição em uma *RPFHT*.

4. *A ocorrência do disparo de t_j, que modifica a marcação M da rede para uma nova marcação M', é denotada por $M[t\rangle M'$, ou (em analogia à* função de próximo estado *δ dos autômatos) $M' = \delta(M, t_j)$.*

Considere a rede de Petri apresentada na Figura 4.3 e sejam $\varphi_1 = [(M(p_1) \geq 2 \wedge M(p_2) = 0]$ a função de habilitação associada à transição t_1 e α o evento que etiqueta t_1. Pela regra de disparo das transições em uma *RPFHT*, t_1 só poderá ser disparada se o número de fichas em p_1 for pelo menos igual a dois e se o número de fichas em p_2 for igual a zero.

A Figura 4.3(a) mostra que t_1 está habilitada e pode ser disparada, ou seja, o evento α pode ocorrer. O disparo de t_1 leva a rede à nova marcação, apresentada na Figura 4.3(b). Nessa nova marcação, t_1 não está habilitada pois o número de fichas em p_2 é maior que zero ($M(p_2) > 0$) e nesse caso $\varphi_1 = 0$.

Para ilustrar a utilização de uma *RPFHT* como rede supervisora, veja o exemplo de um sistema de manufatura simples modelado por uma rede de Petri, apresentada na Figura 4.4.

O sistema da Figura 4.4 consiste de duas estações de processamento com máquinas m_1 e m_2 respectivamente, um robô compartilhado r para descarga e um *buffer* b para armazenamento temporário de peças intermediárias. Cada peça é processada primeiro em m_1 e depois em m_2. As peças entram no sistema e na estação de processamento são automaticamente fixadas a uma bandeja e carregadas na máquina m_1. Após o processamento, o robô r descarrega de m_1 a peça intermediária e a coloca no *buffer* b. Logo a seguir, as peças intermediárias são automaticamente carregadas em m_2 e processadas. Quando m_2 encerra o processamento de uma peça, r descarrega o produto final e libera a bandeja para a primeira estação de trabalho. Assume-se que peças de entrada estão

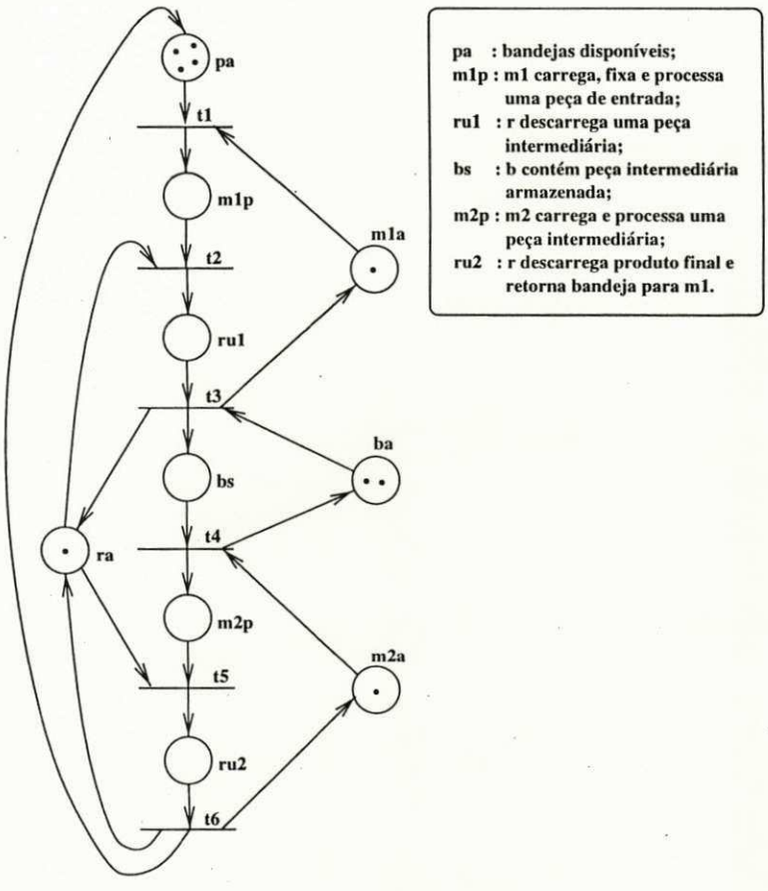


Figura 4.4: Sistema de manufatura com recursos compartilhados.

sempre disponíveis para serem processadas e que o produto final é sempre removido.

A marcação inicial determina como o sistema é inicializado. Nesse exemplo, existem 4 bandejas disponíveis representadas por quatro fichas em pa ($M_0(pa) = 4$), bem como as máquinas m_1 e m_2 e o robô estão disponíveis ($M_0(m_1a) = M_0(m_2a) = M_0(ra) = 1$). Existem também disponíveis no *buffer* espaço para duas peças intermediárias ($M_0(ba) = 2$). Note que, para esta marcação ou estado inicial, somente a transição t_1 está habilitada.

Observe que se a seqüência de transições $t_1t_2t_3t_4t_1t_2t_3t_1t_2t_3t_1t_2$ for disparada a partir da marcação inicial, o sistema alcançará um estado em que $M(ru_1) = 1$, $M(bs) = 2$, $M(m_2p) = 1$ e todos os outros lugares da rede não possuem fichas. Neste estado o sistema estará bloqueado, ou seja, nenhuma transição estará habilitada nesse estado ou marcação.

Para evitar esta situação, três alternativas podem ser estudadas. A primeira seria a modificação da estrutura da rede, ou seja, embutir o controle de prevenção ao bloqueio na própria estrutura da rede. Veja Figura 4.5. Neste caso um lugar de controle lc é acrescentado à estrutura da rede e o mesmo é inicializado com três fichas ($M_0(lc) = 3$). Desta forma evita-se que a seqüência que leva o sistema ao bloqueio seja disparada, pois t_2, que nesta seqüência é disparada quatro vezes, na nova rede com o lugar lc só poderá ser disparada três vezes. Após o disparo de t_5, t_2 poderá ser disparada novamente.

A segunda alternativa seria restringir a marcação inicial da rede da Figura 4.4. Nesse caso, se inicialmente o número de fichas no lugar pa for, por exemplo igual a 3 ($M_0(pa) = 3$), a seqüência de transições acima mencionada não pode ser disparada e o bloqueio é evitado.

Note que a primeira alternativa aumenta o tamanho da estrutura da rede pois são acrescentados, nesse caso, um lugar e dois arcos. Na segunda alternativa, restringe-se a marcação inicial a um conjunto de marcações que evitem o bloqueio na rede quando da evolução do sistema.

Uma terceira alternativa seria a utilização de um supervisor para o sistema capaz de prevenir o bloqueio. Utilizando-se uma *RPFHT* com a mesma estrutura da rede que modela o sistema original, evita-se o bloqueio impondo-se uma restrição ao disparo da transição t_2 através da função de habilitação $\tilde{\varphi}_2 = [M(bs) \leq 1 \vee M(m_2p) = 0]$. Ou seja, t_2 só poderá disparar se o número de fichas em bs for menor ou igual a 1 ou se o número

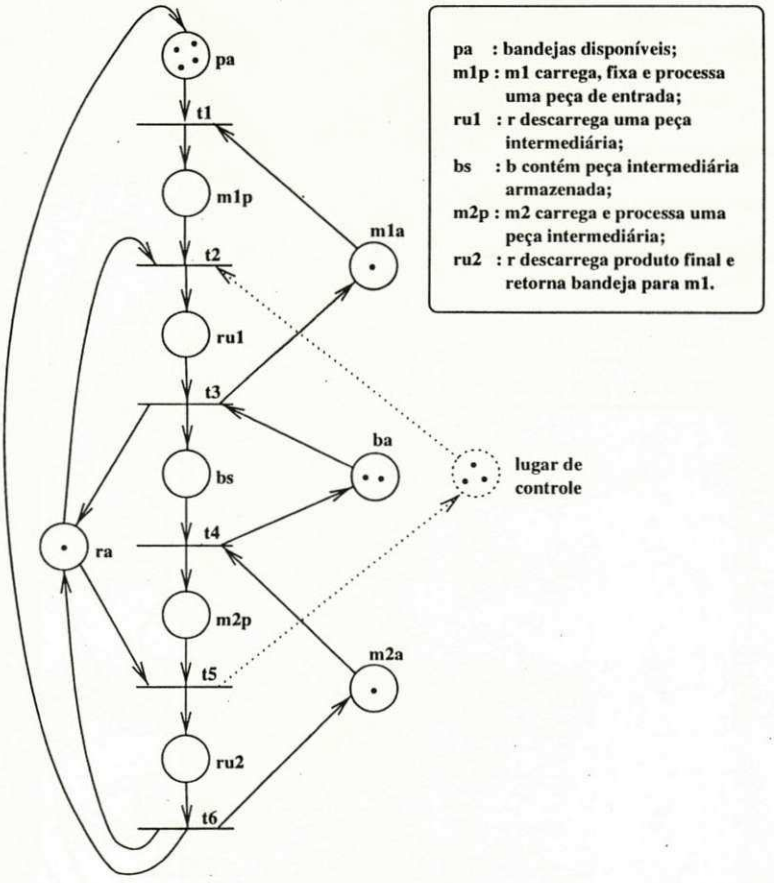


Figura 4.5: Sistema de manufatura com estrutura de controle embutida.

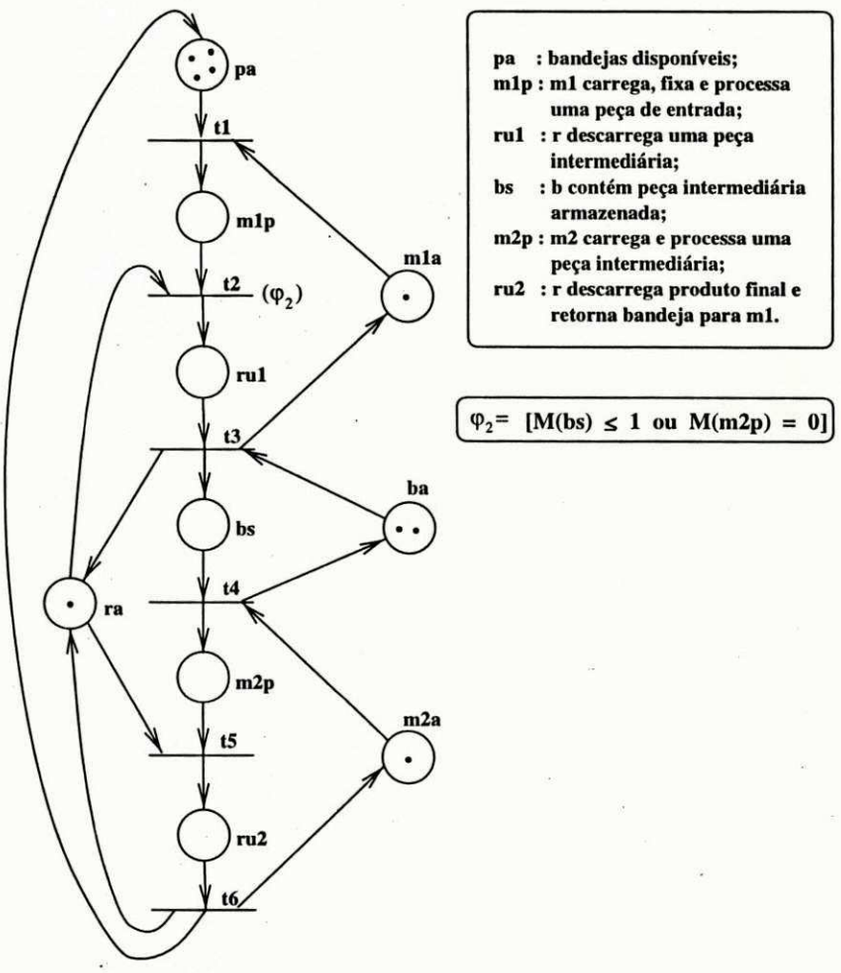


Figura 4.6: *RPFHT* supervisora para o sistema de manufatura.

de fichas em m_2p for igual a zero. A *RPFHT* supervisora é apresentada na Figura 4.6.

Nesse caso, não há restrições à marcação inicial da rede que modela o sistema e a estrutura da rede supervisora é idêntica à sua. Assim, o sistema modelado pela rede de Petri da Figura 4.4 e o supervisor modelado pela *RPFHT* da Figura 4.6 podem ser executados sincronamente, de tal forma que os mesmos estarão sempre em estados equivalentes. Desta forma, o supervisor poderá sempre desabilitar a ocorrência de eventos indesejáveis no sistema pela restrição imposta por φ_2 à transição t_2.

4.2.2 Algoritmos de Síntese

Os principais métodos de análise de redes de Petri são: simulação, método dos invariantes, redução e método da árvore de cobertura.

A maneira mais direta de análise é a simulação, a mesma é muito útil para o entendimento e depuração de um sistema. Este aspecto é relevante durante a fase de concepção e validação de um sistema complexo, entendendo por sistema complexo um sistema com um grande número de estados. Entretanto, por meio de simulação não é possível obter uma prova completa das propriedades dinâmicas de um sistema. Portanto, é importante utilizar métodos formais (baseados em técnicas de prova matemática) para a análise das redes de Petri.

Utilizando-se técnicas formais, duas classes de propriedades das redes de Petri podem ser analizadas, são elas as propriedades estáticas e as propriedades dinâmicas. As propriedades estáticas, ou estruturais, não consideram as seqüências de disparo das transições e caracterizam propriedades especiais da estrutura da rede. Murata [Mur89] apresenta uma excelente introdução a estas propriedades.

As propriedades dinâmicas ou comportamentais caracterizam o comportamento dinâmico das redes, por exemplo, se é possível alcançar uma marcação bloqueada a partir de uma dada marcação inicial. Exemplos de propriedades comportamentais, discutidas em [Mur89], são alcançabilidade, vivacidade, limitação, reversibilidade, cobertura (*coverability*) e estados originais (*home states*).

O método dos invariantes tem a vantagem da análise poder ser realizada em subredes locais, ignorando o comportamento do sistema global. O método utiliza equações lineares para a análise do comportamento da rede, mas infelizmente a solução destas equações é

limitada aos inteiros não negativos.

Para simplificar a análise de redes mais complexas (grande número de lugares, transições e arcos), muitas vezes é necessário reduzir o modelo para um mais simples, garantindo a preservação das propriedades do modelo original. Infelizmente os métodos de redução não garantem que uma determinada propriedade que não se encontra na rede reduzida, não se encontre também na rede original.

Pelo método da árvore de cobertura de uma rede de Petri [Mur89], a partir da marcação inicial M_0, podem ser obtidas tantas marcações quantas forem as transições habilitadas. Assim, a partir de cada nova marcação podem ser obtidas outras novas marcações. Este processo resulta em uma árvore de marcações. Entretanto, se a rede de Petri é não limitada (número de marcações infinito), a árvore crescerá indefinidamente. De modo a manter a árvore finita, introduz-se o símbolo especial w, o qual pode ser considerado como *infinito*. Este símbolo apresenta as seguintes propriedades: para cada inteiro não negativo $n, w > n, w \pm n = w$ e $w \geq w$. A árvore de cobertura para uma rede de Petri com marcação inicial M_0, pode ser construída aplicando-se o algoritmo apresentado em [Mur89]. No caso de uma rede limitada (número de marcações finito), a árvore de cobertura é denominada árvore de alcançabilidade, uma vez que esta contém todas as possíveis marcações alcançáveis e, neste caso, todos os problemas de análise poderiam ser resolvidos pela análise da árvore, não fosse a desvantagem do método ser exaustivo.

Note que na construção da árvore de alcançabilidade de uma rede de Petri, todas as possíveis seqüências de transições da rede são disparadas, a partir de uma dada marcação inicial. Associando a ocorrência de um evento ao disparo de uma transição da rede (rede de Petri etiquetada), então é possível gerar a ocorrência de todas as seqüências de eventos possíveis.

Como o problema da síntese do supervisor pode ser solucionado pela análise da linguagem gerada pela rede que modela o sistema e como todo sistema a eventos discretos (SED) fisicamente realizável (sistema real) é um sistema com capacidade finita, introduzimos o *Algoritmo Modificado da Árvore de Alcançabilidade (AMArA)* que lista todos os estados (marcações) possíveis de serem alcançados pelo sistema juntamente com as seqüências de transições disparáveis (ocorrência de eventos) a partir de cada estado. Também serão listados os possíveis estados que levariam o sistema a uma falha, tais como um estado

com bloqueio, ou um estado em que a capacidade do sistema seja excedida.

Algoritmo 1 Algoritmo Modificado da Árvore de Alcançabilidade para uma rede com capacidade finita (AMArA):

Início

1. *Rotule a marcação inicial M_0 como raiz e etiquete-a como nova;*
2. *Enquanto existirem marcações nova faça:*
   - *(a) Selecione uma marcação nova M;*
   - *(b) Se M é idêntica a uma marcação já existente, etiquete-a como antiga;*
   - *(c) Se nenhuma transição está habilitada em M, etiquete M como bloqueada;*
   - *(d) Enquanto existirem transições habilitadas em M, faça o seguinte para cada transição habilitada:*
     - *i. Obtenha a marcação M' que resulta do disparo de t em M;*
     - *ii. Se a capacidade de algum lugar p é excedida na marcação M', então substitua $M'(p)$ por w;*
     - *iii. Introduza M' como um nó da árvore, ligue um arco, com rótulo t, de M para M' e etiquete M' como não-permitida se a capacidade de algum lugar foi excedida, de outra forma, etiquete-a como nova.*

Fim.

Assumindo que a árvore possui m estados, ou marcações, e que para cada novo estado encontrado, o mesmo deve ser comparado, no pior caso, a todos os outros estados encontrados anteriormente, o *AMArA* possui a seguinte complexidade[1]

$$O(n) = \frac{n(n-1)}{2}$$

Veja o exemplo de um sistema produtor/consumidor modelado pela rede de Petri apresentada na Figura 4.7(a). Executando o *AMArA* para um sistema com capacidade $K = 2$, obtém-se a árvore de alcançabilidade apresentada na Figura 4.7(b).

[1]Número de operações realizadas, neste caso, número de comparações entre estados encontrados

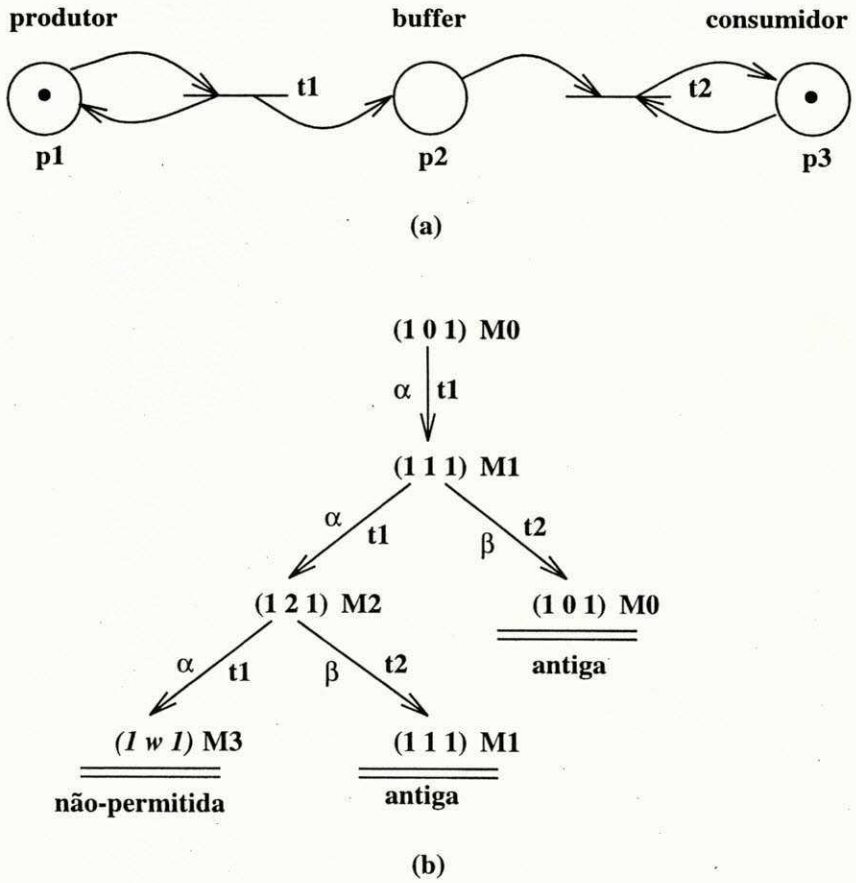


Figura 4.7: Modelo RP e árvore de alcançabilidade modificada de um sistema produtor/consumidor simples.

Note que a seqüência de eventos $\alpha\alpha\alpha$ leva o sistema a uma marcação *não permitida*, o que deve ser evitado através de uma ação de controle externa ao sistema.

O *Algoritmo para a Construção do Gerador da supC(L) (ACGS)* utiliza como dados de entrada a árvore de marcações construída pelo *AMArA*, bem como a especificação do comportamento desejável para o sistema. Ao final da execução do *ACGS*, obtém-se a lista de eventos a serem desabilidados em determinados estados, bem como os respectivos estados.

Baseado na lista de estados e respectivos eventos a serem desabilitados (transições que não podem ser disparadas naquelas marcações), derivam-se as funções de habilitação associadas às transições.

Algoritmo 2 Algoritmo para a Construção do Gerador da supC(L) (ACGS):
Início

1. *Criar uma lista dinâmica,* lista_bloc, *e incluir na mesma os estados ou marcações bloqueadas, incluindo as marcações do tipo não_permitida, onde:*

$$M : M[t_j\rangle\emptyset, \text{ é uma marcação bloqueada e}$$
$$M : M(p_i) = w \text{ é uma marcação não\_permitida;}$$

2. *Adicionar à* lista_bloc *os estados, não marcados, cuja única transição habilitada, se disparada, leva o sistema a um estado bloqueado, ou seja:*

$$M : M[t_j\rangle M',\ t_j \text{ é única transição habilitada em } M,\ M \notin Q_m \text{ e } M' \in \text{lista\_bloc;}$$

3. *Adicionar à lista os estados nos quais exista pelo menos uma transição habilitada, etiquetada por um evento não controlável, cujo disparo da transição leve o sistema para uma marcação na* lista_bloc, *ou seja*

$$\exists\alpha \in \Sigma_u,\ l(t_j) = \alpha \text{ e } M[t_j\rangle M',\ M' \in \text{lista\_bloc;}$$

4. *Criar uma lista, lista_perigo, com os estados antecessores dos elementos (estados) da lista_bloq, juntamente com o evento que os liga, desde que o antecessor não esteja*

na lista_bloq. Esses eventos deverão estar sempre desabilitados quando o sistema se encontrar nesses estados, ou seja:

$$\exists \beta \in \Sigma_c \mid l(t_j) = \beta \text{ e } M[t_j\rangle M',\ M \notin \textit{lista\_bloc} \text{ e } M' \in \textit{lista\_bloc};$$

5. *Dada a especificação desejada para o sistema, encontre a suprema linguagem controlável;*

6. *Adicionar à* lista_perigo *os estados e seus respectivos eventos de saída a serem desabilitados para que a linguagem especificada seja executada, desde que estes estados não estejam ainda na* lista_perigo, *ou seja:*

$$\exists \beta \in \Sigma_c \mid l(t_j) = \beta \text{ e } M[t_j\rangle M',\ M \notin \textit{lista\_perigo} \text{ e } M' \notin G(\sup C(L)).$$

Fim.

Algoritmo 3 *Passo 5 do* ACGS.

Dados o gerador trim G *e o gerador* H *(especificação), faça:*

1. *Adicione à* lista_bloc *os estados que não satisfazem*

$$\Sigma(H(x)) \cap \Sigma_u \subseteq \Sigma(x);$$

2. *Para cada estado* x_i, *na* lista_bloc, *adicione à* lista_bloc*:*

   (a) os estados

$$x_j : (\exists \sigma_u \in \Sigma_u) x_i = \xi(\sigma_u, x_j);$$

   (b) os estados

$$x_k : (\exists \sigma_c \in \Sigma_c) x_i = \xi(\sigma_c, x_k) \wedge x_k \notin X_m \wedge |\Sigma(x_k)| = 1;$$

3. *Encontre a componenete acessível do gerador resultante.*

- O passo um adiciona à *lista_bloc* todos os estados para os quais um evento não controlável, que é fisicamente possível, não é definido na especificação. O passo dois processa e incrementa a lista e o passo três remove qualquer estado inacessível deixado pelos passos anteriores.

- Note que a segunda parte do passo dois preserva a *coacessibilidade* e que, no final destas operações tem-se o gerador da suprema linguagem controlável para a especificação desejável.

Assumindo que os geradores para L (planta) e K (especificação) possuem m e n estados, respectivamente, então o $ACGS$ converge após, no pior caso, mn interações. Isto significa que a computação de $K^{\uparrow}$ possui complexidade polinomial em m e n.

Seja $\Sigma = \{\alpha, \beta\}$ o alfabeto de eventos associado à rede apresentada na Figura 4.7 e seja $\Sigma_c = \{\alpha\}$ e $\Sigma_u = \{\beta\}$. Para este exemplo, a seguinte especificação é desejável:

$$\forall M \in R(N, M_0), \forall p_i \in P, M(p_i) \leq 2,$$

ou seja, cada lugar da rede não pode possuir mais que duas fichas em um determinado instante.

Executando o $ACGS$, tem-se:

$$lista\_bloc : M_3 = [1, w, 1]^T;$$
$$lista\_perigo : M_2 = [1, 2, 1]^T, t_1(\alpha),$$

ou seja, toda vez que o sistema se encontrar na marcação M_2 o evento α deve ser desabilitado para que a marcação M_3 nunca seja alcançada.

Para exemplificar a metodologia de síntese proposta, através da utilização dos algoritmos introduzidos, apresentamos a seguir um outro exemplo simples de um sistema produtor/consumidor.

4.2.3 Síntese de um Supervisor para um Sistema Produtor/-Consumidor

Vejamos o exemplo de um sistema produtor/consumidor, modelado pela rede de Petri apresentada na Figura 4.8(a).

O exemplo apresenta um produtor (representado inicialmente por uma ficha no lugar p_1) que produz itens (disparo da transição t_1) e os coloca em um *buffer* (ficha no lugar p_5). Dado que existe pelo menos um item no *buffer*, o consumidor (representado inicialmente por uma ficha no lugar p_3) pode consumí-lo (disparo da transição t_3). Os eventos α_1, β_1,

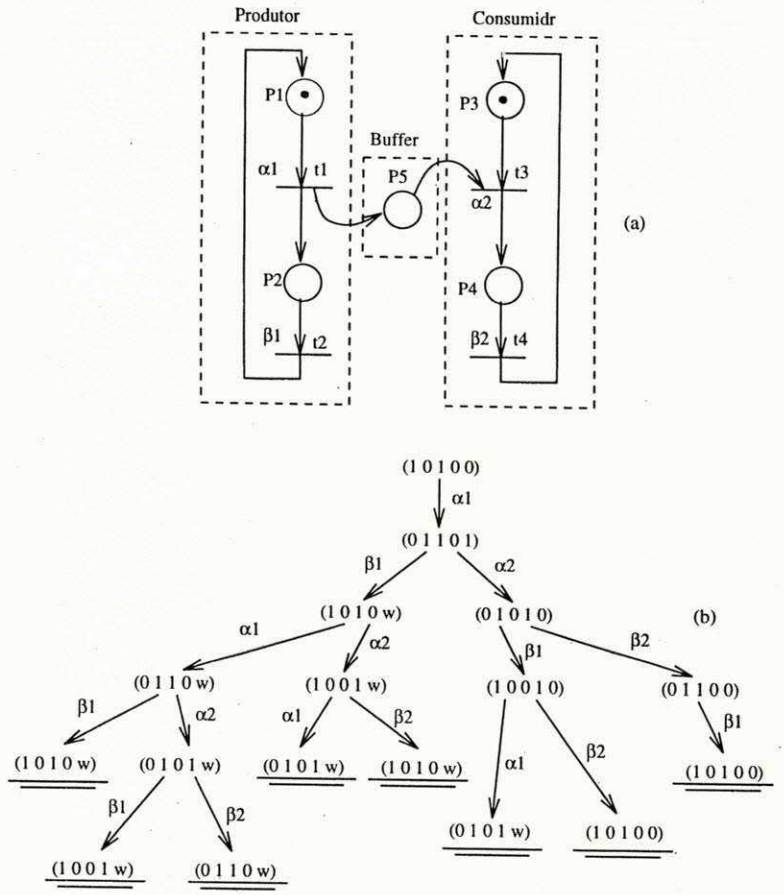


Figura 4.8: Modelo RP e árvore de cobertura do sistema produtor/consumidor.

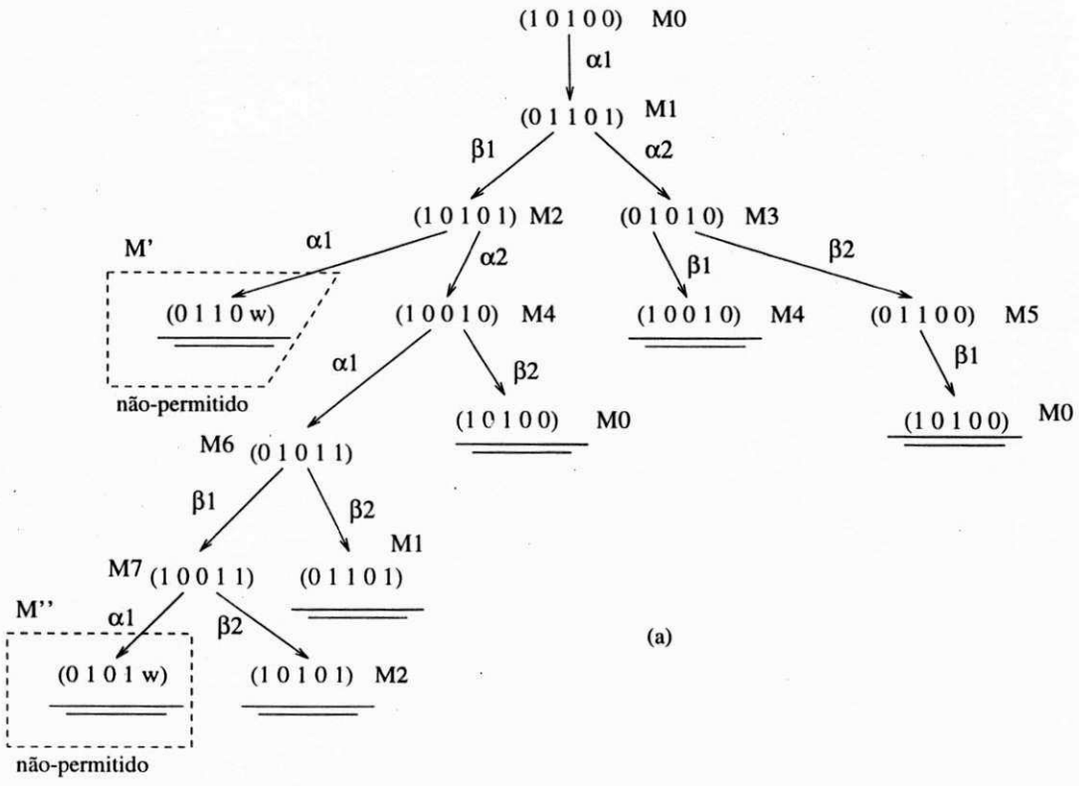


Figura 4.9: Árvore de alcançabilidade do sistema produtor/consumidor, obtida executando o *AMArA*.

α_2 e β_2 estão associados respectivamente ao disparo das transições t_1, t_2, t_3 e t_4. Note que este sistema pode produzir um número muito grande de peças antes que alguma seja consumida, pois a seqüência $(\alpha_1\beta_1)^*$ (disparo da seqüência de transições $t_1t_2t_1t_2\ldots$) pode ocorrer indefinidamente, conforme mostra a árvore de alcançabilidade do modelo, Figura 4.8(b). Para que esse modelo represente um sistema fisicamente possível, deve-se modifica-lo ou exercer sobre ele alguma forma de controle, como também considerar um limite máximo para o número de fichas em cada lugar.

Se o lugar p_5 na rede da Figura 4.8(a) possui capacidade igual a um, por exemplo, utilizando-se o *AMArA* chega-se à nova árvore de alcançabilidade mostrada na Figura 4.9.

Note que aplicando o Algoritmo da árvore de alcançabilidade, a informação do estado do sistema é perdida ao se disparar a cadeia $\alpha_1\beta_1$ a partir da *raiz* (marcação M_0), enquanto

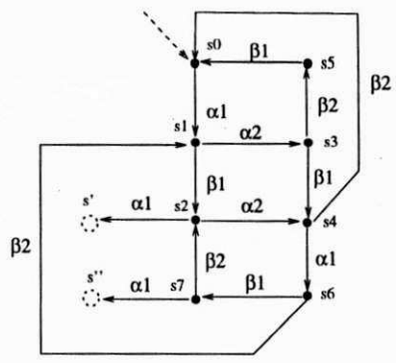


Figura 4.10: Gerador dos eventos fisicamente possíveis no sistema.

que o Algoritmo apresentado ($AMArA$) enumera todos os estados cuja capacidade da rede não seja excedida. Além disso, obtém-se a informação de quais cadeias podem levar o sistema a um estado *não-permitido.*

A partir da árvore de alcançabilidade, obtida executando-se o $AMArA$, pode-se construir a planta ou autômato gerador das cadeias cujos eventos ocorrem a partir do estado inicial do sistema. Veja Figura 4.10. Estão representados tanto na Figura 4.9 quanto na Figura 4.10 todas as cadeias fisicamente possíveis, onde M_i e s_i representam o mesmo estado do sistema.

O modelo do sistema até aqui descrito, é simplesmente um gerador de eventos espontâneo, sem nenhum controle externo. Para se controlar o sistema, segundo a teoria de controle supervisório, é necessário que certos eventos possam ser desabilitados para que a nova especificação do sistema possa ser executada, ou seja, apenas alguns eventos poderão ocorrer em certos instantes para que uma determinda tarefa seja realizada com sucesso.

Uma especificação desejável, por exemplo, pode ser assim definida: o sistema deve sempre retornar à marcação ou estado inicial, ou seja, $M_0 \in Q_m$.

Seja $\Sigma = \{\alpha_1, \alpha_2, \beta_1, \beta_2\}$ onde $\Sigma_c = \{\alpha_1, \alpha_2\}$ e $\Sigma_u = \{\beta_1, \beta_2\}$. Com base na especificação desejável e no conjunto de marcações da rede, executa-se o $ACGS$ e obtém-se a seguinte *lista_perigo*:

$$M_2 \rightarrow t_1(\alpha_1);\ M_7 \rightarrow t_1(\alpha_1),$$

Para que a especificação seja executada, basta que α_1 seja desabilitada nas marcações

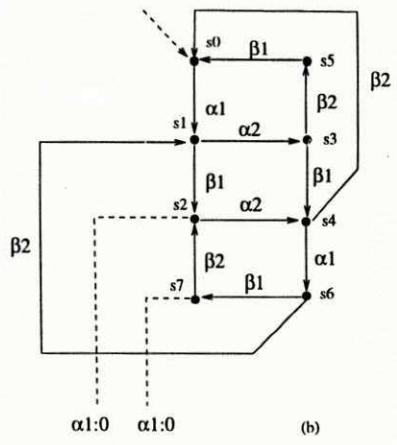


Figura 4.11: Gerador da especificação do sistema.

$M_2 = [\ 1\ \ 0\ \ 1\ \ 0\ \ 1\ ]^T$ e $M_7 = [\ 1\ \ 0\ \ 0\ \ 1\ \ 1\ ]^T$, o que é garantido pela função de habilitação associada a t_1.

Note que a função de habilitação pode ser simplificada, visto que neste caso só a marcação $M(p_5)$ interessa. Observe que nas marcações da *lista_perigo* (M_2 e M_7), $M(p_5) = 1$. Se $t_1(\alpha_1)$ ocorrer a partir destas marcações, então o número de fichas em p_5 será igual a 2, o que não atende à especificação definida pelo *ACGS*. Assim, baseada na especificação e na *lista_perigo*, a função de habilitação pode ser simplificada para $\varphi_1 = \overline{p_5}$.

O supervisor é então modelado pelo autômato apresentado na Figura 4.11 e pela *RPFHT* apresentada na Figura 4.12, onde:

$N = (P, T, F, W)$ é a estrutura da *RPFHT* idêntica à do modelo de rede de Petri da planta G;

l apresenta o seguinte mapeamento:

$$t_1 \rightarrow \alpha_1;\ t_2 \rightarrow \beta_1;\ t_3 \rightarrow \alpha_2;\ t_4 \rightarrow \beta_2;$$

$M_0 = [1\ 0\ 1\ 0\ 0]^T$ é a marcação inicial;

$\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$, onde $\varphi_1 = \overline{p_5}$ e $\varphi_2 = \varphi_3 = \varphi_4 = 1$.

A função de habilitação $\varphi_1 = \overline{p_5}$ significa que t_1 só poderá ser disparada se não existirem fichas depositadas em p_5, ou seja, $\varphi_1 = 1$ se $M(p_5) = 0$.

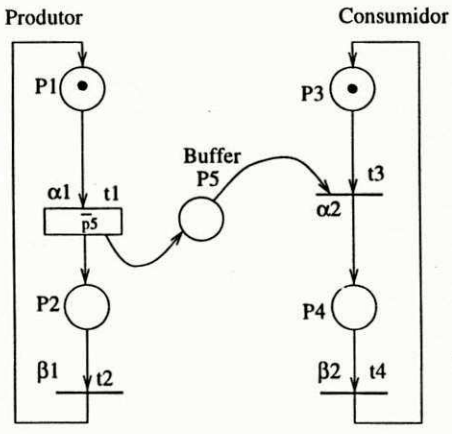


Figura 4.12: Rede de Petri supervisora do sistema produtor/consumidor.

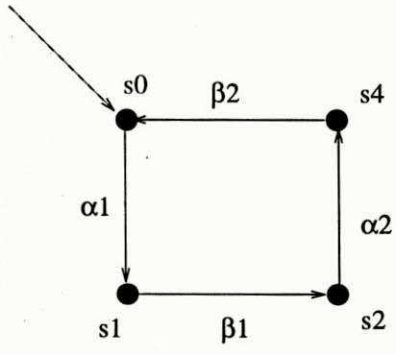


Figura 4.13: Nova especificação de uma tarefa para o sistema produtor/consumidor.

A *RPFHT* supervisora pode então ser executada sincronamente com a planta, obedecendo à regra de disparo de suas transições.

Note que se quisermos especificar uma outra tarefa para o sistema, por exemplo $K = (\alpha_1\beta_1\alpha_2\beta_2)^*$, figura 4.13, basta executar os passos 5 e 6 do Algoritmo *ACGS* para se encontrar os novos estados e os respectivos eventos a serem desabilitados.

Nesse caso, a nova *lista_perigo* será:

$$(M_1)! \rightarrow \alpha_2;$$
$$(M_2, M_4)! \rightarrow !\alpha_1$$

e as novas funções de habilitação serão:

$$\varphi_1 = \overline{p_4} \text{ e } \overline{p_5};$$

$$\varphi_3 = \overline{p_2};$$
$$\varphi_2 = \varphi_4 = 1,$$

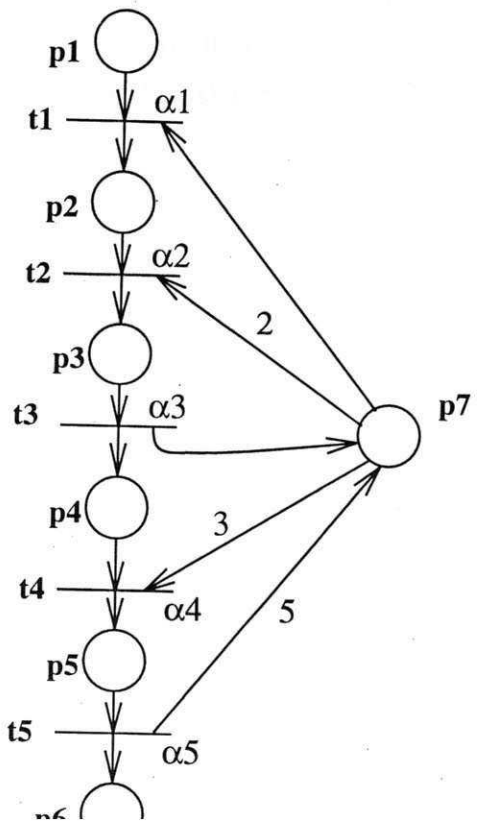
ou seja, a transição t_1 só poderá ser disparada quando os lugares p_4 e p_5 não contiverem fichas, enquanto t_3 só poderá ser disparada quando o lugar p_2 não contiver ficha.

4.2.4 Exemplo de um Sistema de Manufatura Simples

Considere o sistema de manufatura, modelado pela rede de Petri apresentada na Figura 4.14. Este sistema apresenta n usuários ($n = M_0(p_1) \geq 1$), que compartilham recursos para a realização de uma tarefa específica. Cada usuário necessita de 5 recursos para realizar a tarefa. Os recursos são indistinguíveis e o sistema possui um número k limitado de recursos ($k = M_0(p_7) \geq 5$).

Este modelo representa um sistema com partes (fichas em p_1) que serão processadas por quatro estações de processamento, modeladas, respectivamente, pelos lugares p_2, p_3, p_4 e p_5. Para o processamento das partes em cada estação, é necessário o compartilhamento de ferramentas (fichas em p_7). Por exemplo, uma ferramenta é necessária (disparo de t_1) para processar uma parte na primeira estação (ficha em p_2). Para ser processada na segunda estação (modelada por p_3) uma parte necessita de mais duas ferramentas (disparo de t_2). Na terceira estação de processamento (modelada por p_4) são necessárias apenas duas ferramentas, portanto, ao término do processamento na segunda estação uma ferramenta é liberada (disparo de t_3). Para o término do processamento de uma parte na quarta estação (modelada por p_5), mais três ferramentas são necessárias (disparo de t_4). Note que na quarta estação, cinco ferramentas são necessárias para o processamento de uma parte. Após o termino do processamento, as cinco ferramentas são liberadas ao mesmo tempo (disparo de t_5). Note também que mais de uma parte podem ser processadas em uma estação ao mesmo tempo, bastando para tal que os recursos (ferramentas) estejam disponíveis.

Este sistema pode ser levado a um bloqueio, isto é, é possível que o sistema alcance um estado em que nenhum evento é habilitado e assim, nenhum estado marcado poderá ser alcançado. Por exemplo, seja $M_0 = [\ 6\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 8\ ]^T$ o vetor representando a marcação inicial da rede apresentada na Figura 4.14, isto é, $M_0(p_1) = 6$, $M_0(p_7) = 8$,


p1
t1
α1
p2
t2
α2
2
p3
t3
α3
p7
p4
3
t4
α4
5
p5
t5
α5

e $M_0(p_i) = 0, i = \{2, \cdots, 6\}$. Se a seqüência de transições $t_1t_2t_1t_2t_1t_1t_3t_3t_2t_3t_1$ ocorrer, o sistema alcança o estado, ou marcação, $M' = [\ 1\ \ 2\ \ 0\ \ 3\ \ 0\ \ 0\ \ 0\ ]^T$. Nesta marcação nenhuma transição é habilitada, ou seja, não é possível a ocorrência de nenhum evento. Nesta situação, diz-se que o sistema alcançou um estado bloqueado e, por isso, não será capaz de completar uma tarefa. Considerando esta situação, é necessário uma ação de controle externa ao sistema para evitar que o mesmo alcance um estado indesejável.

Seja a marcação inicial $M_0 = [\ 6\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 8\ ]^T$ e seja $\Sigma = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$ o alfabeto de eventos que etiqueta as transições, como mostrado na Figura 4.14, e assuma que $\Sigma_c = \{\alpha_1, \alpha_2, \alpha_4\}$ e $\Sigma_u = \{\alpha_3, \alpha_5\}$. Em outras palavras, o sistema é inicializado com 6 usuários e 8 recursos e os eventos que podem ser desabilitados/habilitados são α_1, α_2 e α_4.

A árvore de alcançabilidade do modelo apresentado na Figura 4.14 é construída executando o *AMArA*. Para a marcação inicial dada, 162 marcações distintas são alcançáveis.

Para manter a coerência com a teoria de controle supervisório, o modelo do sistema até aqui descrito, é simplesmente um gerador de eventos espontâneo, sem nenhum controle externo. Para controlar o sistema, é necessário admitir que certos eventos podem ser desabilitados para que a nova especificação do sistema possa ser executada, ou seja, apenas alguns eventos poderão ocorrer em certos instantes para que um estado *não-permitido* ou *bloqueado* nunca seja alcançado.

Considere a seguinte especificação: o estado $M = [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 6\ \ 8\ ]^T$ tem que ser sempre alcançado. Este estado significa que todos os usuários completaram sua tarefa, ou seja, todos os usuários foram atendidos e todos os recursos estão disponíveis novamente. Formalmente, $M = [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 6\ \ 8\ ]^T \in Q_m$.

Nesse caso, dados a planta G do sistema, modelada pela rede de Petri apresentada na Figura 4.14, e o conjunto de estados possíveis, encontrado após a execução do *AMArA*, executa-se o *ACGS* e obtém-se a *lista_perigo* da Figura 4.15.

O supervisor é então implementado utilizando-se uma *RPFHT* com a mesma estrutura da rede que modela o sistema, associando às transições do supervisor as funções que definem o disparo ou não das transições.

Ao se executar o *ACGS*, encontra-se a componente *ajustada* do modelo e a partir dela o gerador da suprema linguagem controlável, que é a linguagem menos restritiva para a

Eventos a desabilitar	MARCAÇÕES							Eventos a desabilitar	MARCAÇÕES						
	p1	p2	p3	p4	p5	p6	p7		p1	p2	p3	p4	p5	p6	p7
t1	2	4	0	0	0	0	4	t2	2	3	1	0	0	0	2
	2	3	1	0	0	0	2		2	3	0	1	0	0	3
	2	3	0	1	0	0	3		1	3	1	0	0	1	2
	1	4	0	0	0	1	4		1	3	0	1	0	1	3
	1	3	1	0	0	1	2		0	3	1	0	0	2	2
	1	3	0	1	0	1	3		0	3	0	1	0	2	3
	1	1	1	1	0	2	2		0	1	1	1	0	3	2
	1	1	0	2	0	2	3		0	1	0	2	0	3	3
									1	1	1	1	0	2	2
									1	1	0	2	0	2	3
Funções	φ1								φ2						

φ1 =
- M(p7) >= 5 ou
- M(p7) >= 3 e M(p3) >= 1 ou
- M(p7) >= 2 e M(p3) >= 2 ou
- M(p3) >= 3 ou
- M(p5) >= 1

φ2 =
- M(p7) >= 4 ou
- M(p5) >= 1 ou
- M(p3) >= 2 ou
- M(p3) >= 1 e M(p7) >= 3

Figura 4.15: Lista de marcações e respectivos eventos a serem desabilitados e funções φ_1 e φ_2 de habilitação das transições t_1 e t_2, respectivamente.

especificação desejada. Note que a *lista_perigo*, Figura 4.15, possui todos os estados ou marcações com os respectivos eventos a serem desabilitados, ou seja, quando o sistema se encontrar em qualquer um desses estados, os eventos explicitados na *lista_perigo* não devem pertencer à entrada de controle correspondente, caso contrário o sistema pode alcançar um estado a partir do qual a tarefa especificada nunca será completada.

O supervisor é então modelado pela *RPFHT* da Figura 4.16, ou seja:

- $N = (P, T, F, W)$ é a estrutura da *RPFHT* idêntica à da rede de Petri da planta G;
- l apresenta o seguinte mapeamento:

$$t_1 \rightarrow \alpha_1;\ t_2 \rightarrow \alpha_2;\ t_3 \rightarrow \alpha_3;\ t_4 \rightarrow \alpha_4;\ t_5 \rightarrow \alpha_5;$$

- $M_0 = [\ 6\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 8\ ]^T$ é a marcação inicial;
- φ_1 e φ_2 assumem respectivamente o valor zero quando o sistema se encontra nas marcações correspondentes, pertencentes à *lista_perigo*. As funções φ_3, φ_4 e φ_5 são sempre iguais a 1 e portanto são omitidas por simplificação, sem nenhum prejuízo para o entendimento do supervisor.

De outra forma, como apenas α_1 e α_2 serão desabilitados, nas marcações apresentadas na *lista_perigo*, somente nas transições t_1 e t_2 da Figura 4.16 são apresentadas as funções associadas φ_1 e φ_2 respectivamente.

A *RPFHT* supervisora pode então ser executada sincronamente com o sistema, obedecendo à regra de disparo de suas transições.

Note que, como no exemplo do sistema produtor/consumidor apresentado anteriormente, se quisermos especificar uma outra tarefa para o sistema, basta executar os passos 5 e 6 do *ACGS* para se encontrar os novos estados e os respectivos eventos a serem desabilitados.

Note que a rede apresentada na Figura 4.14 modela uma célula de manufatura que trata basicamente do compartilhamento de recursos, ou seja, para que não haja bloqueio (estado em que nenhuma transição está habilitada) os recursos disponíveis ou a disponibilizar em um dado instante devem ser no mínimo iguais aos recursos necessários naquele instante.

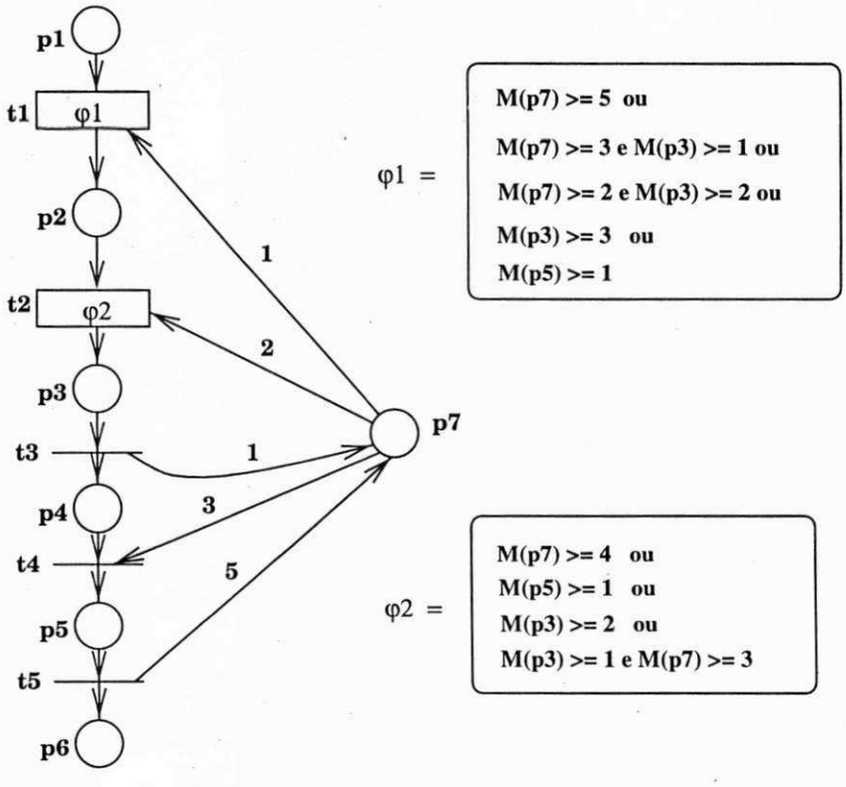


Figura 4.16: *RPFHT* supervisora.

Sendo assim, baseadas na *lista_perigo* e nas características do sistema, as funções de habilitação podem ser simplificadas.

No caso do exemplo, os recursos disponíveis são representados pelas fichas no lugar p_7. Fichas nos lugares p_3 e p_5 indicam que recursos estão prestes a serem disponibilizados. Desta forma, utilizando a *lista_perigo*, pode-se deduzir que $\varphi_1 = 1$ sempre que $[M(p_3) \geq 1$ e $M(p_7) \geq 3]$ ou $[M(p_3) \geq 2$ e $M(p_7) \geq 2]$. De modo semelhante, $\varphi_2 = 1$ sempre que $[M(p_3) \geq 1$ e $M(p_7) \geq 3]$. Baseado na *lista_perigo* e nas características do sistema podemos deduzir que $\varphi_1 = 1$ sempre que $M(p_7) \geq 5$, pois cada usuário necessita de no mínimo cinco recursos. Note que $M(p_7) \leq 4$ em todas as marcações na *lista_perigo*. $\varphi_1 = 1$ sempre que $M(p_7) \geq 4$, pois cada usuário (representados por uma ficha em p_2) necessita de no mínimo quatro recursos. Note que $M(p_7) \leq 3$ em todas as marcações da *lista_perigo* cujo evento α_2 deve ser desabilitado. Note ainda que se o número de recursos a disponibilizar é suficiente para os recursos necessários naquele instante, então $\varphi_1 = \varphi_2 = 1$ ($M(p_5) \geq 1$ significa que pelo menos 5 recursos serão disponibilizados e $M(p_3) \geq 3$ significa que pelo menos 3 recursos serão disponibilizados). Note que $M(p_5) = 0$ em todas as marcações da *lista_perigo*.

4.2.5 Redes de Petri Supervisoras Equivalentes

Como visto nas Seções anteriores, dado um modelo de rede de Petri de um sistema a eventos discretos e uma especificação desejável, é possível encontrar um supervisor, modelado por uma *RPFHT*, com a mesma estrutura da rede que modela o sistema. Para tanto, os algoritmos *AMArA* e *ACGS* são utilizados. As funções de habilitação de transições garantem que a especificação desejável, ou possível, seja executada de forma minimamente restritiva, evitando a ocorrência de eventos indesejáveis, ou estados não permitidos, como definido pelos algoritmos.

A *RPFHT* supervisora pode ser transformada em uma rede de Petri, bastando para tanto que as funções de habilitação de transições sejam substituidas por arcos ponderados que ligam as transições, antes associadas às funções de habilitação, aos lugares da rede. de tal forma que as restrições impostas ao disparo da transição não sejam modificadas. Assim, para uma dada função de habilitação de transição, pode-se encontrar uma rede de Petri equivalente. Esta rede pode então ser analisada pelos métodos de análise de redes

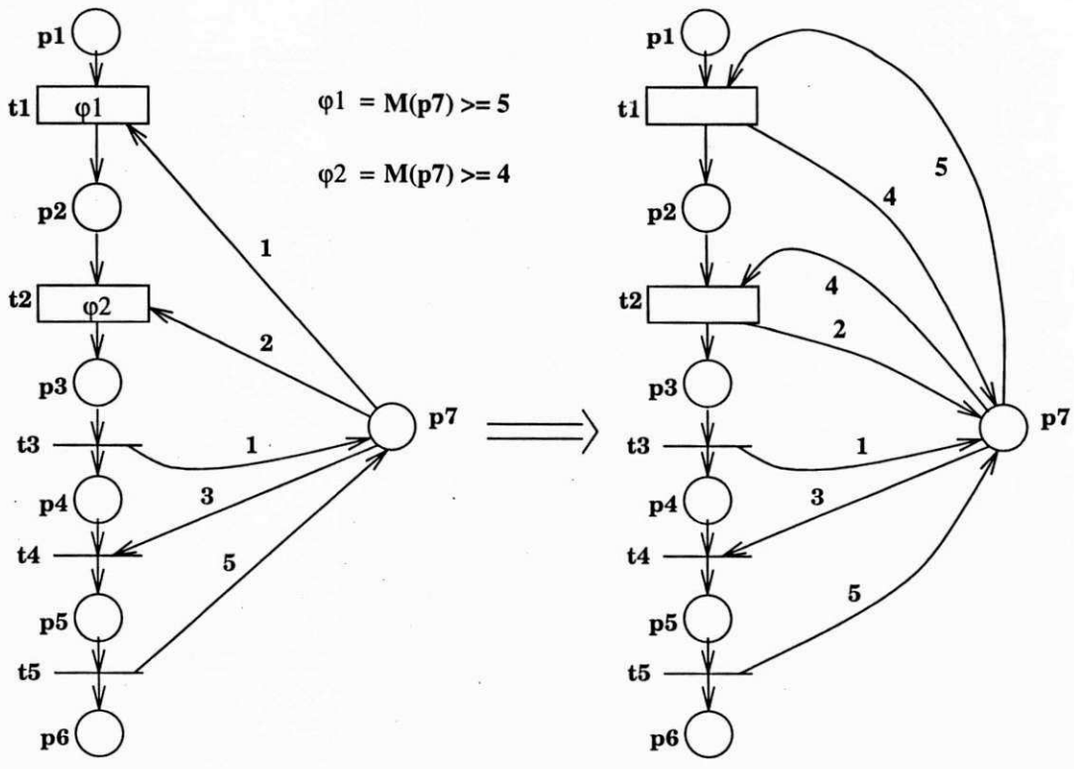


Figura 4.17: Rede de Petri supervisora equivalente.

de Petri, para verificação de propriedades desejáveis. Vejamos um exemplo a partir da *RPFHT* apresentada na Figura 4.16.

Sejam $\varphi_1 = [M(p_7) \geq 5]$ e $\varphi_2 = [M(p_7 \geq 4]$ casos particulares de φ_1 e φ_2 respectivamente. Nesse caso, $t_1(t_2)$ só poderá disparar se o número de fichas em p_7 for maior ou igual a cinco (quatro). Disparando t_1, uma ficha será retirada de p_1, uma ficha será retirada de p_7 e uma ficha será depositada em p_2. Da mesma forma, disparando t_2, uma ficha será retirada de p_2, duas fichas serão retiradas de p_7 e uma ficha será depositada em p_3.

Observe que para a rede supervisora apresentada na Figura 4.16, podemos então substituir a função $\varphi_1 = [M(p_7) \geq 5]$ e o arco de peso igual a um (p_7, t_1), que liga p_7 a t_1, pelos seguintes arcos:

- um arco com peso igual a 5, que liga p_7 a t_1;
- um arco com peso igual a 4, que liga t_1 a p_7.

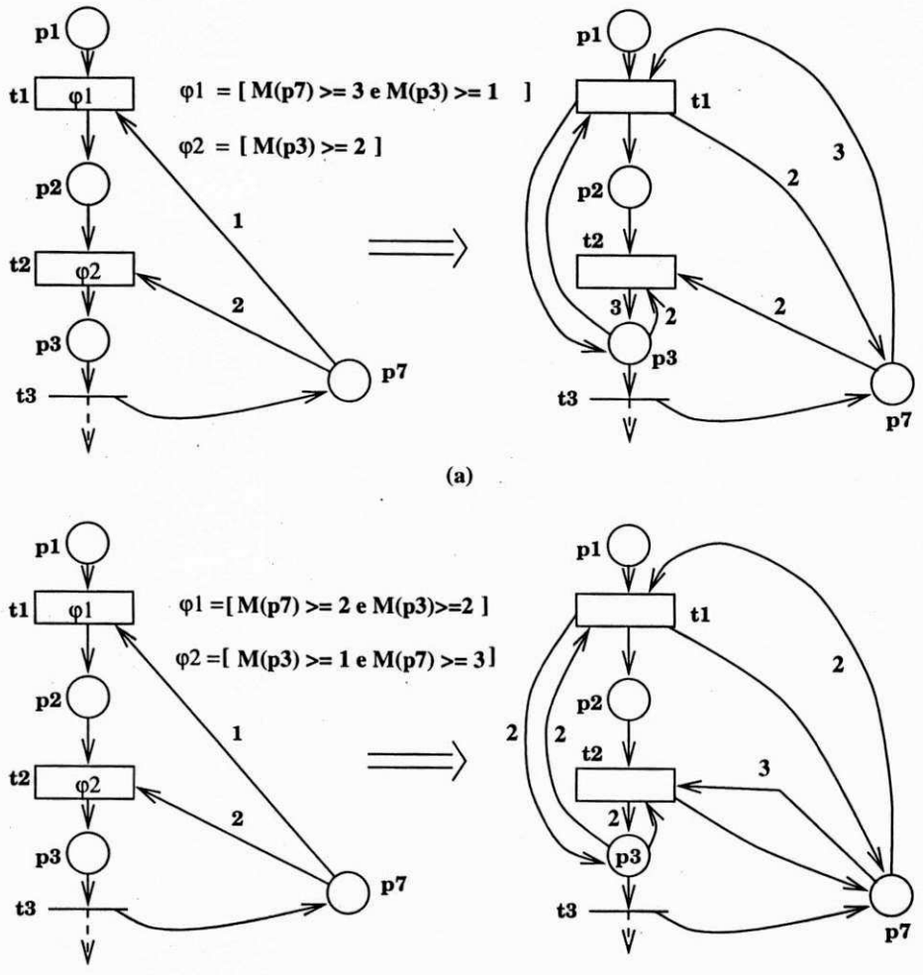


Figura 4.18: Subredes equivalentes.

Os arcos (p_1, t_1) e (t_1, p_2) não sofrem modificações.

Da mesma forma, podemos substituir a função $\varphi_2 = [M(p_7) \geq 4]$ e o arco de peso igual a dois, que liga p_7 a $t_2(p_7, t_2)$, pelos seguintes arcos:

- um arco com peso igual a 4, que liga p_7 a t_2;
- um arco com peso igual a 2, que liga t_2 a p_7.

Os arcos (p_2, t_2) e (t_2, p_3) não sofrem modificações. Veja Figura 4.17.

Note que para a rede da Figura 4.17, t_1 só estará habilitada, portanto poderá disparar, se $M(p_1) \geq 1$ e $M(p_7) \geq 5$. Disparando t_1, uma ficha é retirada de p_1, cinco fichas são retiradas de p_7, uma ficha é depositada em p_2 e quatro fichas são depositadas em p_7. Isso assegura que t_1 só irá disparar para $M(p_7) \geq 5$ e que ao final do disparo, o número de fichas efetivamente retiradas de p_7 seja igual a um. Assim, o disparo de t_1 na rede da Figura 4.16 é equivalente ao disparo de t_1 na rede da Figura 4.17.

O mesmo raciocínio vale para a habilitação e disparo das transições t_2 nas referidas redes.

Analizando a rede apresentada na Figura 4.17, para $M_0 = [\ 6\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 8\ ]^T$, conclui-se que não existe bloqueio do ponto de vista da teoria de controle supervisório, ou seja, $M = [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 6\ \ 8\ ]^T \in Q_m$ é sempre alcançável.

A Figura 4.18 apresenta outras subredes equivalentes às funções φ_1 e φ_2 respectivamente.

4.2.6 Redes de Petri com Temporização Nebulosa e Funções de Habilitação de Transições - *RPTN/FHTs*

Pelo que foi apresentado anteriormente, pode-se afirmar que as *RPFHTs* são uma excelente ferramenta para a supervisão de sistemas a eventos discretos do ponto de vista lógico, ou seja, uma *RPFHT* supervisora possibilita habilitar ou desabilitar os eventos necessários à execução de uma dada especificação, sem entretanto considerar as restrições de tempo envolvidas na realização da tarefa.

Considerando que, em um sistema a eventos discretos real, os intervalos de tempo envolvidos na realização de qualquer tarefa são de importância fundamental na determinação da eficiência do sistema em relação ao custo/desempenho do mesmo, um supervisor deve ser capaz de controlar um sistema tanto do ponto de vista lógico (corretude das tarefas especificadas) quanto do ponto de vista temporal (corretude das restrições de tempo impostas ao sistema).

Pensando em um supervisor com estas características, introduzimos uma extensão às redes de Petri com funções de habilitação de transições e às redes de Petri com temporização nebulosa, denominada *Redes de Petri com Temporização Nebulosa e Funções de Habilitação de Transições* (*RPTN/FHTs*).

Definição 4.3 *Uma rede de Petri com temporização nebulosa e funções de habilitação de transições é assim definida:*

$$RPTN/FHT = (N_N, \Phi, M_0), \text{ onde:}$$

- $N_N = (P, T, I, O, \mathcal{E}, \mathcal{D})$ *é uma estrutura de rede de Petri com temporização nebulosa;*
- $\Phi = (\varphi_1, \cdots, \varphi_m) : R(N_N, M_0) \rightarrow \{0, 1\}$ *é a função de habilitação das transições;*
- $M_0 : P \rightarrow FTF(p)$ *é a marcação inicial da rede.*

O comportamento dinâmico de uma *RPTN/FHT* é descrito pela seguinte regra de disparo das transições:

1. A transição t é dita estar habilitada para disparar se existe pelo menos uma ficha em cada um de seus lugares de entrada (assumindo peso 1 para cada um de seus arcos) no mesmo instante da escala de tempo e a função φ, associada à transição é verdadeira, isto é, dada uma transição t e o conjunto de seus lugares de entrada $(p_1, p_2, ..., p_n)$, com funções de temporização nebulosa $FTF(p_1)$, $FTF(p_2)$, ..., $FTF(p_n)$, respectivamente, a transição t é habilitada se e somente se:

$$FTF(p_1) \cap FTF(p_2) \cap ... \cap FTF(p_n) \neq \emptyset \text{ e } \varphi = 1.$$

2. A transição permanece habilitada por um período E antes de poder disparar;
3. A transição começa e continua o disparo por um período D;
4. Como resultado de um disparo, uma ficha com uma nova função de temporização nebulosa é colocado em cada lugar de saída da transição. Essa nova função de temporização nebulosa é obtida da composição das fichas nebulosas dos lugares de entrada com os intervalos de temporização nebulosos associados à transição.

Note que a regra de disparo de uma transição em uma *RPTN/FHT*, é igual a de uma transição em uma *RPTN*, acrescida da condição imposta pela função de habilitação da transição.

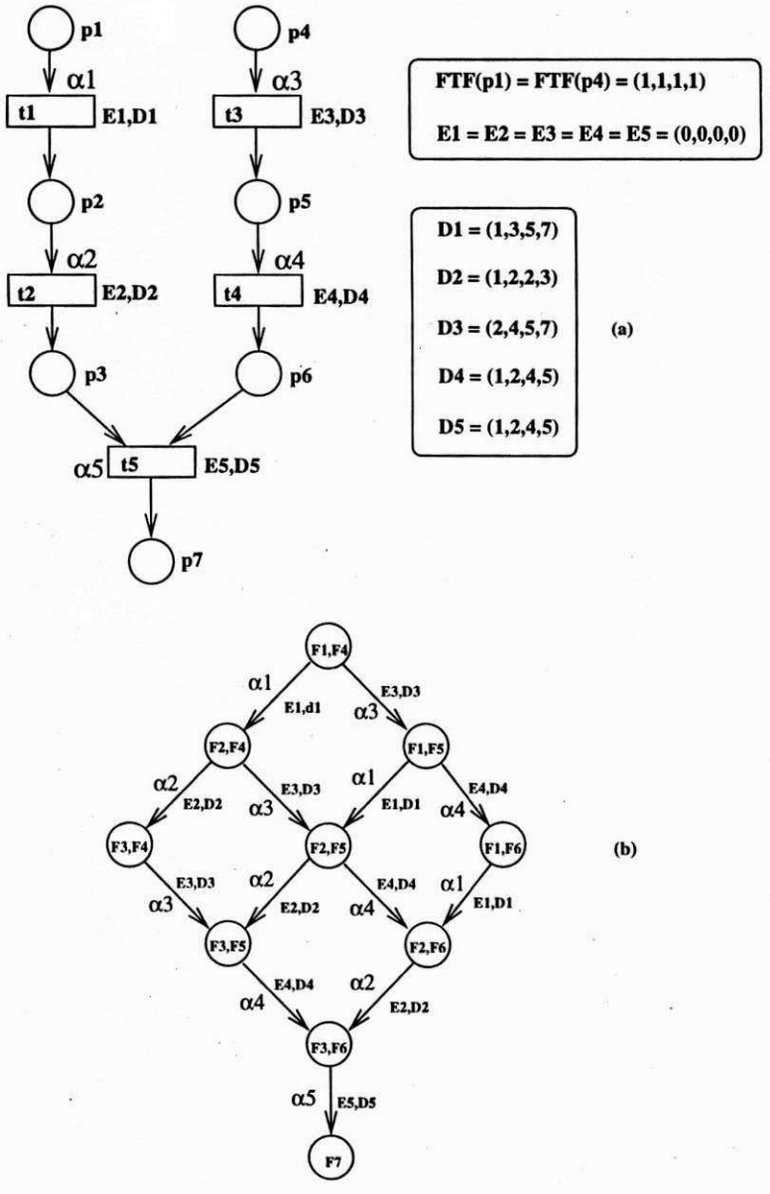


Figura 4.19: (a) Modelo *RPTN* de uma célula de manufatura; (b) Grafo de alcançabilidade nebuloso.

Para ilustrar a utilização de uma *RPTN/FHT* como rede supervisora para uma célula de manufatura, considere a *RPTN* apresentada na Figura 2.10. Para uma melhor leitura, o exemplo é reapresentado na Figura 4.19(a).

Para esta célula de manufatura, deseja-se a seguinte especificação:

- Para cada ficha depositada em p_6, é necessário que já exista uma ficha correspondente em p_3, isto é, para que uma peça final seja montada, é necessário que as partes processadas pela máquina representada por p_2 sejam colocadas primeiro na plataforma de montagem, ou seja,

$$\forall M \in R(N_N, M_0), \text{ se } M(p_3) \geq M(p_6) \text{ então } M \in Q_m.$$

Baseados no *grafo de alcançabilidade nebuloso*, que apresenta todos os possíveis estados da *RPTN*, e na especificação desejável, executa-se o *ACGS* e encontra-se a seguinte *lista_perigo*:

$$M_2 \rightarrow \alpha_4;\ M_4 \rightarrow \alpha_4.$$

Dada a *lista_perigo*, encontra-se a função de habilitação φ_4 da transição t_4 e, consequentemente a *RPTN/FHT* supervisora. Veja Figura 4.20.

Note que a função $\varphi_4 = \{M(p_6) < M(p_3)\}$ força a que o número de fichas em p_3 seja sempre maior ou igual ao número de fichas em p_6. Considerando a marcação inicial do exemplo anterior ($M_0 = [\ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ ]^T$), é sempre possível alcançar o estado desejável ($M_9 = [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ ]^T$), visto que não ocorrem mudanças nem nos intervalos de tempo nebulosos, associados às transições, nem nas funções de temporização nebulosas, associadas às fichas, ou seja, as funções de temporização nebulosas das fichas, na *RPTN/FHT* supervisora, são equivalentes àquelas do modelo *RPTN*, bem como os intervalos nebulosos D e E. Assim, ao serem executados sincronamente (supervisor e sistema), os estados em que os mesmos se encontram são equivalentes.

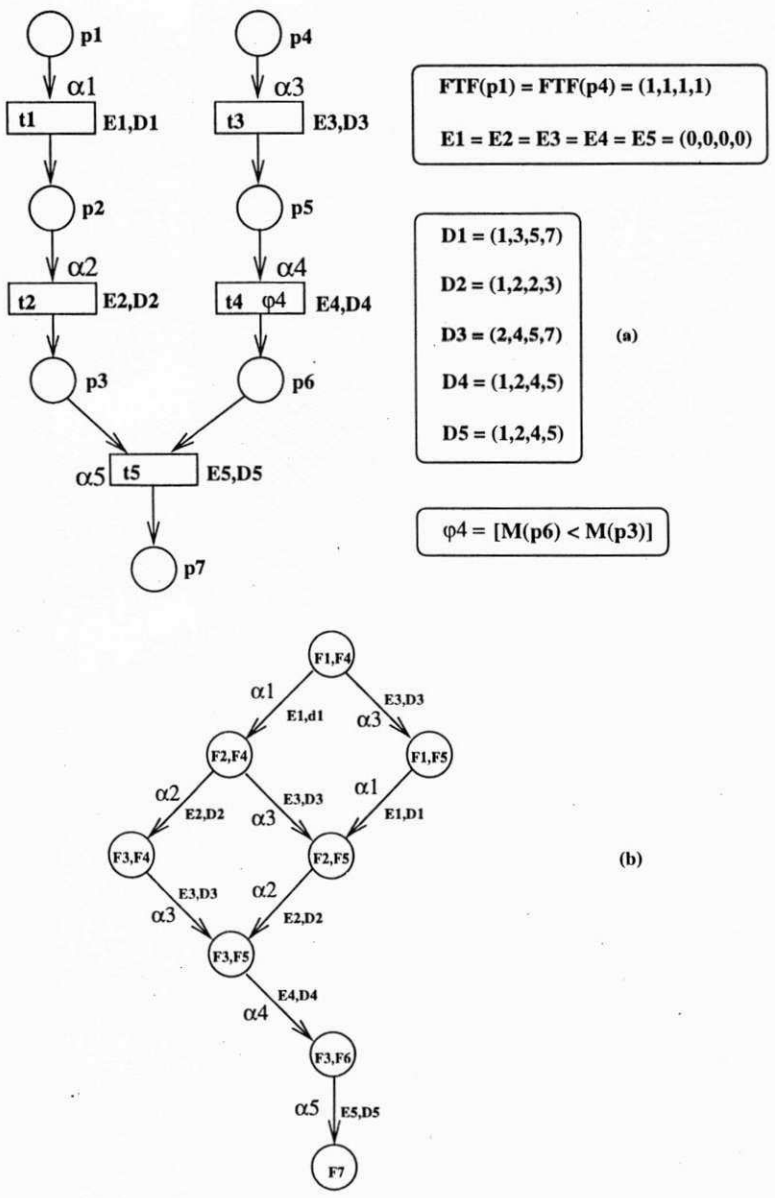


Figura 4.20: (a) Supervisor *RPTN/FHT*; (b) Grafo de alcançabilidade nebuloso.

4.3 Abordagem modular

Na Seção anterior, introduzimos as *RPFHT*s. Essa classe de redes de Petri foi aplicada na modelagem e síntese de supervisores. Também foram apresentados o Algoritmo Modificado da Árvore de Alcançabilidade (*AMArA*), que permite a obtenção de todas as marcações de uma rede de Petri com capacidade finita e o Algoritmo para a construção do gerador da *supC*(*L*) (*ACGS*). Usando o modelo de *RPFHT* e os algoritmos citados, introduziu-se uma abordagem sistemática para a solução do problema de síntese de supervisores, sem modificações *ad-hoc* na estrutura da rede que modela o sistema.

Também foram introduzidas as *RPTN/FHT*s na síntese de supervisores com restrições de tempo para controle de sistemas a eventos discretos tanto do ponto de vista lógico quanto dos limites de tempo estabelecidos para a realização das tarefas especificadas.

Por outro lado, a utilização de redes de Petri na modelagem e síntese de supervisores de sistemas a eventos discretos reais mais complexos, torna-se proibitiva, visto que o número de estados que esses sistemas podem alcançar é muito grande. Nestes casos, é desejável, e muitas vezes necessário, a adoção de uma metodologia composicional ou modular. Esta metodologia, no entanto, deve permitir a verificação de propriedades locais aos módulos, ou componentes individuais do sistema, bem como permitir a verificação do comportamento correto na integração entre componentes do sistema.

Pensando na solução para o problema de síntese de supervisores de sistemas a eventos discretos reais, nesta Seção propomos uma abordagem modular, baseada em sistemas de *G-Nets*, para a síntese do supervisor para SEDs. Utilizando esta abordagem, é possível simplificar a construção do supervisor, visto que a análise pode ser restrita aos módulos ou *G-Net*s de interesse, dado que em um sistema de *G-Nets* as várias *G-Nets* que compõem o sistema podem ser consideradas independentemente. Desta forma, a análise e o controle podem ser localizados e executados para cada *G-Net*, desde que a interface entre as *G Nets* se mantenha imutável, como definido por Perkusich [Per94]. O objetivo principal da utilização de *G-Nets*, nesta abordagem de síntese de supervisores, é evitar a explosão de estados e assim simplificar a construção do supervisor para sistemas mais complexos. Assim, em vez de analisar todo o sistema a ser controlado, é necessário analisar apenas os módulos de interesse a serem efetivamente controlados, diminuindo a complexidade da análise e gerenciando melhor o problema de explosão de estados do sistema.

Note que esta abordagem de síntese modular, difere daquela proposta por Ramadge e Wonham [RW88, RW87a]. Na abordagem Ramadge e Wonham, a especificação global é dada como uma coleção de linguagens, em que cada linguagem representa uma propriedade desejável para o sistema controlado. A intersecção destas linguagens produz a linguagem especificada global. Assim, o problema de síntese do supervisor é solucionado para cada especificação e os supervisores resultantes são então compostos para formar a solução global do problema especificado.

Infelizmente, a composição de supervisores não preserva a propriedade *não bloqueável* do comportamento em malha fechada. Um supervisor global completo, tal que o sistema em malha fechada é não bloqueável, nem sempre pode ser construído pela composição de supervisores modulares, a não ser que as linguagens envolvidas sejam *não conflitantes.*

Em nossa abordagem, como os módulos em um sistema de *G-Nets* são independentes e fracamente acoplados, o supervisor é sintetizado para cada módulo de interesse, não sendo necessária a composição para a formação de um supervisor global, desde que garantida a imutabilidade da interface entre as *G-Nets*, como citado anteriormente.

Em adição, pode-se dizer que para as duas abordagens, um supervisor modular é sempre mais fácil de ser modificado, atualizado e mantido. Por exemplo, se a especificação de um módulo (subespecificação na abordagem Ramadge Wonham) é modificada, então é necessário se redefinir somente o supervisor daquele módulo e não o supervisor do sistema global.

Os algoritmos de síntese de supervisores, apresentados na Subseção 4.2.2 são também utilizados na construção do supervisor. A nova abordagem é introduzida através da modelagem, via sistemas de *G-Nets*, de uma célula de manufatura.

4.3.1 Aplicação da Abordagem Modular para uma Célula de Manufatura

Considerando a célula de manufatura, apresentada na Figura 4.21, é possível identificar dois depósitos de matéria prima **RM1** e **RM2**; dois centros de processamento **MP1** e **MP2**; e um centro de montagem **CM**. Um robot **R** move a matéria prima dos depósitos **RM1** e **RM2** para os centros de processamento **MP1** e **MP2**, respectivamente, e destes centros para o centro de montagem **CM**. A Figura 4.22 apresenta o modelo de sistema

de *G-Nets* para esta célula, não levando em consideração as partes pontilhadas.

O sistema é constituído por três módulos, a saber: a *G-NET GN(MAQ)* que modela os centros de processamento; a *G-NET GN(R)* que modela o robô; e a *G-NET GN(CM)* que modela o centro de montagem. Utiliza-se o modelo de rede de Petri Colorida de Jensen [Jen92, Jen80] para modelar a estrutura interna de cada *G-Net*. Para este exemplo, duas partes montadas constituem um item desta célula. Os itens são transportados para outra célula por um agente externo. Esta ação não é modelada.

O estado inicial do sistema é representado por duas fichas de cores mp_1 e mp_2, respectivamente, no lugar $\mathbf{P}_1$ de *GN(MAQ)*, significando que os centros de processamento estão inicialmente livres, e uma ficha no lugar *RL* de *GN(R)*, significando que o robô está inicialmente livre.

As inscrições nos três retângulos, de linhas não pontilhadas, posicionados ao lado de *GN(MAQ)*, representam, respectivamente, as variáveis, conjuntos de cores e funções de arcos associados às redes de Petri coloridas que implementam a estrutura interna de cada *G-Net*.

Os conjuntos de cores associados aos lugares e transições, são assim definidos:

- cor $MP = with\ \ m1|\ m2$, ou seja, o conjunto de cores *MP* contém os elementos *m1* e *m2*;
- cor $RM = with\ \ r1|\ r2$, ou seja, o conjunto de cores *RM* contém os elementos *r1* e *r2*;
- cor $RL = unit\ \ with\ \ e$; ou seja, o conjunto de cores *RL* contém um único elemento *e*.

As variáveis são assim definidas:

- $var\ \ x : MP$, ou seja, à variável x pode-se atribuir o valor *m1* ou o valor *m2*;
- $var\ \ y : RM$, ou seja, à variável y pode-se atribuir o valor *r1* ou o valor *r2*;
- $var\ \ m, n : int$. As variáveis m e n são do tipo inteiro.

As funções associadas aos arcos são assim definidas:

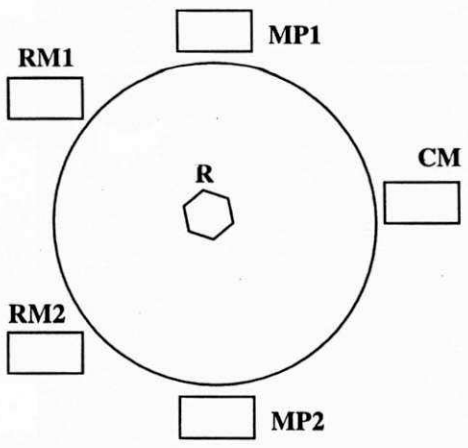


Figura 4.21: Exemplo de uma célula de manufatura.

- fun $FMAQ(x) = case\ x\ of\ m1 => 1'r1 \mid m2 => 1'r2$. Considerando a transição t_1 de *GN(MAQ)*, a mesma só estará habilitada se para uma ficha de cor m_i no lugar p_1 existir pelo menos uma ficha de cor r_i $(1'r_i)$ no lugar p_2, onde p_1 e p_2 são os lugares de entrada de t_1.

- fun $FAC(y) = n'r1 + m'r2$. Considerando a transição ta_1 de *GN(CM)*, a mesma só estará habilitada quando em seu lugar de entrada *Ia* existirem pelo menos n fichas de cor *r1* e m fichas de cor *r2*.

Note que o sistema é inicializado quando *GN(MAQ)* é invocada com o método *inicio*. Nesse caso, fichas de cores rm_1 e rm_2 são depositadas no lugar $\mathbf{P_2}$ de *GN(MAQ)*. Da forma em que está modelado o sistema, as matérias primas poderão ser processadas e enviadas para o centro de montagem **CM** sem restrições, independentemente se **CM** está ou não pronto para receber as peças processadas.

O número máximo de peças que o centro de montagem **CM** suporta é dois. Duas peças são o bastante para a montagem de um item, ou seja, o lugar I_a de *GN(CM)* pode suportar no máximo duas fichas, qualquer que seja a marcação da rede.

Seja $\Sigma = \Sigma_p \cup \Sigma_r \cup \Sigma_a$, onde $\Sigma_p = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, $\Sigma_r = \{\beta_1, \beta_2, \beta_3\}$, $\Sigma_a = \{\gamma_1\}$, e $\Sigma_c = \{\alpha_1, \alpha_3\}$ e $\Sigma_u = \{\alpha_2, \alpha_4, \beta_1, \beta_2, \beta_3, \gamma_1\}$. Veja Figura 4.22 para enteder o significado dos eventos α_i, β_i e γ_i.

Considere as seguintes especificações:

- Todas as peças têm que ser processadas e a seguir montadas;
- O centro de montagem **CM** deve receber no máximo duas peças para a montagem de cada item;
- Um item é constituído de uma peça de **MP1** e outra de **MP2**;
- A peça de **MP1** tem que chegar primeiro em **CM**.

Formalmente, tem-se:

$$\forall M(isp(R.st)) : r_1 \geq r_2 \wedge r_1, r_2 \leq 1$$

Para o modelo do sistema de *G-Nets* apresentado na Figura 4.22, a primeira especificação pode ser vista da seguinte forma: qualquer ficha de cor $rm_i (i = 1, 2)$ depositada em P_2 na *GN(MAQ)* deve alcançar o lugar alvo *GP*. A segunda especificação pode ser vista da seguinte forma: o lugar I_a de *GN(CM)* não pode conter mais que duas fichas, uma de cor rm_1 e outra de cor rm_2. Note que uma ficha no lugar *isp(R.st)* de *GN(MAQ)* significa que *GN(R)* foi invocada com o método *st*, ou seja, o robô deve pegar uma peça em $\mathbf{MP}_i$ e colocá-la em **CM**. Como os eventos em *GN(R)* são não controláveis, tem-se que evitar um número excessivo de fichas (conforme especificação) em *isp(R.st)* de *GN(MAQ)*. Isso pode ser feito visto que os eventos α_1 e α_3 são controláveis.

Como somente os eventos α_1 e α_3 são controláveis nesta célula, é necessário se executar os Algoritmos *AMArA* e *ACGS* somente para a *G-Net GN(MAQ)*.

Quando se executa o sistema de *G-Nets* e uma ficha de cor rm_i alcança o lugar *isp(R.st)*, a rede *GN(R)* é invocada com o método *st* e uma ficha é colocada no lugar I_r de *GN(R)*. Quando a transição tr_2 é disparada, uma ficha é colocada no lugar *isp(CM.it)*. A rede *GN(CM)* é então invocada e uma ficha é colocada no lugar I_a de *GN(CM)*. Isso significa que uma peça foi transportada, pelo robô, de um dos centros de processamento para o centro de montagem **CM**.

Note que a transição t_4 de *GN(MAQ)* não poderá ser disparada enquanto uma mensagem de reconhecimento não for recebida, ou seja, enquanto o lugar alvo GP_2 da rede *GN(R)* não for alcançado. Mais ainda, a transição tr_3 em *GN(R)* somente poderá ser disparada quando o lugar alvo GP_a na rede *GN(CM)* for alcançado. GP_a será alcançado

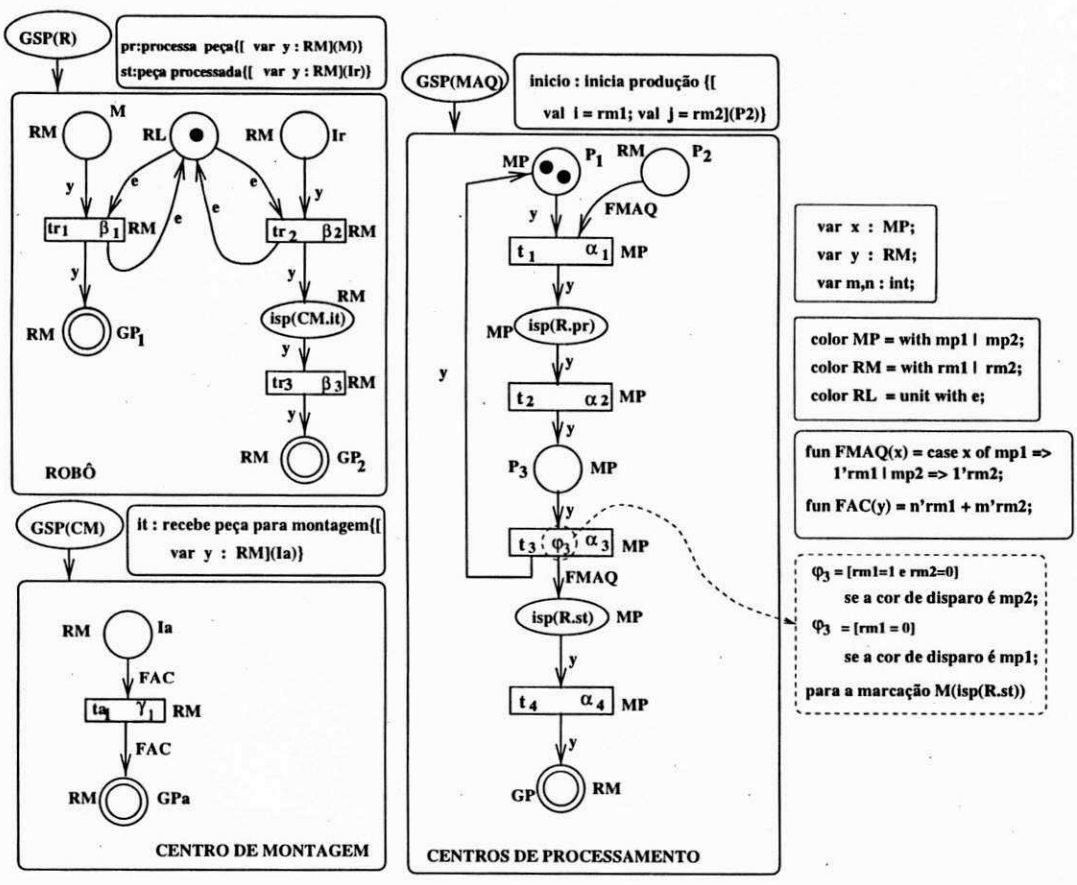


Figura 4.22: Modelo do sistema de *G-Nets* para a célula de manufatura e *RPFHT* supervisora.

Tabela 4.1: Tabela de marcações e respectivos eventos a serem desabilitados

Σ_c	Marcações					
	P1	**P2**	**ispRpr**	**P3**	**ispRst**	**GPc**
	mp2	2rm2	0	mp1	rm1	0
	0	rm2	mp2	mp1	rm1	0
	0	rm2	0	mp1,mp2	rm1	0
t3	0	0	mp2	mp1	rm1,rm2	0
α3	mp2	rm2	0	mp1	rm1	rm2
	0	0	mp2	mp1	rm1	rm2
	0	0	0	mp1,mp2	rm1	rm2
	mp2	0	0	mp1	rm1,rm2	rm2
	mp2	0	0	mp1	rm1	2rm2
φ	φ_3=[rm1=0] para a cor de disparo = mp1					

Σ_c	Marcações					
	P1	**P2**	**ispRpr**	**P3**	**ispRst**	**GPc**
	mp1	2rm1,rm2	0	mp2	0	0
	0	rm1,rm2	mp1	mp2	0	0
	0	rm1,rm2	0	mp1,mp2	0	0
t3	mp1	2rm1	0	mp2	0	rm2
α3	0	rm1	mp1	mp2	0	rm2
	mp1	rm1	0	mp2	rm2	rm1
	0	0	0	mp1,mp2	rm1,rm2	0
	mp1	rm1	0	mp2	rm2	rm1
	0	0	0	mp1,mp2	rm1,rm2	0
	0	0	mp1	mp2	rm2	rm1
	0	0	0	mp1,mp2	rm2	rm1
	mp1	0	0	mp2	rm1,rm2	rm1
	mp1	0	0	mp2	rm2	2rm1
φ	φ_3=[rm1=1 and rm2=0] para a cor de disparo = mp2					

quando um item for montado. Enquanto isso, o lugar *isp(R.st)*, de *GN(MAQ)*, contém uma cópia de cada ficha (peça) enviada para *GN(CM)*.

Assim, para que as tarefas expecificadas sejam executadas, é necessário que:

1. o lugar alvo *GP* em *GN(MAQ)* seja alcançado por todas as fichas colocadas em p_2 quando da inicialização de *GN(MAQ)* (Isso é uma garantia de que todas as peças foram enviadas para o centro de montagem);

2. garantir que no lugar *isp(R.st)* seja colocada no máximo uma ficha de cor rm_1(peça processada em **MP1**);

3. e que uma ficha de cor rm_2 (peça processada em **MP2**) seja colocada no lugar *isp(R.st)* somente quando já existir uma ficha de cor rm_1 naquele lugar;

4. finalmente, é necessário se garantir a existência de no máximo duas fichas no lugar *isp(R.st)*.

Após a execução do Algoritmo *AMArA*, para uma invocação de *GN(MAQ)* em que duas fichas de cor rm_1 e duas de cor rm_2 são colocadas em P_2, obtém-se a árvore de alcançabilidade de *GN(MAQ)*. Esta árvore possui 147 estados ou marcações, portanto apresentaremos somente as marcações encontradas após a execução do Algoritmo *ACGS*.

Note que dentre estas marcações encontram-se algumas que não devem ser alcançadas, por exemplo a marcação $M' = [(mp_2)(2rm_2)(0)(0)(2rm_1)(0)]^T$, e a marcação final, ou estado marcado, $M_f = [(mp_1, mp_2)(0)(0)(0)(0)(2rm_1, 2rm_2)]^T$.

Executa-se então o *ACGS* e encontra-se o gerador da suprema linguagem controlável para as expecificações acima. Ao se executar o *ACGS*, a *lista_perigo*, como mostrado na Tabela 4.1, é construída. Assim, todos os estados e respectivas transições, a serem desabilitadas quando o sistema se encontrar nesses estados, são apresentados. Baseados na *lista_perigo*, podemos inferir que a transição t_3 (evento α_3) em *GN(MAQ)* só pode ser disparada em relação a uma ficha de cor mp_1 se não existirem fichas no lugar *isp(R.st)*. Da mesma forma, o evento α_3 só pode ocorrer em relação a uma ficha de cor mp_2 se no lugar *isp(R.st)* já existir uma ficha de cor rm_1. Em qualquer outra situação, a transição t_3 deverá estar desabilitada. Isso é garantido pela função φ_3 associada à transição t_3 (α_3)

da *G-Net* supervisora, como mostra a Figura 4.22, levando em consideração as partes pontilhadas.

Note que a ação de controle exercida sobre o módulo *GN(MAQ)* é completamente transparente aos outros dois módulos, dado que a a interface entre os mesmos não foi modificada pela ação de controle, ou seja, *GN(MAQ)* continua solicitando os mesmos serviços que solicitava a *GN(R)* antes do controle e este, por sua vez, continua provendo os mesmos serviços, ou seja, transferindo ítens dos depósitos para os centros de processamento e destes para o centro de montagem. A diferença é que os ítens a serem transportados são selecionados, pela ação de controle, em *GN(MAQ)*.

Capítulo 5

Aspectos de Segurança e Tolerância a Falhas

5.1 Introdução

A crescente disponibilidade e baixo custo dos sistemas computacionais tem possibilitado sua aplicação para a solução de uma grande variedade de problemas que vão desde simples aplicações até tarefas mais complexas e críticas, como por exemplo o controle de tráfego aéreo, sistemas de transportes, sistemas médicos e sistemas de manufatura, para citar apenas alguns. Assim como os sistemas de computação, os sistemas, por eles controlados, estão sujeitos a falhas e, dependendo da natureza das mesmas, as conseqüências podem variar desde uma simples inconveniência até uma catástrofe. Desta forma, aspectos relacionados com a confiabilidade, segurança e tolerância a falhas são de grande importância quando se usam sistemas de computação.

Para os sistemas a eventos discretos (SEDs) em tempo real, propriedades lógicas, tais como *vivacidade (liveness)* e *segurança (safeness)*, bem como propriedades temporais têm que ser verificadas. Portanto, em se tratando do projeto de supervisores para SEDs em tempo real, é necessária uma metodologia que inclua mecanismos de verificação de propriedades dependentes e não dependentes do tempo. Para um grande número de sistemas reais, as propriedades relacionadas com a segurança são de vital importância, pois estes sistemas interagem com o ambiente que pode se comportar de forma não prevista

na especificação, ou ainda devido a falhas de *hardware, software* ou mesmo humanas.

Em sistemas em tempo real, a maioria das falhas está relacionada com erros de desempenho e sincronização que aparecem como falhas transitórias do mesmo. Assim, para garantir desempenho como especificado para os domínios do tempo e de valor, para um SED controlado em tempo real, deve-se levar em conta a possibilidade de se introduzir de forma sistemática, em seu supervisor, propriedades de tolerância a falhas dependentes e não dependentes do tempo.

Neste Capítulo discutiremos como a metodologia de síntese apresentada pode ser aplicada no projeto de supervisores, para embutir nos mesmos propriedades de segurança e tolerância a falhas dependentes e não dependentes do tempo, ou seja, dado que ocorreu uma falha no sistema supervisionado, o supervisor deve ser capaz de detectar a falha, inibir a ocorrência de novos eventos indesejáveis e retornar a um estado anterior correto quando o sistema estiver novamente operacional, após solucionado o problema no mesmo. Ao discutirmos aspectos de tolerância a falhas dependentes do tempo, mostraremos como são aplicadas as redes de Petri com temporização nebulosa para modelar esquemas tolerantes a falhas dependentes do tempo, bem como a sua integração com os sistemas de *G-Nets*.

Nas Seções seguintes apresentamos como propriedades de segurança e tolerância a falhas podem ser embutidas no supervisor, destacando os principais resultados. Na Seção 5.2 são apresentados os conceitos básicos relativos à segurança e tolerância a falhas em sistemas a eventos discretos. A Seção 5.3 apresenta a utilização da metodologia de síntese para embutir propriedades de segurança em supervisores, através do exemplo do cruzamento de uma rodovia com uma ferrovia. A Seção 5.4 apresenta a síntese de supervisores para a introdução de propriedades de tolerância a falhas não dependentes do tempo nos mesmos. Apresenta-se, na Seção 5.5, a síntese modular de supervisores para a introdução de caractersticas de tolerância a falhas dependentes do tempo em supervisores de sistemas a eventos discretos, utilizando sistemas de *G-Nets* e redes de Petri com temporização nebulosa.

5.2 Conceitos Básicos

Segurança de um sistema é definida como a probabilidade de que o sistema ou desempenhará suas funções corretamente ou as descontinuará, de modo que operações sendo executadas em outros sistemas associados a ele não sejam afetadas em caso de falha [Joh89].

De acordo com Avizienis e Laprie [AL86] os termos *falha, erro* e *falta* podem ser definidos por:

> Uma *falta* ocorre no sistema quando os serviços sendo executados desviam-se de sua especificação, onde a especificação do serviço é uma descrição do que se espera do serviço. A falta ocorre pois o sistema está em um estado de erro: um *erro* é a parte do estado do sistema que é sujeito a faltas. A causa de um erro é uma *falha*. Um erro é então a manifestação de uma falha no sistema, e uma falta é o efeito de um erro no serviço.

De acordo com a bibliografia brasileira, os termos *falta* e *falha* serão tratados indistintamente pelo termo *falha*. Existem diferentes tipos de falhas, dentre as quais, podem ser incluídas as falhas permanentes e transitórias, as falhas de *hardware* e *software* e as falhas temporais.

No caso de sistemas dedicados, por exemplo sistemas de controle aéreo, o controle do sistema não pode ser interrompido abruptamente quando uma falha ocorre [LS87, LS85]. Portanto, devem ser embutidas no sistema capacidades de resposta a falhas de *hardware*, *software* e talvez a condições ambientais não esperadas. Estas respostas, segundo Levenson [LS87] e Perkusich [dFPM93], podem ser classificadas como:

1 - tolerância a falhas é definida como a capacidade de um sistema de continuar desempenhando completamente suas funções mesmo na presença de falhas de operação.

2 - falha segura significa que o sistema tenta limitar o total do prejuízo causado por uma falha. Neste caso, nenhuma tentativa é feita para satisfazer as especificações funcionais, exceto quando for necessário garantir segurança.

3 - falha suave significa que o sistema continua operando, mas provendo índices de desempenho degradados ou capacidades funcionais reduzidas, até a remoção da falha.

4 - vital significa que o sistema garante que algumas funcionalidades e respectivos desempenhos são cumpridas durante um determinado intervalo de tempo, depois do qual o sistema pode ser totalmente desabilitado em um estado seguro.

Um sistema é tolerante a falha quando ele pode prevenir que falhas causem faltas [Abb90].

Portanto, para que sistemas reais se tornem confiáveis, tolerância a falhas é necessário para garantir que o sistema possa continuar a funcionar e prover serviços, mesmo na presença de falhas. Para diferentes tipos de falhas, diferentes técnicas de tolerância a falhas podem ser adotadas, dentre as mais importantes se incluem:

1. *Blocos de Recuperação (Recovery Blocks)*: é uma técnica tolerante a falhas que encapsula cada função crítica de um processo em um bloco de recuperação [BJR+87]. Um bloco de recuperação consiste de três partes: uma rotina primária que executa a função crítica; um teste de aceitação que decide se os resultados providos pelo bloco primário são aceitáveis e; uma ou mais rotinas de cópia (*backup*) que executam as mesmas funções das rotinas primárias que são gatilhadas se os resultados providos pelas rotinas primárias não são satisfatórios;

2. *Programação N-Versões*: é uma técnica tolerante a falhas que é adequada para falhas de *software*. Neste caso, diferentes versões dos módulos de *software* são projetados e implementados [AL86]. As diferentes versões são executadas em paralelo e um mecanismo de votação é utilizado para avaliar os resultados e apenas um resultado é selecionado;

3. *Recuperação Regressiva com Pontos de Verificação*: é uma técnica tolerante a falhas tanto para *hardware* quanto para *software* [Svo84, TH86]. A ideia é salvar, periodicamente, os estados de computação (pontos de verificação) em algum dispositivo confiável. Os pontos de verificação são usados para reiniciar o processo no caso de falhas, isto é, o processo é reiniciado a partir do último ponto de verificação. No caso de ambiente distribuído, é necessária uma técnica de manutenção de consistência em adição ao mecanismo de recuperação para evitar um número incontrolável de retornos. Este efeito pode ocorrer se os pontos de verificação entre os processos comunicantes são inconsistentes e é conhecido como *Efeito Dominó*;

4. *Conversação*: é uma forma restrita de recuperação regressiva com pontos de verificação [SGCT88]. Nesta técnica, um conjunto de processos comunicantes pode ter pontos de verificação independentes um dos outros. No entanto, uma vez começada a conversação, nenhum ponto de verificação é permitido e todos os processos são obrigados a deixarem a conversação em conjunto. Esta técnica previne a ocorrência do *efeito dominó*;

5. *Troca*: é uma forma restrita de conversação [AK83]. Todos os processos tomam seus pontos de verificação ao mesmo tempo antes de entrarem na troca. Os processos são restringidos a saírem da troca em conjunto. Este tipo de técnica também previne o *efeito dominó.*

A recuperação regressiva com pontos de verificação, troca e conversação são classificadas como técnicas de tolerância a falhas para trás. Os blocos de recuperação e programação N-Versões são ditas técnicas de tolerância a falhas para frente.

As técnicas de tolerância a falhas para trás, em muitos casos, não são apropriadas para ambientes em tempo real. A aplicabilidade de técnicas de tolerância a falhas em sistemas em tempo real deve levar em conta a latência de recuperação que pode causar a violação das restrições de tempo. Por latência de recuperação, entende-se o tempo decorrido entre a detecção do erro até o fim da recuperação do sistema. Portanto, as técnicas de de tolerância a falhas para frente são mais apropriadas para sistemas em tempo real. As abordagens de tolerância a falhas mais comuns em sistemas em tempo real são: a abordagem *primário/secundário* e a abordagem de *redundância modular* [Mos93].

A abordagem *primário/secundário* é equivalente à técnica de blocos de recuperação. Um módulo primário é executado e, em caso de falhas, o módulo secundário é gatilhado. A abordagem de *redundância modular* é adequada para sistemas em tempo real devido à baixa latência de detecção e recuperação. O principal problema com esta abordagem é a falta de flexibilidade na especificação de requisitos de tolerância a falhas.

Dentre os formalismos utilizados para modelar e analizar sistemas com propriedades de segurança e tolerância a falhas, redes de Petri têm se mostrado bastante eficientes [PdFC94, LS87, LS85, dFPM93, SGCT88, BG91, Lev86, dLSA92]. Leveson [LS87, Lev86], por exemplo, aplica redes de Petri temporizadas [MF76] para a modelagem e análise de

sistemas em tempo real críticos seguros, tais como sistemas de transporte metroviário e ferroviário.

Além de ser uma ferramenta formal, as redes de Petri podem combinar componentes de *hardware*, *software* e humanos em um único modelo. Desta forma, é possível determinar, por exemplo, os efeitos da falha de um componente em outros. Pode-se também utilizar o modelo para definir as rotinas de segurança e tolerância a falhas.

Para sistemas complexos, deve-se considerar uma abordagem modular ou composicional que permita a introdução sistemática de propriedades de tolerância a falhas no projeto de um componente.

Deve-se ainda considerar quais propriedades de tolerância a falhas um sistema deve possuir. Durante a fase de projeto, um sistema pode ser tornado *auto-protegido* ou *auto-verificante*. Um sistema é dito auto-protegido se ele pode se proteger de interferência em seu comportamento, devido a falhas em seus componentes. Se ocorrer um erro em um componente, o mesmo deve falhar ou pelo menos tratar seu erro, desta forma evitando interferência no comportamento de outros componentes que interagem com ele. Quando ocorre uma ação auto-protetora, o componente que detecta o erro deve recusar interferência em seu comportamento. Este tipo de ação pode ser implementada através do envio de uma mensagem a outros componentes, indicando que a ação não será executada como solicitado, caso contrário um estado de erro será alcançado. Um componente é dito auto-verificante se ele pode detectar e sinalizar seu erro. Desta forma, sua falha não interferirá no comportamento de outros componentes que fazem uso de seus serviços ou em componentes com os quais o componente faltoso interage. Para tanto o componente faltoso deve gerar sinais de erro explícitos toda vez que um erro é detectado e não pode ser corrigido.

5.3 Introduzindo Propriedades de Segurança

Quando as falhas em um sistema podem provocar danos catastróficos, o problema de segurança adquire importância significativa. No contexto desta abordagem o termo sistema é empregado para designar um ambiente computacional destinado às funções de monitoração e controle em ambientes de automação industrial. Mais especificamente, o

interesse reside naqueles ambientes que podem ser caracterizados como Sistemas a Eventos Discretos.

A necessidade de analizar a concepção de um sistema para identificar a possibilidade de que estados indesejáveis sejam alcançados é de grande importância. A utilização de técnicas formais permite a verificação da validade lógica da concepção, e provê mecanismos para a identificação de estados de risco.

De modo geral o emprego de técnicas formais tem sido feito de forma específica para cada problema, sem uma abordagem genérica e sistematizada. A investigação de formalismos que permitam a sistematização da análise do problema de segurança, principalmente para SEDs, é um tema atual e relevante.

Nesta Seção mostramos como a metodologia de síntese apresentada permite embutir propriedades de segurança em supervisores de sistemas a eventos discretos. Note que nesta abordagem, já na etapa de concepção do sistema, devem ser detectados aqueles estados que podem mas não devem ser alcançados pelo sistema. Assim, é necessária a construção de um supervisor que garanta a execução da especificação do sistema, bem como sua integridade. Utiliza-se como exemplo o sistema de cruzamento de uma ferrovia com uma rodovia.

5.3.1 Exemplo de um Cruzamento Seguro de uma Ferrovia com uma Rodovia

Nesta Seção apresentamos o exemplo clássico do cruzamento de um trem com uma rodovia. Nesse caso, a segurança é o fator mais importante no sistema, visto que vidas humanas estão em jogo. Em hipótese alguma o trem deve cruzar uma rodovia sem que a *cancela de segurança* esteja abaixada.

Não estamos considerando aqui falhas humanas, tais como as de um maquinista, visto que estamos interessados no controle de sistemas automatizados, tais como sistemas de metrô ou de manufatura.

Na Figura 5.1(a) é apresentado o modelo de rede de Petri de um trem cruzando uma rodovia. Veja que a marcação inicial ($M_0 = [\ 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0\ ]^T$) mostra que o trem está se aproximando do cruzamento e a *cancela de segurança* está levantada, ou seja, os veículos continuam a cruzar a ferrovia. Quando o trem está a uma certa proximidade

do cruzamento, um sinal tem que ser enviado (disparo da transição t_1) para que a *cancela de segurança*, na rodovia, seja abaixada para proteção dos veículos.

Seja $\Sigma = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$ o conjunto de eventos associados aos disparos das transições, conforme mostrado na Figura 5.1(a), onde $\Sigma_c = \{\alpha_2, \alpha_4, \alpha_5\}$ e $\Sigma_u = \{\alpha_1, \alpha_3\}$, ou seja, a ocorrência dos eventos α_2 (trem inicia cruzamento), α_4 (*cancela de segurança é abaixada*) e α_5 (*cancela de segurança é levantada*) é controlável. Especificamente para o caso do evento α_2, o mesmo tem que ser controlável, pois sua ocorrência em certas condições pode acarretar um desastre.

Executando o Algoritmo *AMArA* obtêm-se a árvore de alcançabilidade modificada apresentada na Figura 5.1(b).

Pode-se detectar duas marcações importantes na rede da Figura 5.1(a). São elas a marcação final desejada (*estado marcado*), em que os lugares p_4 e p_5 possuem fichas ($M_5 = [\ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ ]^T$); e a outra marcação é aquela em que os lugares p_3 e p_5 estão marcados (estado *não_permitido*) ($M' = [\ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ ]^T$).

A primeira marcação significa que o trem cruzou a rodovia e a *cancela* foi novamente levantada. A segunda marcação significa que o trem está cruzando a rodovia e a *cancela* não foi abaixada. Logicamennte, este é um estado que não se deseja alcançar.

As especificações desejáveis, para garantir a segurança e a realização das tarefas do sistema, são assim definidas:

1. O sistema nunca deve alcançar o estado representado pela marcação

$$M' = [\ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ ]^T;$$

2. O sistema deve sempre alcançar o estado representado pela marcação

$$M_5 = [\ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ ]^T, \text{ ou seja, } M_5 \in Q_m.$$

Note que a seqüência $\alpha_1\alpha_2$, (veja Figura 5.1(b)), leva o sistema a um estado *não_permitido*. Executando o Algoritmo *ACGS*, chega-se aos estados e respectivos eventos a serem desabilitados para que o estado ou marcação M', apresentada acima, nunca seja alcançada.

Após a execução do *ACGS* chega-se à seguinte *lista_perigo*:

$$M_1 \longrightarrow t_2(\alpha_2),$$

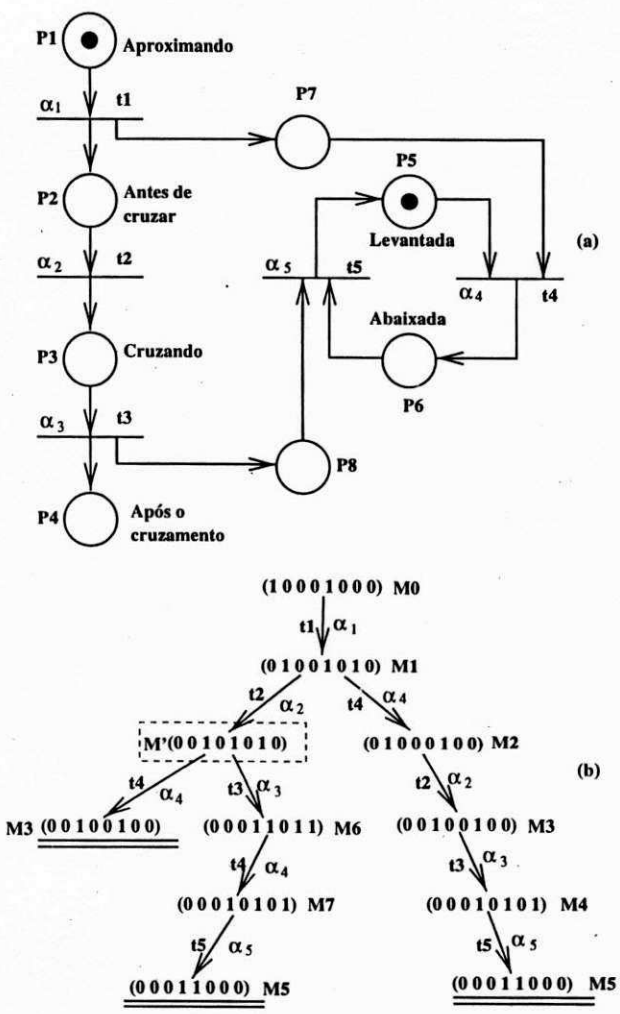


Figura 5.1: (a) Modelo da aproximação de um trem para cruzar uma rodovia; (b) Árvore de alcançabilidade modificada do sistema.

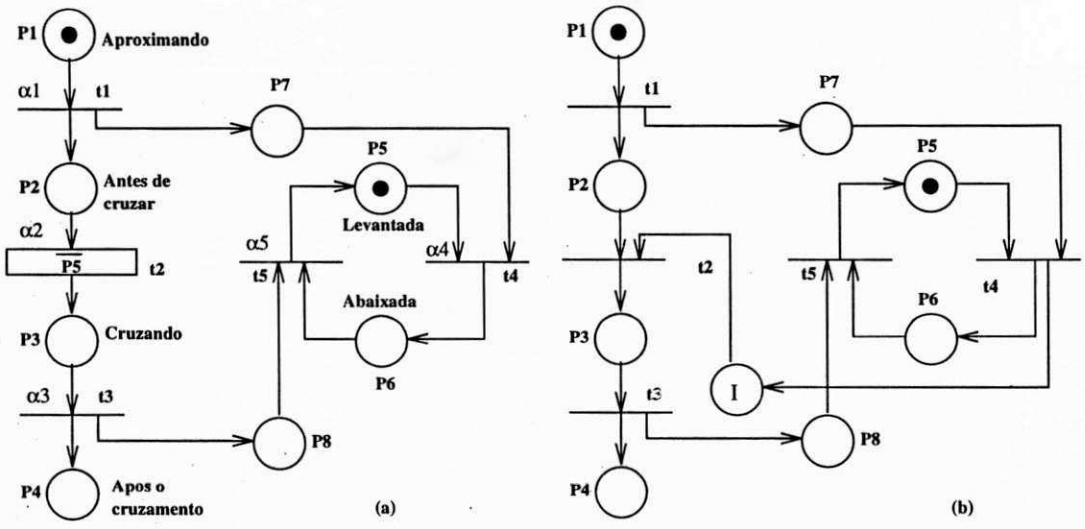


Figura 5.2: (a) *RPFHT* supervisora para garantir a segurança do funcionamento do sistema; (b) Rede de Petri equivalente.

ou seja, é necessário desabilitar o evento α_2 sempre que o sistema se encontrar na marcação M_1. Assim, se a uma certa distância do cruzamento um sinal de que a *cancela de segurança* foi abaixada não for recebido pelo sistema de controle, o mesmo tem que tomar providências antes do cruzamento do trem.

Utilizando-se um sistema computacional como controle, uma *RPFHT* supervisora (com a mesma estrutura da rede que modela o sistema) pode ser executada sincronamente com o sistema, onde a transição t_2 (evento α_2) sempre estará desabilitada na marcação ou estado M_1, ou mais especificamente, t_2 nunca irá disparar para qualquer marcação em que $M(p_5) = 1$, significando que o trem não poderá nunca cruzar a rodovia se a *cancela de segurança* não estiver abaixada. Veja a Figura 5.2(a).

Note que a função de habilitação da transição t_2 da *RPFHT* supervisora garante que o evento α_2 nunca ocorrerá antes que o evento α_4 ocorra. Nesse caso, a *RPFHT* supervisora é equivalente à rede de Petri que modela o sistema, acrescida de um *intertravamento*, impondo que α_4 sempre ocorrerá antes de α_2. Veja lugar I da rede de Petri apresentada na Figura 5.2(b).

5.4 Introduzindo Propriedades de Tolerância a Falhas

Como já dito neste Capítulo, a maioria das falhas que ocorrem em sistemas a eventos discretos estão relacionadas à sincronização e erros de desempenho que se manifestam como falhas transientes do sistema.

Diferentes tipos de falhas, caracterizadas como dependentes e não dependentes do tempo, podem ocorrer. Dentre as falhas não dependentes do tempo, podem ser citadas:

- a não ocorrência de um evento;
- a ocorrência de um evento não desejável;
- a ocorrência simultânea de dois eventos incompatíveis.

Considerando uma metodologia formal para a síntese de supervisores de sistemas a eventos discretos, é necessário a introdução de propriedades de tolerância a falhas nos supervisores de tal forma que os mesmos sejam capazes de detectar e possivelmente recuperar o sistema de uma falha através de sua ação de controle.

Considere a ocorrência de falhas tais como: uma seqüência de eventos que leva o sistema a um bloqueio ou sobrecarga; uma seqüência de eventos que simplesmente degrada o desempenho do sistema, mas que mesmo assim leva o sistema a realizar a tarefa especificada.

Nesses casos, é necessário que o supervisor reconheça a falha e, através de sua ação de controle, conduza o sistema a um estado desejável, se possível.

Utilizando a metodologia de síntese de supervisores apresentada e a técnica de tolerância a falhas *Recuperação Regressiva com Pontos de Verificação*, apresentamos a seguir a síntese de um supervisor com propriedades de tolerância a falhas para o sistema de manufatura modelado pela rede de Petri apresentada na Figura 4.14 e, por comodidade, reapresentada na Figura 5.3. O objetivo é construir um supervisor que previna o sistema de alcançar um estado indesejável, mas que, caso ocorra uma falha transitória no sistema, o supervisor seja capaz de realizar as ações de controle necessárias para que o sistema volte a atuar dentro dos limites do comportamento desejável, anteriormente definido. O

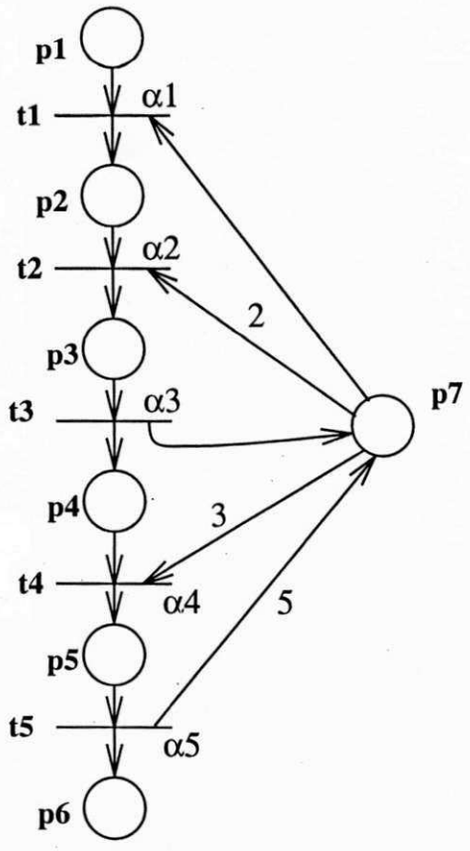


Figura 5.3: Sistema com recursos compartilhados.

supervisor deve ser capaz de retornar a um estado seguro, previamente definido, após detectada a falha e, após solucionado o problema no sistema, o mesmo deve também ser reinicializado a partir de um estado equivalente ao do supervisor.

5.4.1 Exemplo de um Supervisor com Propriedades de Tolerância a Falhas

Seja o sistema modelado pela rede de Petri apresentada na Figura 5.3, com n usuários ($n \geq 1$), que solicitam recursos para a realização de uma tarefa específica. Os recursos são indistinguíveis e o sistema possui um número limitado k de recursos ($k \geq 5$), conforme apresentado na Seção 4.2.4.

Considere a mesma especificação desejável para sistema, apresentada na Seção 4.2.4,

ou seja: Todos os usuários têm que ser atendidos. Como cada usúario é representado por uma ficha no lugar p_1 na marcação inicial, então uma tarefa completada significa um número de fichas no lugar p_6, na marcação final, igual ao número de fichas do lugar p_1 na marcação inicial, ou seja, $M = [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 6\ \ 8\ ]^T \in Q_m$.

Aplicando a metodologia de síntese do supervisor para um sistema com os mesmos 6 (seis) usuários e 8 (oito) recursos, executa-se o Algoritmo *AMArA* e se obtêm todas as marcações, ou estados, possíveis para o sistema não controlado. Dados os estados possíveis e a especificação desejável, executa-se o Algoritmo *ACGS*. Obtém-se, então, a *lista_perigo*, apresentada na Figura 4.15, ou seja, todos os estados e respectivos eventos a serem desabilitados para que a especificação possível seja realizada.

A partir da *lista_perigo*, obtêm-se as funções de habilitação das transições da *RPFHT* supervisora, que possui a mesma estrutura da rede que modela o sistema, veja Figura 4.16 na Seção 4.2.4.

Note que os eventos α_1 e α_2, modelados respectivamente pelos disparos das transições t_1 e t_2, não podem ocorrer a partir dos respectivos estados apresentados na *lista_perigo*. Se qualquer dos dois eventos ocorrer, então o sistema fatalmente será levado ao bloqueio, ou seja, estes são os pontos críticos do sistema. Sendo assim, estes pontos (estados) poderão ser utilizados como pontos de verificação do sistema, ou seja, quando o sistema se encontrar em qualquer uma das marcações da *lista_perigo* e consequentemente o supervisor, pode ocorrer uma falha (ocorrência de um evento desabilitado pelo supervisor), o que levaria o sistema a um bloqueio.

Especificamente, para o exemplo apresentado, se por uma falha no sistema, ocorre o evento associado a t_1 (t_2) quando o mesmo está desabilitado pelo supervisor, é necessário que o supervisor e o sistema retornem aos respectivos estados anteriores à ocorrência do evento, ou a um outro estado que satisfaça à especificação, sob pena dos recursos se tornarem indisponíveis. Por exemplo, se o sistema se encontrar na marcação ou estado $M = [\ 2\ \ 4\ \ 0\ \ 0\ \ 0\ \ 0\ \ 4\ ]^T$, pertencente à *lista_perigo*, e a transição t_1, por uma falha transitória, é disparada, então a marcação $M' = [\ 1\ \ 5\ \ 0\ \ 0\ \ 0\ \ 0\ \ 3\ ]^T$ será alcançada. Note que nesta marcação, o sistema estará bloqueado, pois as funções φ_1 e φ_2 não permitem o disparo de t_1 e t_2, respectivamente. De qualquer forma, mesmo sem o controle, a partir dessa marcação a rede encontrará um estado de bloqueio qualquer que

seja a seqüência de transições disparada. Assim, se o sistema alcançar a marcação M', alguma ação tem que ser tomada pelo supervisor para que os recursos fiquem novamente disponíveis. Uma atitude é fazer com que o usuário que solicitou indevidamente o recurso possa retornar à fila de espera ou ser descartado, tornando os recursos novamente disponíveis para os usuários que têm permissão para tal. Para tanto é necessário que o sistema possua mecanismos para se recuperar de tal falha. Por exemplo, em um sistema de manufatura, um robô poderia retirar a peça da linha de produção e tornar os recursos solicitados novamente disponíveis, sob a ação de controle do supervisor.

Para que o supervisor e o sistema possam retornar aos respectivos estados anteriores à ocorrência do evento indesejável (*recuperação regressiva*), acrescentam-se à rede que modela o sistema e ao supervisor blocos de recuperação de falhas, da seguinte forma: para cada transição t_i associada a um evento indesejável, acrescenta-se à rede uma outra transição t_{fi}, tal que

$$I(t_i) = O(t_{fi}) \text{ e } O(t_i) = I(t_{fi})$$

Para se encontrar as funções de habilitação das transições t_{fi}, procede-se da seguinte forma:

1. Dada a *lista_perigo*, encontra-se a *lista_recupera* disparando-se as transições da *lista_perigo* a partir dos respectivos estados associados, ou seja, a *lista_recupera* é formada pelos estados indesejáveis a partir do disparo das transições na *lista_perigo*,
2. Dada a *lista_recupera*, encontram-se as funções associadas às transições t_{fi}.

Assim, essas transições só irão disparar se um estado indesejável for alcançado. Disparando essas transições retorna-se ao estado anterior, ou ponto de verificação. A *RPFHT* supervisora com propriedades de tolerância a falhas, para o exemplo dado, é apresentada na Figura 5.4

Veja que, pelas funções de habilitação das transições t_{f1} e t_{f2}, os eventos associados a estas transições só ocorrerão se houver uma falha no sistema, ou seja, um estado não desejável for alcançado. Por exemplo, na marcação $M = [\ 2\ \ 4\ \ 0\ \ 0\ \ 0\ \ 0\ \ 4\ ]^T$, citada acima, as transições t_{f1} e t_{f2} estão ambas desabilitadas pelas funções φ_1 e φ_2,

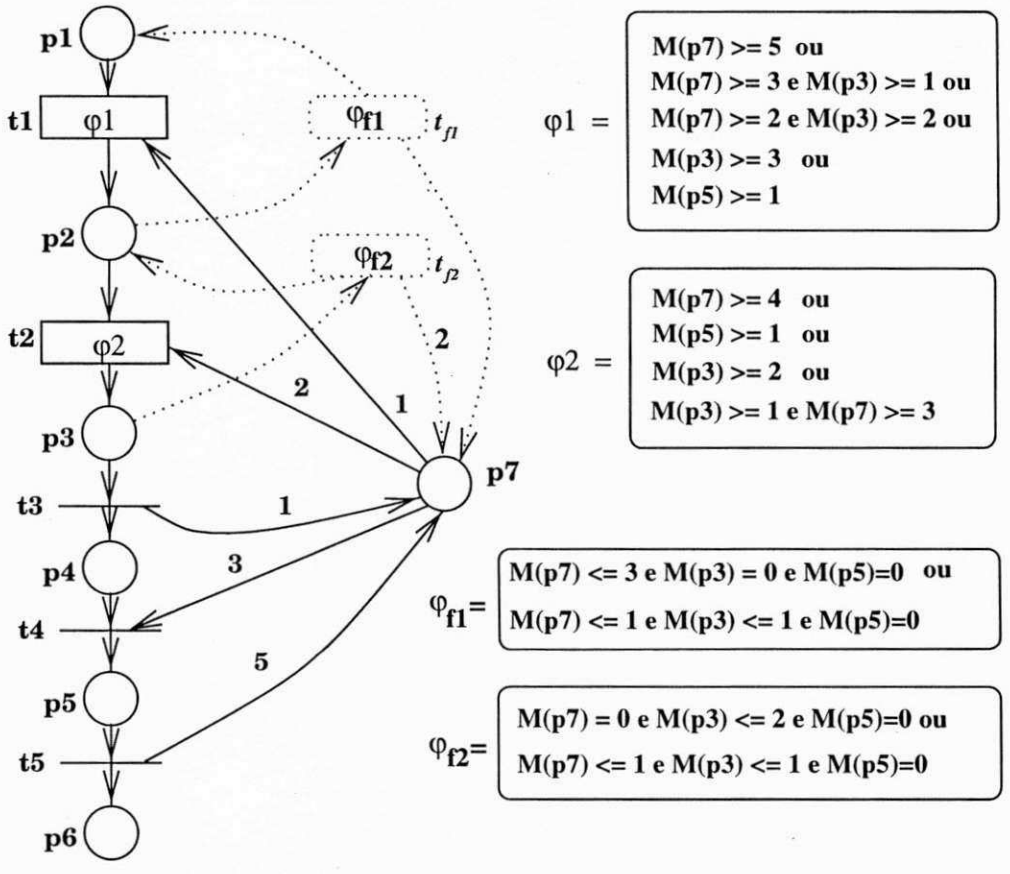


Figura 5.4: *RPFHT* supervisora com propriedades de tolerância a falhas.

respectivamente. Se o sistema alcança a marcação $M' = [\ 1 \quad 5 \quad 0 \quad 0 \quad 0 \quad 0 \quad 3\ ]^T$, por uma falha no mesmo, então t_{f1} estará habilitada visto que $M(p_7) \leq 3$, $M(p_3) = 0$ e $M(p_5) = 0$. Veja função φ_{f1} na Figura 5.4.

5.5 Síntese de Supervisores Utilizando sistemas de *G-Net*s e *RPTN*s

O modelo de redes de Petri com funções de habilitação de transições (*RPFHT*s), bem como os Algoritmos *AMArA* e *ACGS* foram introduzidos na sistematização de uma solução para o problema de síntese de supervisores para sistemas a eventos discretos (SEDs), sem modificações *ad-hoc* na estrutura da rede que modela o sistema. Mas apesar do poderoso mecanismo de estruturação disponivel na teoria de redes de Petri (RPs), as mesmas sofrem da falta de modularidade, colocando-se então o problema da explosão de estados quando da verificação e análise de grandes sistemas modelados por redes de Petri. Assim. na especificação, análise, verificação ou síntese de supervisores de sistemas mais complexos (em relação ao seu tamanho ou número de componenetes do mesmo), é desejável adotar uma metodologia composicional para que se possa verificar propriedades locais dos módulos individuais, bem como verificar o correto comportamento do sistema modelado, na interação entre os seus componentes. Com o objetivo de aplicar esta metodologia para sistemas reais, foi introduzida, também neste capítulo, uma abordagem modular baseada nos sistemas de *G-Net* para a análise e síntese de supervisores. Desta forma, a construção do supervisor, para sistemas mais complexos, é simplificada porque é possível restringir a análise, durante a fase de síntese do supervisor, aos módulos ou *G-Nets* de interesse. Assim, mesmo utilizando um algoritmo de construção da árvore de alcançabilidade, em vez de construir a árvore de alcançabilidade de todo o sistema, é necessário construir somente as árvores de alcançabilidade dos módulos que serão efetivamente controlados. Nessa abordagem, as redes de Petri coloridas (*RPC*s) são utilizadas na estrutura interna de cada módulo ou *G-Net*.

Utilizando esta abordagem de síntese de supervisores para sistemas em tempo real, é necessário incorporar ao supervisor as restrições de tempo impostas ao sistema. Assim, o sistema pode ser controlado levando em consideração tanto restrições de valor quanto

restrições de tempo. Para tanto, é preciso que o supervisor seja capaz de reconhecer os limites de tempo e o comportamento lógico do sistema. Desta forma, se uma falha (no comportamento lógico ou temporal) no sistema ocorre, a mesma pode ser detectada pelo supervisor, que através de sua ação de controle tenta recuperar o sistema.

Nesta abordagem, o supervisor e o sistema são modelados por *G-Nets*. A estrutura interna de cada *G-Net* do sistema é modelada por uma nova classe de redes de Petri derivada das redes de Petri coloridas e redes de Petri com temporização nebulosa, denominada *Redes de Petri Coloridas com Temporização Nebulosa* (*RPC/TN*). A estrutura interna de cada *G-Net* do supervisor é modelada por uma nova classe de redes de Petri derivada das redes de Petri coloridas, redes de Petri com temporização nebulosa e redes de Petri com funções de habilitação de transições, denominada *Redes de Petri Coloridas com Temporização Nebulosa e Funções de Habilitação de Transições* (*RPC/TN/FHT*).

Para a utilização das *RPC/TN*s na estrutura interna de cada *G-Net*, o método de decomposição para o *isp* (*instantiated switching place*) e o *GSP* (*generic switching place*) de uma *G-Net*, definido por Perkusich [Per94], é integrado com os intervalos nebulosos das *RPTN*s, como definido por Figueiredo [dFP95].

Esta abordagem é introduzida através do exemplo da célula de manufatura apresentada na Figura 4.21.

5.5.1 Exemplo de um Supervisor com Propriedades de Tolerância a Falhas dependentes do tempo

Considere a mesma célula de manufatura apresentada na Figura 4.21. O sistema de *G-Nets* que modela a célula é apresentado na Figura 5.5, não levando em consideração o que está entre as linhas tracejadas. A estrutura interna de cada *G-Net* é modelada por uma *RPC/TN*.

Como no exemplo anterior, para que uma peça seja fabricada, é necessário que uma parte processada em $\mathbf{M}P1$ e uma parte processada em $\mathbf{M}P2$ sejam montadas no centro de montagem $\mathbf{C}M$. As peças processadas são transportadas para outra célula por um agente externo cuja ação não está modelada.

Neste exemplo, os intervalos de habilitação nebulosos E (onde e_1 e e_2 são números nebulosos) representam o tempo necessário para que uma dada tarefa seja completada,

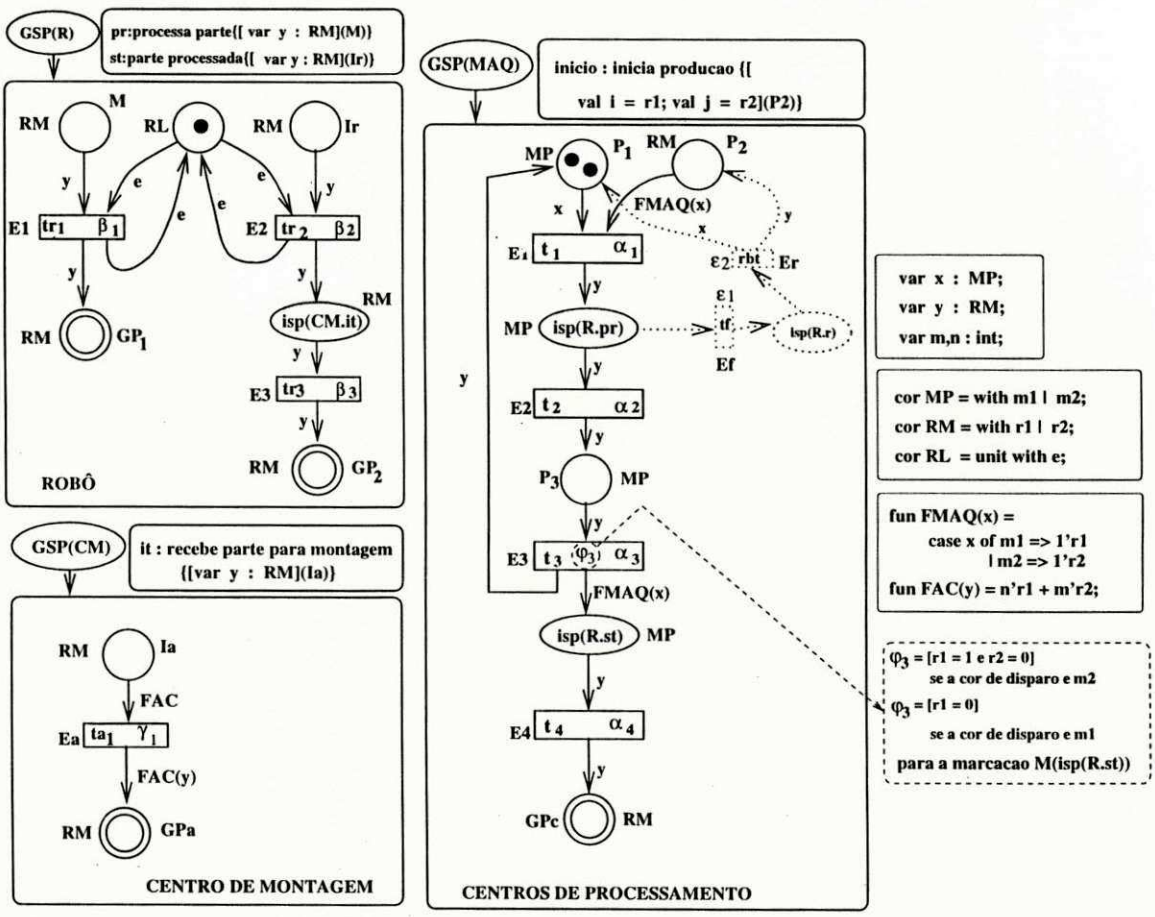


Figura 5.5: Sistema de *G-Nets* com restrições de tempo modelando uma célula de manufatura e o respectivo sistema supervisor da célula.

por exemplo o tempo de processamento de uma parte em uma máquina ou tempo de montagem das partes no centro de montagens. O intervalo de disparo $D([d_1, d_2])$ de todas as transições é $[0, 0]$ (disparo instantâneo). Baseados nas funções de temporização nebulosas associadas às transições e nos intervalos de habilitação e disparo associados às transições, é possível determinar os tempos máximo e mínimo para que uma determinada tarefa seja completada. Mais ainda, baseados nos intervalos de habilitação e disparo é possível detectar violações nas restrições de tempo impostas à execução de uma determinada tarefa por um ou mais componentes do sistema. Baseados nestas características, é possível sintetizar um supervisor que seja capaz de detectar falhas e, possivelmente, recuperar o sistema das falhas detectadas.

Seja $\Sigma = \Sigma_p \cup \Sigma_r \cup \Sigma_a$, onde $\Sigma_p = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \varepsilon_1, \varepsilon_2\}$, $\Sigma_r = \{\beta_1, \beta_2, \beta_3\}$ e $\Sigma_a = \{\gamma_1\}$. Σ_p, Σ_r e Σ_a são os conjuntos de eventos associados aos disparos das transições das *G-Nets GN(MAQ), GN(R)* E *GN(CM)* respectivamente. $\Sigma_c = \{\alpha_1, \alpha_3\}$ é o conjunto de eventos controláveis e $\Sigma_u = \{\alpha_2, \alpha_4, \beta_1, \beta_2, \beta_3, \gamma_1, \varepsilon_1, \varepsilon_2\}$ é o conjunto de eventos não controláveis.

Considere que a especificação desejável para o sistema é dada por:

- Uma peça é montada em $\mathbf{C}M$;
- Para a montagem de uma peça são necessárias uma parte processada em $\mathbf{M}P1$ e uma parte processada em $\mathbf{M}P2$;
- A parte processada em $\mathbf{M}P1$ tem que ser enviada primeiro para $\mathbf{C}M$;
- $\mathbf{R}B$ coloca em $\mathbf{C}M$ uma parte processada em $\mathbf{M}P1$, e logo após, uma parte processada em $\mathbf{M}P2$;
- Após montada uma peça, $\mathbf{R}B$ coloca novamente partes de $\mathbf{M}P1$ e $\mathbf{M}P2$ em $\mathbf{C}M$ para serem montadas.

Considerando que somente os eventos α_1 e α_3 (transições t_1 e t_3 de *GN(MAQ)*, respectivamente) são eventos controláveis, a atenção pode ser voltada somente para a *G-Net GN(MAQ)*, módulo no qual a ação de controle será efetivamente realizada, ou seja, executa-se o *AMArA* e o *ACGS* somente para este módulo.

Por outro lado dois estados críticos, em relação às restrições de tempo impostas ao sistema, se apresentam quando fichas são colocadas ou no lugar *isp(R.pr)* ou no lugar

isp(R.st). Tomando como exemplo o lugar *isp(R.pr)*, as transições e arcos pontilhados, no modelo apresentado na Figura 5.5, representam a técnica de recuperação regressiva temporizada. Nesse caso, uma ficha no lugar *isp(R.pr)* de *GN(MAQ)*, indica que o módulo *GN(R)* foi invocado. A transição t_2 possui um tempo de habilitação nulo $E_2 = [0,0]$ enquanto a transição t_f possui um tempo de habilitação $E_f = [e_{f1}, e_{f2}]$, onde e_{f1} representa o tempo máximo para que a tarefa, para a qual *GN(R)* foi invocada, seja executada, ou seja, se *GN(R)* executar a tarefa para a qual foi invocada dentro dos limites de tempo especificados, uma ficha retorna ao lugar *isp(R.pr)* com *status* = *após*. Nesse caso t_2 estará habilitada instataneamente e poderá disparar. Se, por algum erro no módulo *GN(R)*, o tempo e_{f1} for excedido, então a transição t_f dispara colocando uma ficha no lugar *isp(R.r)*, significando que *GN(R)* foi invocada com o método r (método, não modelado, de recuperação de falhas em *GN(R)*) para recuperar *GN(R)* de uma falha. Quando a transição *rbt* dispara, o supervisor retorna ao estado anterior seguro. Neste caso, isto significa que uma ficha de cor r_i é colocada no lugar p_2 e uma ficha de cor m_i é colocada no lugar p_1 de *GN(MAQ)*. Uma falha como esta causa um retardo na execução de *GN(MAQ)*, então é necessário limitar o número de vezes que esta técnica de recuperação pode ser usada.

Note que em relação às *G-Nets GN(R)* e *GN(CM)*, o interesse reside em determinar somente os tempos máximos de execução dos métodos com os quais as respectivas *G-Nets* foram invocadas. Assim, as funções de temporização nebulosa (*FTF*s) iniciais associadas às fichas da *G-Net* invocada são inicializadas com valor $[0,0]$. O tempo de execução da rede invocada é representado pela *FTF* associada à ficha em seu lugar alvo.

Desta forma, executando o *AMArA*, o conjunto de todos os estados fisicamente possíveis da célula de manufatura é obtido.

Executando o *ACGS*, encontra-se a especificação desejável ou a máxima especificação possível. Em outras palavras, encontra-se a *lista_perigo*, como mostrado na Tabela 5.1. Note que esta Tabela é equivalente à Tabela 4.1. visto que os estados possíveis de serem alcançados pelos dois modelos (com e sem restrições de tempo), para o sistema não controlado, neste caso, são equivalentes. O supervisor é apresentado na Figura 5.5, cuja função de habilitação φ_3, associada a t_3, restringe o disparo de t_3 para que a tarefa especificada seja realizada, bem como as restrições de tempo impostas também ao controlador garantem tanto a realização da tarefa especificada nos limites de tempo definidos, como

possibilitam ao controlador recuperar o sistema de possíveis falhas temporárias.

Tabela 5.1: Tabela de marcações e respectivos eventos a serem desabilitados

Σ_c	Marcações					
	P1	P2	ispRpr	P3	ispRst	GPc
	m2	2r2	0	m1	r1	0
	0	r2	m2	m1	r1	0
	0	r2	0	m1,m2	r1	0
t_3	0	0	m2	m1	r1,r2	0
α_3	m2	r2	0	m1	r1	r2
	0	0	m2	m1	r1	r2
	0	0	0	m1,m2	r1	r2
	m2	0	0	m1	r1,r2	r2
	m2	0	0	m1	r1	2r2
φ	φ_3=[r1=0] para a cor de disparo = m1					

Σ_c	Marcações					
	P1	P2	ispRpr	P3	ispRst	GPc
	m1	2r1,r2	0	m2	0	0
	0	r1,r2	m1	m2	0	0
	0	r1,r2	0	m1,m2	0	0
t_3	m1	2r1	0	m2	0	r2
α_3	0	r1	m1	m2	0	r2
	m1	r1	0	m2	r2	r1
	0	0	0	m1,m2	r1,r2	0
	m1	r1	0	m2	r2	r1
	0	0	0	m1,m2	r1,r2	0
	0	0	m1	m2	r2	r1
	0	0	0	m1,m2	r2	r1
	m1	0	0	m2	r1,r2	r1
	m1	0	0	m2	r2	2r1
φ	φ_3=[r1=1 and r2=0] para a cor de disparo = m2					

Capítulo 6

Discussão e Conclusões

6.1 Sumário

Nos capítulos anteriores introduziu-se uma nova abordagem para a solução do problema de síntese de supervisores de sistemas a eventos discretos, utilizando teoria de controle supervisório e redes de Petri.

Esta abordagem permite que, dados um sistema modelado por uma rede de Petri e uma especificação funcional desejável, se construa um supervisor, modelado por uma rede de Petri com funções de habilitação de transições. A rede supervisora possui a mesma estrutura da rede que modela o sistema. Quando em operação, o sistema e o supervisor são inicializados em estados iniciais equivalentes e executados sincronamente. Assim, sistema e supervisor estarão sempre em estados equivalentes, cujos respectivos eventos de saída só poderão ocorrer se estiverem habilitados pelo supervisor.

Introduziu-se uma metodologia de transformação de uma *RPFHT* em uma rede de Petri equivalente, tornando possível a análise por métodos formais bem conhecidos.

Introduziu-se também uma abordagem modular para a síntese de supervisores, utilizando sistemas de *G-Nets*. A motivação para a proposição desta abordagem é a necessidade de melhor gerenciar o problema da explosão de estados quando analizando sistemas mais complexos. Com esta abordagem, é possível construir supervisores independentes para cada *G-Net* do sistema de *G-Nets*, dado que as mesmas são fracamente acopladas e possuem um mecanismo de comunicação bem definido. Assim, a ação de um supervisor

modificando a dinâmica interna de uma *G-Net*, será transparente para as outras *G-Nets* do sistema.

Note que, para a construção do supervisor, o espaço de estados é analizado separadamente para cada *G-Net* a ser controlada. Desta forma, modificações na especificação funcional de uma *G-Net*, ou seja modificações no supervisor, ficam restritas às *G-Nets* de interesse, evitando assim modificações no supervisor global.

Neste trabalho, as redes de Petri coloridas (modelo de Jensen [Jen92]) foram utilizadas na implementação da estrutura interna de cada *G-Net*, preservando as características de imutabilidade das interfaces e comunicação inter-*G-Nets*.

Na implementação do supervisor modular, introduzimos a classe de redes de Petri denominada *redes de Petri coloridas com funções de habilitação de transições*, que utilizadas na estrutura interna das *G-Nets*, permitem uma modelagem mais clara e sucinta de sistemas mais complexos.

Dado o interesse crescente nas possíveis soluções para os problemas de falhas em um sistema real, introduzimos uma sistemática de construção de supervisores com propriedades de tolerância a falhas, ou seja, o supervisor é capaz de detectar e avisar ao sistema que uma falha ocorreu. Ao mesmo tempo, o supervisor deve retornar a um estado anterior correto quando o sistema estiver novamente operacional, se possível a partir de um estado equivalente pois assim os mesmos poderão ser reinicializados daquele ponto.

Pensando na correção, do ponto de vista lógico e temporal, das tarefas a serem realizadas pelo sistema, integrou-se as redes de Petri com temporização nebulosa com as redes de Petri coloridas com funções de habilitação de transições, na implementação da estrutura interna das *G-Nets* supervisoras. Desta forma, é possível restringir a ocorrência de eventos indesejáveis utilizando tanto as funções de habilitação de transições como os intervalos nebulosos de habilitação e retardo associados à ocorrência dos eventos. Utilizando esta classe de redes de Petri, também é possível, ao supervisor, detectar falhas nos limites de tempo impostos ao sistema e, caso uma falha ocorra, o mesmo pode se reconfigurar em um estado anterior seguro para reinicialização.

6.2 Trabalhos Futuros

O trabalho desenvolvido nesta Tese não pretende ser auto-contido, pelo contrário, pretende-se que seja estendido em vários aspectos.

Um aspecto relevante, em relação às restrições de tempo do sistema, seria a introdução de conceitos tais como eventos iminentes, remotos, preempitivos e forçados, bem como a introdução de um relógio global, na tentativa da otimização da supervisão de sistemas a eventos discretos, baseados naqueles definidos por Ostroff e Wonham [OW90] e estendido por Ionescu e Lin [IL95], dentre outros, com modelos de lógica temporal.

Uma outra linha de pesquisa a se seguir, seria o desenvolvimento de algoritmos de síntese do supervisor baseados, não em um método exaustivo, como o apresentado, mas sim na estrutura da rede, de forma a definir as funções de habilitação, como o fazem Holloway e Krogh [HK90] e Boel et al. [BLNB95] para definir os lugares de controle para certas classes de redes de Petri controladas.

Na área de aplicação, pretende-se utilizar esta metodologia para aplicações industriais a partir da construção de protótipos tanto para controle centralizado como para controle distribuído utilizando sistemas de *G-Nets*.

Referências Bibliográficas

[Abb90] R. J. Abbott. Resourceful system for fault tolerance, reliability, and safety. *ACM Computing Surveys*, 22(1):35–68, Março 1990.

[AK83] T. Anderson and J. C. Knight. A framework for software fault tolerance in real-time systems. *IEEE Transactions on Software Engineering*, SE-9(3):355–364, Maio 1983.

[AL86] A. Avizienis and J. C. Laprie. Dependability computing: From concepts to design diversity. *IEEE Proceedings*, 74(5):67–80, Maio 1986.

[AW90] K. J. Astrom and B. Wittenmark. *Computer-Controlled Systems.* Prentice-Hall, 1990.

[AW95] K. J. Aström and B. Wittenmark. *Adaptive Control - Second Edition.* Addison Wesley, 1995.

[BC92] L. Bernardinello and F. De Cindio. A survey of basic models and modular net classes. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 304–351. Springer-Verlag, 1992.

[BG91] F. Belli and K-E. Grosspeitsch. Specification of fault-tolerant systems issues by predicate/transition nets and regular expressions-approach and case study. *IEEE Transactions on Software Engineering*, 17(6):513–526, Junho 1991.

[BHG+93] S. Balemi, G. J. Hoffman, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. Supervisory control of a rapid themal multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059, Julho 1993.

[BJR+87] K. P. Birmam, T. A. Joseph, T. Raeuchle, R. Abadi A. E. Koo, and S. Toueg. Checkpoint and rollback-recovery for distributed system. *IEEE Transactions on Software Engineering*, SE-13(1):23–31, Janeiro 1987.

[BLNB95] K. Boel, L.Ben-Naoun, and V. Van Breusegem. On forbidden state problems for a class of controlled petri nets. *IEEE Transactions on Automatic Control*, 40(10):1717–1731, october 1995.

[BLP95a] G. C. Barroso, A. M. N. Lima, and A. Perkusich. Análise de segurança baseada em controle supervisório e redes de petri. In *Proceedings of 6th Simpósio Brasileiro de Tolerância a Falhas*, pages 345–355, Canela, RS, BR, 1995.

[BLP95b] G. C. Barroso, A. M. N. Lima, and A. Perkusich. Modular synthesis of supervisors based on a petri net approach. In *Proceedings of 22th Simpósio Integrado de Software e Hardware, SEMISH'95*, pages 882–894, Canela, RS, BR, 1995.

[BLP95c] G. C. Barroso, A. M. N. Lima, and A. Perkusich. Transition enabling petri nets to supervisory control theory. In *Proceedings of IEEE/ECLA/IFIP International Conference on Architetures and Design Methods for Balanced Automation Systems*, pages 56–63, Vitoria, ES, BR, 1995.

[BLP96a] G. C. Barroso, A. M. N. Lima, and A. Perkusich. Petri nets with transition enabling functions in the supervision of discrete event systems. XI Congresso Brasileiro de Automática, 1996.

[BLP96b] G. C. Barroso, A. M. N. Lima, and A. Perkusich. Supervision of discrete event systems using petri nets and supervisory control. In *First International Workshop on Manufacturing and Petri Nets*, pages 77–96, Osaka, Japan, 1996.

[BLP96c] G. C. Barroso, A. M. N. Lima, and A. Perkusich. Synthesis of supervisors using petri nets. In *Proceedings of the 13th World Congress of IFAC - International Federation of Automatic Control - Volume J*, pages 449–454, San Francisco, CA, USA, 1996.

[Boo91] G. Booch. *Object Oriented Design: With Application.* Benjamin/Cummings, 1991.

[BRA83] G. W. BRAMS. *Reseaux de Petri: Théorie et Pratique, Tomo II.* Masson S.A., Paris, FR, 1983.

[BS92] M. Broy and T. Streicher. Modular functional modelling of petri nets with individual tokens. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 70–88. Springer-Verlag, 1992.

[BW94] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–341, Fevereiro 1994.

[CDQV85] G. Cohen, D. Dubois, J. P. Quadrat, and M. Viot. A linear system theoretic view of discrete event processes and its use for performance evaluation in manufacturing. *IEEE Transactions on Automatic Control*, 30(3):210–220, 1985.

[CH90] X-R. Cao and Y-C. Ho. Models of discrete event dynamic systems. *IEEE Control System Magazine*, 10(4):69–76, 1990.

[Che91] G. Chehaibar. Use of reentrant nets in modular analysis of coloured nets. In G. Rozenberg, editor, *Advances in Petri Nets 1991*, volume 524 of *Lecture Notes in Computer Science*, pages 58–77. Springer-Verlag, 1991.

[CP92] S. Christenssen and L. Petrucci. Towards a modular analysis of coloured petri nets. In K. Jansen, editor, *Application and Theory of Petri Nets 1992, 13th International Conference*, volume 616 of *Lecture Notes in Computer Science*, pages 113–133. Springer-Verlag, Sheffield, UK, 1992.

[CR90] C. G. Cassandras and J. P. Ramadge. Toward a control theory for discrete event systems. *IEEE Control System Magazine*, 10(4):66–68, 1990.

[DC91] Y. Deng and S. K. Chang. A framework for the modeling and prototyping of distributed information systems. *International Journal of Software Engineering and Knowledge Engineering*, 2(3):203–226, Setembro 1991.

[DC92] Y. Deng and S. K. Chang. Unifying multi-paradigms in software system design. In *Proceedings of the 4th International Conference on Software Engineering and Knowledge Engineering*, Capri, Itália, Junho 1992.

[DCdFP93] Y. Deng, S. K. Chang, J. C. A. de Figueiredo, and A. Perkusich. Integrating software engineering methods and petri nets for the specification and prototyping of complex software systems. In M. Ajmone Marsan, editor, *Lecture Notes in Computer Science*, volume 691 of *Application and Theory of Petri Nets*. Springer-Verlag, Chicago, USA, Junho 1993.

[dF94] J. C. A. de Figueiredo. *Redes de Petri com Temporização Nebulosa.* PhD thesis, Universidade Federal da Paraíba - Campus II, Campina Grande, PB - BR, 1994.

[dF95] J. C. A. de Figueiredo. Fuzzy time petri net: A generalized timing petri net extension for real-time systems. In *Proceedings of 6th International Fuzzy Systems Association Congress*, volume 1, pages 341–344, 1995.

[dFP95] J. C. A. de Figueiredo and A. Perkusich. Tratamento antecipado de falhas: Uma abordagem por redes de petri. In *Anais Do VI Simpósio de Computadores Tolerantes a Falhas*, Agosto 1995.

[dFPM93] J. C. A. de Figueiredo, A. Perkusich, and M. E. Morais. Tolerância a falhas em sistemas de software utilizando uma abordagem por redes de petri. In *Anais Do V Simpósio de Computadores Tolerantes a Falhas*, Outubro 1993.

[DHR93] D. Driankov, H. Hellendoorn, and M. Reinfrank. *An Introduction to Fuzzy Control.* Springer-Verlag, 1993.

[dLSA92] R. de Lemos, A. Saeed, and T. Anderson. Analysis of timeliness in safety-critical systems. In J. Vytopil, editor, *Formal Techniques in Real-Time Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, pages 571–592. Springer-Verlag, 1992.

[DS95] J. Desel and J. Sparza. *Free Choice Petri Nets.* Cambridge University Press, 1995.

[FP80] G. F. Franklin and J. D. Powell. *Digital Control of Dynamic Systems.* Addison-Wesley, 1980.

[GD91] A. Giua and F. DiCesare. Supervisory design using petri nets. In *Proceedings of 30th IEEE International Conference on Decision and Control*, pages 92–97, 1991.

[GD92] A. Giua and F. DiCesare. On the existence of petri net supervisors. In *Proceedings of 31th IEEE International Conference on Decision and Control*, 1992.

[GD93] A. Giua and F. DiCesare. Weak petri net languages in supervisory control. In *Proceedings of the 32nd Conference on Decision and Control*, pages 229–234, San Antonio, Texas, USA, Dezembro 1993.

[GD94a] A. Giua and F. DiCesare. Blocking and controllability of petri nets in supervisory control. *IEEE Transactions on Automatic Control*, 39(4):818–823, 1994.

[GD94b] A. Giua and F. DiCesare. Petri net strutural analysis for supervisory control. *IEEE Transactions on Robotics and Automation*, 10(2):185–195, Abril 1994.

[Gen87] H. J. Genrich. Predicate/transition nets. In *Lecture Notes in Computer Science*, volume 254 of *Petri Nets:Central Models and Their Properties*, pages 207–247. W. Brauer, W. Reisig e G. Rozenberg, 1987.

[GL79] H. J. Genrich and K. Lautembach. The analysis of distributed systems by means of predicate/transition-nets. *Lecture Notes in Computer Sciences*, 70:123–146, 1979.

[Gly89] P. W. Glynn. A GSMP formalism for discrete event systems. *Proceedings of the IEEE*, 77(1):14–23, 1989.

[HG93] L. E. Holloway and X. Guan. A generalization of state avoidance policies for controlled petri nets. In *Proceedings of the 32nd Conference on Decision and Control*, pages 204–210, San Antonio, Texas, USA, december 1993.

[HH79] J. E. Hopcroft and J. D. Hullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[HK90] L. E. Holloway and B. H. Krogh. Synthesis of feedback control logic for a class of controlled petri nets. *IEEE Transactions on Automatic Control*, 35(5):514–523, 1990.

[HK92] L. E. Holloway and B. H. Krogh. On closed-loop liveness of discrete event systems under maximally permissive control. *IEEE Transactions on Automatic Control*, 37(5):692–697, may 1992.

[HKG95] L. E. Holloway, B. H. Krogh, and A. Giua. Petri nets for the control of discrete event systems: A tutorial survey. Center of Robotics and Manufacturing Systems and Department of Electrical Engineering, University of Kentuky, 1995.

[Ho89] Y-C. Ho. Dynamics of discrete event systems. *Proceedings of the IEEE*, 77(1):3–6, 1989.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Process.* Prentice-Hall, London, UK, 1985.

[IH88] A. Ichikawa and K. Hiraishi. Analysis and control of discrete event systems represented by petri nets. In P. Varaiya and A. B. Kurshanski, editors, *Discrete Event Systems: Models and Applications*, volume 103 of *Lecture*

Notes in Control and Information Sciences, pages 115–134. Springer-Verlag, New York, USA, 1988.

[IL95] D. Ionescu and J. Lin. Optimal supervision of discrete event systems in a temporal logic framework. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(12):1595–1605, December 1995.

[IV88] K. Inan and P. Varaiya. Finitely recursive process models for discrete event systems. *IEEE Transactions on Automatic Control*, 33(7):626–639, 1988.

[Jan87] M. Jantzen. Language theory of petri nets. *Central Models and their Properties, Advances in Petri Nets, Lecture Notes in Computer Sciences*, 254-I:397–412, 1987.

[JCJO92] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1992.

[Jen80] K. Jensen. Coloured petri nets and the invariant-method. Technical report, Aarhus University, DAIMI PB-104, Dinamarca, 1980.

[Jen92] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, volume 1 : Basic Concepts. Springer-Verlag, 1992.

[Joh89] B. W. Johnson. Design and analysis of fault-tolerant systems for industrial application. In W. Goke and H. Sorensen, editors, *Informatik-Fachberichte, Fault-Tolerante Computing Systems*, volume 214, pages 57–73. Springer-Verlag, 1989.

[KG88] A. Kaufmann and M. M. Gupta. *Fuzzy Mathematical Models in Engineering and Management Science.* Elsevier Science Publishers B. V., 1988.

[Kle75] L. Kleinrock. *Queueing Systems*, volume 1. Wiley, New York, USA, 1975.

[Kro87] B. H. Krogh. Controlled petri nets and maximally permissive feedback logic. In *25th Annual Allerton Conference*, pages 317–326, University of Illinois, Urbana, USA, Setembro 1987.

[Kuo70] B. C. Kuo. *Automatic Control Systems.* Prentice-Hall, 1970.

[Kuo80] B. C. Kuo. *Digital Control Systems.* Holt, Rinehart and Winston, 1980.

[Lev86] N. G. Leveson. Software safety: Why, what, and how. *ACM Computing Surveys*, 18(2):125–163, Junho 1986.

[LS85] N. G. Leveson and J. L. Stolzy. Analyzing safety and fault tolerance using time petri nets. In G. Goos and J. Hartmanis, editors, *Anais do TAPSOFT'85*, Lecture Notes in Computer Science, pages 339–355. Springer-Verlag, 1985.

[LS87] N. G. Leveson and J. L. Stolzy. Safety analysis using petri nets. *IEEE Transactions on Software Engineering*, SE-13(3):386–397, Março 1987.

[LW88a] Y. Li and W. M. Wonham. On supervisory control of real-time discrete-event systems. *Information Sciences*, (46):159–183, 1988.

[LW88b] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, (44):173–198, 1988.

[LW93] Y. Li and W. M. Wonham. Control of vector discrete-event systems i - the base model. *IEEE Transactions on Automatic Control*, 38(8):1214–1227, august 1993.

[MBC+95] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceshinis. *Modeling with Generalized Stochastic Petri Nets.* John Wiley and Sons, 1995.

[MF76] P. M. Merlin and D. J. Farber. Recoverability of communication protocols - implications of a theoretical study. *IEEE Transactions on Communication*, COM-24(9):1036–1046, Setembro 1976.

[Mil80] R. Milner. *A Calculus of Communicating Systems.* Springer Verlag, New York, USA, 1980.

[Mos93] D. Mossé. *Design, Development, and Deployment of Fault-Tolerant Applications for Distributed Real-Time Systems.* PhD thesis, University of Maryland, Colege Park, 1993.

[Mur89] T. Murata. Petri net: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[OW90] J. S. Ostroff and W. M. Wonham. A framework for real-time discrete event control. *IEEE Transactions on Automatic Control*, 35(4):386–396, Abril 1990.

[Pap65] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, New York, USA, 1965.

[PC92] Y. E. Papelis and T. L. Casavant. Specification and analysis of parallel/distributed software and systems by petri nets with transition enabling functions. *IEEE Transactions on Software Engineering*, 18(3):252–261, 1992.

[PdFC94] A. Perkusich, J. C. A. de Figueiredo, and S. K. Chang. Embedding fault-tolerant properties in the design of complex systems. *Journal of System and Software*, 25(2):23–37, 1994.

[Per94] A. Perkusich. *Análise de Sistemas Complexos Baseada na Decomposição de Sistemas de* G-Nets. PhD thesis, Universidade Federal da Paraíba -Campus II, Campina Grande, PB - BR, 1994.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, N.J., USA, 1981.

[PF95a] A. Perkusich and J. C. A. Figueiredo. Anticipated faults in real-time distributed systems. In *Anais Do 7th International Conference on Software Engineering and Knowledge Engineering*, Junho 1995.

[PF95b] A. Perkusich and J. C. A. Figueiredo. A petri net based approach to model objects for a track-vehicle control system. In *Anais Do 7th International Conference on Computing and Information, ICCI'95*, Trent University, Peterborough, Canadá, Julho 1995.

[Rei85] W. Reisig. *Petri Nets: An Introduction.* Springer-Verlag, 1985.

[Res92] W. Resig. *A Primer in Petri Net Design.* Springer-Verlag, 1992.

[RW87a] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM Journal of Control and Optimization*, 25(5):1202–1218, 1987.

[RW87b] P. J. G. Ramadge and W. M. Wonham. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optmization*, 25(3):637–659, 1987.

[RW87c] P. J. G. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

[RW88] P. J. G. Ramadge and W. M. Wonham. Modular supervisory control of discrete event systems. *Mathematics Control, Signals and Systems*, 1(1):13–30, 1988.

[RW89] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–97, 1989.

[SB93] C. Sibertin-Blanc. Client-server protocol for high-level nets. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 353–372. Springer-Verlag, Chicago, USA, june 1993.

[SGCT88] Y. Shieh, D. Ghosal, P. R. Chintamaneni, and K. Tripathi. Application of petri net models for the evaluation of fault-tolerant techniques in distributed systems. Technical report, University of Maryland, Colege Park, Maryland, 20742, USA, Outubro 1988. Tecnical Report Series CS-TR-2128.

[Sil85] M. Silva. *Las Redes de Petri: en la Automática y la Informática.* Editorial AC, 1985.

[SK92] R. S. Sreenivas and B. H. Krogh. On petri net models of infinite state supervisors. *IEEE Transactions on Automatic Control*, 37(2):818–823, 1992.

[Sre93] R. S. Sreenivas. A note on deciding the controllability of a language k with respect to a language l. *IEEE Transactions on Automatic Control*, 38(4):658–662, 1993.

[SV89] M. Silva and R. Valette. Petri nets and flexible manufacturing. In *Lecture Notes in Computer Science*, pages 374–417. Springer-Verlag, 1989. Advances in Petri Nets.

[Svo84] L. Svobodova. Resilient distributed computing. *IEEE Transactions on Software Engineering*, SE-10(3):257–268, Maio 1984.

[TH86] A. M. Tyrrel and D. J. Holding. Design of reliable software in distributed systems using conversation schame. *IEEE Transactions on Software Engineering*, SE-12(9):921–928, Setembro 1986.

[TRA70] Y. Takahashi, M. J. Rabins, and D. M. Auslander. *Control and Dynamic Systems.* Addison-Wesley, 1970.

[Uni95] Universidade Federal da Paraíba - Centro de Ciências e Tecnologia - Campus II, Campina Grande - PB, Brasil. *Second International Course on Petri Nets for Latin America*, Campina Grande - PB - Brasil, novembro 1995.

[Ush89] T. Ushio. On controlabillity of controlled petri nets. *Control-Theory and Advanced Technology*, 5(3):265–275, september 1989.

[Zad65] L. A. Zadeh. Fuzzy sets. *Information and Control*, (8):338–353, 1965.

[ZC94] R. M. Ziller and J. E. R. Cury. On the supremal L-controllable sublanguage of a non prefix-closed language. In *Proceedings of 10th Congresso Brasileiro de Automática*, pages 260–265, Rio de Janeiro, RJ, BR, 1994.

[ZD93] M. C. Zhou and F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems.* Kluwer Academic Publishers, Boston, USA, 1993.

[Zil93] R. M. Ziller. A abordagem ramadge-wonham no controle de sistemas a eventos discretos: Contribuições à teoria. Master's thesis, Universidade Federal de Santa Catarina, Santa Catarina, BR, 1993.