Uma Abordagem para a Síntese de Supervisores de Sistemas a Eventos Discretos a partir de Modelo Temporizado

Vanderley Pereira da Silva

Dissertação de Mestrado 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 Mestre em Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Antonio Marcus Nogueira Lima, Dr.Ing.
Orientador

Campina Grande, Paraíba, Brasil

S586a Silva, Vanderley Pereira da
Uma abordagem para a sintese de supervisores de sistemas a eventos discretos a partir de modelo temporizado / Vanderley Pereira da Silva. - Campina Grande, 1999.
146 f.

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

1. Processamento da Informacao 2. Engenharia Eletrica 3. Sintese de Supervisores - Eventos Discretos 4. Dissertacao I. Lima, Antonio Marcus Nogueira II. Universidade Federal da Paraiba - Campina Grande (PB)

CDU 621:681.3.02(043)

UMA ABORDAGEM PARA A SÍNTESSE DE SUPERVISORES DE SISTEMAS A EVENTOS DISCRETOS A PARTIR DE MODELO TEMPORIZADO

VANDERLEY PEREIRA DA SILVA

Dissertação Aprovada em 16.08.1999

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

PROF. GIOVANNI CORDEIRO BARROSO, D.Sc., UFC
Componente da Banca

PROF. MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL, Ph.D., UFPB
Componente da Banca

PROF. ANGELO PERKUSICH, D.Sc., UFPB
Componente da Banca

CAMPINA GRANDE - PB
Agosto - 1999

Dedicatória

Dedico este trabalho aos meus pais Valderino e Vanda, aos meus irmãos Valderino e Vanderlan, e a minha namorada Darlinda, pelo amor, carinho, dedicação, amizade e confiança.

Agradecimentos

Agradeço ao professor Antonio Marcus Nogueira Lima, pela orientação e contribuição necessárias para a realização deste trabalho.

Um agradecimento especial a todos os amigos do LEIAM, Bione, Edgar, Eduard, Fabiano, Inaldo, Laurinda, Levi, Marcelo, Maurício, Rui e Talvanes, pela ajuda e companheirismo durante todo o desenvolvimento deste trabalho.

Agradeço a ajuda recebida de todos os colegas do LIEC, LPP e LabPetri.

Agradeço a Ângela, Eleonora, Pedro e a todos aqueles que contribuiram direta ou indiretamente para a realização deste trabalho.

À CAPES e a Universidade do Amazonas que proporcionaram o suporte financeiro para viabilizar a realização deste trabalho.

Resumo

Esse trabalho trata da síntese de supervisores para sistemas a eventos discretos. A abordagem proposta utiliza a teoria de controle supervisório em conjunto com as redes de Petri. Para a realização desta síntese são utilizados algoritmos para a enumeração do espaço de estados do modelo e para obtenção das funções de habilitação das transições da rede que representa o supervisor. A rede de Petri que modela o comportamento sistema a eventos discretos inclui a informação relativa aos tempos de execução das tarefas do sistema. Para tratar os modelos temporizados é desenvolvido um algoritmo para a construção do espaço de estados do modelo e é introduzida uma nova classe de rede de Petri que é utilizada para representar o supervisor. O trabalho também apresenta um algoritmo para converter o supervisor sintetizado em um programa a ser executado num controlador lógico programável. A utilidade da metodologia proposta é demonstrada aplicando-a na síntese de supervisores para vários exemplos típicos de sistemas de manufatura.

Abstract

This work deals with the synthesis of the supervisors for discrete events systems. The proposed approach combines the supervisory control theory and Petri nets. In order to achieve the supervisor synthesis an algorithm for enumerating the state space and another to determine the transition enabling functions of the Petri net that represents the discrete event system are employed. The Petri net that models the behavior of the discrete event system includes the timming of the tasks to be executed by the system. In order to take into account the timmed models an algorithm for constructing the state space of the model has been developed and a new extension of Petri net is introduced to represent the supervisor. The work also presents an algorithm to convert the supervisor issued from the synthesis procedure into a program to be executed in a programmable logic controller. The usefulness of the proposed methodology is demonstrated by its application to synthesize the supervisors for various typical manufacturing systems.

Lista de Símbolos e Abreviaturas

$\in$	Pertence a
$\notin$	Não pertence a
:	Tal que
$\exists$	Existe
$\forall$	Para Todo
$\emptyset$	Conjunto vazio
$\cup$	Operador união de conjuntos
$\cap$	Operador interseção de conjuntos
$\wedge$	Operador "e"
$\vee$	Operador "ou"
N	Conjunto dos números naturais
Q	Conjunto dos números irracionais
$\alpha, \beta, \lambda, \mu...$	Eventos
Σ	Alfabeto ou conjunto de eventos
$r, s, u, w, ...$	Palavras
$\|s\|$	Comprimento da palavra s
ε	Palavra nula
$Pre\,(s)$	Conjunto de todos os prefixos da palavra$s$
Σ^+	O conjunto de todas as palavras que podem ser formadas com símbolos de Σ
Σ^*	Idem, incluindo a palavra ε
Σ^k	O conjunto das palavras $\{s : s \in \Sigma^* \wedge \|s\| = k\}$
$\delta\,(\sigma, q)!$	A função δ é definida para o par (σ, q)
$\delta\,(s, q)!$	A função δ é definida para o par (s, q)
Q, X	Conjunto de estados
$\|Q\|$	Cardinalidade do conjunto Q
A	Autômato

G, S	Geradores
K, L	Linguagens
$\overline{K}$	Prefixo_fechamento de K
KL	A linguagem $\{st : s \in K \wedge t \in L\}$
$K - L$	A linguagem $\{s : s \in K \wedge s \notin L\}$
$L(G)$	Linguagem gerada de G
$L_m(G)$	Linguagem marcada de G
Σ_c	Conjunto de eventos controláveis
Σ_u	Conjunto de eventos não controláveis
$\Sigma(q)$	O conjunto de eventos tais que $\sigma \in \Sigma(q) \Longrightarrow \delta(\sigma, q)!$, isto é, o conjunto de eventos que podem ocorrer a partir do estado q
$< G, \Gamma >$	Planta formada pelo gerador G com conjunto de entradas de controle válidas Γ
$< S, \Phi >$	Supervisor formado pelo gerador S e pelo mapa de controle Φ
$L(S, G)$	Linguagem gerada de $< G, \Gamma >$ sob supervisão de $< S, \Phi >$
$L_c(S, G)$	Linguagem controlada de $< G, \Gamma >$ sob supervisão de $< S, \Phi >$
$L_m(S, G)$	Linguagem marcada de $< G, \Gamma >$ sob supervisão de $< S, \Phi >$
$supC(K), K^{\uparrow}$	Máxima sublinguagem $L(G)$-controlável de K
P	Conjunto finito de lugares
T	Conjunto finito de transições
I	Função que especifica os arcos que ligam lugares a transições
O	Função que especifica os arcos que ligam transições a lugares
K	Função de capacidade dos lugares
M_0	Marcação inicial de uma rede de Petri
${}^{\bullet}p$	Conjunto das transições de entrada do lugar p
$p^{\bullet}$	Conjunto das transições de saída do lugar p
${}^{\bullet}t$	Conjunto dos lugares de entrada da transição t
$t^{\bullet}$	Conjunto dos lugares de saída da transição t
$M(p)$	Número de fichas no lugar p na marcação M
M_i	Uma marcação (estado) da rede
N	Estrutura de rede de Petri
$R(M_0)$	Conjunto de todos os estados alcançáveis a partir de M_0

$L(M_0)$	Conjunto de todas as seqüências de disparo possíveis a partir de M_0
l	Função que etiqueta as transições
Φ	Conjunto de Funções de habilitação de transição
φ_i	Função de habilitação da transição i
E	Conjunto de eventos externos
FE	Função que mapeia eventos externos para as transições
$Tempo, Tempo\_t$	Função que mapeia tempo para as transições
τ	Quantidade de tempo
M^r	Marcação composta pelas fichas reservadas
M^n	Marcação composta pelas fichas não reservadas
M^d	Marcação composta pelas fichas disponíveis
M^i	Marcação composta pelas fichas indisponíveis
IP	Mapeamento de intervalos de tempo para lugares
SE	Conjunto não vazio de sinais binários de entrada
SS	Conjunto não vazio de sinais binários de saída
CE	Mapeamento de condições externas para as transições
MA	Mapeamento de lugares para ações externas
Sen	Conjunto de sinais binários de sensores
Act	Conjunto de sinais binários para atuadores
FD	Função de disparo temporizada
R^+	Conjunto dos números reais não-negativos
FX	Função sinal de entrada
Y	Função sinal de saída
Est	Estado de uma TPN
Id	Intervalo de disparo
D	Domínio de disparo da classe
IE	Intervalo estático
TDI	Tempo de disparo estático inferior
TDS	Tempo de disparo estático superior
$AEPCLP$	Algoritmo para elaboração do programa para CLP

AAAC	Algoritmo da árvore de alcançabilidade de classes
ACGS	Algoritmo para a construção da suprema linguagem controlável
AGV	Veículo guiado automaticamente
AMArA	Algoritmo modificado da árvore de alcançabilidade
CLP	Controlador Lógico Programável
CTL	*Computation tree logic*
FBD	Programação por diagrama de blocos
FTPN	Rede de Petri com temporização nebulosa
GRAFCET, SFC	Grafo de comando etapa-transição
GSMP	Processos semi-markovianos generalizados
IL	Lista de instruções
IPN, SIPN	Rede de Petri Interpretada
LD	Diagrama de relés
PdPN	Rede de Petri P-temporizados
PID	Proporcional integral derivativo
PPN	Rede de Petri P-temporal
PTdPN	Rede de Petri P-temporal T-temporizada
RP	Rede de Petri marcada
RPFHT	Rede de Petri com função de habilitação de transições
RPS	Rede de Petri sincronizada
RPT	Rede de Petri com extensão temporal
RTPN	Rede de Petri em tempo real
RPTIFHT	Rede de Petri temporal interpretada com funções de habilitação de transições
SDVC	Sistema dinâmico de variáveis contínuas
SED	Sistema dinâmico a eventos discretos
ST	Texto estruturado
TCS	Teoria de controle supervisório
TdPN	Rede de Petri Temporizada
TPN	Rede de Petri Temporal

Lista de Figuras

Lista de Tabelas

Conteúdo

Capítulo 1

Introdução

A necessidade atual de modernização tecnológica dos sistemas de manufatura de bens, imposta tanto pela globalização e, principalmente, pelas mudanças de comportamento da sociedade de consumo, aponta para a questão do aumento de qualidade, produtividade, flexibilidade e competitividade, levando as empresas a, cada vez mais, usarem sistemas de manufatura automatizados. Esses sistemas apresentam um alto nível de integração dos equipamentos de manufatura com a tecnologia computacional que em conjunto com as necessidades de produção resultam em complexos sistemas dinâmicos feitos pelo homem. Devido ao grande capital aplicado e a complexidade desses sistemas industriais, o projeto e operação dos mesmos requer um significativo esforço de modelagem e análise para selecionar o melhor projeto e a política de operação.

Na composição dos sistemas de manufatura automatizados, podemos encontrar equipamentos como as máquinas de controle numérico, os controladores de processos industriais, os sensores e os atuadores. As máquinas de controle numérico, são dispositivos de produção responsáveis pelo processamento e transporte da matéria, como por exemplo os tornos mecânicos, as fresadoras, algumas máquinas especiais para montagem e para corte de materiais, robôs, veículos guiados automaticamente ($AGVs$), etc. Os controladores de processos industriais, são dispositivos microprocessados dedicados ao controle e monitoração de processos industriais de uma maneira geral; podem ser dedicados ao controle discretizado de variáveis, onde se destacam os controladores digitais de processo, os controladores lógico programáveis (CLP) e os controladores de processos contínuos. Diversos são os tipos de sensores utilizados na automação de processos de produção discreta, dependendo da aplicação a que se destine, como por exemplo os de posiçvelocidade, os de proximidade, presença, passagem e fim de curso, entre outros. Os atuadores são dispositivos utilizados para a conversão de sinais elétricos de controle provenientes dos controladores em ações requeridas pelo controle do sistema [10].

Esses sistemas são caracterizados por suas *condições* e *eventos*. As condições represen-

tam os estados, como por exemplo, "Robô 1 disponível", "AGV deslocando-se da estação A para a B", etc. Os eventos, representam a forma pela qual estes sistemas percebem as ocorrências no ambiente, são ações que podem ocorrer, como por exemplo, "Início do transporte da peça 1 (que está na máquina 2) para a máquina 3", "AGV deixando a estação A", "AGV chegando a estação B", "A detecção de uma mudança de estado por um sensor", etc. Os eventos são, por natureza, instantâneos, o que lhes confere um caráter discreto no tempo [1, 2].

Para sistemas desta natureza, o espaço de estados é um conjunto discreto, que descreve todos os estados possíveis que o sistema pode assumir, e a evolução dos seus estados depende da ocorrência súbita de um evento físico, a resposta de um sensor por exemplo, a intervalos de tempo em geral irregulares e desconhecidos. Em resposta a ocorrência de um evento, o sistema reage, acionando atuadores por exemplo, acomodando-se numa nova situação, onde permanece até que ocorra um novo evento. Sistemas com estas características são denominados de Sistemas Dinâmicos a Eventos Discretos (*SDED*, ou *DEDS* em inglês) ou simplesmente Sistemas a Eventos Discretos (*SED* ou *DES*) em oposição aos familiares Sistemas Dinâmicos de Variáveis Contínuas (*SDVC* ou *CVDS*) [5, 42, 3, 44, 22, 6] .

Para que um evento ocorra, suas *pré-condições* precisam ser verdadeiras. Uma vez que as pré-condições são verdadeiras o evento pode ocorrer fazendo com que o sistema assuma novas condições que são chamadas de *pós-condições.*

Um *SED* pode apresentar *condições de conflito* e *concorrência de eventos.* Por exemplo, a condição "AGV deslocando-se para a estaccão A" e "AGV deslocando-se para a estação B", representam um conflito e não podem ocorrer simultaneamente para um mesmo AGV. Já os eventos "Início do carregamento da máquina 3 pelo robô 2" e "Início do processamento do produto 1 na máquina 1" são eventos independentes e podem ocorrer simultaneamente.

Um exemplo de um *SED* é mostrado na Figura 1.1. O sistema consiste de duas máquinas diferentes, um robô, dois depósitos e uma esteira. Todas as peças do depósito 1 precisam ser processadas pela máquina 1 ($M1$), e todas as peças do depósito 2 pela máquina 2 ($M2$). O robô, é utilizado para carga e descarga das máquinas. A evolução dinâmica deste sistema pode ser exemplificada pela Figura 1.2, que representa graficamente a trajetória percorrida pelo *SED* no seu espaço de estados. Na Figura 1.2, os eventos são representados pelas letras $\alpha, \beta, \gamma, \nu, \rho, \tau, \eta, \sigma$. O estado inicial e_0, é o estado em que se encontra o *SED* antes da ocorrência do primeiro evento. Em muitos casos é desejável que o sistema, após a realização de uma tarefa especificada retorne a este estado, denominado estado de repouso (*home state*).

No exemplo da Figura 1.2, e_0 representa o estado inicial, ou estado de repouso, no qual o robô e as máquinas estão livres e os depósitos possuem peças. Na ocorrência do evento α, o robô começa a carregar $M1$ e o sistema passa ao estado e_1, robô carregando $M1$.

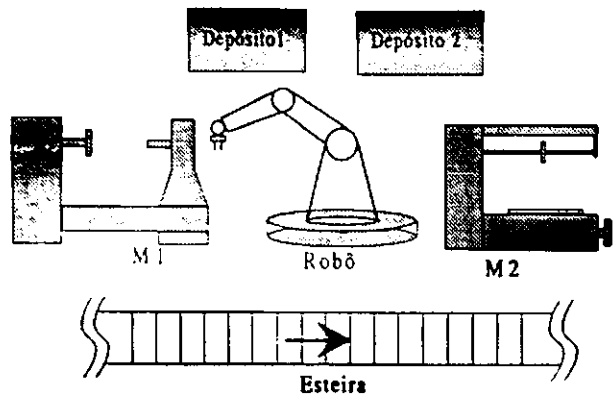


Figura 1.1: Exemplo de um *SED*

Ocorrendo o evento β, indicando término do carregamento e início do processamento por $M1$, o sistema passa ao estado e_2. O sistema alcança o estado e_3, após a ocorrência de γ, robô carregando $M2$. Quando ocorre υ, indicando o final do carregamento da $M2$ e início do processamento pela mesma, o sistema atinge o estado e_4. Ocorrendo o evento ρ, o robô começa a descarga de $M1$, o sistema chega ao estado e_5. A mudança de estado, de e_5 para o estado e_6, ocorre com o evento τ. O sistema alcança o estado e_7, início da descarga de M_2, na ocorrência de η. Com a conclusão da descarga de $M2$, indicado pelo evento σ, o sistema volta ao seu estado inicial.

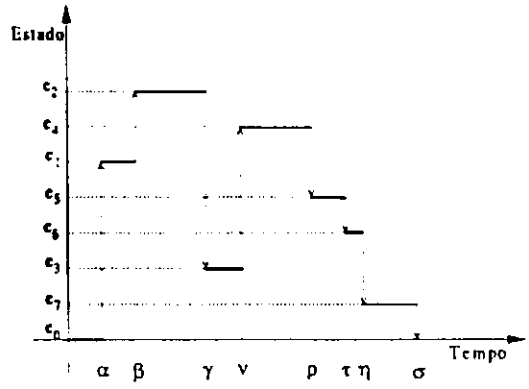


Figura 1.2: Exemplo de trajetória de um *SED*

Atualmente há uma significativa atividade de pesquisa voltada à busca de modelos matemáticos adequados à representação dos *SEDs*, pois sua natureza discreta faz com que os modelos matemáticos convencionais baseados em equações diferenciais, não sejam adequados para tratá-los. É desejável encontrar um modelo matematicamente conciso e computacionalmente adequado, para que se possa estudar e apresentar soluções para os problemas relacionados ao controle deste tipo de sistema.

1.1 Modelos para *SEDs*

Sistemas a eventos discretos, por exemplo os modernos sistemas industriais, exibem muitas características como a evolução dependente de eventos, o assincronismo, a relação seqüencial, a concorrência, o conflito, a exclusão mutua, o não determinismo e o bloqueio. Essas características, são muito difíceis de serem descritas pela teoria de controle tradicional que utiliza equações diferenciais ordinárias e parciais para descrevê-las.

Para capturar essas propriedades, muitos mecanismos têm sido propostos e desenvolvidos para modelagem e análise desses sistemas. O modelo pode contemplar tempo ou não. A diferença é que o modelo que não contempla enfatiza a "seqüência de estados" de um *SED* e omite inteiramente a "dependência temporal" das especificações para cada estado e é adequada apenas para responder questões de natureza qualitativa ou lógica. Em modelos temporizados, diferentemente, o "tempo" é incorporado como parte integrante do modelo e é mais adequada para responder questões relacionadas ao desempenho. Entre esses modelos podem ser citados:

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

1.1.1 Cadeias de Markov

As cadeis de Markov são processos estocásticos, modelos matemáticos usados para a descrição de fenômenos aleatórios como função de um parâmetro que geralmente tem o significado de tempo e cujo espaço de estados S, do processo, é discreto [39, 16, 26, 5, 44]. O espaço de estados de uma cadeia é usualmente o conjunto dos números inteiros naturais $\mathrm{N} = \{0, 1, 2, ...\}$, ou um subconjunto deste. Do ponto de vista matemático, um processo estocástico é uma família de variáveis aleatórias $\{X(t), t \in T\}$ definida sobre um mesmo espaço probabilístico, indexado pelo parâmetro t, e levando a valores no conjunto S. Os valores assumidos pelo processo estocástico são chamados *estados*, tal que o conjunto S é chamado o *espaço de estado* do processo.

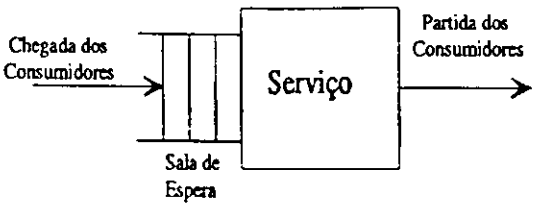


Figura 1.3: Representação de uma fila

A descrição probabilística de um processo estocástico é dado por meio da união de suas *funções de distribuição de probabilidade* [1] (*PDF* - também conhecida como função de distribuição comulativa) das variáveis aleatórias $\{X(t_i), i = 1, 2, ..., n\}$.

Neste tipo de modelo, o comportamento do sistema a partir de um instante qualquer independe dos estados passados; apenas o estado atual influencia o comportamento futuro. Esta propriedade recebe o nome de propriedade de Markov. A referida propriedade [26], é formalmente dada por:

$$\begin{aligned} & P\{X(t) \leq x | X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, ..., X(t_0) = x_0\} \\ = \; & P\{X(t \leq x | X(t_n) = x_n\} \quad t > t_n > t_{n-1} > ... > t_0 \end{aligned}$$

para todo $n \in \mathrm{N}$ e $x_0, ..., x_n \in S$.

Este modelo é adequado à representação de sistemas quando se deseja considerar a probabilidade de ocorrência de eventos, como acontece na análise de desempenho.

1.1.2 Teoria das Filas

Sistemas de filas são um caso particular da classe dos *sistemas de fluxo*. Num sistema de fluxo, objetos ou informações fluem, movem-se ou são transportados através de um ou mais canais de capacidade limitada, de um ponto a outro do sistema. Devido ao limite de capacidade dos canais, podem ser formadas filas de espera pela disponibilidade de um ou mais canais [44, 3].

As filas constituem uma outra ferramenta de modelagem de *SEDs*. Uma fila é um sistema para o qual clientes chegam a uma estação de trabalho para solicitar um serviço. A estação de serviço pode ter um ou mais servidores. Quando todos os servidores estão ocupados, os clientes precisam esperar em uma sala de espera. Quando o serviço é concluído, o cliente deixa a fila. A Figura 1.3 ilustra as principais caracteríexemplo, tarefas são consumidores e máquinas são os servidores. A sala de espera é análoga a um depósito.

Conhecer o processo de chegada do cliente à estação e a forma como ele é atendido é importante para se descrever um sistema deste tipo. Diferentes representações originam diversos tipos de filas, cada qual com características e aplicações próprias.

[1] Usamos a notação $P\{A\}$ para indicar a probabilidade de ocorrência do evento A.

Os sistemas de filas encontram aplicação na análise de desempenho de *SEDs*, como por exemplo, redes telefônicas, redes de computadores e redes de manufatura. Exemplos de modelagem de sistemas de manufatura, utilizando esta teoria, são apresentados por Viswanadham [39] e Cassandras [5].

1.1.3 Processos Semi-Markovianos Generalizados

Um processo semi-markoviano é um processo estocástico $\{X(t) : t \geq 0\}$. A variável aleatória $X(t)$ representa o estado do autômato no tempo t. Um processo semi-markoviano processa a propriedade de Markov no valor do próximo estado no ponto onde um evento ocorre e depende somente do corrente estado. Os tempos entre eventos, no entanto, têm distribuição aleatória [5, 39, 26].

Nesta abordagem, procura-se 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 representadas por estados. As partes contínuas, tais como o tempo restante de serviço, são chamadas 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 [5, 39, 20].

1.1.4 Álgebra de Processo

A um conjunto de eventos ocorridos em uma dada seqüência dá-se o nome de *processo*. Desta forma, uma seqüência de eventos gerada por um *SED* pode ser vista como um processo.

O modelo para um dado *SED* pode ser obtido pela composição de modelos menores, que representam partes do sistema. Isto é útil porque, em geral é mais fácil modelar o sistema passo a passo do que de uma só vez. Ao se representar um *SED* por um conjunto de processos, é necessário definir um conjunto de operações sobre processos que permitam representar a composição dos mesmos.

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

Várias álgebras de processo têm sido desenvolvidas para o tratamento de *SEDs*, inspiradas nos trabalhos de Milner [29] e Hoare [23].

1.1.5 Álgebra Max-Plus

Outra estrutura para modelagem é baseada no uso de um dióide, estrutura algébrica constituída de um conjunto e duas operações sobre este conjunto: $max\{a, b\}$ para qualquer número real a e b, e adição $(a + b)$.

O termo "dióide" (siguinifica "dois") refere-se ao fato que esta álgebra é baseada em dois operadores. As operações são formalmente chamadas *adição* e *multiplicação* e representadas por $\oplus$ e $\otimes$, respectivamente. Entretanto, seus significados (em termos de álgebra regular) são diferentes. Para qualquer dois números reais a e b, definimos

$$\begin{aligned} Adição &: a \oplus b \equiv \max\{a, b\} \\ Multiplicação &: a \otimes b \equiv a + b \end{aligned}$$

Observação 1.1 *A álgebra* $(\max, +)$ *pode facilmente ser trocada pela álgebra* $(\min, -)$. *Isto é simples pois*

$$\max\{a, b\} = -\min\{(-a), (-b)\}$$

As propriedades algébricas desta estrutura mostram-se úteis na descrição de sistemas a eventos discretos, permitindo uma representação próxima à da teoria de controle para *SD-VC*, que envolvem operações regulares de multiplicação $(\times)$ e adição $(+)$ em suas equações de estado: $X(k+1) = Ax(k) + Bu(k)$. Em particular, o comportamento de um *SED* que envolve um conjunto de atividades repetitivas, pode ser caracterizado pela solução de uma equação matricial escrita nesta álgebra. Detalhes da teoria e exemplos podem ser encontrados em Cassandras [5] e Cohen [8].

1.1.6 Lógica Temporal

A Lógica Temporal (LT), proporciona a base formal para a descrição e raciocínio relativos ao comportamento correto dos sistemas, especialmente de sistemas não seqüenciais, expressando propriedades relativas à ocorrência de eventos no tempo.

Existem basicamente duas versões de lógica temporal que são a lógica temporal linear e a lógica temporal de tempo ramificado. Na lógica temporal linear considera-se que as execuções do sistema, as quais representam seu comportamento e são representadas tanto

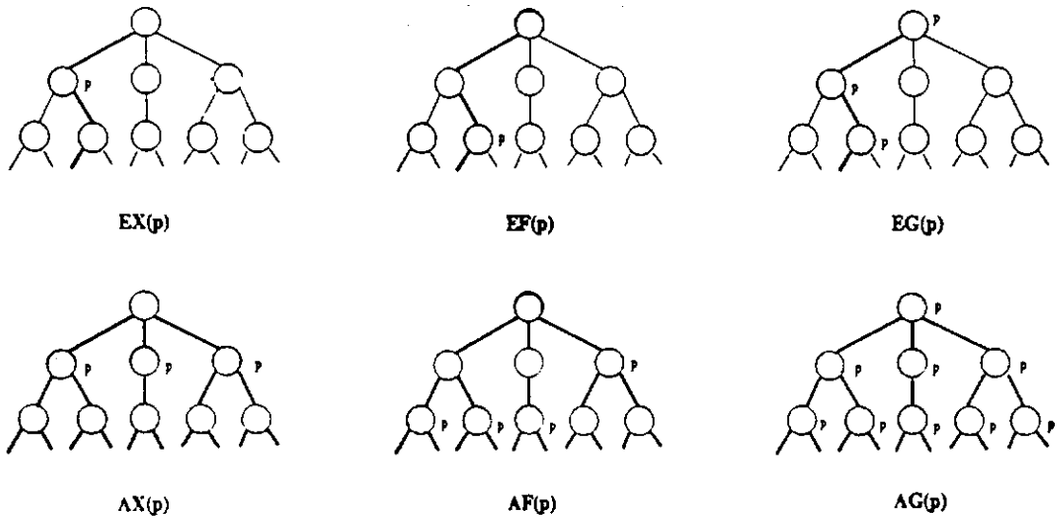


Figura 1.4: Ilustração do significado de operadores temporais

por seqüências de eventos ordenados no tempo ou por seqüências de eventos em ordem causal, não são relacionadas. Por outro lado, na lógica temporal de tempo ramificado as execuções de sistema são representadas por seqüências de eventos ordenados no tempo ou por seqüências de eventos em ordem causal e estão relacionados numa estrutura em árvore com ramos que indicam alternativas e conflitos.

A lógica de tempo ramificado, [9, 21, 40], é conhecida como *Computation Tree Logic (CTL)*, na qual os operadores temporais ocorrem apenas aos pares combinando quantificadores de caminho universal (A) ou existencial (E) com algumas fórmulas de caminho. As fórmulas de caminho são da forma: sempre ($G(p)$), eventualmente ($F(p)$), próximo estado ($X(p)$), ou até que ($[pUq]$), onde p e q são fórmulas *CTL*. Alguns exemplos de fórmulas *CTL* válidas são $AG(p)$, $EF(p)$, e $A[pUEF(q)]$, onde p e q são proposições atômicas. Vale resaltar que os conectivos lógicos da lógica proposicional, $\neg$ (não), $\wedge$ (e), $\vee$ (ou) e $\rightarrow$ (implica), também são usados nesta lógica.

A Figura 1.4 ilustra a semântica destes operadores, em termos da árvore de transições característica de um *SED*. Nesta figura, os nós representam o estado do sistema a cada instante, as fórmulas se aplicam a partir do nó raiz e p é verdadeira em alguns estados de forma que a correspondente fórmula seja satisfeita. Por exemplo, $EF(p)$ significa que a partir do estado inicial é possível a ocorrência de p, $AF(p)$ especifica que partindo do estado inicial o sistema eventualmente alcançará um estado no qual a propriedade p é verdadeira, $AG\,(p)$ especifica um invariante, isto é, p precisa ser verdadeiro em todos os estados. Esta fórmula pode ser usada para verificar exclusão mútua (dois processos nunca estão em uma seção crítica ao mesmo tempo) ou se o sistema não possui estados de bloqueio.

A lógica temporal pode ser utilizada de duas maneiras distintas no tratamento de *SEDs*: no primeiro caso, o sistema, condicionado à ação de um controlador que tem por objetivo

assegurar seu funcionamento de acordo com a especificação, é modelado empregando-se uma técnica que não a lógica temporal, como por exemplo um autômato ou uma rede de Petri; a lógica temporal é utilizada para formular asserções que refletem o funcionamento desejado do sistema. É necessário que exista um mapeamento entre as duas representações, através do qual se procura então verificar que as asserções não são violadas pelo modelo. Conseguindo isto, assume-se que este é correto. Esta abordagem é denominada *dual*, por empregar dois formalismos distintos. Um exemplo desta abordagem pode ser visto no trabalho de Costa [9], no qual a lógica temporal é utilizada para a especificação de comportamento de modelos de *SEDs* em redes de Petri e cujo supervisor são obtidos pela teoria de controle Supervisório. Por outro lado, é possível empregar a própria lógica temporal para a modelagem do sistema, de maneira que tanto o modelo quanto as asserções que garantem sua correção sejam dadas pelo mesmo formalismo. O que se faz neste caso é tentar provar as asserções como se fossem teoremas, a partir da representação do modelo. Este é considerado correto se todas as asserções puderem ser provadas. Esta abordagem é denominada *primal*, por se utilizar um único formalismo.

Cabe notar aqui que, apesar do nome, a lógica temporal não permite considerar instantes de tempo determinados (ou seja, o tempo real). A lógica temporal pode ser dotada de meios que possibilitem explicar os instantes de tempo em que os eventos ocorrem. Tem-se então a chamada *lógica temporal de tempo real*.

1.1.7 Teoria de Autômatos e de Linguagens

Uma das maneiras formais de se estudar o comportamento lógico de um *SED* é baseado nas teorias de *linguagens* e *autômatos* (ou máquinas de estado). O ponto de partida é o fato que qualquer *SED* tem um conjunto de eventos subjacentes E associado a ele. O conjunto E é visto como o *alfabeto* da linguagem, e as seqüências de eventos como as *palavras* desta linguagem. Um autômato, então, é um dispositivo que gera uma linguagem pela manipulação do alfabeto (eventos) de acordo com um conjunto especificado de regras. Entretanto, implementar um *SED*, modelado por uma máquina de estados, se torna difícil por causa do aumento exponencial do número de estados do modelo 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 teorias de autômatos e linguagens serão apresentadas no capítulo 2 e podem também ser encontradas em Cassandras [5].

Essas teorias tem sido bastante pesquisadas. Dentre os trabalhos podemos citar o modelo proposto por Ramadge e Wonham [34, 41], pois difere das demais abordagens apresentadas nesta seção pelos resultados obtidos sobre a existência de um supervisor para impor à planta um dado padrão de comportamento minimamente restritivo (único) e os meios para sintetizá-los.

1.1.8 Redes de Petri

As redes de Petri [31] são uma ferramenta gráfica e matemática que apresentam um método unificado para modelagem, análise e implementação de sistemas. Como ferramenta gráfica, tem grande poder de comunicação visual, similar aos diagramas de fluxo e diagramas de bloco, e combinada a ferramentas computacionais, possibilita uma simulação gráfica interativa. Ela possibilita uma análise formal do modelo, enquanto ferramenta matemática, pois permite uma descrição por equações de estado, equações algébricas e outros modelos matemáticos. Comparadas com os modelos discutidos, elas apresentam as seguintes vantagens [42]:

- 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 das dependências 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 indesejadas do sistema, 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 de eventos discretos diretamente a partir do modelo;
- Informação do estado atual do sistema que permite monitorização em tempo real.

Devido a estas vantagens, a modelagem dos *SEDs*, deste trabalho, será realizada em redes de Petri.

A possibilidade de análise formal do modelo em rede de Petri é especialmente importante para sistemas críticos em tempo real, como os sistemas de controle de trafego áereo e ferroviário e sistemas de controle de reatores nucleares, entre outros. Dois tipos de

propriedades podem ser estudadas com modelos de rede de Petri: as propriedades comportamentais e as propriedades estruturais. As propriedades comportamentais, como alcançabilidade, vivacidade, limitação e reversibilidade são bastante importantes no estudo de bloqueios, estouro da capacidade de depósitos e capacidade de reinicialização.

A alcançabilidade permite-nos saber se um determinado estado faz parte do conjunto de estados possíveis no sistema ou não, isto é, se o sistema pode ou não atingir um estado específico, ou exibe um comportamento funcional particular. A vivacidade nos diz se o sistema apresenta alguma situação que leve-o ao bloqueio. A limitabilidade é a propriedade que nos ajuda a identificar, no sistema modelado, a existência de estouro da capacidade dos locais, que freqüentemente representam áreas de armazenamento de produtos e de ferramentas. A reversibilidade nos possibilita saber se o sistema é capaz de retornar de um estado de falha para o estado correto precedente ou para o estado inicial [31, 43].

As redes de Petri são compostas por transições, lugares, arcos e fichas. As transições são graficamente representadas por barras ou retângulos. Os lugares são representados por círculos. Os arcos ligam os lugares às transições ou as transições aos lugares. As fichas podem ou não estar presentes nos lugares, indicando a veracidade ou não da condição representada pelo lugar.

As redes de Petri podem ser utilizadas para modelar *SEDs*, usando os conceitos de condições e eventos. Quando isto é feito, lugares representam condições e transições representam eventos. Cada transição (um evento) pode ter um certo número de lugares de entrada (pré-condições), e de lugares de saída (pós-condições).

As transições podem ser usadas para modelar atividades, de modo que, por exemplo, o período de habilitação das transições corresponda a execução da atividade e o disparo da transição corresponda ao término da atividade. Assim, tempo pode ser naturalmente associado às transições [26]. Aos lugares, também pode ser associado tempo de forma a possibilitar a modelagem de estados como: "Produto 1 está sendo processado na máquina 1 por 5 minutos" [1, 2].

Uma das maiores vantagens de usar modelos de redes de Petri, é que o mesmo modelo é usado para a análise das propriedades comportamentais e avaliação de desempenho, bem como a construção sistemática de simuladores a eventos discretos e controladores [43]. No entanto, é muito difícil verificar modelos complexos de rede de Petri de grandes sistemas de manufatura [45, 42]. Para solucionar este problema, muitas metodologias de síntese de construção de modelos em redes de Petri de sistemas de manufatura foram proposta das quais se destacam as abordagens bottom-up e top-down [42].

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 *GSPM* Redes de Petri Estocásticas	

Tabela 1.1: Classificação dos Modelos de *SEDs*

1.2 Concepção de controladores para *SEDs*

Pode-se observar nos modelos apresentados que a maioria tem recursos formais apenas para a análise, mas não para a síntese de controladores para *SEDs*, obrigando o projetista a executar os passos de síntese com base em outros métodos, muitas vezes fortemente dependentes de sua experiência. Os modelos para *SED* 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 de equações, ao uso de asserções lógicas ou à formulação probabilística, respectivamente. A tabela 1.1, baseada em [22] e atualizada em [44], foi montada baseada nesses critérios.

Diferente das técnicas de projeto de controladores para sistemas contínuos, que são sistematizadas e consolidadas, as técnicas de projeto de controladores para *SEDs* estão em desenvolvimento e não estão sistematizadas. Não existe a predominância de uma determinada técnica sobre as demais propostas. Paralelamente, observa-se um aumento no interesse do uso de redes de Petri para a modelagem, controle, simulação e análise de desempenho de *SEDs*. Este aumento, deve-se ao ambiente uniforme de modelagem (dos componentes de *hardware*, *software*, humano e suas interações), de análise formal e projeto que as redes de Petri oferecem.

Uma estrutura de controle bastante discutida, para a concepção de controladores de sistemas a eventos discretos, como os sistemas de manufatura e outros sistemas distribuídos, é a de controle hierárquico. Para Zhou e DiCesare [42], a estrutura de controle de um sistema a eventos discretos em tempo real pode ser vista como uma estrutura de dois níveis. O primeiro nível é o controle supervisório ou controle relacionado a eventos, e o segundo nível é o controle de processos tradicional. Esta estrutura de controle é apresentada na Figura1.5.

O controle hierárquico, em um ambiente automatizado, pode ser implementado utilizando-

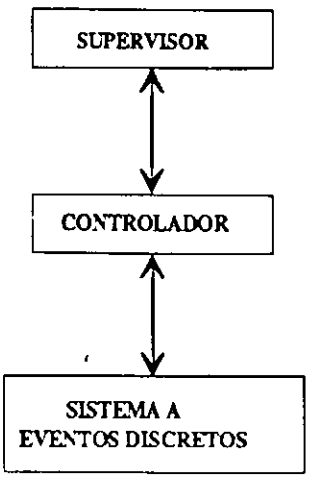


Figura 1.5: Arquitetura de Controle de um *SED* real

se dois métodos básicos: controle centralizado e controle distribuído. No primeiro método, um computador é usado para realizar e sincronizar todas as atividades relacionadas a eventos e controlar processos de baixo nível. No segundo, os componentes de um sistema podem ser sub-sistemas independentes que se comunicam entre si, para solucionar o problema de controle. Um controlador supervisório, deve ser concebido para a coordenação de tarefas assíncronas entre esses componentes distribuídos para satisfazer os requisitos do sistema.

Outra teoria de controle para *SEDs*, denominada *Teoria de Controle Supervisório* (*TCS*), proposta por Ramadge e Wonham [41, 34], é baseada em linguagens formais e autômatos. Nesta teoria, é feita uma distinção clara entre o sistema a ser controlado, denominado *planta*, e a entidade que o controla, que recebe o nome de *supervisor*. A planta é um modelo que reflete o comportamento fisicamente possível do sistema, isto é, todas as ações que este é capaz de executar na ausência de qualquer ação de controle. O supervisor é aquele que exerce uma ação de controle restritiva sobre a planta, de modo a confinar seu comportamento àquele que corresponde a uma dada especificação.

Uma vantagem desta teoria é a de permitir uma abordagem sistemática de síntese de supervisores, semelhante ao que se faz em controle convencional, sendo estes obtidos de forma a restringir o comportamento da planta para evitar que esta realize ações proibidas. Desta forma, pode-se verificar se uma dada especificação de comportamento pode ou não ser cumprida e, caso não possa, identificar a parte dessa especificação que pode ser implementada de forma minimamente restritiva.

Usando a Teoria de Controle Supervisório e as Redes de Petri, Barroso [3] propôs uma abordagem para a síntese de supervisores. Está técnica serviu de base para a dissertação que será apresentada a seguir.

1.3 Apresentação da Dissertação

Os modernos sistemas industriais exibem as características de um sistema a eventos discretos. Estes sistemas apresentam grande complexidadde e requerem ferramentas para modelagem e análise que ajudem na obtenção da melhor política de operação dos mesmos. Várias são as ferramentas de modelagem e análise para este tipo de sistema, como as citadas na seção 1.1.

Uma teoria interessante, proposta por Ramadge e Wonham [41] e objeto de estudo de Ziller [44], denominada Teoria de Controle Supervisório, permite a síntese de entidades de controle, denominadas supervisores. Nessa teoria há uma distinção clara entre o sistema a controlar, denominado genericamente como planta, e o supervisor que realiza a função semelhante a de um controlador convencional. Usando esta teoria e uma extensão de uma classe das redes de Petri, denominada *Função de Habilitação de Transições (RPFHT)*, Barroso [3] elaborou uma abordagem para síntese de supervisores. Neste trabalho são apresentados dois algoritmos: o primeiro é baseado no conhecido *Algoritmo da Árvore de Alcançabilidade* e o segundo, nos algoritmos apresentados por Ramadge e Wonham [41] e Ziller [44]. A síntese do supervisor é realizada executando-se seqüencialmente os algoritmos, que tem como dados o modelo do sistema e o seu comportamento desejado.

Na abordagem de Barroso o modelo, que representa o sistema, não contempla tempo. Modelos que não conteplam tempo só permitem análise de propriedades qualitativas ou lógicas, e não permitem a caracterização das propriedades temporais, essenciais em muitos casos, na modelagem de sistemas em tempo real.

A proposta deste trabalho é reformular a abordagem de síntese de Barroso de forma a permitir a síntese de supervisores, a partir de modelos construídos com redes de Petri temporais, e a implementação dos mesmos em um Controlador Lógico Programável, *CLP*.

O uso de um modelo temporal é motivada pela necessidade de representar os tempos de execução das tarefas de *SED*, que podem ter seus términos indicados por sinais provenientes de sensores. Observa-se que a inclusão de tempo, no modelo que representa a planta, pode tanto alterar a ordem relativa de ocorrência de eventos quanto a inibição de alguns eventos, em alguns estados [18]; isto fica claro quando se compara o espaço de estados obtido considerando-se o disparo das transições dependente somente da marcação, e o obtido considerando-se o disparo dependente também dos intervalos de tempo associados às transições do modelo. Consequentemente, o espaço de estado obtido levando-se em conta também o tempo representa melhor o comportamento do sistema real. Assim, o uso do espaço de estados obtido a partir de um modelo lógico, pode levar a especificações de comportamento desejado que possuam estados que não existirão ao longo do tempo. Isto pode eventualmente levar a erros, na definição do supervisor, que podem ser evitados pelo

uso de modelos temporizados.

Devido ao uso de modelos temporizados são introduzidos um algoritmo para a construção do espaço de estados de um modelo em rede de Petri temporais, o *Algoritmo da Árvore de Alcançabilidade de Classes - AAAC* e uma nova classe de rede de Petri, a *Rede de Petri Temporal Interpretada com Funções de habilitação de Transições - RPTIFHT*, utilizada para modelar o supervisor, ambos fase de síntese do supervisor. Na fase destinada a implementação do supervisor é introduzido um algoritmo utilizado para a elaboração do programa para *CLP*, o *Programa para CLP - AEPCLP*.

Esta abordagem será implementada em um programa, em linguagem C, que a partir de dois arquivos de dados (contendo a descrição do modelo e especificação funcional), gera um programa em lista de instruções para o *CLP*.

Esta Dissertação está estruturada da seguinte forma: No Capítulo 2 serão apresentados os conceitos básicos de linguagens formais, automatos, geradores e a teoria de controle supervisório. No Capítulo 3 são apresentadas as redes de Petri e a abordagem de síntese de supervisores para *SED* de Barroso. No Capítulo 4 são apresentadas algumas extensões das redes de Petri e o estudo das redes *TPN*. No Capítulo 5 a abordagem para a síntese e implementação de supervisores para *SED* que utiliza modelo temporizado do sistema. Finalmente no Capítulo 6 apresentamos as conclusões.

Capítulo 2

Definições Gerais

No procedimento de síntese de supervisores para Sistemas a Eventos Discretos (*SED*), são utilizados conceitos da Teoria de Controle Supervisório, *TCS*. Esta teoria é baseada nos conceitos da Teoria de Linguagens e de Autômatos. Portanto, neste capítulo, iremos apresentar os conceitos da Teoria de Linguagens e de Autômatos e a *TCS* [44, 3, 34, 41, 5].

A primeira seção traz os conceitos básicos da teoria de linguagens, mostrando como um *SED* pode ser representado por uma linguagem. A seção apresenta ainda os conceitos de autômatos e geradores, bem como as linguagens a eles associadas e as propriedades que tornam os geradores adequados à representação de *SEDs*.

Autômatos e geradores de modo geral não admitem a idéia de simultaneidade, de modo que não se pode levar em conta a ocorrência simultânea de dois ou mais eventos distintos. Assume-se portanto que os sistemas tratados aqui apresentam evolução seqüencial. Além disso, a representação por autômatos e geradores não considera os instantes de tempo em que os eventos ocorrem, mas apenas a ordem em que acontecem.

A segunda, e última, seção deste capítulo, apresenta a *TCS*. Veremos que esta teoria distingue claramente a planta a ser controlada de seu controle.

2.1 Linguagens Formais, Autômatos e Geradores

2.1.1 Linguagens Formais

Os eventos que ocorrem em um *SED* são representados por símbolos; por exemplo na Figura 1.2 temos $\alpha, \beta, \gamma, \iota, \rho, \tau, \eta, \sigma$. A partir de um conjunto de símbolos, um *alfabeto* é assim definido:

Definição 2.1 *Ao conjunto não vazio de símbolos, representado por letra grega maiúscula, em geral por* Σ*, denomina-se alfabeto.*

Portanto $\Sigma = \{\alpha, \beta, \gamma, \nu, \rho, \tau, \eta, \sigma\}$ é o alfabeto do *SED* da Figura 1.2, supondo que não existam eventos adicionais não representados na figura.

Assim como na escrita habitual, justapondo-se símbolos do alfabeto de um *SED*, obtem-se uma "palavra" que inicia com comprimento nulo (situação em que o sistema ainda não iniciou a geração).

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

Assim, à medida que um *SED* evolui, ele gera palavras de comprimento crescente, pelas mudanças de estado. Isto nos leva á noção de comprimento de uma palavra que é formalizada como segue:

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

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

É importante observar que $\varepsilon \notin \Sigma$, pois ε é uma palavra e não um símbolo de Σ.

Conjuntos de palavras são estruturas fundamentais para a Teoria de Linguagens. Um conjunto de palavras é definido como:

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

Exemplo 2.1 Dado o alfabeto $\Sigma = \{\alpha, \beta\}$, tem-se:

$$\begin{aligned}
\Sigma^0 &= \{\varepsilon\} \\
\Sigma^1 &= \{\alpha, \beta\} \\
\Sigma^2 &= \{\alpha\alpha, \alpha\beta, \beta\alpha, \beta\beta\} \\
\Sigma^3 &= \{\alpha\alpha\alpha, \alpha\alpha\beta, \alpha\beta\alpha, \alpha\beta\beta, \beta\alpha\alpha, \beta\beta\alpha, \beta\alpha\beta, \beta\beta\beta\}
\end{aligned}$$

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

$$\Sigma^+ = \bigcup_{k=1}^{\infty} \Sigma^k = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ... e$$

$$\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup ...$$

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: $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

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

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, $\Sigma^0 = \{\varepsilon\}$.

O conjunto de todas as palavras que um sistema de alfabeto Σ é capaz de gerar é uma linguagem $L \subseteq \Sigma^*$. Em geral, um sistema não gera qualquer seqüência de eventos de Σ^* que se possa imaginar: escolhendo símbolos de Σ arbitrariamente, é possível construir palavras que não correspondam a seqüência de eventos fisicamente possíveis no sistema. Isto significa que, em geral, a linguagem gerada pelo sistema é um subconjunto próprio[1] de 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 $s_1 = \sigma_1\sigma_2...\sigma_k$ e $s_2 = \sigma_{k+1}\sigma_{k+2}...\sigma_n$, denota-se por s_1s_2, a palavra formada por $\sigma_1\sigma_2...\sigma_k\sigma_{k+1}...\sigma_n$.*

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.*

Exemplo 2.2 Seja o alfabeto $\Sigma = \{\alpha, \beta, \gamma, \nu, \rho, \tau, \eta, \sigma\}$, para o sistema da Figura 1.2, $u = \alpha\beta\gamma\nu$ é um prefixo de $s = \alpha\beta\gamma\nu\rho\tau\eta\sigma$, porque $\exists(\lambda = \rho\tau\eta\sigma) \in \Sigma^*$ tal que $u\lambda = s$.

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

Exemplo 2.3 *No sistema da Figura 1.2, $Pre(\alpha\beta\gamma\nu\rho\tau\eta\sigma) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta\gamma, \alpha\beta\gamma\nu, \alpha\beta\gamma\nu\rho, \alpha\beta\gamma\nu\rho\tau, \alpha\beta\gamma\nu\rho\tau\eta, \alpha\beta\gamma\nu\rho\tau\eta\sigma\}$. Note que a palavra $s = \alpha\beta\gamma\nu\rho\tau\eta\sigma$ é prefixo de si mesma pois a igualdade $\alpha\beta\gamma\nu\rho\tau\eta\sigma\varepsilon = \alpha\beta\gamma\nu\rho\tau\eta\sigma$ satisfaz a condição da Definição 2.9.*

[1] Dados dois conjuntos A e B, diz-se [14, 7] que A é subconjunto próprio de B, ou que A é parte própria de B, ou que A está contido propriamente em B ou que B contém propriamente A, se $A \subset B$ e $A \neq B$, simbolicamente

$$A \subset B \wedge A \neq B \longleftrightarrow (\exists x)\,(x \in B \wedge x \notin A) \wedge (A \subset B)$$

Dada uma linguagem $L \subseteq \Sigma^*$, existe uma linguagem a ela associada, 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:*

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

Como conseqüência desta definição tem-se que:

$$L \subseteq \bar{L} \tag{2.1}$$

Definição 2.12 *Uma linguagem L é prefixo-fechada ou simplesmente fechada se e somente se $L = \bar{L}$.*

Conseqüentemente, se L é prefixo-fechada, então para cada palavra r pertencente a L, vale $\text{Pre}(r) \subseteq L$.

Considerando-se a evolução seqüencial dos *SEDs*, pode-se afirmar que, se um dado sistema produziu uma palavra r, então ele necessariamente produziu anteriormente todas as palavras do conjunto $\text{Pre}(r)$. Em outras palavras, a linguagem gerada por um *SED* incluirá, para cada palavra, também todos os seus prefixos.

Partindo-se disto, tem-se a seguinte proposição:

Proposition 2.1 *O comportamento lógico de qualquer sistema a eventos discretos em que não ocorram eventos simultâneos pode ser representado por uma linguagem prefixo-fechada.*

Definição 2.13 *A linguagem prefixo-fechada que representa o comportamento lógico de um sistema a eventos discretos é denominada linguagem gerada do sistema.*

Após partir do estado inicial e percorrer uma determinada trajetória em seu espaço de estados, um *SED* acabará, de modo geral, completando uma ou mais tarefas. As seqüências de eventos (palavras) que levam a tarefas completadas formam também uma linguagem.

Definição 2.14 *A linguagem que representa o conjunto de tarefas que o sistema a eventos discretos é capaz de executar é denominada linguagem marcada do sistema.*

Cabe observar que, para gerar qualquer palavra desta linguagem, o *SED* em questão tem que gerar todos os seus prefixos. Em outras palavras, um *SED* que produza as palavras contidas numa linguagem (não necessariamente fechada) L_m também produzirá as palavras contidas em $\overline{L_m}$. Se L é a linguagem gerada de um sistema e L_m sua linguagem marcada, então

$$L_m \subseteq \overline{L_m} \subseteq L = \bar{L} \tag{2.2}$$

Existe uma classe importante de linguagens que podem ser expressas pelas chamadas *expressões regulares*. 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 por 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.

Exemplo 2.4 *Sejam $\Sigma = \{\alpha, \beta\}$ e L uma linguagem prefixo-fechada na qual os símbolos α e β ocorrem alternadamente, sendo que α ocorre primeiro. A linguagem L pode ser assim representada*

$$L = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta\alpha, \alpha\beta\alpha\beta, ...\} = \{(\alpha\beta)^*(\alpha + \varepsilon)\}$$

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, 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 seqüencialmente 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 uma função de transição de estados, 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.

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 2.15 *Um autômato determinístico finito ou simplesmente autômato é uma quíntupla*

$$A = (\Sigma, Q, \delta, q_0, Q_m), \text{ na qual:}$$

Σ *é um alfabeto;*

Q *é um conjunto finito de estados;*

$\delta : \Sigma \times Q \to Q$ *é uma função de transição de estados;*

$q_0 \in Q$ *é o estado inicial e*

$Q_m \subseteq Q$ *é o conjunto de estados marcados.*

Definir um autômato significa definir cada um dos elementos da quíntupla apresentada na Definição 2.15. O alfabeto, os conjuntos de estados e o estado inicial são listas de símbolos; já a função de transição pode ser dada em forma de uma lista relacionando os pares de $\Sigma \times Q$ a elementos de Q (v. exemplo 2.5), ou pelos *diagramas de transição de estados*, ou ainda por uma *tabela de transições de estados*.

Um diagrama de transição de estados é um grafo orientado, cujos vértices representam os estados e cujos arcos representam a função de transição:

Definição 2.16 *Dado o autômato* $A = (\Sigma, Q, \delta, q_0, Q_m)$, *o diagrama de transição de estados associado a* A *é o grafo orientado* $G = \{V, W\}$, *onde:*

$$V = Q \text{ e}$$

$$W = \{(p, q, \sigma) : p, q \in Q \wedge \sigma \in \Sigma \wedge \delta(\sigma, p) = q\}.$$

Além disso, estados marcados são caracterizados por vértices desenhados com linhas duplas e o estado inicial é identificado por uma seta. Com isso, obtém-se uma representação gráfica completa para o autômato.

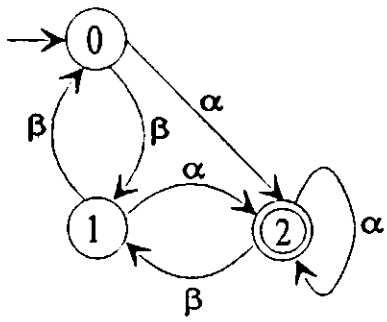


Figura 2.1: Diagrama de transição representando o autômato do exemplo 2.4.

	α	β
0	2	1
1	2	0
2	2	1

Tabela 2.1: Tabela de transição para o autômato do exemplo 2.4

A tabela de transição de estados é uma matriz cujas linhas são enumeradas pelo conjunto de estados do autômato e cujas colunas são enumeradas pelo seu conjunto de transições. O elemento t_{ij} da matriz é o estado para o qual o autômato irá se, no estado i, ocorrer o evento j.

Exemplo 2.5 *Seja um autômato com*

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

De acordo com a definição 2.16, o grafo correspondente é $G = \{V, W\}$, com

$$V = \{0,1,2\} \text{ e}$$
$$W = \{(0,2,\alpha),(0,1,\beta),(1,2,\alpha),(1,0,\beta),(2,2,\alpha),(2,1,\beta)\}.$$

O diagrama de transição correspondente a este autômato é dado na Figura 2.1; a tabela de transição de estado é apresentada na Tabela 2.1.

Conhecidas a função de transição e o estado atual do autômato, é possível determinar seu estado após processar um dado símbolo. Para processar uma cadeia de símbolos (isto é, uma palavra), pode-se simplesmente repetir este procedimento para cada um dos símbolos que a compõe. No entanto, é conveniente estender a definição da função de transição para processar palavras:

Definição 2.17 *Dado um autômato $A = (\Sigma, Q, \delta, q_0, Q_m)$, a função de transição estendida, denotada por $\bar{\delta}$, é a função $\bar{\delta}$: $\Sigma^* \times Q \to Q$ tal que:*

$$\bar{\delta}(\varepsilon, q) = q \text{ e}$$

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

Neste trabalho dispensa-se a notação $\bar{\delta}$, representando a partir daqui a função de transição estendida simplesmente por δ.

Definida esta função, pode-se determinar qual será o estado do autômato após processar uma palavra sobre o seu alfabeto, a partir de um dado estado. Dependendo da palavra escolhida, o estado final a que se chega poderá ou não pertencer ao conjunto de estados marcados. Isto significa que existem dois conjuntos de palavras com respeito a um dado autômato: as que levam do estado inicial a um estado marcado e as que levam do estado inicial a um estado não marcado. Isto é motivo para a seguinte definição:

Definição 2.18 *Dado um autômato $A = (\Sigma, Q, \delta, q_0, Q_m)$, diz-se que A reconhece a palavra $s \in \Sigma^*$ se e somente se $\delta(s, q_0) \in Q_m$.*

Exemplo 2.6 *O autômato do exemplo 2.5 reconhece todas as palavras sobre o alfabeto $\Sigma = \{\alpha, \beta\}$ que terminam com o símbolo α.*

Definição 2.19 *Dado um autômato $A = (\Sigma, Q, \delta, q_0, Q_m)$, a linguagem marcada (ou linguagem aceita ou ainda linguagem reconhecida) de A, denotada por $L_m(A)$, é:*

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

A linguagem marcada de um autômato é o conjunto de todas as palavras por ele reconhecidas. Note que a linguagem marcada é, no caso geral, uma linguagem não prefixo-fechada.

Exemplo 2.7 *O autômato do exemplo 2.5 reconhece a linguagem $\{(\alpha^*\beta^*)^*\alpha\}$.*

Viu-se que as tarefas que um dado *SED* é capaz de completar podem ser representadas por uma linguagem, denominada linguagem marcada do sistema. Conclui-se daí que é possível representar um *SED* de linguagem marcada L_m por um autômato A tal que $L_m(A) = L_m$.

É também desejável ter uma representação por máquinas de estados finitos para a linguagem gerada do sistema. No entanto, os autômatos definidos acima não possuem um mecanismo que permita fazê-lo. 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 $\Sigma \times Q$. Diz-se então que δ é uma *função parcial*. Embora se possa falar de "autômatos com função de transição parcial", utiliza-se aqui o termo "gerador" para denominar esta classe de máquinas, discutida a seguir.

2.1.3 Geradores

Definição 2.20 *Um gerador é uma quíntupla $G = (\Sigma, Q, \delta, q_0, Q_m)$, na qual Σ, Q, q_0 e Q_m já foram definidos na definição 2.15.*

A única diferença entre autômatos e geradores é o fato de que nestes a função de transição pode ser parcial, ou seja, definida apenas para um subconjunto de eventos para cada estado do gerador. Assim como os autômatos, os geradores podem também ser representados por diagramas de transição, com a diferença de que, neste caso, estão presentes apenas os arcos para os quais a função de transição esta definida.

Definição 2.21 *Dado um gerador $G = (\Sigma, Q, \delta, q_0, Q_m)$, associa-se a cada estado $q \in Q$ o conjunto de eventos definidos $\Sigma(q)$, dado por*

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

em que $\delta(\sigma, q)!$ significa "δ está definida para o par (σ, q)".

Este fato tem várias conseqüências, a começar pela função de transição estendida, que só pode ser definida para aquelas palavras que, quando processadas pelo gerador, se mativerem dentro dos limites de definição da função de transição:

Definição 2.22 *Dado um gerador $G = (\Sigma, Q, \delta, q_0, Q_m)$, a função de transição estendida, denotada por $\bar{\delta}$, é uma função $\bar{\delta}$: $\Sigma^* \times Q \to Q$ tal que:*

$$\bar{\delta}(\epsilon, q) = q \text{ e}$$
$$\bar{\delta}(s\sigma, q) = \delta(\sigma, \bar{\delta}(s, q)), \text{ para } q \in Q \text{ e } s \in \Sigma^* \text{ sempre que}$$
$$q' = \bar{\delta}(s, q) \text{ e } \delta(\sigma, q') \text{ estiverem ambos definidos.}$$

Da mesma forma, como foi 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 2.23 *Dado um gerador $G = (\Sigma, Q, \delta, q_0, Q_m)$, a linguagem gerada de G, denotada por $L(G)$, é:*

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

Enquanto os autômatos, com sua função de transição completa, podem processar qualquer palavra sobre seu alfabeto, o caráter parcial da função de transição dos geradores faz com que a linguagem gerada seja, em geral, um subconjunto próprio de Σ^*. Além disso, esta linguagem é prefixo-fechada, como mostrado a seguir.

Proposition 2.2 *Dado um gerador $G = (\Sigma, Q, \delta, q_0, Q_m)$, a linguagem gerada $L(G)$ é prefixo-fechada.*

Dem. : Seja uma palavra qualquer $s : s \in L(G)$. Então $\exists q \in Q : \delta(s, q_0) = q$. Considerando que q é necessariamente alcançado através de outro estado $q' \in Q$, pode-se afirmar que s pode ser escrita na forma $s = s'\sigma$, com $\sigma \in \Sigma$ e $s' \in \text{Pre}(s)$ e que valem as expressões $\delta(s', q_0) = q'$ e $\delta(\sigma, q') = q$. Procedendo recursivamente desta forma, chega-se ao estado inicial q_0, mostrando que $\text{Pre}(s) \in L(G)$.

Portanto, para todo gerador G vale:

$$L(G) = \overline{L(G)} \tag{2.3}$$

De forma análoga ao caso dos autômatos, o conjunto de estados marcados dos geradores pode ser utilizado para distinguir dois conjuntos de palavras geradas:

Definição 2.24 *Dado um gerador $G = (\Sigma, Q, \delta, q_0, Q_m)$, a linguagem marcada de G (ou linguagem aceita ou ainda linguagem reconhecida por G), denotada por $L_m(G)$, é:*

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

Portanto, dado um *SED* de linguagem gerada L e de linguagem marcada L_m, pode-se representá-lo por um gerador G tal que $L(G) = L$ e $L_m(G) = L_m$. Assim, os geradores permitem representar ambas as linguagens associadas a um *SED*, o que não era possível no caso dos autômatos.

Pelas definições 2.23 e 2.24 é claro que

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

Também é fácil ver que, se duas linguagens quaisquer K e L são tais que $K \subseteq L$, então $\overline{K} \subseteq \overline{L}$. Portanto, tomando o prefixo-fechamento de ambos os membros de 2.4,vem:

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

e, pela equação 2.3,

$$\overline{L_m(G)} \subseteq L(G) \tag{2.6}$$

Considerando-se as equações (2.1) a (2.6), tem-se:

$$L_m(G) \subseteq \overline{L_m(G)} \subseteq L(G) = \overline{L(G)} \tag{2.7}$$

É interessante comparar esta equação com a equação (2.2).

Note que a definição 2.20 não impõe qualquer restrição à estrutura do gerador. Em particular, é possível definir um gerador com estados inacessíveis, isto é, estados que jamais podem ser alcançados a partir do estado inicial. Tais estados formam "ilhas" que não são ligadas ao estado inicial por qualquer caminho. Isto é motivo para a seguinte definição:

Definição 2.25 *A componente acessível do gerador* $G = (\Sigma, Q, \delta, q_0, Q_m)$ *é*

$$A_c(G) = (\Sigma, Q_{ac}, \delta_{ac}, q_0, Q_{ac,m}), \text{ com}$$

$$\begin{aligned} Q_{ac} &= \{q : \exists w \in \Sigma^* \wedge \delta(w, q_0) = q\}, \\ Q_{ac,m} &= Q_{ac} \cap Q_m \text{ e} \\ \delta_{ac} &= \delta|(\Sigma \times Q_{ac}), \end{aligned}$$

em que $\delta_{ac} = \delta|(\Sigma \times Q_{ac})$ é a função δ restrita ao domínio $\Sigma \times Q_{ac}$.

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

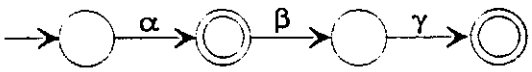
Uma outra propriedade que um gerador pode ou não ter é a coacessibilidade:

Definição 2.26 *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* $L(G) \subseteq \overline{L_m(G)}$.

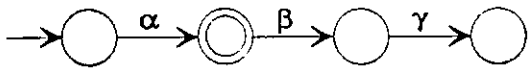
Esta definição diz que um gerador é coacessível se e somente se, a partir de qualquer um de seus estados, existir pelo menos um caminho que leve a um estado marcado. Considerando-se que a equação (2.6) vale para todo gerador G, é possível substituir a inclusão na Definição 2.26 pela igualdade. Portanto, um gerador é coacessível se e somente se

$$L(G) = \overline{L_m(G)} \tag{2.8}$$

Exemplo 2.8 *A Figura 2.2 mostra dois geradores, o primeiro coacessível e o segundo não coacessível. A linguagem gerada por ambos é* $\{\alpha, \alpha\beta, \alpha\beta\gamma\}$*. Qualquer palavra desta linguagem é um prefixo de uma palavra marcada no primeiro gerador, cuja linguagem marcada é* $\{\alpha, \alpha\beta\gamma\}$*, mas não no segundo: as palavras* $\alpha\beta$ *e* $\alpha\beta\gamma$ *não são prefixos de* $\{\alpha\}$.



(a)



(b)

Figura 2.2: Geradores: (a) coacessível, (b) não coacessível.

Definição 2.27 *Um gerador que é ao mesmo tempo acessível e coacessível é dito ajustado (trim).*

2.2 Teoria de Controle Supervisório

Mostrou-se na seção 2.1.3 que um sistema a eventos discretos (*SED*) de linguagem gerada L e linguagem marcada L_m pode ser representado por um gerador G tal que a linguagem gerada $L(G) = L$ e a linguagem marcada $L_m(G) = L_m$. Todavia, sabe-se da teoria de linguagens que existem linguagens não representáveis por autômatos ou geradores finitos. Através da teoria de linguagens, prova-se ainda que a classe de linguagens representáveis por autômatos ou geradores finitos é a classe de linguagens regulares.

Toda teoria e resultados da *TCS*, desenvolvida 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 definidos se limitem àqueles representados por geradores finitos. O exemplo a seguir ilustra um *SED* simples modelado por um gerador finito.

Exemplo 2.9 *A Figura 2.3 mostra um gerador G que modela a operação de uma máquina com três estados possíveis, a saber R (em repouso), A (em atividade) e M (em manutenção). São quatro as transições entre estados, cada uma identificada por um evento do alfabeto* $\Sigma = \{\alpha, \beta, \lambda, \mu\}$. *A máquina está inicialmente no estado de repouso, do qual pode passar ao estado ativo* (α) *e deste voltar ao repouso* (β) *ou então sofrer uma pane* (λ) *e entrar em manutenção, de onde voltará eventualmente à condição de repouso* (μ).

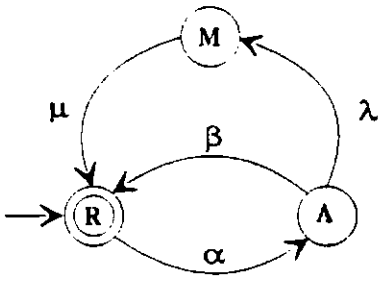


Figura 2.3: Gerador representando uma máquina com três estados

A linguagem gerada, $L(G)$, é o conjunto de todas as palavras obtidas partindo-se do estado inicial R e seguindo o grafo. A expressão regular que representa esta linguagem pode ser escrita como:

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

Como o único estado marcado de G é o inicial, a linguagem marcada $L_m(G)$ compreende as palavras que representam um ciclo completo no grafo, seja pelo caminho $\alpha\beta$ ou pelo caminho $\alpha\lambda\mu$:

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

A linguagem $L(G)$ representa o comportamento fisicamente possível do sistema modelado, enquanto a linguagem $L_m(G)$ representa o conjunto de tarefas que o sistema é capaz de executar. Nesse caso, pode-se utilizar a definição de coacessibilidade de um gerador como critério para a existência ou não de bloqueio no sistema: num gerador coacessível, $L(G) = \overline{L_m(G)}$ (equação 2.8), significando que toda palavra gerada é prefixo de alguma palavra marcada, ou seja, toda seqüência de eventos fisicamente possível tem pelo menos uma continuação que leva a uma tarefa completa. O gerador representado na Figura 2.3 é coacessível.

Dentro desta interpretação, um gerador coacessível e o sistema por ele representado são ditos *não bloqueantes.*

2.2.1 Geradores Controlados

O modelo de um *SED* como descrito até aqui, por exemplo o gerador da Figura 2.3, é simplesmente um gerador espontâneo de cadeias de eventos, sem 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. Para introduzir um mecanismo de controle no gerador, admite-se que existe um conjunto de eventos $\Sigma_c \subseteq \Sigma$, denominados *eventos controláveis* e que podem ser inibidos, isto é, impedidos de ocorrer por um agente externo. Os demais eventos são ditos *não controláveis* e considerados permanentemente habilitados. Denota-se este conjunto por Σ_u, de maneira que valem as relações:

$$\Sigma = \Sigma_u \cup \Sigma_c$$

e

$$\Sigma_u \cap \Sigma_c = \emptyset$$

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.

Esta partição do alfabeto de eventos reflete características de sistemas reais. Por exemplo, o início de operação de uma máquina e o envio de uma mensagem num sistema de comunicação, são em geral, controláveis, ao contrário de uma pane, da perda de uma mensagem ou o término de certas operações, geralmente modelados como eventos não controláveis.

Para que seja possível interferir no funcionamento do gerador, este precisa ser dotado de uma interface através da qual se possa informar quais eventos devem ser habilitados e quais

devem ser inibidos, em um determinado estado. Convenciona-se especificar o conjunto de eventos que se deseja habilitar, o qual recebe o nome de *entrada de controle*. Formalmente, tem-se:

Definição 2.28 *Dado um gerador* $G = (\Sigma, Q, \delta, q_0, Q_m)$, *cujo alfabeto é particionado em* $\Sigma = \Sigma_u \cup \Sigma_c$, *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 2.29 *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.

Por analogia com a teoria de controle clássica, utiliza-se o termo *planta* para designar o sistema a ser controlado. A linguagem gerada pelo gerador que a representa é denominada *linguagem da planta* e representa o comportamento do sistema na ausência de qualquer ação de controle que possa restringir sua operação.

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. É este o princípio de funcionamento do mecanismo de controle adotado pela teoria de controle supervisório, que consiste em chavear as entradas de controle em resposta ao comportamento observado do sistema, de modo a restringir a linguagem gerada a uma dada especificação.

Considerando o gerador da Figura 2.3, 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, apresentamos a seguir a noção de supervisor e as condições para a existência do mesmo.

2.2.2 Supervisores

Um supervisor é um agente que tem a capacidade de observar os eventos gerados por uma planta, um gerador G, e, em resposta, determinar entradas de controle $\gamma, \gamma', \gamma'', \ldots \in$

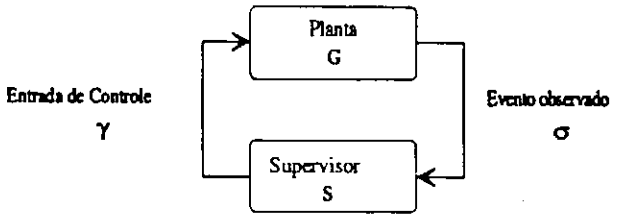


Figura 2.4: Supervisão de um *SED*.

Γ, as quais são realimentadas à planta, determinando o comportamento do sistema sob supervisão. Formalmente tem-se:

Definição 2.30 *Um supervisor para um gerador controlado* $G_c = (G, \Gamma)$ *é um par* $\mathrm{S} = (S, \Phi)$, *composto de um gerador* $S = (\Sigma, X, \xi, x_0, X_m)$ *e de um mapa de controle* Φ, *no qual:*

- Σ é o mesmo alfabeto do gerador G;
- X é um conjunto de estados;
- $\xi : \Sigma^* \times X \to X$ 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
- $\Phi : X \to \Gamma$ é uma função que associa a cada $x \in X$ uma entrada de controle $\gamma \in \Gamma$.

A Figura 2.4 apresenta o modelo do sistema composto em malha fechada, do gerador controlado G_c supervisionado pelo supervisor S.

A especificação de um alfabeto idêntico para S e G é necessária para que se possa interconectá-los como na Figura 2.4. Com isso, o supervisor pode ser construído de modo a rastrear as palavras geradas pela planta, evoluindo no seu espaço de estados de acordo com a função ξ. A função Φ, denominada *mapa de controle*, associa a cada estado $x \in X$ uma entrada de controle $\gamma \in \Gamma$ para aplicar em G.

Linguagens

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 sob supervisão é formalizado pelas linguagens definidas a seguir:

Definição 2.31 *Dados uma planta* $G_c = (G, \Gamma)$ *e um supervisor* $\mathrm{S} = (S, \Phi)$, *a linguagem gerada pelo sistema sob supervisão, denotada por* $L(S/G)$, *é definida pelas seguintes asserções:*

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

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

A Definição 2.31 diz que os eventos que podem ocorrer sob supervisão são os eventos fisicamente possíveis e habilitados. Dado que uma palavra s pertence a $L(S/G)$ somente se pertence a $L(G)$, então

$$L(S/G) \subseteq L(G) \tag{2.9}$$

Dado que uma palavra s pertence a $L(S/G)$ somente se $s \in L(S/G)$, pode-se afirmar que $L(S/G)$ é prefixo-fechada, isto é:

$$L(S/G) = \overline{L(S/G)} \tag{2.10}$$

Definição 2.32 *Dados uma planta* $G_c = (G, \Gamma)$ *e um supervisor* $\mathrm{S} = (S, \Phi)$, *a linguagem controlada do sistema sob supervisão, denotada por* $L_c(S/G)$, *é definida como:*

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

A linguagem controlada é simplesmente a parte da linguagem marcada original que "sobrevive" sob a ação de controle: Se $L_m(G)$ representa as tarefas que podem ser completadas pela planta, então $L_c(S/G)$ representa as tarefas que podem ser completadas sob supervisão. Em conseqüência desta definição,

$$L_c(S/G) \subseteq L(S/G) \tag{2.11}$$

e

$$L_c(S/G) \subseteq L_m(G) \tag{2.12}$$

Convenciona-se que uma palavra gerada no sistema em malha fechada é reconhecida se e somente se tanto G quanto S a reconhecerem. Isto dá origem à terceira linguagem associada à planta supervisionada:

Definição 2.33 *Dados uma planta* $G_c = (G, \Gamma)$ *e um supervisor* $\mathrm{S} = (S, \Phi)$, *a linguagem marcada do sistema sob supervisão, denotada por* $L_m(S/G)$, *é definida como:*

$$L_m(S/G) = L_c(S/G) \cap L_m(S). \tag{2.13}$$

As equações 2.9, 2.11 e 2.13 nos levam à seguinte conclusão:

$$L_m(S/G) \subseteq L_c(S/G) \subseteq L(S/G) \subseteq L(G) \tag{2.14}$$

Supervisores Próprios

Para que um supervisor seja capaz de rastrear o funcionamento da planta, exige-se que sua função de transição seja definida para todo evento habilitado fisicamente possível. Embora se possa, teoricamente, conceber supervisores em que isto não aconteça sem contrariar a definição 2.30, só terão sentido prático aqueles concebidos para acompanhar a evolução da planta. Formalmente, temos:

Definição 2.34 *Um supervisor* $\mathrm{S} = (S, \Phi)$ *para uma planta* $G_c = (G, \Gamma)$ *é dito completo quando,* $\forall s \in \Sigma^* \wedge \sigma \in \Sigma$, *as três condições seguintes são verdadeiras:*

$$(a)\ s \in L(S/G),$$
$$(b)\ s\sigma \in L(G)\ e$$
$$(c)\ \sigma \in \Phi(\xi(s, x_0))$$

e juntas implicam em

$$(d)\ \xi(s\sigma, x_0)!,\ \textit{isto é,}\ \xi(s\sigma, x_0)\ \textit{éé definido.}$$

Pela definição 2.34, 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 $L(S/G)$, $L_c(S/G)$ e $L_m(S/G)$. As mesmas são assim definidas:

Definição 2.35 *Um supervisor* $\mathrm{S} = (S, \Phi)$, *para uma planta* $G_c = (G, \Gamma)$ *é dito não bloqueante se e somente se*

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

Definição 2.36 *Um supervisor* $\mathrm{S} = (S, \Phi)$ *para a planta* $G_c = (G, \Gamma)$ *é dito não rejeitante se e somente se*

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

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

Naturalmente, é sempre desejável construir supervisores não bloqueantes e não rejeitantes. Estes conceitos são aliados ao de supervisor completo (Definição 2.34), na seguinte definição:

Definição 2.37 *Um supervisor completo, não bloqueante e não rejeitante é dito supervisor próprio. Como conseqüência, um supervisor próprio é um supervisor completo tal que*

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

2.2.3 Condições de Existência de Supervisores

A utilidade do mecanismo de controle, proposto na teoria de controle supervisório, consiste em se determinar sob que condições existe um supervisor que realize uma dada especificação e, caso exista, como encontrá-lo. 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 de *fechamento* e *controlabilidade*.

Definição 2.38 *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 2.39 *Dadas duas linguagens arbitrárias $K, L \subseteq \Sigma^*$ 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:

Proposition 2.3 *Seja* S *um supervisor completo para G_c. Então $L(S/G)$ é prefixo-fechada e $L(G)$-controlável.*

Dem.: Seja $S = (S, \Phi)$ um supervisor completo para a planta $G_c = (G, \Gamma)$. O prefixo-fechamento de $L(S/G)$ decorre de sua definição (ver equação 2.10).

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

$s\sigma \in L(S/G)\,\Sigma_u \cap L(G)$,	isto é,
$s\sigma : s \in L(S/G) \wedge \sigma \in \Sigma_u \wedge s\sigma \in L(G)$.	Por ser σ não controlável,
$s\sigma : s \in L(S/G) \wedge \sigma \in \Phi(\xi(s, x_0)) \wedge s\sigma \in L(G)$	e, sendo S completo,
$s\sigma \in L(S/G)$,	$\forall$ palavra $s\sigma$, ou seja,
$L(S/G)\,\Sigma_u \cap L(G) \subseteq L(S/G)$.	

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

Teorema 2.4 *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* S *tal que $L(S/G) = K$ se e somente se K for prefixo-fechada e $L(G)$-controlável.*

Teorema 2.5 *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 bloqueante S tal que $L_c(S/G) = K$ se e somente se K for $L_m(G)$-fechada e $L(G)$-controlável.

2. O supervisor S será próprio se e somente se o gerador S for tal que $X_m = X$.

As demonstrações desses teoremas podem ser encontradas em [33] e [44].

Para problemas formulados em termos de linguagens geradas, o Teorema 2.4 estabelece as condições necessárias e sufucientes para a existência do supervisor S, o gerador S e o mapa Φ podem ser encontrados a partir das equações

$$L(S) = K \text{ e} \tag{2.15}$$

$$\Phi(x) = \gamma : \gamma \in \Gamma \wedge \gamma \supseteq \Sigma_x^1 \wedge \gamma \cap \Sigma_x^0 = \emptyset \tag{2.16}$$

em que 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 pertençam a $\Phi(x)$. 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 $\sigma \in \Phi(x)$.

Para problemas formulados em termos de linguagens marcadas, se K satisfaz às condições do Teorema 2.5, então o supervisor é tal que $L(S/G) = \overline{K}$, visto que nesse caso, $L_c(S/G) = \overline{K} \cap L_m(G) = K$.

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 sublinguagem controlável* $K^{\uparrow}$ ou $\sup C(L)$ [41].

Desta forma, se $K^{\uparrow}$ solucionar de forma satisfatória o problema, isto é, se $K \supseteq A$, em que A é uma linguagem que representa o comportamento mais retrito 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 estados finitos, $K^{\uparrow}$ é sempre computável [41].

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 $K^{\uparrow}$ pode ser a linguagem vazia.

Para o caso em que o 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 [41].

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

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

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

Nos casos em que 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. O restante desta seção trata da existência de uma solução minimamente restritiva para os casos em que E deixe de satisfazer essas condições.

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

$$C(E) = \left\{K' : K' \subseteq E \wedge K'\Sigma_u \cap L(G) \subseteq K'\right\} \tag{2.17}$$

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

$$F(E) = \left\{K' : K' \subseteq E \wedge \overline{K'} \cap L_m(G) = K'\right\} \tag{2.18}$$

Para esclarecer a terminologia empregada a seguir, considere a seguinte definição:

Definição 2.40 *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),*

$$w_1, w_2 \in W \Longrightarrow w_1 Z w_2 \in W.$$

Teorema 2.6 *Seja* $H(K) = \left\{K' : K' \subseteq K \wedge \left[\forall K'_1, K'_2 \in H(K)\right] K'_1 \cup K'_2 \in H(K)\right\}$ *um conjunto não vazio de subconjuntos de* K*, fechado sob a operação de união. Então existe em* $H(K)$ *o elemento supremo*

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

Dem.: Para demonstrar o 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, e o segundo da hipótese de ser $H(K)$ fechado sob essa operação. Além disso, sup $H(K)$ está sempre definido, pois $H(K)$ é, por hipótese, não vazio.

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

Com isto, pode-se apresentar a seguinte proposição:

Proposition 2.7 $C(E)$ *e* $F(E)$ *são não vizinhos 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 A para enumerar os membros desse conjunto, de modo que

$$K_\alpha \in C(E),\ \alpha \in A$$

Sendo os K_α controláveis, vale:

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

então:

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

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 B para enumerar os membros desse conjunto, de modo que

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

Sendo os K_β $L_m(G)$-fechados, vale:

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

então:

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

Logo a união de quaisquer elementos de $F(E)$ é $L_m(G)$-fechada e pertence a $F(E)$.

Corolário 2.1 *Existem em $C(E)$ e $F(E)$, respectivamente, os elementos supremos*

$$supC(E) = \cup\{K : K \in C(E)\} \text{ e}$$
$$supF(E) = \cup\{K : K \in F(E)\}.$$

Dem.: Pelo teorema 2.6.

Os elementos sup$C(E)$ e sup$F(E)$ são denominados respectivamente de *máxima sublinguagem* $L_m(G)$-controlável e de *máxima sublinguagem* $L_m(G)$-fechada de E.

Além disso, vale a seguinte proposição:

Proposition 2.8 *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 ver o fechamento sob união, considere duas linguagens quaiquer $K_1, K_2 \in C(E) \cap F(E)$. 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, vem:

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

e assim

$$K_1 \cup K_2 \in C(E) \cap F(E).$$

Corolário 2.2 *Existe em $C(E) \cap F(E)$ o elemento supremo $supCF(E) = \cup\{K : K \in C(E) \cap F(E)\}$.*

Dem.: Pelo Teorema 2.6.

O elemento $supCF(E)$ é denominado de *máxima sublinguagem* $L(G)$-controlável e $L_m(G)$-fechada de E.

Pelo Corolário 2.2, o teorema seguinte é demonstrado.

Teorema 2.9 *O problema de controle supervisório tem solução se e somente se $supCF(E) \supseteq A$, sendo $supCF(E)$ 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 2.9 e pela definição 2.32,

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

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

$$\forall E : E \subseteq L_m(G),\ E = supF(E)$$

e

$$supCF(E) = supC(E).$$

Com isso, o problema de controle supervisório tem a seguinte versão para linguagens geradas:

Dados um gerador G tal que $L_m(G)$, 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 2.5 dado que $L(S/G) \subseteq K$.

Teorema 2.10 *O problema de controle supervisório para linguagens geradas tem solução se e somente se* $supC(E) \supseteq A$, *sendo* $supC(E)$ *a solução minimamente restritiva.*

Cabe observar ainda que caso todos os estados do gerador sejam marcados, significa que qualquer supervisor obtido será *não-bloqueante.*

Capítulo 3

Síntese de Supervisores de *SEDs*

Como foi dito no capítulo 1, não existe uma metodologia de projeto de controladores de sistemas a eventos discretos (*SED*), sistematizada e consolidada como existe para os sistemas contínuos. A abordagem de Barroso [3] é a metodologia que serve de base para o presente trabalho. Esta abordagem utiliza a Teoria de Controle Supervisório (*TCS*), apresentada na seção 2.2 e uma extensão de uma classe de rede de Petri denominada *Rede de Petri com Função de Habilitação de Transição (RPFHT)*. Para a compreensão desta abordagem falta-nos, a esta altura, o conhecimento da *RPFHT*, e dos algoritmos utilizados para a síntese do supervisor. Neste capítulo apresentaremos os conceitos básicos de redes de Petri para então introduzirmos as *RPFHT* e os algoritmos de síntese.

3.1 Redes de Petri

O conceito de rede de Petri foi originalmente apresentado na dissertação de Carl Adam Petri, submetida em 1962 a Universidade Técnica de Darmstadt, Oeste da Alemanha. A rede, que é uma ferramenta gráfica e matemática, foi originalmente desenvolvida para modelar e analisar sistemas de comunicação, que têm como características a concorrência, o paralelismo, o assincronismo, a distribuição e o não determinismo. No entanto ela mostrou-se aplicável a muitos outros tipos de sistemas, como por exemplo os sistemas de manufatura.

Uma rede de Petri é um tipo particular de grafo direcionado, ponderado e bipartido, composto por três tipos de objetos e com um estado inicial chamado *marcação inicial*, M_0. Esses objetos são *lugares, transições e arcos* direcionados, que ligam lugares a transições e transições a lugares. Um lugar é representado por um círculo e uma transição por uma barra ou retângulo. Aos arcos são atribuidos pesos (inteiros positivos), onde um arco w-ponderado deve ser entendido como um conjunto de w arcos paralelos ligando um lugar a uma transição ou vice-versa.

Para o estudo do comportamento dinâmico de uma rede, em termos de seus estados

e suas mudanças, cada lugar pode conter um número inteiro positivo ou zero de fichas (*tokens*), representadas por pontos. A marcação da rede, M, é um vetor coluna de m-elementos, onde m é o número total de lugares. Os elementos de M, representados por $M(p)$, são os números de fichas no correspondente lugar p.

Os lugares podem comportar um número finito ou infinito de fichas. As redes de Petri em cujos lugares podem ser colocados um número ilimitado de fichas denomina-se *rede de Petri com capacidade infinita*. De outra forma, uma rede em cujos lugares podem ser colocados um número limitado de fichas, denomina-se *rede de Petri com capacidade finita*.

Definição 3.1 *Uma rede de Petri marcada é definida como um conjunto,*

$$RP = (P, T, I, O, K, M_0), \text{ no qual:}$$

- $P = \{p_1, p_2, ..., p_m\}$ é um conjunto finito de m lugares,
- $T = \{t_1, t_2, ..., t_n\}$ é um conjunto finito de n transições, com $P \cup T \neq \emptyset$ e $T \cap P = \emptyset$,
- $I : P \times T \rightarrow \mathbf{N}$ é uma função que especifica os arcos que ligam lugares a transições,
- $O : T \times P \rightarrow \mathbf{N}$ é uma função que especifica os arcos que ligam transições a lugares,
- $K : P \rightarrow \mathbf{N} \cup \{\infty\}$ é a função de capacidade,
- $M_0 : P \rightarrow \mathbf{N}$ é a marcação inicial,

Definição 3.2 *Uma estrutura de rede de Petri, sem nenhuma marcação inicial, é denotada por $N = (P, T, I, O)$. 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 $RP = (N, M_0)$.*

A Figura 3.1(a) representa uma estrutura de rede de Petri com sete lugares, seis transições e quinze arcos direcionados. Nela o conjunto de lugares é $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ e o de transições é $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. Acrescentando uma marcação a esta estrutura, obtemos a rede de Petri marcada da Figura 3.1(b). Nesta, a marcação inicial é $M_0 = [1, 0, 1, 0, 0, 2, 0]^T$.

Definição 3.3 *Uma rede de Petri ordinária, é uma rede na qual todos os seus arcos tem peso igual a um.*

Por esta definição, uma rede ordinária possui o mapeamento de seus arcos da seguinte forma:

$$I : P \times T \rightarrow \{0, 1\} \text{ e}$$
$$O : T \times P \rightarrow \{0, 1\}.$$

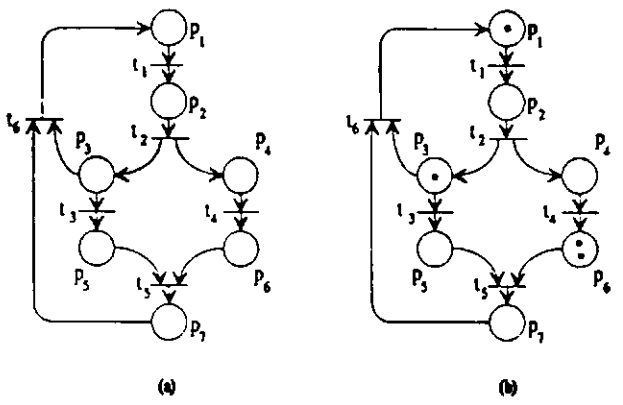


Figura 3.1: Rede de Petri. (a) Estrutura da rede de Petri. (b) Rede de Petri Marcada (David [12])

Tanto lugares quanto transições possuem conjuntos de entrada (pré-condições) e saída (pós-condições), definidos como:

$$
\begin{aligned}
{}^{\bullet}p &= \{t \in T | (t,p) \in O\}, \text{ é o conjunto de transições de entrada do lugar } p. \\
p^{\bullet} &= \{t \in T | (p,t) \in I\}, \text{ é o conjunto de transições de saída do lugar } p. \\
{}^{\bullet}t &= \{p \in P | (p,t) \in I\}, \text{ é o conjunto de lugares de entrada da transição } t. \\
t^{\bullet} &= \{p \in P | (t,p) \in O\}, \text{ é o conjunto de lugares de saída da transição } t.
\end{aligned}
$$

Na Figura 3.1, p_1 é lugar de entrada para t_1 assim como p_6 é para t_5. O lugar p_5 é lugar de saída para t_3 e p_1 é para t_6. A transição t_6 é transição de entrada de p_1 assim como t_3 é para p_5. A transição t_5 é transição de saída de p_5 e p_6.

O par formado por uma transição t e um lugar p, onde p é tanto lugar de entrada como de saída de t, é chamado *auto-laço.*

Definição 3.4 *Uma rede de Petri é dita ser pura se ela não possui auto-laços.*

De modo geral evita-se trabalhar com redes de Petri que contenham auto-laços. Desse modo, uma rede de Petri que contenha *auto-laço* pode ser convertida para uma rede de Petri pura como mostrado na Figura 3.2.

Uma transição sem nenhum lugar de entrada é denominada de *transição fonte (source)*, que está sempre habilitada (ver regra de habilitação a seguir). Uma transição que não possui lugares de saída é chamada *transição sorvedouro (sink)*, que consome fichas mas não produz nenhuma.

O comportamento de muitos sistemas pode ser descrito em termos da mudança de seus estados. Para simular o comportamento dinâmico de um sistema, representado por uma rede de Petri, um estado ou marcação é mudada de acordo com as regras de habilitação e disparo das transições, dadas a seguir:

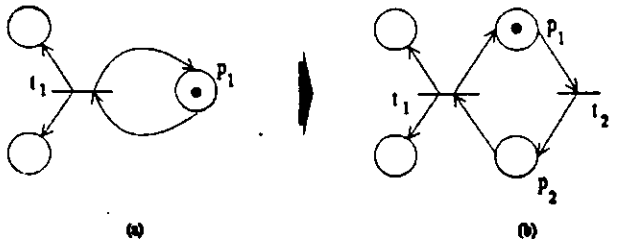


Figura 3.2: (a) Rede com *auto-laço* e (b) Conversão para uma rede sem *auto-laço*.(Zhou [43]).

Regra de Habilitação: uma transição $t \in T$ está habilitada se

$$\forall p \in {}^{\bullet}t : M(p) \geq I(p,t) \text{ e}$$
$$\forall p \in t^{\bullet} : M(p) \leq K(p) - O(t,p).$$

Isto é, uma transição está habilitada se cada um de seus lugares de entrada contém um número de fichas maior ou igual ao peso do arco que os liga e se a capacidade de cada um de seus lugares de saída não for excedida pelo acréscimo das fichas decorrentes do disparo.

Regra de Disparo: uma transição habilitada pode ou não disparar (dependendo da ocorrência ou não do evento atual). Uma vez disparada, a transição retira de cada lugar de entrada $I(p,t)$ fichas e acrescenta $O(t,p)$ fichas a cada lugar de saída. A ocorrência do disparo de t, que modifica a marcação M para uma nova marcação M', é denotada por $M[t > M'$, ou (em analogia à função de transição de estado δ dos autômatos) $M' = \delta(M,t)$. A marcação M' resultante do disparo de t na marcação M é:

$$M'(p) = \begin{cases} M(p) - I(p,t), \text{ se } p \in {}^{\bullet}t \\ M(p) + O(t,p), \text{ se } p \in t^{\bullet} \\ M(p), \text{ se } p \notin {}^{\bullet}t \wedge p \notin t^{\bullet} \end{cases}$$

As regras de habilitação e de disparo são ilustradas na Figura 3.3. Essas redes possuem capacidade infinita. As transições na Figura 3.3(a, b e c) antes do disparo estão habilitadas pois em cada caso os lugares p_1 e p_2 contêm um número de fichas igual ou superior ao peso dos arcos. Mas este não é o caso para o exemplo da Figura 3.3(d), no qual a transição t_1 não está habilitada devido a quantidade de fichas em p_2 ser inferior ao peso do arco.

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

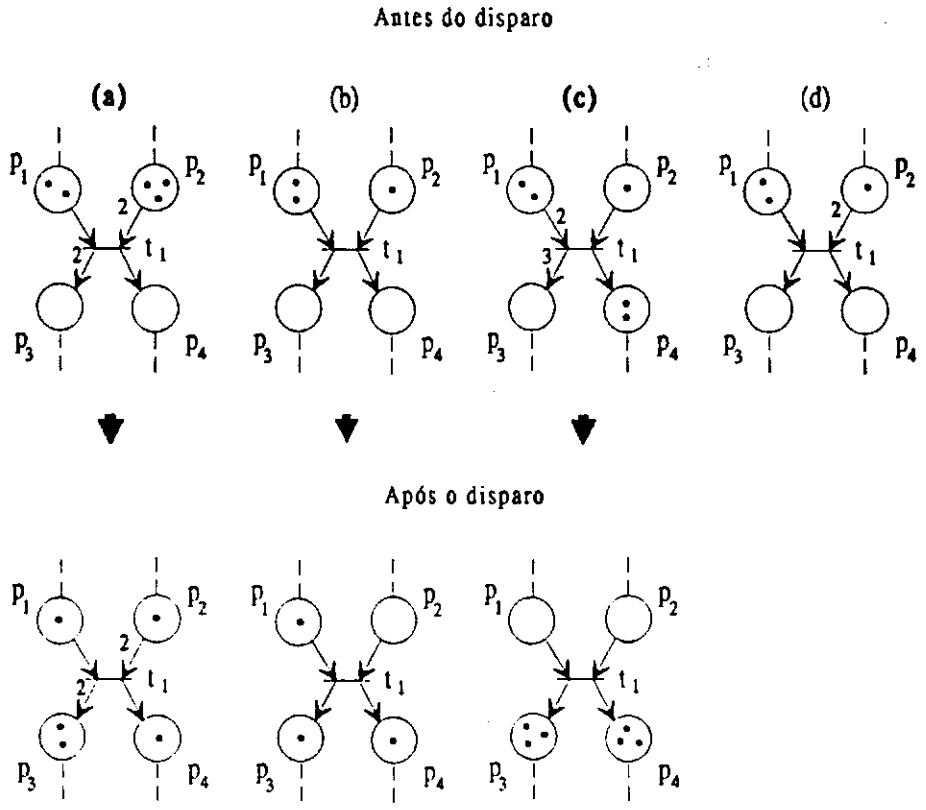


Figura 3.3: Ilustração da regra de disparo de uma transição.

de transições e M_0, pode-se facilmente encontrar a seqüência de marcações pela execução de uma rede de Petri.

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 $t_{j0}, t_{j1}, ..., t_{jk}$ e uma marcação M, a marcação

$$M' = \delta\left(M, t_{j1}, t_{j2}, ..., t_{jk}\right)$$

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

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

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

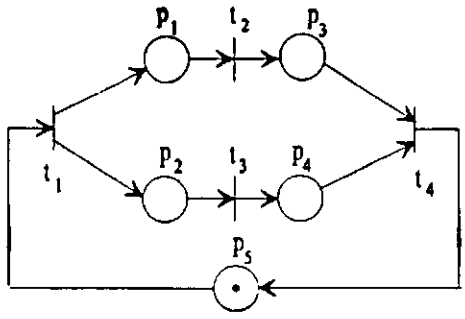


Figura 3.4: Rede de Petri com marcação inicial.

3.1.1 Propriedades das Redes de Petri

Considerando que o modelo em rede de Petri reflita exatamente os requesitos especificados para o modelo do sistema, o estudo de suas propriedades pode revelar características e problemas desse sistema. Dois tipos de propriedades podem ser estudadas por meio do modelo de rede de Petri: as comportamentais e as estruturais. As propriedades comportamentais são aquelas que dependem da marcação inicial enquanto as estruturais não apresentam esta dependência. Um bom tratamento das propriedades estruturais é feito por Murata em [31]. Nesta seção são tratadas as propriedades comportamentais de alcançabilidade, vivacidade, limitabilidade e segurança, e reversibilidade e estado de repouso [31, 12, 43, 46, 26].

Alcançabilidade

Alcançabilidade é uma base fundamental para o estudo das propriedades dinâmicas de um sistema. Com ela podemos saber se o sistema pode alcançar um estado específico, ou exibe um comportamento particular. Em geral, a preocupação é se o sistema modelado em rede de Petri exibe todas as propriedades desejadas, como especificado nos requisitos da especificação, e nenhuma indesejada.

Para determinar se o sistema modelado pode alcançar um estado específico, contido num comportamento desejado, é necessário encontrar pelo menos uma seqüência de disparo de transições que transforme a marcação M_0 em M_n, em que M_n representa o estado específico. Deve-se atentar para o fato de que um sistema real deve alcançar um determinado estado que é especificado pelo comportamento desejado. Num modelo de rede de Petri, isto precisa ser refletido na existência de uma seqüência específica de disparo de transições, representando o comportamento desejado, que transformaria M_0 no requerido M_n.

Seja a Figura 3.4. Para uma marcação inicial $M_0 = [0,0,0,0,1]^T$, há uma transição habilitada, t_1. O disparo de t_1 a partir de M_0 resulta na marcação $M_1 = [1,1,0,0,0]^T$. Isto é escrito como $M_0[t_1 > M_1$.

Na marcação M_1, há duas transições habilitadas, t_2 e t_3. As marcações decorrentes do

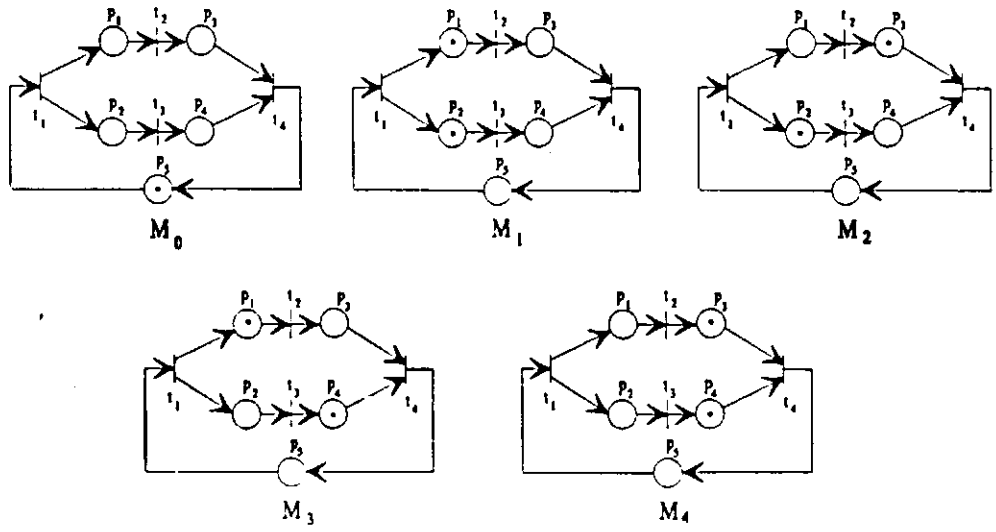


Figura 3.5: Marcações alcançáveis de uma rede de Petri.

disparo das transições são M_2 e M_3, respectivamente (ver Figura 3.5), tem-se:

$M_1[t_2 > M_2 = [0,1,1,0,0]^T$

$M_1[t_3 > M_3 = [1,0,0,1,0]^T$

Na marcação M_2, somente a marcação t_3 está habilitada. Seu disparo resulta na marcação M_4:

$M_2[t_3 > M_4 = [0,0,1,1,0]^T$

Na marcação M_3, somente a transição t_2 está habilitada. Seu disparo também resulta na marcação M_4. Para a marcação M_4 somente a transição t_4 está habilitada e seu disparo leva à marcação inicial M_0.

Uma seqüência de disparos de transições resultará em uma seqüência de marcações. Diz-se que a marcação M_n é alcançável a partir da marcação M_0 se existe uma *seqüência de disparos* que transforma M_0 em M_n. Uma seqüência de disparo é denotada por $\sigma_d = M_0\ t_1\ M_1\ t_2\ M_2\ ...\ t_n\ M_n$ ou simplesmente $\sigma_d = t_1\ t_2\ ...\ t_n$. Neste caso, M_n é alcançável a partir de M_0 por σ e nós escrevemos $M_0[\sigma_d > M_n$. Para a rede da Figura 3.4, uma seqüência válida, a partir de M_0, é $\sigma_d = t_1\ t_2\ t_3$ que resulta na marcação M_4, $M_0[\sigma_d > M_4$. Observamos também, que $M_0[\sigma_d^{'} > M_4$ é válida para $\sigma_d^{'} = t_1\ t_3\ t_2$.

Ao conjunto formado por todos os estados alcançáveis a partir de M_0, chama-se *conjunto de alcançabilidade*. Ele é representado por $R(M_0)$. O conjunto de todas as seqüências de disparo possíveis a partir de M_0 em uma RP é denotado por $L(M_0)$.

Limitabilidade e Segurança

Uma rede de Petri RP é dita ser *k-limitada* ou simplesmente *limitada* se o número de fichas em cada um dos lugares não excede um número finito k para qualquer marcação alcançável a partir de M_0, isto é, $M(p) \leq k$ para todo p e toda marcação $M \in R(M_0)$.

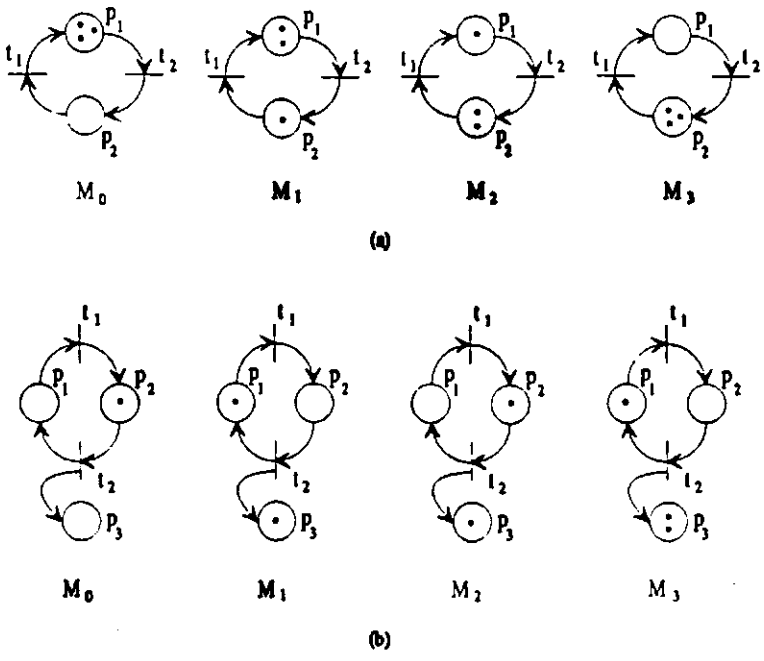


Figura 3.6: (a) Rede limitada. (b) Rede Ilimitada

Definição 3.6 *Uma RP é dita ser segura (ou binária) para uma marcação M_0 se ela é 1-limitada, isto é, se para todas as marcações alcançáveis a partir de M_0, cada um dos lugares contém no máximo uma ficha.*

Em redes de Petri os lugares podem representar depósitos e registradores para armazenamento intermediário de dados. Pela verificação que uma rede é limitada ou segura, garante-se que não haverá estouro da capacidade dos depósitos ou registradores, não importando qual seqüência de disparo é realizada.

A rede da Figura 3.6(a) é 3-limitada, isto é, para toda marcação alcançável $M(p) \leq 3$ $\forall p$, e a rede da Figura 3.6(b) é ilimitada, pois $M(p_3)$ cresce indefinidamente. A rede da Figura 3.4 é uma rede segura.

Vivacidade

O conceito de vivacidade está intimamente relacionado a ausência de bloqueios (*deadlocks*) na operação do sistema. Para ilustrar esse conceito, imaginemos um sistema de manufatura simples composto por duas máquinas diferentes, um robô e um depósito intermediário. Nele, todas as peças do depósito de entrada precisam ser processadas pela máquina 1 e depois pela máquina 2, para produzir o produto final. O robô é usado para carregar e descarregar as máquinas. O depósito intermediário é usado para armazenar peças processadas pela máquina 1 e tem capacidade unitária. Se o sistema atingir um estado no qual as máquinas estejam processando peças, o depósito intermediário contenha uma peça e a máquina 1 terminar o processamento antes da máquina 2, ele entrará num estado de

bloqueio. Isto ocorre porque o robô pegará a peça da máquina 1 para descarregá-la no depósito intermediário e não conseguirá pois o depósito está cheio. Desta forma, o robô não estará mais disponível para a carga e descarga das máquinas e o sistema não irá evoluir.

A vivacidade garante a ausência de bloqueios no sistema modelado, não importando a seqüência de disparo escolhida. Para um melhor entendimento algumas definições são apresentadas a seguir.

Definição 3.7 *Um deadlock (ou estado de bloqueio) é uma marcação na qual nenhuma transição está habilitada.*

Definição 3.8 *Uma transição t_j é viva para uma marcação inicial M_0 se para toda marcação alcançável $M_n \in R(M_0)$ existe uma seqüência σ a partir de M_n tal que $t_j \in \sigma$.*

Definição 3.9 *Uma RP é viva para uma marcação inicial M_0 se todas as transições são vivas para M_0.*

Isto significa que eventualmente todas as transições da rede são disparáveis por alguma seqüência de disparo. Um exemplo de uma rede viva é a rede da Figura 3.4.

Entretanto, de modo geral, a determinação da vivacidade é impraticável. Desse modo vários tipos de vivacidade foram definidos e podem ser mais facilmente analisados [31, 16, 43, 42, 46]. Uma transição t em uma *RP* é dita ser:

- *L0-viva* (ou morta) se a transição nunca puder ser disparada a partir de M_0.
- *L1-viva* se a transição for potencialmente diparável, isto é, se t pode ser disparada pelo menos uma vez em alguma seqüência de disparo em $L(M_0)$.
- *L2-viva* se t pode ser disparada pelo menos um número inteiro k, positivo, de vezes em alguma seqüência de disparo em $L(M_0)$.
- *L3-viva se* t aparece infinitamente, freqüentemente em alguma seqüência de disparo em $L(M_0)$.
- *L4-viva* se t é *L1-viva* para todas as marcações $M \in R(M_0)$.

Seguindo esta classificação, uma *RP* é chamada *Lk-viva*, para a marcação M_0, se toda transição na rede é *Lk-viva*.

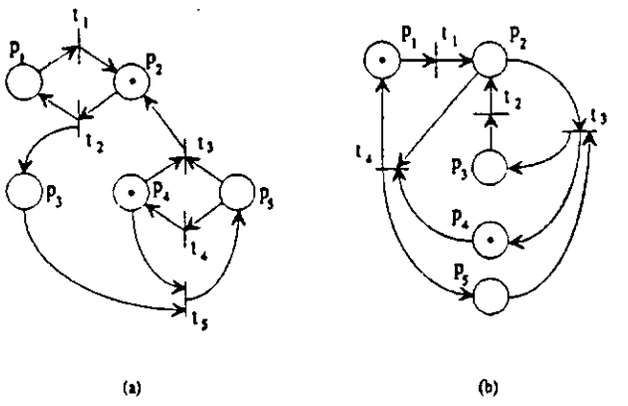


Figura 3.7: (a) Rede reversível. (b) Rede não reversível

Reversibilidade e Estado Recorrente

Uma importante característica desejável na operação de sistemas reais, tais como sistemas de manufatura, é a capacidade desses sistemas recuperarem-se automaticamente de um erro. Por exemplo, um robô pode deixar cair uma peça, no transporte da mesma de um ponto a outro. Portanto, seria interessante que o robô pudesse recuperar-se desta situação sem nenhuma intervenção humana. Uma solução seria o retorno do estado de falha para o estado inicial. Outra seria o retorno para o estado correto precedente. Estas soluções estão relacionadas às propriedades de reversibilidade e estado recorrente de uma rede de Petri. Uma rede de Petri é dita ser reversível se, para cada marcação $M \in R(M_0)$, M_0 é alcançável a partir de M. Um estado M' é dito ser um estado recorrente, se para cada marcação $M \in R(M_0)$, M' é alcançável a partir de M. Esta é uma propriedade menos restritiva e mais prática. A Figura 3.7 mostra duas redes, uma reversível e outra não.

3.1.2 Métodos de Análise Comportamental

Os métodos de análise para redes de Petri podem ser classificados nos seguintes três grupos: 1) método da árvore de cobertura (alcançabilidade), 2) abordagem por equações matriciais [31, 12, 16, 26], e 3) técnicas de redução ou decomposição [31, 12]. O primeiro método envolve essencialmente a enumeração de todas as marcações alcançáveis. Este método é limitado a pequenas redes devido a complexidade da explosão do espaço de estados. Por outro lado, o segundo e terceiro métodos são poderosos mas em muitos casos são aplicáveis somente a subclasses de redes de Petri ou a situações especiais. Nesta seção trataremos apenas do primeiro método [31, 12, 16, 43, 46], pois ele será utilizado neste trabalho.

A Árvore de Cobertura

Este método baseia-se na enumeração de todas as possíveis marcações alcançadas a partir da marcação inicial. Começando com a marcação inicial M_0, pode-se construir o conjunto de alcançabilidade pelo disparo de todas as possíveis transições habilitadas em todas as marcações alcançadas a partir da marcação inicial M_0. Este processo resulta em uma representação em forma de árvore das marcações. Os nós representam marcações geradas a partir de M_0 (a raiz) e seus sucessores, e cada um dos arcos representa o disparo de uma transição, que transforma uma marcação em outra.

Esta representação em árvore crescerá indefinidamente se a rede for ilimitada. Para manter finita a representação da ávore, é introduzido um símbolo especial w, que pode ser entendido como "infinito". Assim para qualquer inteiro n, $w > n$, $w \pm n = w$ e $w \geq w$.

Definição 3.10 *Uma marcação M' cobre uma marcação M'' se a marcação de cada lugar p_i de M' é maior ou igual a sua marcação em M'' :*

$$M' \geq M'' \Longleftrightarrow M'(p_i) \geq M''(p_i), \forall p_i$$

A árvore de cobertura pode ser construída pelo seguinte algoritmo.

1. Rotule a marcação inicial M_0 como raiz e etiquete-a como *nova*;
2. Enquanto existirem marcações do tipo *nova* faça:
   (a) Selecione uma marcação etiquetada como *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 em M:
      (i) Obtenha a marcação M' que resulta do disparo de t em M;
      (ii) Se no caminho da raiz para M, existir uma marcação M'' tal que $M'(p) \geq M''(p)$ para cada um dos lugares p, e $M' \neq M''$, isto é, M'' é coberto, então troque $M'(p)$ por w para cada p tal que $M'(p) > M''(p)$.
      (iii) Introduza M' como um nó da árvore, ligue um arco, com rótulo t, de M para M' e etiquete M' como *nova*.

Um exemplo para este método é mostrado na Figura 3.8. Para a marcação inicial $M_0 = (1\ 0\ 0)^T$, as duas transições t_2 e t_3 estão habilitadas. Disparando t_2 transforma-se M_0 em $M_1 = (0\ 0\ 1)^T$, que é um nó bloqueado, desde que nenhuma transição está habilitada

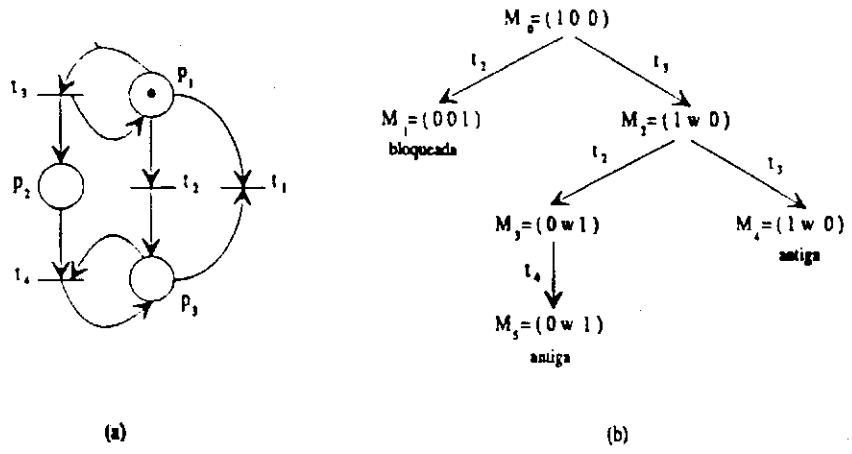


Figura 3.8: (a) Uma rede de Petri, (b) A árvore de cobertura.

nesta marcação. Agora, disparando t_3 em M_0 obtemos $M_2' = (1\ 1\ 0)^T$, que cobre $M_0 = (1\ 0\ 0)^T$. Por isso, a nova marcação é $M_2 = (1\ w\ 0)^T$, onde duas transições estão habilitadas, t_2 e t_3, novamente. Disparando t_2 transforma-se M_2 em $M_3 = (0\ w\ 1)^T$, da qual t_4 pode ser disparada, resultando em um nó antigo $M_5 = M_2$. Assim, nós temos a árvore de cobertura da rede da Figura 3.8(a) mostrada na Figura 3.8(b).

Algumas das propriedades que podem ser estudadas usando a árvore de cobertura são as seguintes:

- A rede RP é limitada e assim $R(M_0)$ é finita se e somente se w não aparece em qualquer nó da árvore. Para este caso a árvore é chamada *árvore de cobertura*.
- A rede RP é segura se e somente se cada nó da árvore contém somente zeros e uns.
- Uma transição é morta se e somente se ela não aparece como um rótulo de um arco na árvore.
- Se M é alcançado a partir de M_0, então existe um nó rotulado M' tal que $M \leq M'$.
- Se dado quaisquer dois nós na árvore de alcançabilidade, existir um caminho direto no qual todas as transições estão presentes, então a rede é viva.
- Se existe um caminho direto de qualquer nó para a marcação inicial na árvore, então a rede é reversível.

3.1.3 Modelagem com Redes de Petri

Em sistemas a variáveis contínuas e a eventos discretos, tanto projetistas quanto analistas de sistemas de controle necessitam ter um modelo matemático das características do sistema e a relação entre elas para desenvolver seus projetos, suas análises e avaliações. Sabe-se

da necessidade da obtenção de um modelo correto para o sistema, pois erros no modelo podem contribuir substancialmente para o aumento do tempo e custo de desenvolvimento, bem como na eficiência operacional. Para sistemas a variáveis contínuas existem leis físicas que governam os componentes e suas interconexões que podem ser usadas para construir o modelo, por exemplo, por equações diferenciais.

Em sistemas a eventos discretos, também se faz necessário um modelo matemático pelos mesmos motivos. As redes de Petri, como ferramenta gráfica e matemática, proporcionam um ambiente para modelagem, análise formal e projeto de sistemas a eventos discretos. Várias são as vantagens do uso desta ferramenta, citadas no capítulo 1.

Nesta seção apresentaremos modelos em rede de Petri de relações básicas entre processos ou operações encontradas em sistemas de manufatura. Apresentamos, também, uma sugestão de método de modelagem de sistemas de manufatura.

Modelos básicos de um sistema de manufatura

Um sistema de manufatura consiste de uma variedade de componentes como robôs, máquinas, matéria-prima, sensores, atuadores, computadores e acessórios relacionados a um processo específico. As características típicas exibidas pelas atividades contidas nos processos, tais como concorrência, sincronização e conflito, podem ser modeladas por redes de Petri. Vamos identificar aqui a construção de redes de Petri para representar as características das atividades dos processos de sistema de manufatura. A Figura 3.9, descreve essas estruturas.

Execução Seqüencial Na Figura 3.9(a), a transição t_2 pode disparar somente após o disparo de t_1. Isto impõe a restrição de precedência "t_2 após t_1". Tais restrições de precedência são típicas da execução de peças em um sistema de manufatura. Também, esta estrutura em RP modela a relação causal entre estas atividades.

Concorrência e Sincronização Muitas operações de manufatura ocorrem em paralelo, iniciadas por um evento. Por exemplo, dois tipos diferentes de peças podem ser manufaturadas por duas linhas de manufatura separadas. Quando cada uma é completada, uma outra operação pode ocorrer. Esta situação exibe as características de concorrência e sincronização. Na Figura 3.9(b), as operações concorrentes, p_2 e p_3, são iniciadas pelo disparo da transição t_1, na ocorrência do evento associado a esta transição. O início da atividade p_6 está sincronizado ao término das operações p_2 e p_3, indicado por uma ficha em p_4 e p_5.

Conflito Se duas ou mais operações podem ser iniciadas como conseqüência de uma outra, então duas ou mais transições serão as saídas de um mesmo lugar. Esta estrutura é chamada de conflito, escolha ou decisão; elas não podem nunca ser concorrentes. Elas são

habilitadas na mesma marcação e a ocorrência de uma delas desabilita as demais. A rede da Figura 3.9(c) representa uma estrutura deste tipo. Nela as transições t_1, t_2 e t_3 estão em conflito.

Concorrência e conflito são complementares no seguinte sentido: Se duas transições estão habilitadas no mesmo estado então ou elas estão em concorrência ou em conflito.

Cíclica Se uma seqüência de operações segue uma após a outra e ao término da última inicia-se a primeira, então uma estrutura cíclica é formada entre essas operações. A Figura 3.9(d) mostra um exemplo de uma estrutura cíclica.

Exclusão Mútua Outra situação comum ocorre quando duas ou mais operações precisam compartilhar o mesmo recurso, por exemplo duas máquinas compartilhando o mesmo robô para carga e descarga das mesmas. Esta disputa pelo recurso leva a um conflito. A Figura 3.9(e) mostra uma estrutura de exclusão mútua. Nela o recurso p_1 é compartilhado pelas operações p_2 e p_3.

Confusão Confusão é a situação na qual concorrência e conflito co-existem. Um exemplo é descrito na Figura 3.9(f). Tanto t_1 como t_3 são concorrentes, enquanto t_1 e t_2 estão em conflito, e t_2 e t_3 também estão em conflito.

Módulos de Depósitos em Redes de Petri

Áreas de armagenagem, estoques, ou depósitos são muito comuns em sistemas de manufatura. Nestes sistemas, um depósito é comumente usado e a capacidade apropriada do mesmo pode levar a um bom desempenho do sistema. Um depósito simples ligando dois componentes, por exemplo, duas máquinas, pode ser modelado em rede de Petri usando-se dois lugares como mostrado na Figura 3.10(a). Tal estrutura é bastante utilizada na literatura. No entanto, devido ao uso de redes de Petri seguras neste trabalho, nós precisamos de módulos de depósitos que sejam construídos em rede de Petri seguras.

Módulos para depósitos em redes de Petri seguras

Alguns módulos para depósitos modelados através de rede de Petri segura são apresentados por Zhou [42]. Por exemplo, no depósito de capacidade b, mostrado na Figura 3.10(b), os elementos são ordenados e sujeitos à política *FIFO* (First In First Out). Nesse caso, cada peça que entra no depósito precisa disparar b transições para sair do mesmo, isto é, t_2 à t_{b+1}, e alcançar a operação seguinte.

Para evitar, que uma peça necessite disparar b transições para sair do depósito, e permitir que ela saia seja qual for sua posição no depósito, outro módulo em rede de Petri de

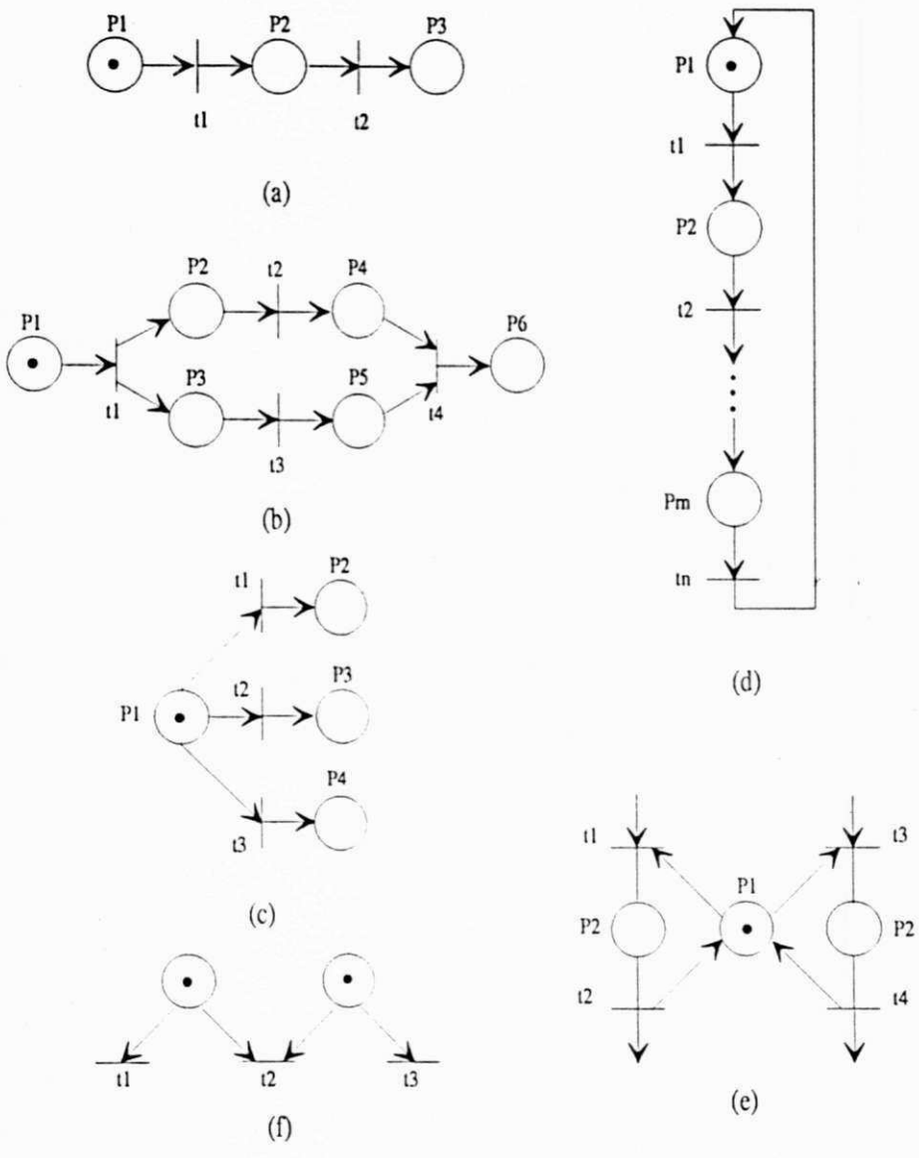


Figura 3.9: Operações fundamentais. (a) execução seqüêncial; (b) concorrência e sincronização; (c) conflito; (d) cíclica; (e) exclusão mútua; (f) confusão.

um depósito com capacidade b é mostrado na Figura 3.10(c) com marcação inicial (1, 0, ..., 0)^T usando $b+1$ lugares e $2(b+1)$ transições.

Assumindo que $b > 1$. Para $i = 1, 2, ..., b+1$,

t_i : transição que liga à operação precedente ao depósito;

r_i : transição que liga à operação sucessiva ao depósito;

p_i : lugar e se ele é marcado, o depósito possui $i-1$ peças.

Portanto, se p_1 estiver marcado, então o depósito está vazio. Se p_{b+1} estiver marcado, então o depósito está cheio.

Considerando que a estação de trabalho que está antes de um depósito possa produzir uma peça de cada vez e que a estação de trabalho que está depois necessita consumir duas peças por vez; então, os módulos de depósitos, modelados por redes de Petri seguras, mostrados anteriomente não são aplicáveis. Uma realização é mostrada na Figura 3.10(d).

É fácil verificar na Figura 3.10(d) que quando o depósito contém uma única peça, isto é, $m(p_2) = 1$, a operação após o depósito não pode ser realizada. Quando o depósito posuir duas ou mais peças, a operação seguinte pode ser realizada.

Esses módulos de depósitos em redes de Petri seguras oferecem a possibilidade de explorar as operações internas de um depósito.

Método de Modelagem

Uma metodologia para modelagem é a chave para a aplicação prática das redes de Petri. O procedimento de modelagem nesse trabalho é baseado no método proposto por Zhou [43, 42]. Nesse método o modelo do sistema pode ser obtido através dos seguintes passos:

1. Identificar as atividades e os recursos existentes no sistema de manufatura.

2. Identificar as relações de seqüência, concorrência, conflito e exclusão mútua das atividades do sistema de manufatura.

3. Ordenar as atividades que formam uma seqüência pela ordem de precedência. Adicionar a esta seqüência as atividades de conflito, as concorrentes e as que se excluem mutuamente.

4. Para cada atividade: criar e rotular um lugar para representar a situação da atividade; adicionar uma transição (início da atividade) com um arco(s) para o(s) lugar(es); adicionar uma transição (término da atividade) com um arco(s) que parta(m) do(s) lugar(es). Observar que uma transição de início de atividade poderá ter dois ou

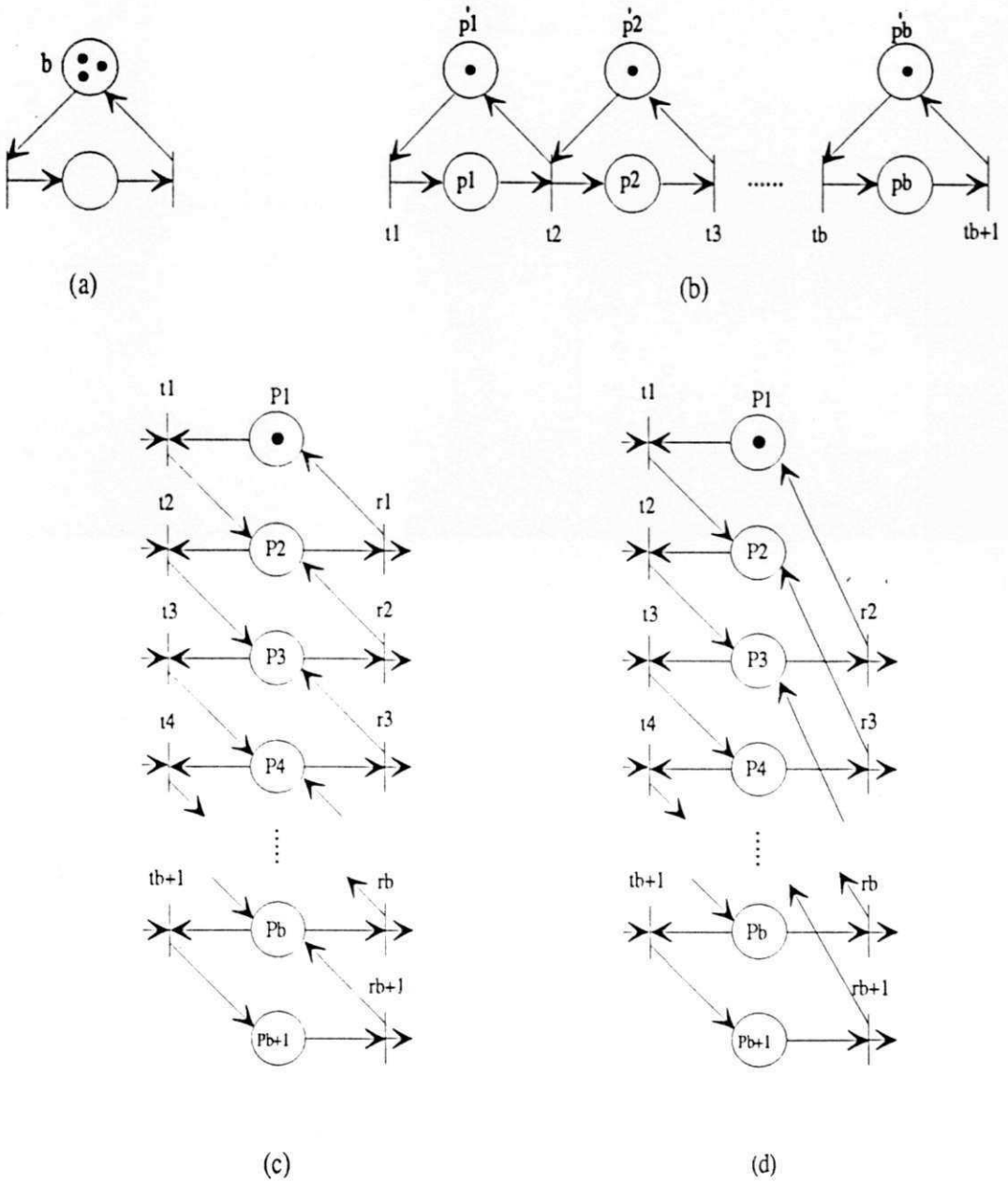


Figura 3.10: Modelos de depósitos.

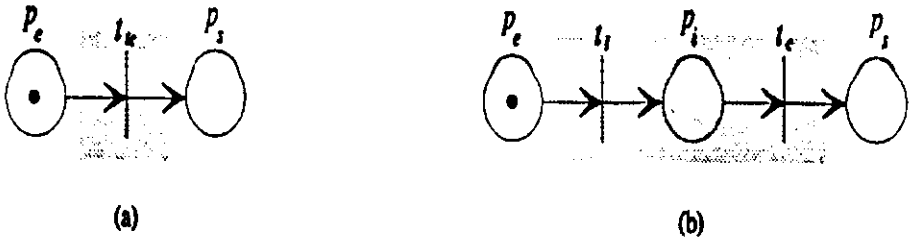


Figura 3.11: Separação de uma transição que representa ao mesmo tempo um evento não controlável e um controlável em uma transição não controlável e uma controlável.

transições de início terão o mesmo lugar de entrada representando o conflito de atividades. Quando a rede for executada, uma ficha em um lugar de atividade indicará que a atividade está sendo executada. O início e o término de uma atividade é marcado pelos disparos das transições de entrada e saída do lugar que a representa e podem também representar o início da próxima atividade.

5. Para cada atividade: se ainda não houver sido criado, criar e rotular um lugar para cada recurso que precisa estar disponível para o início da atividade. Conecte todos os lugares à transição que representa o início da atividade. Criar arcos de saída para conectar as transições de término de atividade aos lugares que representam recursos que tornam-se disponíveis ao término da atividade.

6. Especificar a marcação inicial.

No passo 5, devemos observar que nos casos onde uma transição representa ao mesmo tempo o término de uma operação e o início de uma outra, podemos estar representando em uma mesma transição um evento não controlável e outro controlável. Os eventos não controláveis, são em geral associados às transições de saída de um lugar que representa uma operação. Os eventos controláveis, são associados às transições de entrada de um lugar que representa uma operação. Se esta dupla representação ocorrer, o supervisor não saberá distinguir a transição controlável da não controlável, prejudicando a ação de controle.

A solução para este problema está na separação da transição que representa ao mesmo tempo um evento não controlável e um evento controlável em duas transições: uma representando o evento não controlável e outra o controlável. A Figura 3.11, mostra como isto é feito. Desta forma, uma vez de posse do modelo do sistema modelado segundo o método proposto, precisamos identificar quais transições têm dupla representação e substituí-las conforme é sugerido na Figura 3.11.

Vamos utilizar o seguinte exemplo de uma célula de manufatura para ilustrar como dividir as transições quando necessário. A célula consiste de duas máquinas diferentes m_1 e m_2, um robô r e uma esteira e_1, dispostos como a Figura 3.12. Nela, as máquinas são alimentadas automaticamente e quando terminam o processamento emitem um sinal

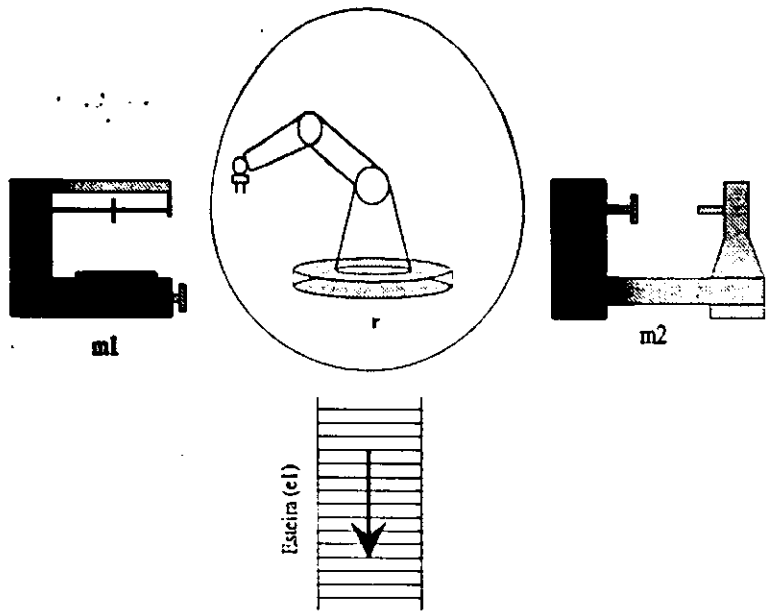


Figura 3.12: Célula de manufatura.

Lugar (com ficha)	Interpretação
p_1 (p_4)	m_1 (m_2) processando
p_2	r transportando peça de m_1 para e_1
p_3	r disponível
p_5	r transportando peça de m_2 para e_1
p_6	m_1 disponível
p_7	m_2 disponível

Tabela 3.1: Interpretação dos Lugares

indicando o término da operação. Uma vez que a operação de processamento termina e o robô está livre a peça é descarregada para a esteira e a máquina pode começar a processar novamente. Uma vez iniciado o trabalho de uma peça em uma máquina, esta não pode ser interrompida até que o trabalho seja concluído.

O modelo obtido, segundo o método proposto, poderia ser o mostrado na Fig 3.13(a). As tabelas 3.1 e 3.2 contêm a descrição tanto dos lugares quanto das transições. Neste modelo, tanto a transição t_2 quanto a t_5, representam ao mesmo tempo um evento não controlável e um controlável: o término das operações das máquinas m_1 e m_2, e o início da operação do robô r. Assim, elas precisam ser separadas, conforme o mostrado na Figura 3.11. Realizando esta alteração obtemos a rede da Fig 3.13(b). A transição t_{2i} (t_{5i}), é uma transição não controlável e representa o fim da operação realizada pela máquina m_1 (m_2), e o lugar p_i (p_{ii}), uma peça pronta em m_1 (m_2). A transição t_{2c} (t_{5c}), é uma transição controlável e representa o início da operação realizada pelo robô r.

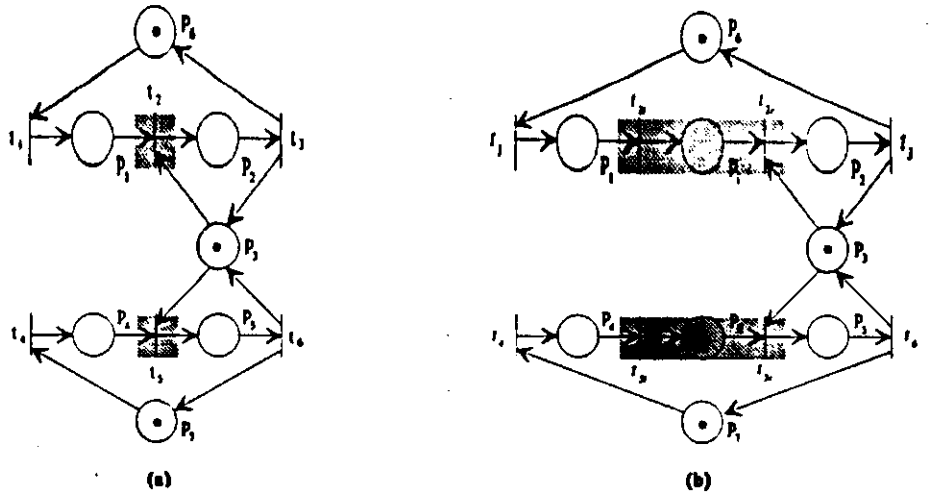


Figura 3.13: Rede de Petri da célula de manufatura: (a) modelo contendo transições com dupla representação; (b) modelo após a divisão das transições com dupla representação.

Transição	Interpretação
$t_1(t_4)$	início do processamento por $m_1(m_2)$
$t_2(t_5)$	fim do proces. por $m_1(m_2)$ e início da descarga por r
$t_3(t_6)$	fim do transporte de uma peça de $m_1(m_2)$ para e_1

Tabela 3.2: Interpretação das Transições

Este método busca representar o sistema livre de qualquer controle. O controle do sistema é realizado pelo supervisor, obtido pelo método que será descrito adiante. Assim, não é necessário mudar o modelo do sistema, como ocorre na abordagem de Zhou [42], para cada nova especificação funcional, somente as restrições à ocorrência de eventos no sistema.

3.2 Método de Síntese de Supervisores

Dado o modelo de um *SED*, representado por um gerador G, a teoria de controle supervisório (TCS) estabelece as condições necessárias para a existência de um supervisor S para um *SED*, baseado em um comportamento desejável, representado pela linguagem K. Tanto o gerador G quanto o supervisor S podem ser modelados por autônomos ou por redes de Petri, dentre outros formalismos.

A utilização da teoria de controle supervisório e das redes de Petri em conjunto, para o projeto, análise e controle de um *SED*, utilizada por Barroso [3] é apresentada na Figura 3.14. Por este diagrama podemos observar que a partir do sistema sintetiza-se o supervisor baseado em uma ou mais especificações funcionais a serem realizadas pelo sistema sob supervisão. Essa abordagem possui uma flexibilidade que permite a especificação de várias tarefas, em diferentes tempos, provocando mudanças somente no supervisor, pois as ações de controle não estão embutidas no modelo do sistema e sim no supervisor.

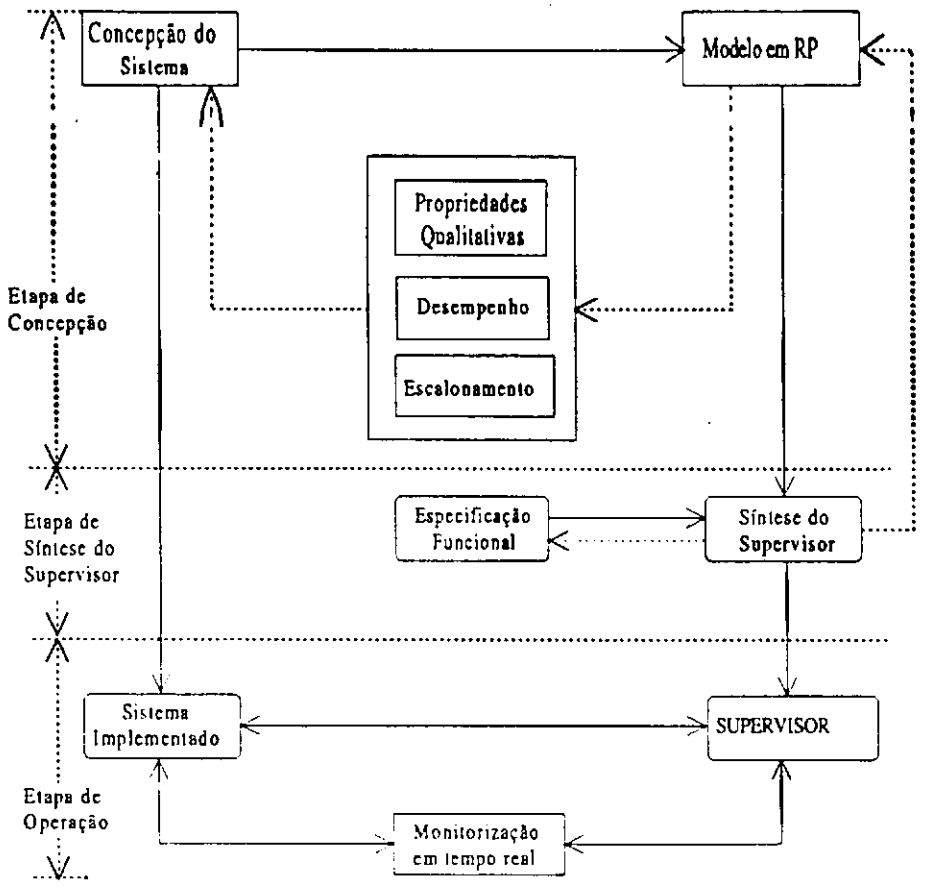


Figura 3.14: Utilização de redes de Petri e teoria de controle supervisório para a concepção, análise e controle de um *SED*.

Seguindo esta abordagem, foi proposta a metodologia de síntese de supervidores para sistemas a eventos discretos mostrada na Figura 3.15. O objetivo era dispor de uma metodologia de síntese do supervisor que não exigisse modificações *ad-hoc* na estrutura do modelo do sistema. Assim, dado que um *SED* é modelado por uma rede de Petri, então a rede de Petri supervisora, *denominada Rede de Petri com Funções de Habilitação de transições (RPFHTs)*, deve possuir a mesma estrutura da rede que modela o sistema. 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 mudança do modelo a cada vez que a especificação funcional é modificada, como ocorre na abordagem de Zhou [42], por exemplo. A mudança no supervisor se dá apenas na alteração das restrições à ocorrência de eventos no sistema.

Para definir que restrições serão impostas à ocorrência de eventos do 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 3.15 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 fun-

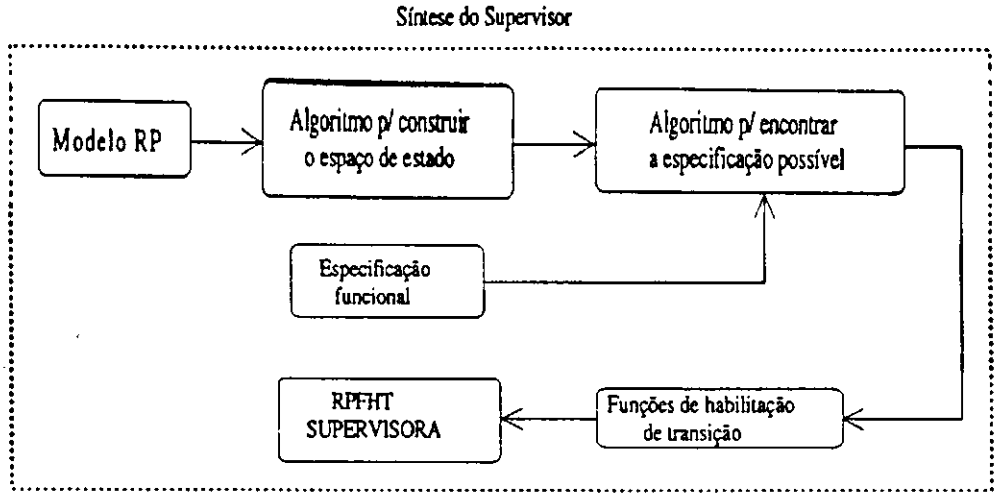


Figura 3.15: Diagrama em blocos do procedimento de síntese do supervisor.

cional desejável, executa-se o $ACGS$ para encontrar a especificação possível e a seguir as funções que restringirão a ocorrência de eventos (representado pelo disparo das transições na $RPFHT$) para que a especificação possível seja executada pelo sistema sob supervisão.

Nesta seção apresentaremos as redes $RPFHT$ utilizadas por Barroso assim como os algoritmos desenvolvidos. As redes são utilizadas para implementar supervisores de sistemas a eventos discretos controlados. Elas possuem a mesma estrutura básica das redes de Petri, de acordo com a definição 3.1, e se distiguem pelas funções de habilitação que podem ser associadas às suas transições, funções estas que modificam a regra de disparo das transições.

3.2.1 Redes de Petri com Funções de Habilitação de Transições

Definição 3.11 *Uma rede de Petri com funções de habilitação de transições é uma tupla*

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

$N = (P, T, I, O)$ é uma estrutura de rede de Petri;

$K : P \rightarrow N \cup \{\infty\}$ 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;

$\Phi = \{\varphi_1, ..., \varphi_m\} : R(M_0) \rightarrow \{0, 1\}$ 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 $M_i \in R(M_0)$ se $\varphi_j(M_i) = 0$.

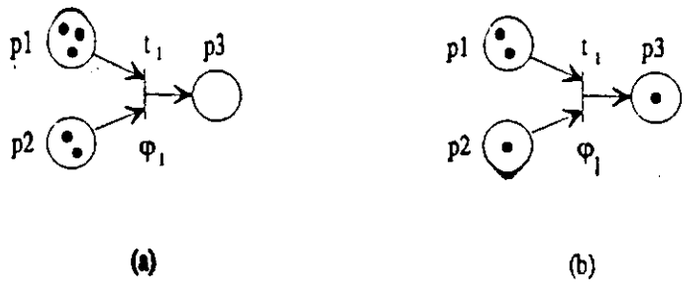


Figura 3.16: Exemplo de uma *RPFHT*.

De outra forma, se $\varphi_j(M_i) = 1$, então t_j estará habilitada sujeita às condições impostas pela regra de disparo das transições (seção 3.1), como em uma rede de Petri.

Definição 3.12 *O estado, ou marcação, de uma rede de Petri RPFHT muda 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
   - Satisfaz a regra de habilitação das redes de Petri e
   - $\varphi_j = 1$.
2. Uma transição habilitada pode ou não disparar;
3. O disparo de uma transição $t \in T$, habilitada na marcação M, resulta em uma nova marcação M' dada pela regra de disparo das redes de Petri;

Um exemplo de *RPFHT* está mostrado na Figura 3.16, na qual para a transição t_1 está definida a função de habilitação dada por

$$\varphi_1 = [M(p_1) \geq 2 \wedge M(p_3) = 0].$$

Neste exemplo, vemos que pela função de habilitação φ_1, no caso da Figura 3.16(a), a transição t_1 está habilitada e pode disparar, já que a função $\varphi_1 = 1$. Após o disparo, Figura 3.16(b), a primeira condição da função é verdadeira, contudo a condição $M(p_3) \neq 0$, o que torna a função $\varphi_1 = 0$, desabilitando a transição t_1.

3.2.2 Algoritmos de Síntese

A síntese de supervisores de *SEDs* de Barroso [3], utiliza dois algoritmos que em suas execuções criam a árvore de alcançabilidade do modelo em *RP* (*AMArA*) e determinam, a partir desta árvore, juntamente com a especificação funcional desejável, todas as funções que devem ser aplicadas às transições do supervisor modelado, para que o sistema não entre em *bloqueio* (*ACGS*). A seguir, são apresentados estes algoritmos.

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 em M:
      (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.

Este algoritmo tem uma modificação simples com respeito ao algoritmo da árvore de alcançabilidade original, que é um passo a mais na determinação de estados não permitidos (ramos da árvore com crescimento infinito). Esta mudança no algoritmo é determinante para modelagem de *SEDs*, visto que estes, para serem fisicamente realizáveis, têm de ter capacidade finita. Observamos, então, que o mesmo é válido para redes com capacidade limitada ou finita.

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]

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

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

Início

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

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

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

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

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

3. Adicionar à *lista_bloq* 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_bloq*, ou seja

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

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:

$$\exists \beta \in \Sigma_c | l(t_j) = \beta \text{ e } M[t_j > M', M \notin \textit{lista\_bloq} \text{ e } M' \in \textit{lista\_bloq};$$

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:

$$\exists \beta \in \Sigma_c | l(t_j) = \beta \text{ e } M[t_j > M', M \notin \textit{lista\_perigo} \text{ e } M' \notin G(supC(l)).$$

Fim.

Algoritmo 3 Passo 5 do *ACGS*.

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

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

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

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

(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 componente acessível do gerador resultante.

- O passo um indica à *lista_bloq* 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 funcional desejável.

O *ACGS* é um algoritmo que trabalha com a utilização dos dados referentes à árvore de alcançabilidade gerada pelo *AMArA* e da especificação do comportamento que se deseja para o sistema. Logo, este algoritmo só pode ser executado após a execução do *AMArA* e do fornecimento desta especificação. De posse destes dados, a execução do *ACGS* nos dá uma lista de eventos que devem ser desabilitados nos devidos estados, determinando a *supC(L)*, e com estas saídas, implementam-se as funções de habilitação que devem ser impostas às respectivas transições para impedir que o sistema entre em bloqueio e siga a especificação desejada, caso esta seja possível.

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

Para exemplificar a metodogia de síntese apresentada, através dos algoritmos descritos, apresentaremos a seguir um exemplo de uma célula de manufatura simples.

3.2.3 Supervisor para uma célula de manufatura

Vejamos o exemplo de uma célula de manufatura, mostrada na Figura 3.17. Ela consiste de duas máquinas (m_1 e m_2), um braço robótico (r), um depósito (d), e duas esteiras (e_1 e e_2).

Nesta célula, as peças a serem processadas, chegam pela esteira e_1. Cada uma dessas peças, é processada pela máquina m_1 e em seguida pela máquina m_2. O braço robótico, r, é o responsável pelo transporte das peças de e_1 para m_1, das peças de m_1 para d ou para

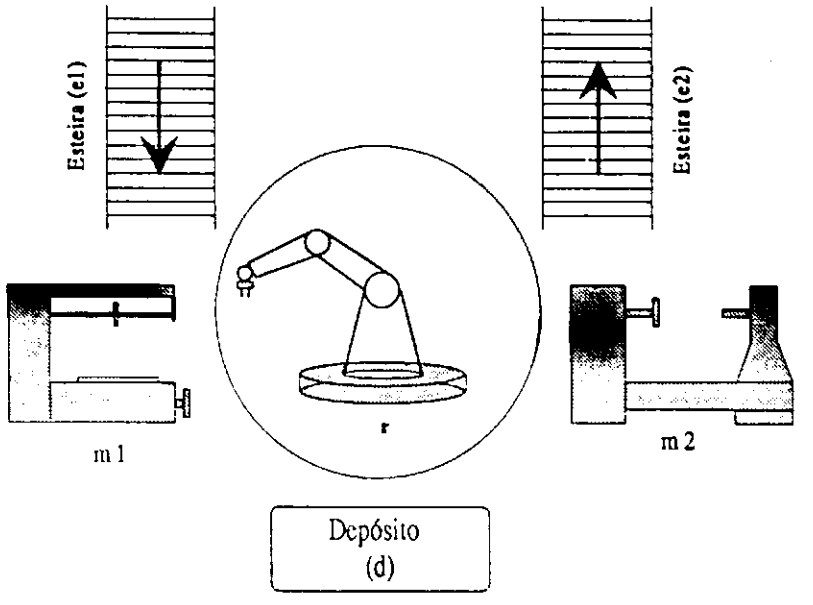


Figura 3.17: Célula de manufatura com recurso compartilhado.

m_2, e de m_2 para e_2. O depósito, d, tem capacidade para armazenar apenas uma peça. A outra esteira é usada para transportar as peças trabalhadas em m_2 para outra célula.

A Figura 3.18, apresenta a rede de Petri, para esta célula, livre de qualquer controle. Nela, o lugar p_1, representa a esteira e_1. Uma ficha em p_1 indica que há uma peça em e_1. Os lugares p_2, p_6, p_7, p_9 e p_{11} representam o transporte de uma peças de e_1 para m_1, o de uma peça de m_1 para m_2, o de uma peça de m_1 para d, o de uma peça de d para m_2 e o de uma peça de m_2 para e_2, nesta ordem. Os lugares p_4 e p_{12}, representam a disponibilidade das máquinas m_1 e m_2, os lugares p_5 e p_{10}, o processamento por m_1 e m_2, respectivamente, e o lugar p_8, representa o depósito d. O lugar p_3, representa a disponibilidade do braço. Para tornar o desenho mais legível, o lugar p_3 foi desenhado em três posições diferentes na figura. Nesta os eventos α_1, β_1, α_2, β_2, α_3, β_3, α_4, β_4, β_5 e α_5 estão associados respectivamente ao disparo das transições t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9 e t_{10}.

A marcação inicial determina como a célula é iniciada. Nesse exemplo, existe uma peça na esteira e_1, representada por uma ficha em p_1 ($M_0(p_1) = 1$), bem como o braço e as máquinas estão disponíveis ($M_0(p_3) = M_0(p_4) = M_0(p_{12}) = 1$). Não há nenhuma peça no depósito d ($M_0(p_8) = 0$). Nesta marcação ou estado, somente a transição t_1 está habilitada.

Utilizando-se o *AMArA* chega-se à árvore de alcançabilidade mostrada na Figura 3.19. Nesta, os estados recebem nomes de acordo com uma ordem crescente de números a partir de zero. Os estados que não são *novos* ou são *antigos*, ou *bloqueados*, ou *não-permitidos*. No caso dos *não-permitidos* o nome do estado recebe um p adicional, por exemplo *M2p*.

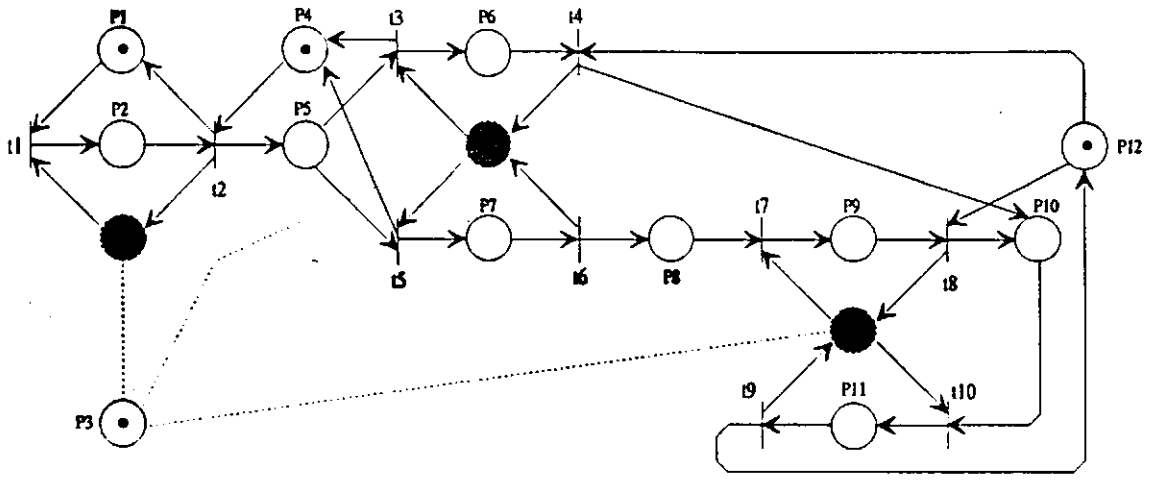


Figura 3.18: Rede de Petri da célula de manufatura simples.

Note que o *AMArA*, enumera todos os estados cuja capacidade da rede não seja excedida. Além dos estados, obtém-se a informação de quais cadeias podem levar a célula a um estado *não-permitido* ou a um estado *bloqueado*.

O modelo do sistema até aqui descrito, é simplesmente um gerador de eventos espontâneo, sem nenhum controle externo. Por isso, pode executar tarefas não desejadas. Observe que, por exemplo, se a seqüência de transições $t_1t_2t_1$ for disparada (ocorrência dos eventos $\alpha_1\beta_1\alpha_1$) a partir da marcação inicial, a célula alcançará um estado no qual $M_3(p_2) = 1$, $M_3(p_5) = 1$, $M_3(p_{12}) = 1$ e todos os outros lugares da rede não possuem fichas. Neste estado a célula estará bloqueada, ou seja, nenhuma transição estará habilitada nesse estado ou marcação. Pode também, atingir um estado *não-permitido*, pela seqüência $t_1t_2t_5t_6t_1t_2t_5t_6$ $(\alpha_1\beta_1\alpha_3\beta_3\alpha_3\beta_3)$, no qual o braço tem uma peça para descarregar no depósito mas este já tem uma peça ($M_{1p}(p_1) = M_{1p}(p_3) = M_{1p}(p_4) = M_{1p}(p_{12}) = 1$, $M_{1p}(p_8) = w$ e os demais lugares não tem fichas), entre outros.

Uma especificação funcional desejável para esta célula, por exemplo, pode ser assim definida: a célula deve sempre retornar à marcação ou estado inicial, ou seja, $M_0 \in Q_m$.

Isto, implica que deve-se evitar que a célula alcance os estados não-permitidos e os de bloqueio. Para evitar estes estados, uma solução é a utilização de um supervisor para a célula, implementado com uma *RPFHT* com a mesma estrutura da rede que modela a célula, capaz de prevenir tais estados. Para se controlar a célula, segundo a teoria de controle supervisório, é necessário que certos eventos possam ser desabilitados para que uma especificação funcional desejável possa ser executada, ou seja, apenas alguns eventos poderão ocorrer para que uma determinada tarefa seja realizada com sucesso.

Seja $\Sigma = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$, o alfabeto de eventos associado à rede apresentada na Figura 3.18, no qual $\Sigma_c = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$, o alfabeto de eventos controláveis, e $\Sigma_u = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$, o alfabeto de eventos não controláveis. Com base na

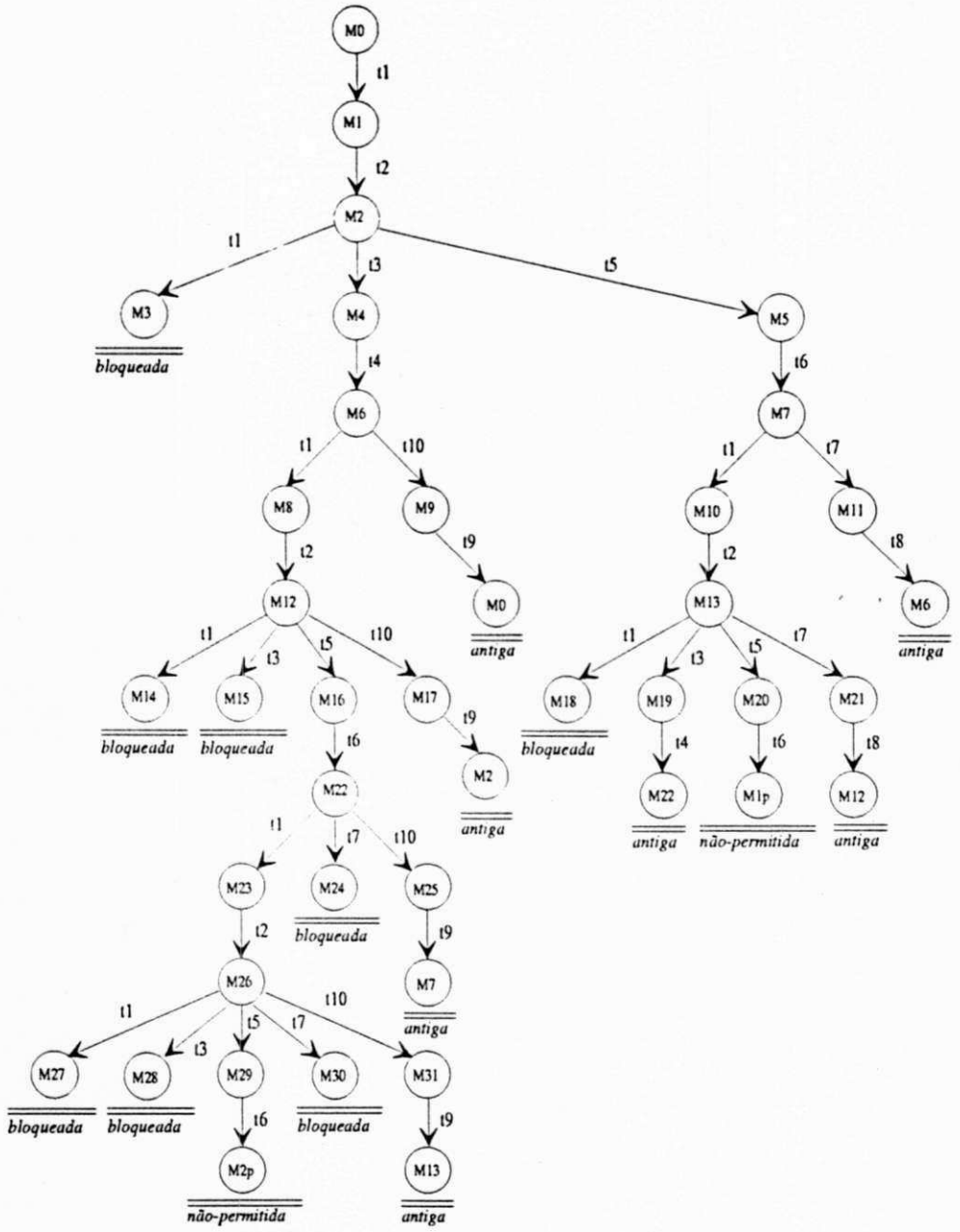


Figura 3.19: Árvore de alcançabilidade da célula de manufatura.

especificação funcional desejável e no conjunto de marcações da rede, executa-se o *ACGS* e obtém-se a seguinte *lista_perigo*:

$$\begin{aligned}
M_2 &\rightarrow t_1/\alpha_1;\\
M_{12} &\rightarrow t_1/\alpha_1;\ M_{12} \rightarrow t_3/\alpha_2;\\
M_{13} &\rightarrow t_1/\alpha_1;\ M_{13} \rightarrow t_5/\alpha_3;\\
M_{22} &\rightarrow t_7/\alpha_4;\\
M_{26} &\rightarrow t_1/\alpha_1;\ M_{26} \rightarrow t_3/\alpha_2;\ M_{26} \rightarrow t_5/\alpha_3;\ M_{26} \rightarrow t_7/\alpha_4;
\end{aligned}$$

A notação $M_i \rightarrow t_j/\alpha_k$, significa que o supervisor deve evitar o disparo da transição t_j (ocorrência do evento α_k) no estado M_i. Assim, para que a especificação funcional desejável seja executada, basta que α_1 seja desabilitada nas marcações M_2, M_{12}, M_{13}, M_{22} e M_{26}, α_2 seja desabilitada nas marcações M_{12} e M_{26}, α_3 seja desabilitada nas marcações M_{13} e M_{26} e α_4 seja desabilitada nas marcações M_{22} e M_{26}.

O supervisor é então modelado pela *RPFHT*, apresentada na Figura 3.20. As funções de habilitação são:

$$\begin{aligned}
\varphi_1 &= \overline{M_2} \wedge \overline{M_{12}} \wedge \overline{M_{13}} \wedge \overline{M_{26}};\\
\varphi_3 &= \overline{M_{12}} \wedge \overline{M_{26}};\\
\varphi_5 &= \overline{M_{13}} \wedge \overline{M_{26}};\\
\varphi_7 &= \overline{M_{22}} \wedge \overline{M_{26}};\\
\varphi_2 &= \varphi_4 = \varphi_6 = \varphi_8 = \varphi_9 = \varphi_{10} = 1,
\end{aligned}$$

a notação $\overline{M_i}$ significa uma marcação diferente de M_i.

A rede supervisora pode então, ser executada sincronamente com a célula, obedecendo a regra de disparo de suas transições.

Observe que se quisermos especificar uma outra tarefa para o sistema, por exemplo $K = (\alpha_1\beta_1\alpha_2\beta_2\alpha_5\beta_5)^*$, basta executar os passos 5 e 6 do algoritmo *ACGS* para encontrar os novos estados e os respectivos eventos a serem desabilitados.

Neste caso, a nova *lista_perigo* será:

$$\begin{aligned}
M_2 &\rightarrow t_1/\alpha_1;\ M_2 \rightarrow t_5/\alpha_3;\\
M_6 &\rightarrow t_1/\alpha_1;
\end{aligned}$$

e as novas funções de habilitação serão:

$$\begin{aligned}
\varphi_1 &= \overline{M_2} \wedge \overline{M_6};\\
\varphi_5 &= \overline{M_2};\\
\varphi_2 &= \varphi_3 = \varphi_4 = \varphi_6 = \varphi_7 = \varphi_8 = \varphi_9 = \varphi_{10} = 1.
\end{aligned}$$

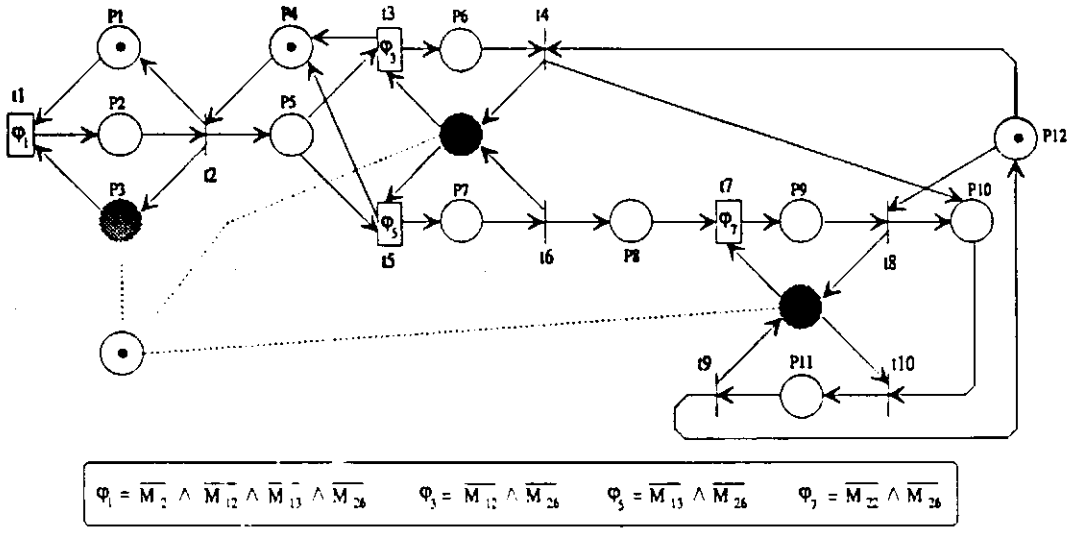


Figura 3.20: Rede de Petri supervisora da célula de manufatura.

A utilização prática das redes de Petri para modelar e analisar sistemas mais complexos, ampliou o estudo teórico das redes fazendo surgir extensões de redes de Petri que buscam representar de forma mais realista os sistemas por elas modelados. Algumas destas extensões serão apresentadas no próximo capítulo e serviram para a escolha da extensão que será usada para modelar o sistema e obtenção da rede supervisora na síntese proposta neste trabalho.

Capítulo 4

Extensões de Rede de Petri

As redes de Petri clássicas, usadas para modelar complexos sistemas a eventos discretos, proporcionam modelos que representam um ponto de vista lógico do sistema. Elas possibilitam a descrição somente do que acontece e não contemplam, em seus modelos, características como disparos sincronizados a eventos externos e restrições temporais.

Nesse capítulo apresentaremos algumas extensões para as redes de Petri, que foram sugeridas na literatura, para possibilitar a sincronização do disparo de transições a ocorrência de eventos externos e a caracterização de propriedades relacionadas ao tempo. Dentre a extensões temporais que serão apresentadas, merece destaque a extensão denominada rede de Petri temporal, pois a mesma será utilizada neste trabalho.

4.1 Redes de Petri Sincronizadas

Em uma rede de Petri clássica, sabemos que uma transição pode ser disparada se ela estiver habilitada, mas não sabemos quando isto ocorrerá. Em uma rede de Petri sincronizada, um evento é associado a cada uma das transições e, agora, o disparo, de uma transição habilitada, acontecera em sincronismo com a ocorrência do evento externo associado [12, 13].

Definição 4.1 *Uma rede de Petri sincronizada é uma tripla*

$$RPS = (RP, E, FE), \text{ na qual}$$

RP é uma rede de Petri marcada;

E é um conjunto de eventos externos;

FE é uma função do conjunto de transições T de RP para $E \cup \{e\}$

em que e é o evento que possui ocorrência permanente, isto é, ele é o elemento neutro do monoide $(E^1 + ... + E^p)^*$, *onde* $E = \{E^1, ..., E^p\}$ *é o conjunto de eventos externos. Em*

outras palavras, e corresponde à seqüência de eventos externos cujo período é zero. Se uma transição é sincronizada com um evento e, ela é disparada assim que esteja habilitada.

A notação E^i corresponde ao *nome* de um evento externo. A notação E_j corresponde ao evento associado à transição t_j.

Nessa representação é admitido a hipotese de que dois eventos externos nunca ocorrem simultaneamente.

Para que uma transição possa disparar, além de satisfazer as regras de disparo de uma rede de Petri clássica (estar habilitada), ela deve esperar pela ocorrência do evento externo, E, a ela associado. Quando t encontra-se habilitada, a transição é considerada receptiva ao evento E. Estando t receptiva e ocorrendo E, t dispara e gera uma nova marcação conforme as regras usadas para as redes de Petri clássicas.

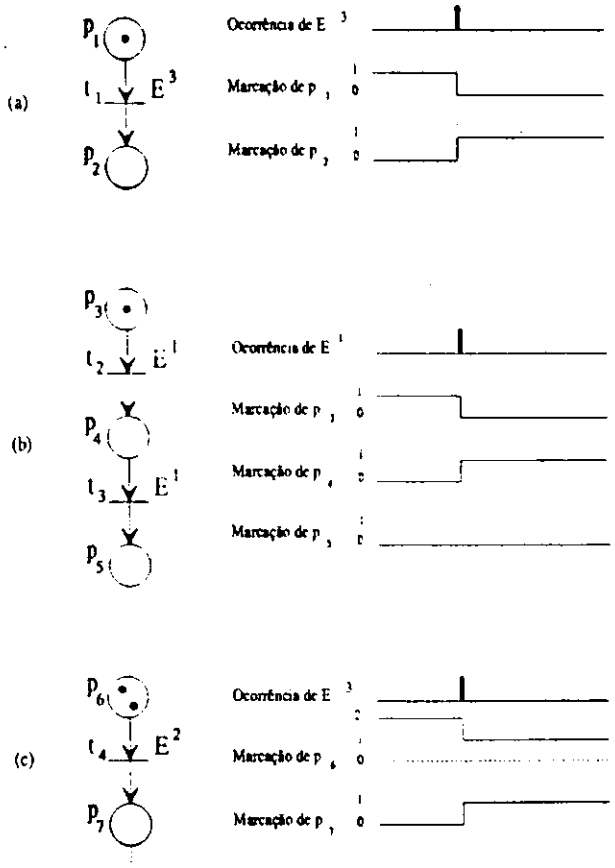


Figura 4.1: Disparo de uma transição sincronizada (David [12]).

A Figura 4.1 apresenta o princípio de disparo de uma transição sincronizada. Na Figura 4.1(a), o evento externo E^3 é associado à transição t_1. Nesta figura, a transição é dita *receptiva* ao evento E^3, porque ela está habilitada. Ela é disparada quando o evento E^3 ocorre. Na Figura 4.1(b), a transição t_2 está *receptiva* ao evento E^1, pois está habilitada. Ela é disparada quando o evento E^1 ocorrer. De outra forma, a transição t_3 não dispara,

embora esteja sincronizada com o evento E^1, pois não estava habilitada quando E^1 ocorreu. Na Figura 4.1(c) há duas fichas em p_6, o suficiente para o disparo da transição t_4 duas vezes, então dois disparos de t_4 poderiam ser realizados quando E^2 ocorresse, isto é, as duas fichas passariam de p_6 para p_7. No entanto, a transição t_4 é disparada somente uma vez, devido a semantica escolhida pelos criadores da rede de Petri sincronizada.

A Figura 4.2 apresenta a evolução da marcação de uma rede quando a seqüência de eventos $Z = E^2E^1E^1E^2E^1$ é aplicada. Nesta, podemos observar que, mesmo na ocorrência do evento associado à transição, se a transição não estiver habilitada (receptiva) ela não dispara, por exemplo a ocorrência de E^2 na marcação M_0. O conjunto de evoluções possíveis é representado pelo gráfico da Figura 4.2(c). Para cada disparo de transição, é indicado após a barra o evento cuja ocorrência causou o disparo.

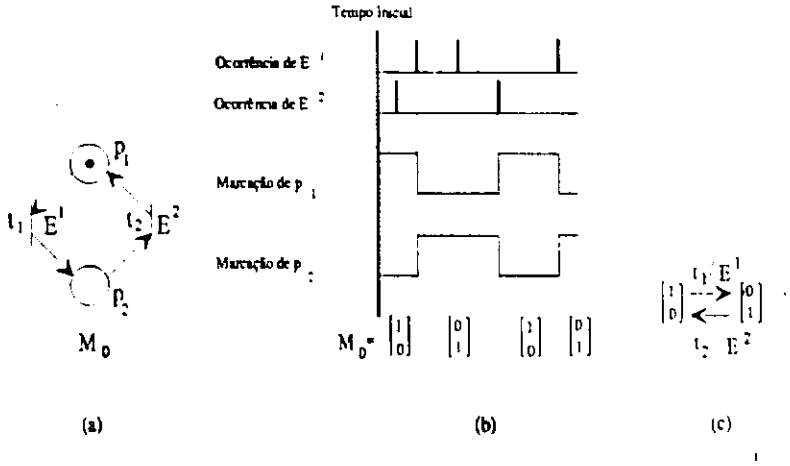


Figura 4.2: Exemplo do comportamento de uma *RPS* (David [12])

Esta rede é adequada para ser usada no contexto do problema de controle pois permite a sincronização do disparo de uma transição com o sinal de resposta de um sensor, por exemplo.

4.2 Extensões Temporizadas das Redes de Petri

As redes de Petri clássicas, autônomas, permitem ao projetista apenas o estudo das propriedades qualitativas (não dependentes do tempo) do sistema sob análise. Com elas, a caracterização das propriedades temporais, que são essenciais em muitos casos, como por exemplo, na modelagem de sistemas em tempo real, não é possível. Com o intuito de tornar possível a representação de propriedades quantitativas (relacionadas ao tempo) dos sistemas, diversas extensões de redes de Petri foram propostas. Elas diferem basicamente em dois aspectos [15]:

Localização: as restrições de tempo podem ser associadas tanto a lugares quanto a tran-

sições.

Tipo: a natureza das especificações das restrições de tempo (atrasos fixos, intervalos, atrasos estocásticos, etc).

Essas extensões são classificadas em duas abordagens principais: a determinística e a estocástica. As extensões determinísticas são basicamente utilizadas para modelar sistemas em tempo real onde as restrições de tempo são determinadas por valores determinísticos. Entretanto, elas são limitadas para fazer análise de desempenho e para representação de incertezas que são comumente encontradas nos sistemas reais. As extensões estocásticas, obtidas quando a duração dos eventos varia de uma maneira probabilística bem definida, são largamente utilizadas para fazer análise de desempenho de sistemas e são principalmente utilizadas para determinar alguns índices de desempenho. No entanto, elas falham quando modelam sistemas em tempo real, no qual a representação dos limites de tempo é fundamental. Neste trabalho serão tratadas apenas as extensões determinísticas.

Segundo Freedman [18], a adição de tempo as redes de Petri clássicas tem as seguintes conseqüências:

- O conjunto de seqüências de disparo de uma rede de Petri com extensão temporal (RPT) é um subconjunto das seqüências de disparo de sua rede de Petri (RP) sem extensão temporal. desde que a informação de tempo pode inibir certas combinações de disparo de transições.

- O *estado* de uma RPT não é completamente expresso pelas marcações de sua RP. Dependendo do modelo particular de tempo, o estado precisa incluir informações sobre as transições disparadas na marcação ou sobre o atraso de disparo das transições habilitadas.

- O $R_T(M_0)$ de uma RPT, definido como o conjunto de marcações alcançáveis de seqüências de disparo válidas, é um subconjunto do $R(M_0)$ de sua RP. Isto porque o tempo pode alterar a ordem relativa de disparo das transições, embora a organização *lógica* dos disparos não mude, isto é, as condições sob as quais as transições são habilitadas e suas conseqüências.

- Uma vez que a adição de tempo não altera a estrutura lógica da RP, os P-invariantes e os T-invariantes da RPT são exatamente os mesmos.

- As propriedades de limitabilidade e alcançabilidade são indecidíveis para uma RPT (Informalmente, um problema indecidível é tão difícil que nenhum algoritmo - polinomial ou outro - pode dar uma solução para ele). Entretanto, a limitabilidade de

uma *RP* é uma condição suficiente mas não necessária para a limitabilidade de uma *RPT*.

4.2.1 Redes de Petri Temporizadas

As redes de Petri Temporizadas *(Timed Petri Net - TdPN)* [18, 12, 13, 15, 46, 43, 32], são uma extensão de redes de Petri, desenvolvidas por C. Ramchandani para estudar o desempenho de estruturas computacionais básicas tais como processadores *pipeline*. Nessas, uma duração de tempo de disparo é associado a cada transição da rede. As *TdPN*, seguem as mesmas regras de habilitação das redes de Petri clássicas. As transições disparam instantaneamente, quando habilitadas, mas só depositam as fichas nos locais de saída após ter decorrido d_j unidades de tempo após o disparo da transição, onde d_j é o tempo associado à transição.

Definição 4.2 *Uma Rede de Petri Temporizada é uma dupla*

$$TdPN = (RP, Tempo) \text{ na qual}$$

- *RP* é uma rede de Petri marcada e
- $Tempo : T \rightarrow \{0, 1, 2, ...\}$.

Tempo é uma função de tempo, mapeando cada transição na rede nos números naturais. $Tempo\,(t_j) = d_j$ representa o tempo associado à transição t_j.

Quanto ao que acontece às fichas dos lugares de entrada quando a transição dispara, os autores apresentam dois comportamentos para elas:

- No primeiro [32, 46, 43], as fichas são removidas dos lugares de entrada quando a transição torna-se habilitada. A transição dispara após um certo período de tempo (tempo de atraso, τ), depositando fichas nos lugares de saída. A Figura 4.3, ilustra este comportamento. Consideremos que o tempo associado à transição é τ, e que o disparo de t_1 começa em T_0. A Figura 4.3(a), representa a marcação do sistema antes do disparo de t_1. A Figura 4.3(b), representa a situação durante o disparo de t_1: as fichas foram removidas dos lugares de entrada de t_1, mas a marcação do lugar de saída permanece o mesmo. A Figura 4.3(c), representa a marcação do sistema no fim do disparo de t_1.

- No segundo [12, 13, 46, 43], a ficha pode ter dois estados: ela pode ser *reservada* para o disparo de uma transição t_j ou ela pode ser *não reservada*. Isto é ilustrado na Figura 4.4. Quando a transição t_1 é disparada, uma ficha é depositada no lugar

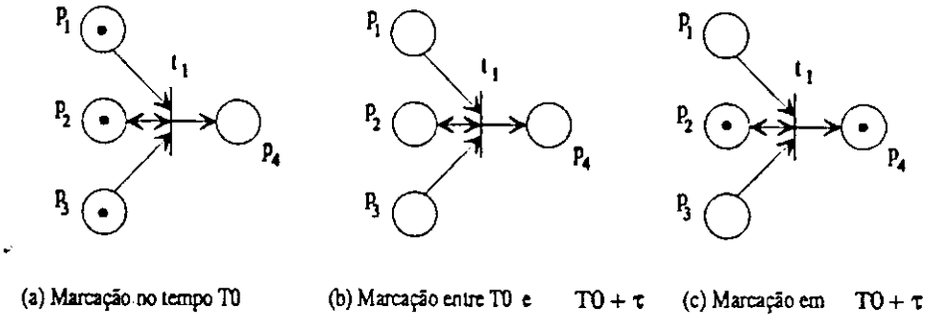


Figura 4.3: Exemplo de uma regra de disparo de uma *TdPN*.

p_1, habilitando a transição t_2. A transição t_2 pode disparar em qualquer momento após isto. Quando a transição dispara, a ficha requerida para o disparo é reservada. Decorrido a duração d_2, após o disparo, a transição é efetivamente disparada. A ficha reservada é então removida de p_1 e uma ficha não reservada é depositada em p_2. Em qualquer instante de tempo τ, a marcação atual, M, do sistema é a composição de duas marcações, M^r e M^n, das quais M^r é a marcação composta pelas fichas reservadas e M^n pelas fichas não reservadas. Uma transição está habilitada para a marcação $M = M^r + M^n$ se ela está habilitada para a marcação M^n.

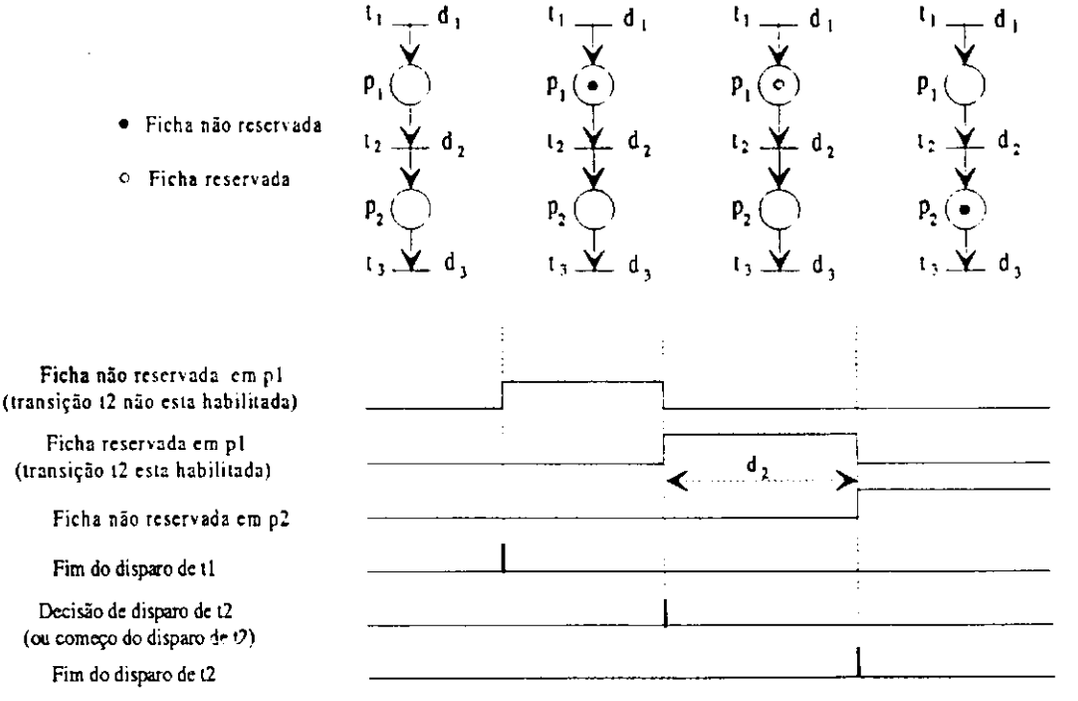


Figura 4.4: Exemplo de reserva de ficha (David [12]).

4.2.2 Redes de Petri P-Temporizados

As Redes de Petri P-Temporizados (*P-Timed Petri Nets - PdPN*) [12, 13, 15], diferem das *TdPNs* devido à localização da caracterização das restrições de tempo. Neste tipo de rede o tempo é associado ao lugar. Uma vez que uma ficha chega a um lugar temporizado, esta ficha permanece indisponível por um tempo d_i. Decorrido o tempo d_i a ficha torna-se disponível para o disparo da transição de saída do lugar. Este comportamento é ilustrado na Figura 4.5. Com o disparo da transição t_1, uma ficha é depositada no lugar p_1, e esta permanece indisponível por um tempo d_i. Decorrido o tempo d_i, a ficha torna-se disponível e a transição t_2 habilitada. Quando ela é disparada (não necessariamente imediatamente), uma ficha é depositada em p_2, e esta permanece indisponível por um tempo d_2, e assim por diante.

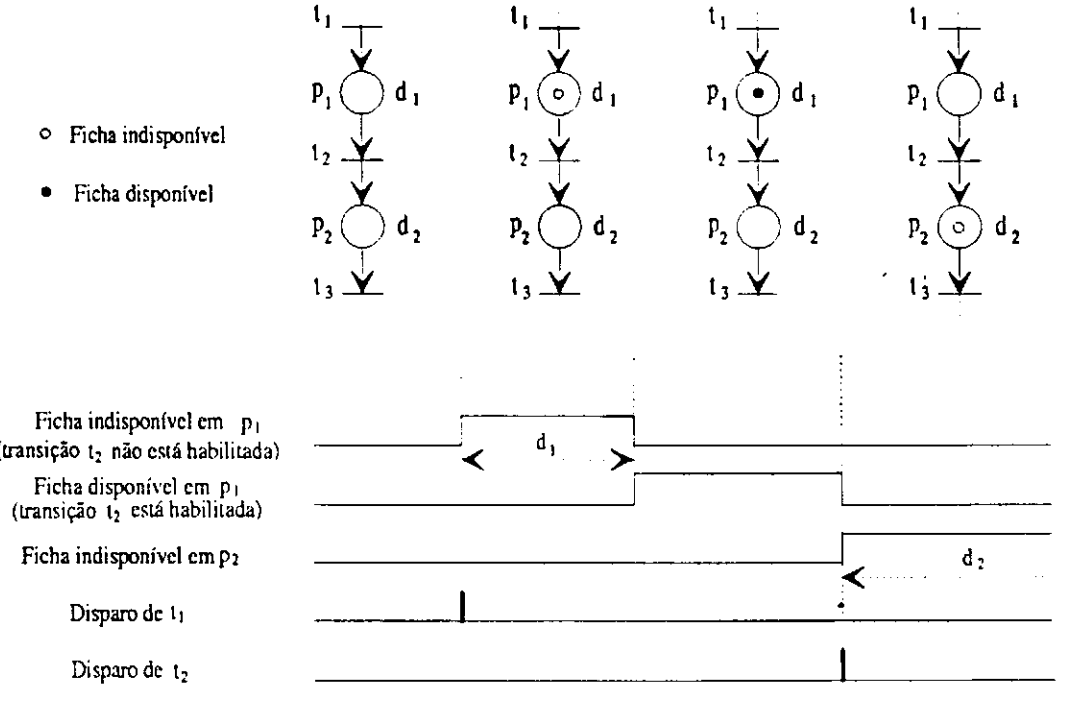


Figura 4.5: Disponibilidade de uma ficha (David [12])

Definição 4.3 *Uma Rede de Petri P-Temporizada é uma dupla*

$$PdPN = (RP, Tempo)\,,\ \textit{na qual}$$

- RP é uma rede de Petri marcada e
- $Tempo : P \rightarrow \{0, 1, 2, ...\}$.

Tempo é uma função de tempo, mapeando cada lugar na rede nos números naturais. $Tempo\,(p_i) = d_i$ representa o tempo associado ao lugar p_i.

Em qualquer instante de tempo τ, a marcação atual, M, do sistema é a composição de duas marcações, M^d e M^i, das quais M^d é a marcação composta pelas fichas disponíveis e M^i pelas fichas indisponíveis. Uma transição está habilitada para a marcação $M = M^d + M^i$ se ela está habilitada para a marcação M^d. O disparo da transição, que tem tempo de duração zero, é executado da mesma forma que para uma rede não temporizada, e a remoção dá-se somente das fichas disponíveis dos lugares de entrada.

Se uma ficha é depositada em um lugar, ao qual está associado o tempo d_i, no tempo t, ela estará disponível para o disparo da transição t_i em $t + d_i$. A evolução, em um caso geral, de uma *PdPN* onde x e y são números positivos ou zeros (correspondendo a um tempo de espera arbitrário entre habilitação e disparo), é mostrada na Figura 4.6. A situação na qual $x = y = 0$, isto é, quando a transição dispara imediatamente após $t + d_i$, é conhecida como *função de disparo à máxima velocidade* e é mostrada na Figura 4.6(b). O gráfico de marcações da rede da Figura 4.6(a), evoluindo sob a função de disparo à máxima velocidade, é apresentado na Figura 4.6(c). Neste gráfico, a marcação, $M = M^d + M^i$, é a composição das fichas disponíveis e indisponíveis, de forma que não pode-se distinguir uma da outra. O arco, que liga a marcação M à M', indicando a mudança de estado do primeiro para o segundo, é marcado t_j/d_k (ou para várias transições habilitadas ao mesmo tempo, $\{t_i, t_j, ...\}/d_k$). A transição t_j, representa a única transição cujo disparo promoveu a mudança da marcação M para a marcação M', e d_k é o tempo decorrido entre a obtenção da marcação M e o disparo da transição t_k.

Na Figura 4.6(c), observamos que M_0, M_1 e M_3 possuem o mesmo número de fichas em cada um dos lugares. O que os torna diferente, embora não apareça claramente na representação, é o *tempo residual de indisponibilidade* de cada uma das fichas no momento de obtenção de cada marcação. Na Figura 4.7, esta informação fica clara pela indicação entre parenteses do tempo residual de indisponibilidade das várias fichas. O resíduo é zero para todas as fichas da marcação inicial (este resíduo poderia assumir outros valores).

Os modelos *PdPN* e *TdPN* são equivalentes [12, 13, 15]. A Figura 4.8 ilustra a transformação de uma *TdPN* (Figura 4.8(a)) para uma *PdPN* (Figura 4.8(b)). A transição temporizada t_1 é decomposta em duas transições t_1' e t_1'' e um lugar p_1' de forma que os lugares de entrada de t_1' são os lugares de entrada de t_1 e os lugares de saída de t_1'' são os de saída de t_1 e o tempo (atraso) que foi associado a transição agora é associado ao lugar p_1'. O tempo associado aos lugares , que antes pertenciam a *TdPN*, é zero, $d_i = 0$, na rede transformada. A marcação inicial dos lugares permanece e a dos lugares adicionados é zero.

De forma inversa, o lugar é decomposto em dois lugares e uma transição, na transformação de *PdPN* (Figura 4.9(a)) para uma *TdPN* (Figura 4.9(b)). Para toda transição t_j da rede inicial (Figura 4.9(a)), $d_j = 0$. A marcação inicial de p_1'' é a marcação inicial de p_1, e a marcação inicial de p_1 é zero.

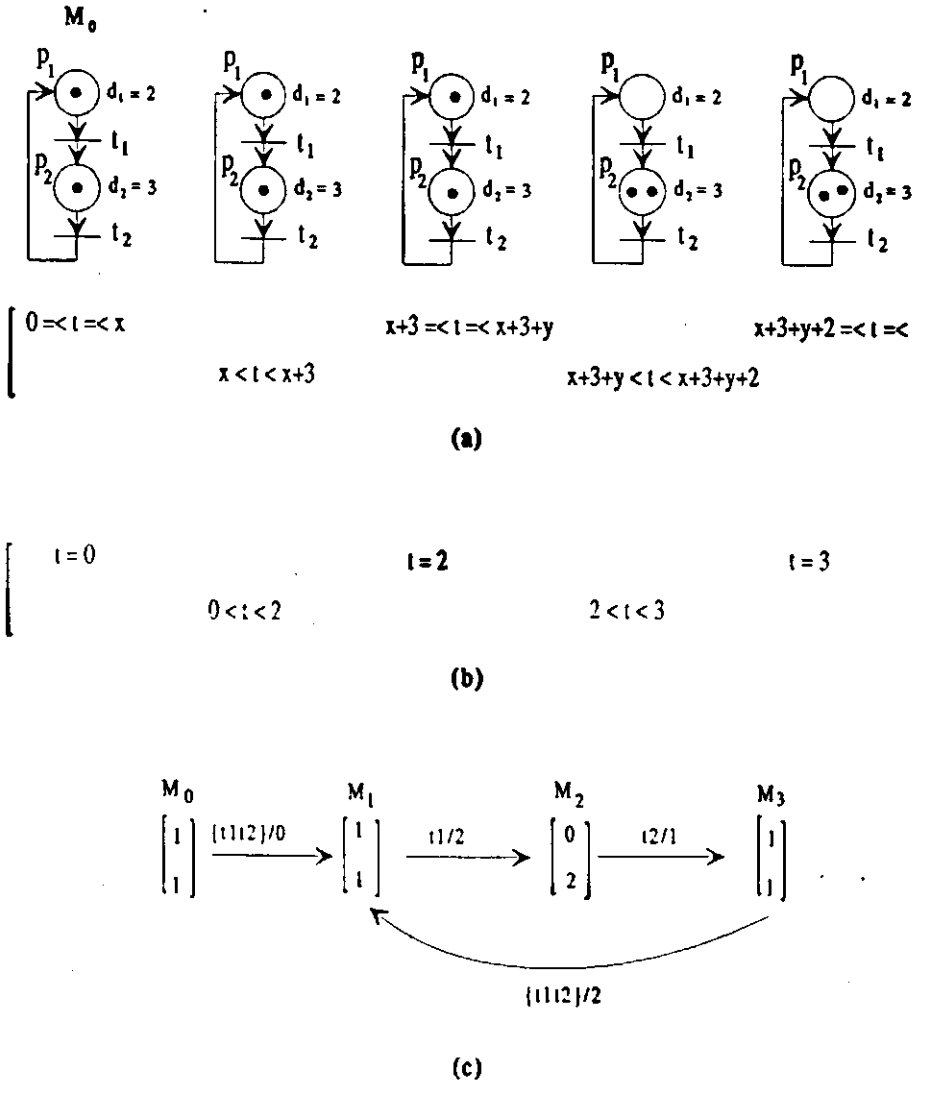


Figura 4.6: Evolução de uma *PdPN* (David [12]).

Portanto é sempre possível passar de uma *TdPN* para uma *PdPN*, e vice-versa, mas no caso geral a rede de Petri não marcada não é a mesma para ambos os casos [13].

4.2.3 Rede de Petri Temporal

O modelo de Rede de Petri Temporal (*Time Petri Net - TPN*), [4, 15], proposto por P. Merlin [27, 28], é um modelo no qual um intervalo de tempo $[a, b]$ é associado a cada transição da rede. O limite inferior do intervalo, a, representa o tempo mínimo que deve decorrer, a partir do instante em que as condições de habilitação de uma transição são satisfeitas, até o tempo no qual a transição pode disparar. O limite superior do intervalo, b, representa o tempo máximo que a transição pode permanecer habilitada sem ser disparada. Após o tempo b, a transição deve disparar. Este modelo é mais geral do que o de Ramchandani [18], desde que a duração d de um evento pode ser simplesmente modelado como $[d, d]$. É

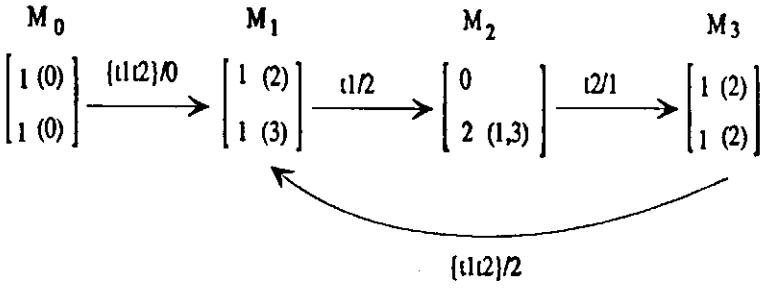


Figura 4.7: Indicação da duração do resíduo de indisponibilidade (David [12]).

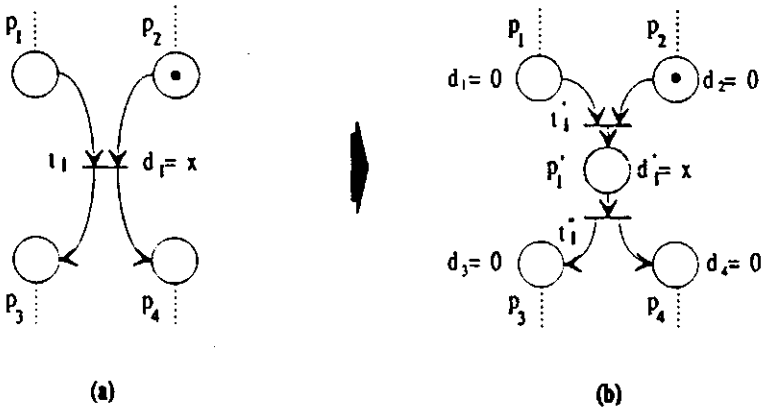


Figura 4.8: Transformação de uma *TdPN* em *PdPN (David [12])*.

claro, o formalismo básico de uma *RP* também pode ser representado como uma classe especial onde os limites de todos os eventos são $[0, \infty[$.

4.2.4 Rede de Petri P-Temporal

O modelo de Rede de Petri P-Temporal (*P-Time Petri Nets - PPN*) [17], difere do modelo *TPN* pois a restrição de tempo, intervalo $[a_i, b_i]$, é associada aos lugares da rede. Neste caso, quando uma ficha é deposiatada em um lugar p_i, ela permanece indisponível para habilitação de sua transição de saída, por um tempo a_i, e permanece disponível para habilitação e disparo da transição até b_i (após b_i a ficha é considerada morta e não pode mais ser utilizada para o disparo da transição). O formalismo de *PdPN* pode ser representado por este como uma classe especial onde os limites de todos os lugares são $[d_i, \infty]$.

Definição 4.4 *Uma Rede de Petri P-temporal é uma dupla*

$$PPN = (RP, IP), \text{ na qual}$$

- *RP* é uma rede de Petri marcada e
- *IP* é uma aplicação definida da seguinte maneira:

$$IP : P \rightarrow (\mathbf{Q}^+ \cup 0) \times (\mathbf{Q}^+ \cup \infty)$$

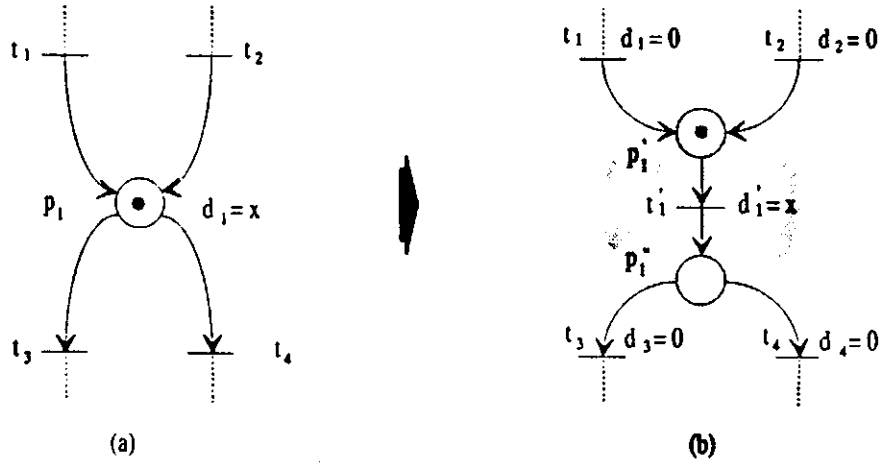


Figura 4.9: Transformação de uma *PdPN* em *TdPN (David [12])*.

$p_i \rightarrow IP_i = [a_i, b_i]$ com $0 \leq a_i \leq b_i; \forall i, 1 \leq i \leq m, m = Card(p)$.

Em Dutilleul [17], este tipo de rede é utilizada para modelar sistemas industriais caracterizados pelo tempo de operação incluídos entre um valor máximo e mínimo. A aplicação considerada é uma linha *eletroplant* para a qual os recursos de transporte precisam estar livres em um tempo compatível com as durações tratadas.

4.2.5 Rede de Petri P-temporal T-temporizada

A Rede de Petri P-Temporal T-temporizada (*P-Time T-Timed Petri Net - PTdPN*), definida por Julia [25, 24], é uma rede onde um intervalo de tempo $[a_i, b_i]$ é associado aos lugares e um tempo d às transições. Poderíamos dizer que esta rede é a combinação das redes *PPN* e *TdPN*. Esse tipo de rede é utilizada por Julia para tratar o problema de escalonamento dos sistemas de produção por lotes ("batch systems").

Definição 4.5 *A rede de Petri p-temporal t-temporizada é uma tríplice*

$$RPPT = (RP, IP, Tempo\_t), \text{ na qual}$$

- RP é uma rede de Petri marcada;
- IP é uma aplicação definida da seguinte maneira:

$$IP : P \rightarrow (\mathbf{Q}^+ \cup 0) \times (\mathbf{Q}^+ \cup \infty)$$

$$p_i \rightarrow IP_i = [a_i, b_i] \text{ com } 0 \leq a_i \leq b_i$$

- $Tempo\_t : T \rightarrow \mathrm{R}^+$.

$Tempo\_t$ é uma aplicação do conjunto das transições no conjunto dos números reais positivos ou nulos. $Tempo\_t\,(t_j) = d_j$ = tempo associado à transição t_j.

À ficha de um lugar p de uma rede $RPPT$, é associado um *intervalo de visibilidade* $\left[(\delta_p)_{\min}\,;(\delta_p)_{\max}\right]$ que define a data, $(\delta_p)_{\min}$, antes da qual a ficha não está disponível para o disparo da transição e, $(\delta_p)_{\max}$, a data a partir da qual a ficha tem *"morte"*, não pode mais ser utilizada para o disparo de nenhuma transição.

Se uma transição t tem n lugares de entrada e cada um destes possui uma ou várias fichas, então o *intervalo de sensibilização* desta transição, $[(\theta_t)_{\min}, (\theta_t)_{\max}]$, é obtido escolhendo-se para cada um dos n lugares uma ficha e o seu intervalo de visibilidade, bem como fazendo-se a intersecção daqueles intervalos de visibilidade.

O intervalo de *tempo de conflito* de duas transições, t_1 e t_2, que estão em conflito estrutural, é obtido fazendo-se a intersecção dos intervalos de sensibilidade das transições. Durante o intervalo de tempo obtido, t_1 e t_2 estão em conflito efetivo em relação a marcação. Fora deste intervalo, elas não estão em conflito efetivo.

A Figura 4.10, apresenta um exemplo onde os intervalos estáticos associados aos lugares são:

$$\begin{aligned}\left[(d_{p_1})_{\min} : (d_{p_1})_{\max}\right] &= [1,6]\\ \left[(d_{p_2})_{\min} : (d_{p_2})_{\max}\right] &= [0,7]\\ \left[(d_{p_3})_{\min} : (d_{p_3})_{\max}\right] &= [2,6]\end{aligned}$$

Na data 0, uma ficha chega a p_1, na data 2 uma ficha chega a p_3 e, na data 3, uma ficha chega a p_2. Segundo a Figura 4.10b, os intervalos de visibilidade das fichas dos lugares p_1, p_2, p_3 são:

$$\begin{aligned}\left[(\delta_{p_1})_{\min} : (\delta_{p_1})_{\max}\right] &= [1,6]\\ \left[(\delta_{p_2})_{\min} : (\delta_{p_2})_{\max}\right] &= [3,10]\\ \left[(\delta_{p_3})_{\min} : (\delta_{p_3})_{\max}\right] &= [4,8]\end{aligned}$$

Os intervalos de sensibilização são $[3,6]$ para t_1 e $[4,8]$ para t_2. O intervalo de conflito associado à dupla (t_1, t_2) é, então $[4,6]$.

4.2.6 Rede de Petri Interpretada

O termo "Rede de Petri Interpretada" pode ser aplicado a várias interpretações de acordo com o uso que se deseja fazer dela. Pelo menos duas interpretações podem ser encontradas: a apresentada por David [12, 13] e a de Litz [19]. O modelo de rede de Petri interpretada permite a modelagem de controladores lógicos e sistemas em tempo real [13].

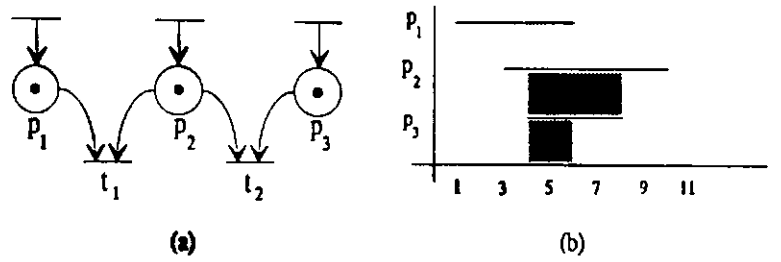


Figura 4.10: (a) Conflito para uma rede de Petri p-temporal t-temporizada. (b) Intervalos de sensibilização e conflito (Julia [25])

Na primeira abordagem, de David, uma rede de Petri Interpretada, *IPN*, exibe três características:

1. Ela é Sincronizada
2. Ela é P-Temporizada
3. Ela compreende uma parte de processamento de dados cujo estado é definido por um conjunto de variáveis $V = \{V_1, V_2, ...\}$. Este estado é modificado por operações $O = \{O_1, O_2, ...\}$ que são associados aos lugares. Ele determina o valor das condições (predicados) $C = \{C_1, C_2, ...\}$ que são associados às transições. A Figura 4.11, exibe estas características.

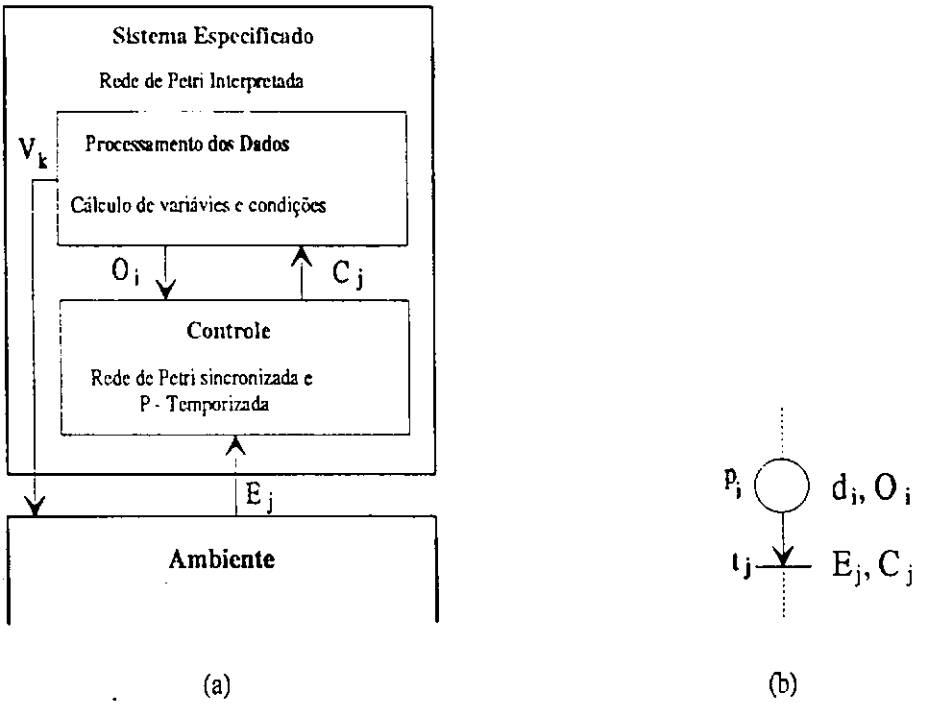


Figura 4.11: Rede de Petri Interpretada.

Neste tipo de rede, uma transição t_j é disparada na ocorrência de um evento E_j se a transição t_j estiver habilitada e a condição C_j for verdadeira. Quando a transição t_j está habilitada e a condição C_j é verdadeira, diz-se que t_j é *disparável na ocorrência* de E_j. Se uma ficha é depositada em um lugar p_i no instante t, a operação O_i é executada e a ficha fica indisponível por d_i.

Como foi visto anteriormente, existe uma equivalência entre *PdPN* e *TdPN*. Portanto pode-se construir uma rede de Petri Interpretada que seja T-temporizada.

A abordagem apresentada por Litz, difere em alguns aspectos da *IPN* e, portanto, o nome usado é *SIPN*.

Definição 4.6 *Uma SIPN é, formalmente, uma tupla* $SIPN = (RP, SE, SS, CE, MA)$ *com:*

- RP é uma rede de Petri elementar pura.
- SE é um conjunto, não vazio, de sinais binários de entrada.
- SS é um conjunto, não vazio, de sinais binários de saída.
- CE é um mapeamento, associando a toda transição $t_i \in T$ uma condição de disparo externa (expressão Booleana em SE).
- MA é um mapeamento, associando todo lugar $p_i \in P$ com uma ação $MA(i) \in \{0, 1, -\}^{|SS|}$ que é ativado assim que o lugar é marcado. $MA(i)[j]$ especifica o valor do sinal de saída ss_j.

Neste tipo de rede, uma transição dispara se ela está habilitada e se a sua condição de disparo é verdadeira. Todas as transições podem disparar (se habilitadas) simultaneamente desde que não estejam em conflito. A Figura 4.12, mostra o disparo em uma *SIPN*. No momento que o sinal i_1 torna-se verdadeiro t_1 dispara e a saída da rede muda de $(o_1, o_2, o_3, o_4) = (1, 1, -, -)$ para $(o_1, o_2, o_3, o_4) = (0, 0, 1, 1)$.

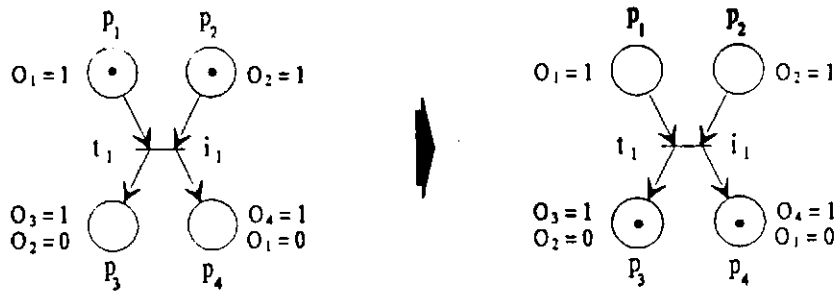


Figura 4.12: Disparo de uma *SIPN*.

A abordagem *SIPN* deixa clara a relação comportamental existente entre a troca de informações do algoritmo de controle e o processo. A planta a ser controlada possui:

Um conjunto de sinais binários de sensores, $Sen = \{s_1, ..., s_n\}$ e

Um conjunto de sinais binários de atuadores, $Act = \{a_1, ..., a_m\}$.

Desta forma o controlador reage aos sinais dos sensores e influencia o processo atuando sobre os atuadores. Portanto, as condições, $SE_{SIPN} = Sen$ e $SS_{SIPN} = Act$, precisam ser satisfeitas.

4.2.7 Redes de Petri em Tempo Real

As redes de Petri em Tempo Real, são uma classe de rede de Petri proposta por Venkatesh [38, 37], que utilizam-se de redes de Petri RP para o desenvolvimento de controladores seqüênciais de sistemas a eventos discretos.

Uma rede de Petri em tempo real pode ser obtida pela associação de tempo e informação de sensores e atuadores à redes de Petri não temporizadas.

Definição 4.7 *Uma rede de Petri em Tempo Real (Real-time Petri nets - RTPN), é definida como:*

$$RTPN = (RP, FD, FX, Y), \text{ na qual}$$

- RP é uma rede de Petri marcada;
- $FD : T \rightarrow \mathbf{R}^+$, é uma função de disparo temporizada;
- $FX : P \rightarrow \{-, 0, 1, 2, ...k\}$ e $FX(p_i) \neq FX(p_j)$, $i \neq j$, é uma função sinal de entrada, onde k é o máximo número de canais de sinais de entrada, e " $-$ " é um atributo que indica que nenhuma entrada está associada ao local;
- $Y : T \rightarrow \mathbf{N}$, é uma função sinal de saída.

Os três últimos elementos da $RTPN$, são assim explicados:

O vetor de tempo (FD) é utilizado para associar atrasos de tempo a transições que modelam as atividades no sistema;

O vetor de sinal de entrada (FX) lê o estado do sinal de entrada da interface digital de entrada. FX associa atributos a todo lugar. $FX_i = FX(p_i)$ e é um atributo associado com o lugar p_i.

O vetor de sinal de saída (Y) escreve um sinal de saída na interface digital de saída. Y associa atributos a todas as transições. $Y_i = Y(t_i)$ é o atributo associado à transição t_i que representa o número que é para ser enviado para a interface digital de saída.

Há dois eventos para o disparo de uma transição, *início do diparo* e o *fim do disparo*. Entre esses, o disparo está em progresso. A remoção das fichas dos locais de entrada da transição ocorre no início do disparo da transição. O depósito de fichas nos lugares de saída da transição ocorre no fim do diparo da transição. Enquanto o diparo da transição está em progresso, o tempo para terminar o disparo, chamado *tempo restante do disparo*, diminui da duração do disparo para zero no qual seu disparo é completado. As regras de execução de uma $RTPN$ incluem regras de habilitação e disparo:

1. Uma transição $t \in T$ está habilitada em M se e somente se
   - Satisfaz a regra de habilitação das redes de Petri e
   - $FX\,(p)$ tem conteúdo igual a 1.
2. O disparo de uma trasição $t \in T$ habilitada em M, resulta em uma nova marcação M', dada pela regra de disparo das redes de Petri.

O procedimento de projeto para a formulação de um controlador baseado em $RTPN$ é mostrado na Figura 4.13. Por este procedimento, observamos que o controlador pode ser formulado por uma dada seqüência de controle. Nesta abordagem a seqüência de controle é baseada no "jogador de fichas" (token game). Um controlador baseado em $RTPN$ pode ser constuído em, e executado por, um computador. Como a execução da $RTPN$ inicia e continua, o sistema controlado também inicia e realizando operações correspondentes a seqüência modelada pela $RTPN$.

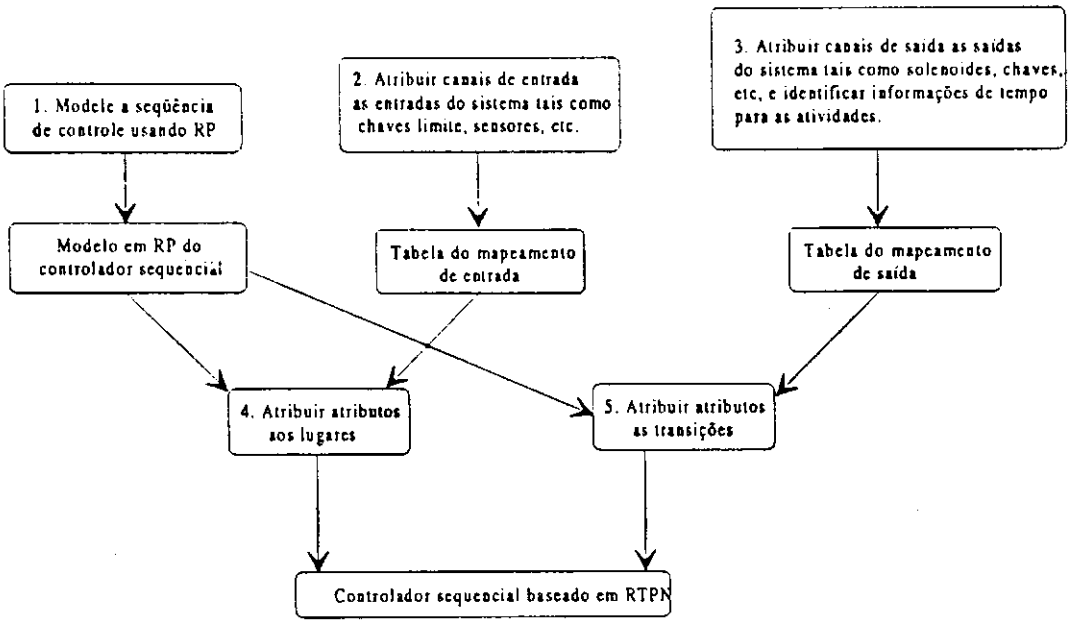


Figura 4.13: Procedimento para formular um controlador baseado em $RTPN$ [38, 37].

4.3 Estudo de Redes *TPN*

Durante o estudo das extensões de rede de Petri apresentadas, o que se buscava inicialmente era uma rede capaz de representar o tempo de resposta dos sensores e o tempo das atividades de um processo de manufatura. Isto porque se desejava representar o comportamento do sistema ao longo do tempo antes de controlá-lo. Depois, buscou-se uma rede que além disso, possibilitasse a associação de sinais de sensores (eventos externos) às transições, a associação de atividades aos lugares (sinais de comando), e ainda, tivesse um mecanismo que pudesse inibir o disparo de transições, quando habilitadas. Esta rede então, que teria a estrutura da inicial seria usada para a implementação da rede supervisora, e a primeira seria usada para modelar o sistema. Isto, estaria de acordo com a abordagem de síntese de supervisores de Barroso, pois a rede supervisora teria a mesma estrutura da rede que modela o sistema.

Dentre os modelos temporizados apresentados, há os mais simples que atribuem durações fixas de tempo a lugares com o objetivo de caracterizar que dada condição é verdadeira por uma certa quantidade de tempo, as *PdPN e as interpretadas de David* [12], ou à transições com o intuito de representar o tempo de duração da ocorrência de um evento, as *TdPN e RTPN*. Mas existem condições que não são fixas, ou não podem ser expressas por valores nominais, permitindo-se apenas a estimação (ou medição) de limites inferiores e superiores, as *PPN*, e da mesma forma para eventos, as *TPN*. Há também o modelo que atribui durações fixas de tempo a transiçoes e intervalos a lugares, as *RPPT*. Em Figueiredo [15], é definida a rede de Petri com temporização nebulosa (*FTPN*), não apresentada neste trabalho. É mostrado que as *FTPN*, através da associação de dois intervalos nebulosos de tempo às transições, são mais gerais do que as extensões temporais determinísticas que foram apresentadas aqui.

Nem todos os modelos aqui apresentados, dispunham de um método formal para a construção da árvore de alcançabilidade, necessária para a síntese do supervisor, no qual fosse considerada o tempo, somente a *TPN* e a *FTPN*. No entanto, as *FTPN* necessitam de descrições nebulosas do tempo para serem usadas, o que as tornam não tão práticas de serem usadas, enquanto as *TPN* necessitam de um intervalo de tempo que pode ser obtido por estimação ou por medição simples. Por isso, escolheu-se a rede *TPN* como modelo temporizado para a modelagem dos sistemas.

Agora faremos um estudo mais detalhado das *TPN*, desenvolvidas por Merlin. Esta seção é largamente baseada em Berthomieu e Diaz [4].

Definição 4.8 *Uma Rede de Petri Temporal (Time Petri Net - TPN), é assim definida*

$$TPN = (RP, IE)\,, \text{ na qual}$$

- RP é uma rede de Petri marcada e
- $IE : T \rightarrow [\alpha^s, \beta^s]$, é um mapeamento chamado intervalo estático, no qual α^s e $\beta^s \in$ Q e $\alpha^s \leq \beta^s$.

Para a análise das *TPNs* será preciso diferenciar os intervalos estáticos e os intervalos dinâmicos associados às transições. Os intervalos associados às transições durante a concepção do modelo são os intervalos estáticos de cada transição t_i, $[\alpha^s, \beta^s]$. O intervalo de tempo $[\alpha_i^s, \beta_i^s]$ é o *intervalo estático de disparo* associado a transição t_i. O limite inferior de disparo estático α_i^s será chamado de tempo de disparo estático inferior, *TDI*. O limite superior de disparo estático β_i^s será chamado de tempo de disparo estático superior, *TDS*.

A Figura 4.14, que reproduz um exemplo de Figueiredo [15], mostra um protocolo comunicando dois processos modelados por uma *TPN*. O exemplo mostra como as *TPNs* podem ser usadas para recuperar mensagens em um protocolo de comunicação. Inicialmente, uma ficha no lugar p_1 e uma ficha no lugar p_3 indicam respectivamente que o processo A está pronto para enviar uma mensagem e o processo B está pronto para receber uma mensagem. Quando o processo A envia uma mensagem, uma ficha é depositada nos lugares p_2 e p_4. O significado de uma ficha em p_4 é manter a informação a ser usada no caso de perda de uma mensagem. A recuperação da mensagem é acionada pelo disparo da transição t_3 que é determinado pelos intervalos de habilitação a ela associados. Nesse exemplo, o limite inferior do intervalo de habilitação, a, é maior do que o tempo estimado para o processo A receber o reconhecimento do processo B, representado pelo disparo da transição t_5. Os demais intervalos de habilitação não foram representados na figura.

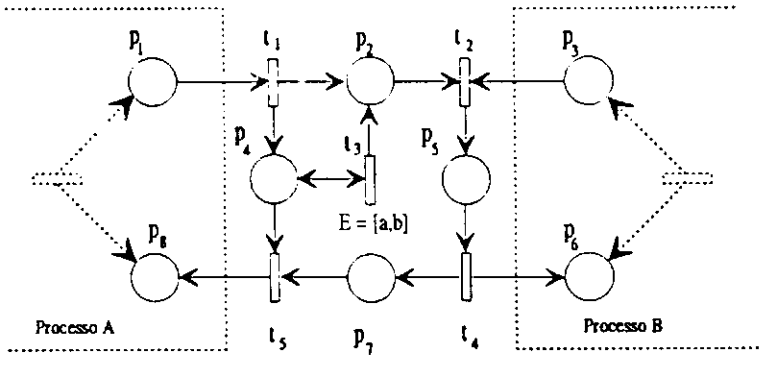


Figura 4.14: Exemplo de uma rede de Petri temporal [15]

4.3.1 Estados em uma *TPN*

Devido a associação de tempo às transições, o estado de uma *TPN* não é completamente expresso pelas suas marcações. O estado necessita incluir informações sobre o *intervalo*

estático de disparo das transições habilitadas. Supondo que a rede é segura, uma forma geral para o estado, Est, de uma TPN pode ser definida [4] como um par $Est = (M, Id)$ constituido de:

- uma marcação M;
- um conjunto de intervalos de disparo Id que é um vetor de possíveis tempos de disparo. O número de entradas deste vetor é igual ao número de transições *habilitadas pela marcação* M. Além disso, como o vetor Id possui uma entrada para cada transição habilitada para uma dada marcação, o número de entradas irá variar durante o comportamento da rede de acordo com o número de transições habilitads. A i-ésima entrada de Id define um par de valores de tempo, entre os quais a i-ésima transição pode disparar individualmente.

Vamos considerar a TPN da Figura 4.15. Ela representa o comportamento bifurcação e união que é tipicamente encontrado tanto em sistemas de manufatura quanto em sistemas distribuidos. Nesta representação a transição t_1 representa a operação de bifurcação, as três transições t_2, t_3 e t_4 representam as três ramificações paralelas, que podem ser tanto de processos de manufatura quanto de programas distribuidos, e a transição t_5 representa a sincronização e o fim da parte paralela. Finalmente, o processo é reiniciado pela transição t_6.

O estado inicial da Figura 4.15 tem marcação inicial $M_0 = (10000000)^T$. Para esta marcação somente t_1 está habilitada e, portanto, o vetor Id possui apenas uma entrada igual a [2,7], portanto t_1 pode disparar a qualquer instante de tempo entre 2 e 7.

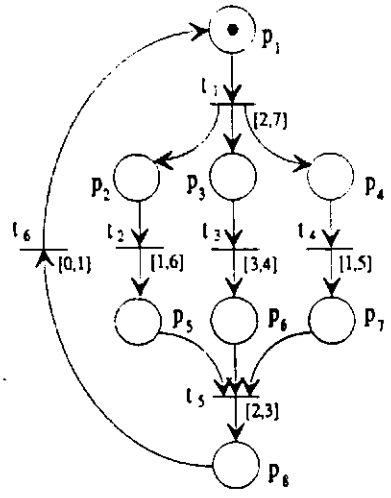


Figura 4.15: Uma rede de Petri Temporal.

4.3.2 Condições de Disparo de um Conjunto de Transições

Com a inclusão de tempo, as condições de disparo de uma transição de uma TPN sofrem alterações em relação as das redes de Petri. Supondo que o estado atual da TPN seja $Est = (M, Id)$, algumas transições podem estar habilitadas pela marcação, mas nem todas elas podem disparar devido a restrição de disparo das transições $[TDI, TDS]$.

Vamos assumir que a transição t_i torna-se habilitada, no tempo τ, no estado $Est = (M, Id)$. Sua *condição de disparabilidade* é formalmente expressa por duas condições:

(a) $\forall p \in {}^{\bullet}t, M(p) \geq I(p,t)$ no tempo τ, isto é, a transição está habilitada pela marcação segundo a regra de habilitação das redes de Petri no tempo τ;

(b) ela deve respeitar seus limites do *intervalo estático de disparo*, não podendo disparar antes do seu TDI e devendo disparar antes ou até o seu TDS, a menos que outra transição dispare antes e desabilite-a.

De acordo com estas condições, a transição t_i, habilitada no tempo τ, no estado $Est = (M, Id)$, é disparável no tempo $\tau + \theta$ se e somente se as seguintes condições são satisfeitas:

1. satisfaz a condição (a) da condição de disparabilidade;

2. o tempo relativo θ, relativo ao tempo absoluto de habilitação τ, não é menor do que o seu TDI e não é maior do que o menor dos TDS de todas as transições habilitadas na marcação M:

$$TDI \text{ de } t_i \leq \theta \leq \min \{TDS \text{ de } t_k\}$$

onde t_k cobre todas as transições habilitadas pela marcação M.

Observe que a condição (2) procede, pois no tempo $\theta = \min\{TDS \text{ de } t_k\}$, a correspondente transição, para a qual o TDS é mímimo, precisa disparar, modificando a marcação e portanto o estado da TPN.

O atraso θ não é um tempo global, ele é um tempo relativo ao tempo τ no qual o estado Est foi alcançado. Assim, o tempo absoluto de disparo de uma transição t_i habilitada no estado Est é "θ+ o tempo absoluto τ no qual o estado Est foi alcançado".

4.3.3 Regra de Disparo entre Estados

Uma vez que a transição t_i é disparada no tempo $\tau + \theta$ a partir do estado $Est = (M, Id)$, um novo estado $Est' = \left(M', Id'\right)$ é alcançado e pode ser calculado como segue:

1. A marcação M' é calculada, para todos os lugares, da mesma forma que para as redes de Petri

$$M'(p) = M(p) - I(t_i, p) + O(t_i, p)$$

2. Id' é calculado em três passos:

   (a) Remoção do vetor Id de todos os intervalos relacionados às transições desabilitadas pelo disparo de t_i, incluindo t_i;

   (b) Deslocamento dos intervalos, do valor θ, para a origem dos tempos, de todas as transições que permanecem habilitadas, e portanto permanecem em Id, e truncar seus TDI, quando necessário para valores positivos;

   (c) Introdução em Id' dos intervalos das transições habilitadas a partir de Est'.

Para exemplificar esta regra de disparo, consideremos a rede da Figura 4.15. O estado inicial Est_0 é composto pela marcação inicial $M_0 = (10000000)^T$ e pelo vetor de intervalos de atraso de disparo inicial $Id_0 = [2,7]$. Para este estado somente t_1 está habilitada e ela pode ser disparada em um tempo θ_1, que pode assumir uma infinidade de valores no intervalo [2,7]. O disparo de t_1 num tempo θ_1, neste caso é irrelevante o valor de θ_1 desde que esteja no intervalo [2,7], leva ao estado $Est_1 = (M_1, Id_1)$ com:

$$\begin{aligned} M_1 &= (01110000)^T \text{ e} \\ Id_1 &= [1,6]\,[3,4]\,[1,5], \end{aligned}$$

no qual Id_1 possui três entradas devido a t_2, t_3 e t_4 estarem habilitadas pela marcação M_1.

O disparo de t_2, por exemplo, pode ocorrer entre o tempo relativo 1, o TDI de $[1,6]$, e o tempo relativo 4, o menor valor relativo entre os maiores valores TDS das transições habilitadas na marcação M_1.

O disparo de t_2, num tempo θ_2 no intervalo $[1,4]$ leva ao estado $Est_2 = (M_2, Id_2)$ com:

$$\begin{aligned} M_2 &= (00111000)^T \text{ e} \\ Id_2 &= [\max(0, 3-\theta_2), 4-\theta_2]\,[\max(0, 1-\theta_2), 5-\theta_2] \end{aligned}$$

no qual Id_2 possui duas entradas devido a t_3 e t_4 estarem habilitadas pela marcação M_2.

Os intervalos de Id_2 podem variar

$$\begin{aligned} \text{de } Id_2 &= [2,3]\,[0,4] \text{ para } \theta_2 = 1 \\ \text{a } Id_2 &= [0,0]\,[0,1] \text{ para } \theta_2 = 4 \end{aligned}$$

Para o caso onde o disparo de t_2 ocorre em $\theta_2 = 4$, t_3 deve disparar imediatamente após t_2, pois seu intervalo é $[0,0]$.

4.3.4 Alcançabilidade de uma TPN

Assim como em uma rede de Petri elementar, o disparo de uma transição habilitada t_i mudará o estado atual para um estado futuro. No entanto, agora, este disparo acontece

em um instante de tempo θ relativo ao tempo de habilitação de t_i. Assim, representamos o disparo de t_i a partir do estado Est no instante de tempo θ resultando no estado Est' como:

$$Est \overset{(t_i,\theta)}{\rightarrow} Est'$$

Uma seqüência de disparo será uma seqüência de pares da forma

$$(t_1,\theta_1).(t_2,\theta_2)...(t_n,\theta_n)$$

na qual $t_1, t_2, ..., t_n$ são transições e $\theta_1, \theta_2, ..., \theta_n$ são tempos.

Diz-se que um estado Est_n é alcançável a partir do estado Est_0 se existe uma seqüência de disparos que transformam Est_0 em Est_n tal que

$$Est_0 \overset{(t_1,\theta_1)}{\rightarrow} Est_1 \overset{(t_2,\theta_2)}{\rightarrow} Est_2 - - - \rightarrow Est_{n-1} \overset{(t_n,\theta_n)}{\rightarrow} Est_n$$

Por exemplo, seja o sistema da Figura 4.15. Para o estado inicial Est_0, com marcação inicial $M_0 = (10000000)^T$ e um vetor de intervalos de atraso de disparo $Id_0 = [2,7]$, o estado Est_5 com marcação $M_5 = (00000001)$ e $Id_5 = [2,3]$, é alcançável pela seqüência de disparo $(t_1,\theta_1).(t_2,\theta_2).(t_3,\theta_3).(t_4,\theta_4).(t_5,\theta_5)$. Isto é escrito como:

$$Est_0 \overset{(t_1,\theta_1)}{\rightarrow} Est_1 \overset{(t_2,\theta_2)}{\rightarrow} Est_2 - - - \rightarrow Est_4 \overset{(t_5,\theta_5)}{\rightarrow} Est_5$$

A regra de disparo permite a obtenção dos estados e uma relação de alcançabilidade entre eles. O conjunto de estados que são alcançáveis ou o conjunto de seqüências de disparo possíveis a partir do estado inicial caracterizam o comportamento da TPN. Infelizmente, em geral, não é possível usar este conjunto de estados para propositos de análise desde que este conjunto pode ser infinito devido a infinidade de valores que os θ podem assumir nos intervalos: somente simulação pode ser conduzida para abordagens simples; um método geral apropriado proposto por Berthomieu e Diaz [4], para uma análise exaustiva, será dada na próxima seção.

4.3.5 Classes de Estados e Alcançabilidade

Quando calcula-se um novo estado em uma TPN, observa-se que para o disparo de uma única transição existe uma infinidade de valores possíveis de tempo de disparo relativo, quantidade esta que está confinada a um intervalo. O caso é que para cada valor de tempo de disparo, um novo estado é gerado, mas todos possuem a mesma marcação. Embora a marcação seja a mesma, os vetores de intervalos de disparo estático têm seus valores diferentes. Então, informalmente falando, uma classe de estados será definida como a união de todos esses estados.

Classe de Estados

A partir de um estado podemos disparar uma transição habilitada em uma infinidade de valores de tempo pertencentes ao seu *intervalo estático de disparo* e conseqüentemente obter uma infinidade de estados decorrentes desses disparos. Agregando-se todos esses possíveis estados em um pseudoestado, teremos uma classe, associada aos possíveis disparos da transição, denominada classe de estados.

Uma classe de estados é um par $C = (M, D)$ no qual:

- M é a marcação da classe: todos os estados na classe possuem a mesma marcação;
- D é o domínio de disparo da classe; ele é definido como a união dos domínios de disparo de todos os estados na classe.

A transformação de estado para classe de estado, dada no diagrama da Figura 4.16 para uma dada transição, mostra que as classes são definidas contendo todos os possíveis tempos de disparo que podem ocorrer a partir de uma dada marcação alcançada.

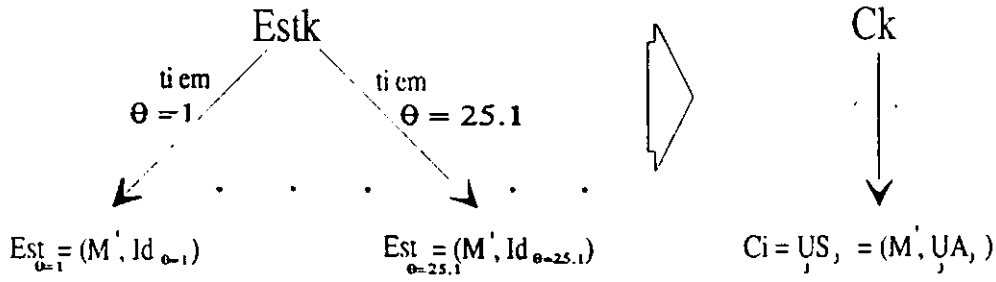


Figura 4.16: Transformação de estados para classes.

Cálculo de Classes

É interessante saber como se obter uma classe de estados a partir de uma existente, para então termos uma árvore de alcançabilidade de classes, de forma a nos possibilitar o estudo do comportamento temporal de uma TPN.

Uma transição, em uma classe, é inicialmente considerada como potencialmente habilitada para o disparo, se ela está habilitada pela marcação, independentemente do seu *intervalo estático de disparo*. Uma vez de posse de todos os intervalos de disparo estático de todas as transições habilitadas no estado Est, é preciso determinar os intervalos de disparo das transições de forma que eles não comprometam as restrições temporais das outras transições. Se existir um intervalo de disparo para a transição, então a transição é dita habilitada para o disparo, e ela pode ser disparada para a obtenção de uma outra classe.

Para a classe gerada a marcação é determinada de acordo com a regra de disparo que rege as redes de Petri. Para a obtenção dos novos intervalos de disparo estático desta classe

deve-se observar a existência das diferentes expressões de intervalos para estados, expressões de domínio para classe de estados são capazes de introduzir e capturar relações sutis entre o tempo de disparo das diversas transições, por exemplo quando elas permanecem habilitadas após o disparo; este é o ponto chave.

Assumindo que t_i seja a i-ésima transição habilitada pela marcação M. Ela é disparável na classe $C = (M, D)$ se e somente se as seguintes condições são satisfeitas:

(a) $\forall p \in {}^\bullet t_i,\ M(p) \geq I(p, t_i)$;

(b) $TDS_{t_i} \leq TDS_{t_j}, \forall j, j \neq i$.

Observe que a condição (b) está de acordo com a condição 2 da condição de disparo.

Seria difícil expressar a condição (b) acima para as classes usando somente os TDI e TDS das transições, como foi feito para os estados. Isto deve-se a relações não triviais existentes entre os tempos de disparo das diferentes transições, como mostraremos agora.

Consideremos dois casos ilustrativos relacionados às classes do sistema da Figura 4.15.

Caso 1, caso no qual nenhuma transição permanece habilitada após o disparo. A classe inicial, C_0 é dada por

$$\begin{aligned} M_0 &= (10000000)^T \\ D_0 &= \text{(todas as soluções de) } 2 \leq \theta_1 \leq 7. \end{aligned}$$

Após o disparo da transição t_1, a classe resultante C_1 é dada por

$$\begin{aligned} M_1 = &\ (01110000)^T \\ D_1 = &\ 1 \leq \theta_2 \leq 6 \\ &\ 3 \leq \theta_3 \leq 4 \\ &\ 1 \leq \theta_4 \leq 5 \end{aligned}$$

Então a transição t_2 pode disparar se além disso,

$$\begin{aligned} \theta_2 &\leq \theta_3 \\ \theta_2 &\leq \theta_4. \end{aligned}$$

Caso 2, é o caso geral no qual algumas transições habilitadas antes do disparo permanecem habilitadas após o disparo. Antes do disparo de t_1 não havia nenhuma transição a mais habilitada na classe C_0, portanto nenhuma transição permaneceu habilitada após o disparo, como foi apresentado no caso anterior. Um caso complexo ocorre quando algumas transições permanecem habilitadas após o disparo; este caso será considerado agora.

Na classe C_1, t_2, t_3 e t_4 estão habilitadas. O disparo de t_2, por exemplo, é possível se o seguinte sistema tem solução:

$$1 \leq \theta_2 \leq 6 \tag{4.1}$$

$$3 \leq \theta_3 \leq 4 \tag{4.2}$$

$$1 \leq \theta_4 \leq 5 \tag{4.3}$$

$$\theta_2 \leq \theta_3 \tag{4.4}$$

$$\theta_2 \leq \theta_4 \tag{4.5}$$

Dele conclui-se que t_2 pode disparar no intervalo $1 \leq \theta_2 \leq 4$, respeitando as demais restrições de t_3 e t_4. As equações (4.1), (4.2) e (4.3) são válidas para t_2, t_3 e t_4. Se for considerado o disparo de t_3 em vez de t_2, então as inequações (4.4) e (4.5) precisam ser trocadas por $\theta_3 \leq \theta_2$ e $\theta_3 \leq \theta_4$. Devido a marcação M_1, o intervalo de disparo para t_3 é $3 \leq \theta_3 \leq 4$. Procedendo da mesma forma, o intervalo de t_4 é obtido, $1 \leq \theta_4 \leq 4$.

Disparando t_2, por exemplo, t_3 e t_4 permanecem habilitadas na nova classe. Para estas transições deve-se calcular seus novos intervalos de atraso de disparo. Esses intervalos são obtidos pelo devido deslocamento de seus intervalos originais de um intervalo igual ao intervalo de disparo da transição t_2. Para entender como é feito este deslocamento vamos supor que t_2 é disparada a partir da classe C_1 no tempo relativo θ_{2f}.

Supondo que t_2 é disparado após um dado tempo θ_{2f} (entre 1 e 4), como mostrado na Figura 4.17. Acontece que, após o disparo de t_2, as transições t_3 e t_4 permanecem habilitadas decorridos θ_{2f} unidades de tempo. Após o disparo, seus novos valores, θ'_3 e θ'_4, podem ser definidos por uma translação onde $\theta_i = \theta'_i + \theta_{2f}$.

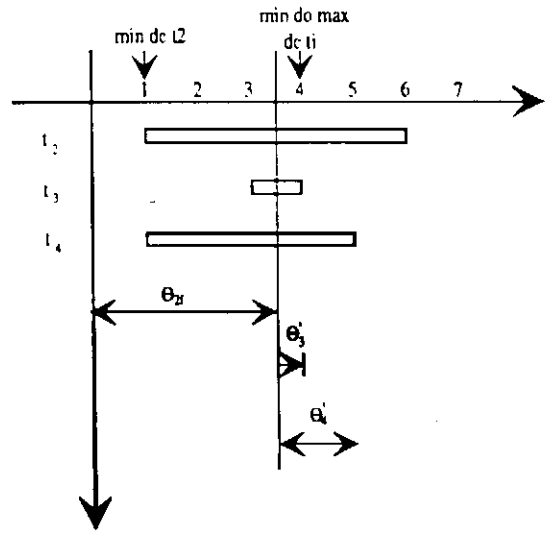


Figura 4.17: Representação de tempo de disparo.

Disparando t_2, usando: $2 \leq \theta_3 \leq 4$, $1 \leq \theta_4 \leq 5$, e $\theta_i = \theta'_i + \theta_{2f}$, dá

$$2 \leq \theta'_3 + \theta_{2f} \leq 4 \tag{4.6}$$

$$1 \leq \theta'_4 + \theta_{2f} \leq 5 \tag{4.7}$$

ou

$$2 - \theta_{2f} \leq \theta'_3 \leq 4 - \theta_{2f} \tag{4.8}$$

$$1 - \theta_{2f} \leq \theta'_4 \leq 5 - \theta_{2f} \tag{4.9}$$

com

$$1 \leq \theta_{2f} \leq 4. \tag{4.10}$$

Como θ_{2f} introduz uma relação entre t_3 e t_4, (4.6) e (4.7) podem ser escritas como

$$2 - \theta'_3 \leq \theta_{2f} \leq 4 - \theta'_3 \tag{4.11}$$

$$1 - \theta'_4 \leq \theta_{2f} \leq 5 - \theta'_4 \tag{4.12}$$

Eliminando θ_{2f} das relações entre os tempos de disparo de t_3 e t_4 e definindo os tempos de disparo da próxima classe:

$$0 \leq \theta'_3 \leq 2 \text{ de (4.8) e (4.10)}$$

$$0 \leq \theta'_4 \leq 2 \text{ de (4.9) e (4.10)}$$

$$\theta'_4 - \theta'_3 \leq 2 \text{ de (4.11) e (4.12)}$$

$$\theta'_3 - \theta'_4 \leq 2 \text{ de (4.11) e (4.12).}$$

As duas últimas inequações definem uma área do plano e expressam a existência e as relações necessárias entre t_3 e t_4.

Regra de Disparo da *TPN*

Consideremos que a transição $t(f)$ é disparada a partir da classe $C = (M, D)$ resultando na classe $C' = (M', D')$. A nova classe $C' = (M', D')$ é obtida conforme os seguintes passos:

(a) A marcação M' é calculada, para todos os lugares, da mesma forma que para as redes de Petri

$$M'(p) = M(p) - I(t_i, p) + O(t_i, p)$$

(b) O domínio D' é calculado a partir do domínio D através de três passos.

(i) Acrescentar ao sistema de inequações, formado pelos intervalos de disparo de D, as condições de disparabilidade para a transição $t(f)$; isto leva ao sistema:

$$\alpha_i \leq t(i) \leq \beta_i, \text{ todos os intervalos de } D, \text{ incluive o de } t(f)$$

$$t(f) \leq t(j) \ \forall j, j \neq f$$

Faça a seguinte mudança de variáveis: expresse todos os tempos relacionados às variáveis $t(j)$, com $j \neq f$, como o somatório do tempo de disparo da transição $t(f)$ e da nova variável $t'(j)$ com

$$t(j) = t'(j) + t(f), \forall j, j \neq f$$

e elimine do sistema a variável $t(f)$ pela derivação dos novos intervalos e as relações de restrições necessárias.

O sistema resultante pode ser escrito como

$$\alpha'_i \leq t'(i) \leq \beta'_i, \forall t(i) \text{ que permaneceu habilitado}$$
$$t(j) - t(k) \leq \gamma_{jk}, \forall j \text{ e } k, \text{ com } j \neq k \text{ e } \forall \text{ par } t(j), t(k),$$

que permanecem habilitados, no qual $t(m)$ representa tanto a m-ésima transição habilitada nesta classe quanto o possível tempo de disparo associado a m-ésima transição.

(ii) Eliminar do sistema obtido após o passo (i), dando-se conta das relações que elas implicam, todas as variáveis correspondentes às transições desabilitadas pelo disparo de $t(f)$: essas transições são habilitadas por M mas não por M'.

(iii) Incluir ao sistema obtido após o passo (b), novas variáveis associadas às transições habilitadas em M', e definir o intervalo destas variáveis como o intervalo estático da respectiva transição que representa.

Definição Formal do Comportamento de uma Rede de Petri Temporal

Quando os disparos são considerados, o domínio da classe de estados pode ser descrito como um conjunto de desigualdades da forma:

$\alpha_i \leq t(i) \leq \beta_i$, $\forall i$, para cada uma $t(i)$ recém habilitada ou que permaneça habilitada,

$t(j) - t(k) \leq \gamma_{jk}$, $\forall j, k$ com $j \neq k$, para cada $t(j), t(k)$ que permanece habilitada.

Lemma 4.1 *O domínio de disparo D da classe de estados para uma TPN segura pode ser expresso como o conjunto de soluções do sistema de desigualdades da seguinte forma:*

$$\alpha_i \leq t(i) \leq \beta_i \ \forall i$$
$$t(j) - t(k) \leq \gamma_{jk} \ \forall j, k \ k \neq j$$

A prova deste Lema pode ser encontrada em Berthomieu e Diaz [4].

A marcação inicial da classe inicial é a marcação inicial da rede. O domínio inicial é definido como o conjunto de soluções do sistema de desigualdades: $\alpha_i^s \leq t(i) \leq \beta_i^s$, $\forall t(i)$,

onde $t(i)$ está associado com a i-ésima transição habilitada pela marcação inicial e α_i^s e β_i^s são os valores dos intervalos estáticos associados às transições.

Valores padrões para γ_{jk} podem ser obtidos com $\gamma_{jk} = \beta_j^s - \alpha_k^s$: desta forma, aquelas desigualdades são redundantes e não afetam o conjunto solução do sistema $\alpha_i^s \leq t(i) \leq \beta_i^s$.

Com o uso da regra de disparo pode-se construir a árvore de classes. A árvore tem como raiz a classe inicial e existe um arco rotulado como t_i ligando C a C' se t_i é disparável a partir da classe C e se seu disparo leva à classe C'. Deve-se dar importância ao fato de cada classe ter somente um número finito de sucessores, em um máximo de um para cada transição habilitada pela marcação da classe. Na árvore, duas classes de estado C_1 e C_2 são consideradas iguais se e somente se

suas marcações são iguais, $M_1 = M_2$

e seus domínios de disparo são iguais, $D_1 = D_2$.

Para maiores detalhes sobre o método de análise de *TPNs* recomenda-se a leitura do trabalho de Berthomieu e Diaz [4] que apresenta ainda algumas propriedades deste tipo de rede.

Capítulo 5

Abordagem Temporal para a Síntese de Supervisores

Neste capítulo apresentaremos a abordagem temporal para a síntese e implementação de supervisores para sistemas a eventos discretos proposta neste trabalho. Na etapa de síntese são introduzidos o *Algoritmo da Árvore de Alcançabilidade de Classes - AAAC*, utilizado para a obtenção do espaço de estados considerando o tempo associado às transições da rede que modela o sistema (para o qual se deseja obter um supervisor), e uma nova classe de rede de Petri para modelar o supervisor, *Rede de Petri Temporal Interpretada com Funções de Habilitação de Transição - RPTIFHT.*

Na etapa de implementação é introduzido o *Algoritmo para a Elaboração do Programa para o CLP - AEPCLP*, que especifica como converter a rede supervisora obtida na etapa de síntese em um programa em lista de instruções para um controlador lógico programável, *CLP.*

Veremos ainda o programa em linguagem C que realiza automáticamente esta abordagem a partir do modelo do sistema, da especificação funcional desejada e dos mapeamentos dos canais de entrada e saída do *CLP* para a rede suparvisora.

5.1 Metodologia de Síntese e Implementação de Supervisores

Na Figura 5.1 temos a descrição, em diagrama de blocos, dos passos a serem seguidos para a síntese e implementação de supervisores a partir de modelos temporais. Podemos observar que o processo começa com a obtenção do modelo do sistema em rede de Petri temporal, *TPN.* Esta extensão de rede de Petri permite a associação de intervalos de tempo às transições da rede e são utilizados, em geral, para representar a quantidade de tempo

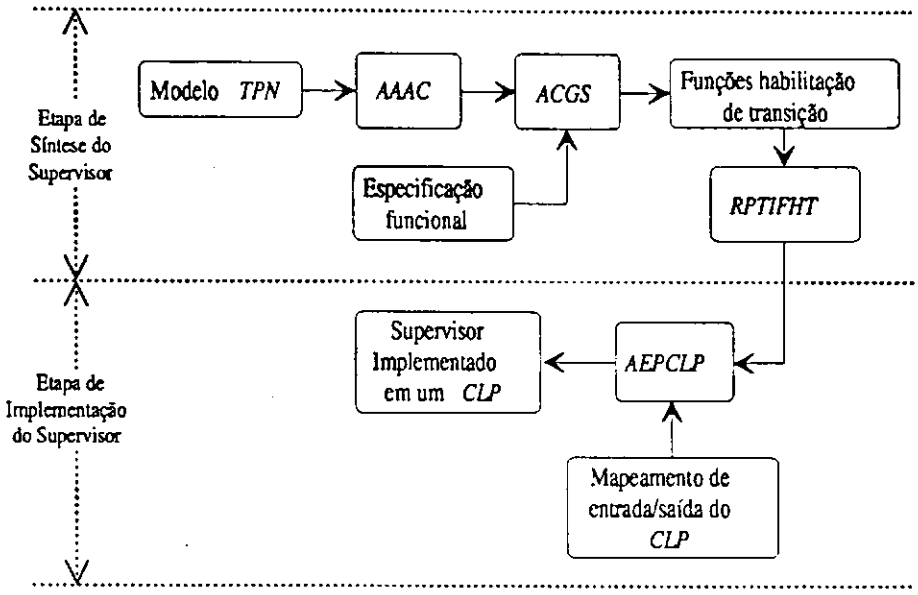


Figura 5.1: Diagrama em blocos do procedimento de síntese e implementação do supervisor.

mínima e máxima que um evento leva para ocorrer, após todas as suas pré-condições terem tornado-se verdadeiras. De posse do modelo e de uma marcação inicial, se constrói o espaço de estados deste modelo.

Na etapa de síntese, a construção do espaço de estados é feita segundo o *Algoritmo da Árvore de Alcançabilidade de Classes*, *AAAC*, que considera o tempo associado às transições para a enumeração do espaço de estados. Este algoritmo é baseado no método enumerativo para análise de redes de Petri temporais de Berthomieu e Dias [4], apresentado no capítulo 4, e no *Algoritmo Modificado da Árvore de Alcançabilidade* de Barroso, *AMArA*, apresentado no capítulo 3. O algoritmo foi elaborado para enumerar o espaço de estados de modelos em redes de Petri temporais seguras.

Com o espaço de estados obtido e a especificação funcional desejada executa-se o *Algoritmo para a Construção do Gerador da Suprema Linguagem Controlável*, *ACGS*, para encontrar a especificação possível e a seguir as funções que restringirão a ocorrência de eventos, representada pelo disparo das transições na *Rede de Petri Temporal Interpretada com Funções de Habilitação de Transição - RPTIFHT*, para que a especificação possível seja executada pelo sistema sob supervisão.

Na etapa de implementação, o supervisor obtido, modelado em *RPTIFHT*, é implementado em um controlador lógico programável, *CLP*, pela execução do *Algoritmo para a Elaboração do Programa para o CLP*, o *AEPCLP*. Este algoritmo, que tem como dados de entrada a rede supervisora e os mapeamentos entre canais de entrada e saída do *CLP* e os lugares e transições da rede supervisora, transforma a rede supervisora em um programa em *Lista de Instruções* para o *CLP*.

Começaremos a discussão sobre as etapas que compreendem o procedimento de síntese

e implementação do supervisor pela fase de obtenção do espaço de estados do modelo do sistema. O modelo do sistema é considerado um dado de entrada e pode ser obtido conforme o método descrito no capítulo 3 para modelagem com redes de Petri, associando-se às transições do modelo obtido, os devidos intervalos de tempo.

5.1.1 Etapa de Síntese do Supervisor

Algoritmo da Árvore de Alcançabilidade de Classes

O espaço de estados do sistema, para o qual se deseja sintetizar o supervisor, é conseguido pela execução do algoritmo *AAAC*. Este algoritmo enumera todos os estados possíveis que o sistema pode atingir, a partir de um estado inicial, considerando o tempo associado às transições da rede *TPN* segura que modela o sistema. A seguir veremos os passos que constituem o *AAAC* e um exemplo.

Algoritmo 1 Algoritmo para construção da Árvore de Alcançabilidade de Classes para uma rede *TPN* segura (*AAAC*):

Início

1. Componha a classe inicial C_0, com a marcação inicial M_0 e o vetor de intervalos de disparo Id_0, intervalos estáticos das transições habilitadas em M_0, e rotule-a como raiz e etiquete-a como *nova*;
2. Enquanto existirem classes do tipo *nova* faça:
   - (a) Selecione uma classe C do tipo *nova*;
   - (b) Resolva o sistema de inequações da classe C e determine as transições habilitadas para o disparo e seus respectivos intervalos de disparo;
     - (i) Se nenhuma transição está habilitada em C, etiquete C como *bloqueada*;
     - (ii) Enquanto existirem transições habilitadas em C, faça o seguinte para cada transição habilitada em C:

       Obtenha a classe C' que resulta do disparo de t em C;

       Se C' é idêntica a uma classe já existente, etiquete-a como *antiga*;

       Se a capacidade de algum lugar p é excedida na classe C', então substitua $M'(p)$ por w e etiquete C' como *não-permitida*;

       Se C' não for *antiga* e nem *não-permitida* etiquetá-la como *nova*;

       Introduza C' como um nó da árvore, ligue um arco, com rótulo t, de C para C';

3. Reconstrua a árvore, a partir da árvore gerada para elimimar as classes repetidas. Nesse estágio, uma classe é considerada igual a uma outra se ela possuir a mesma marcação e as mesmas classes subseqüentes de uma outra.

Fim.

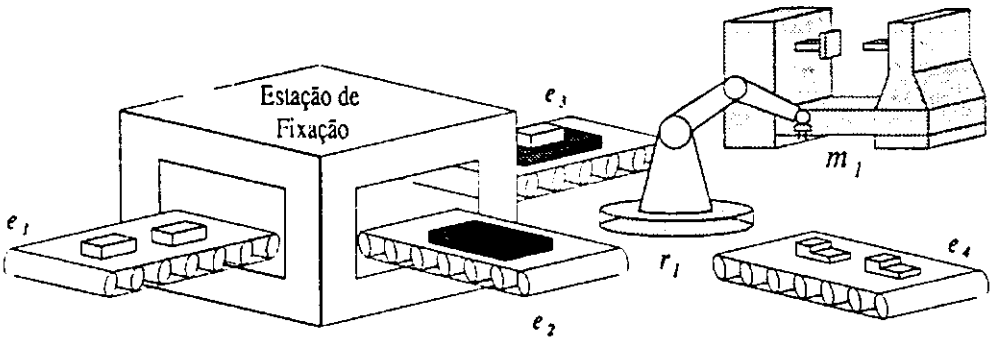


Figura 5.2: Célula de manufatura.

No algoritmo uma classe é constituída de uma marcação e um domínio de disparos (ver seção 4.3.5). As denominações *antiga, bloqueada* e *não-permitida* têm o mesmo significado de quando usadas no *AMArA* (ver seção 3.2.3). No segundo passo do algoritmo duas classes são consideradas idênticas se possuírem a mesma marcação e o mesmo domínio de disparos. O terceiro passo do algoritmo é usado para compactar, sem perda de informação útil, a representação da árvore. A compactação é possível se existirem classes com a mesma marcação e domínios de disparo diferentes mas que possuam todas as classes subseqüentes iguais.

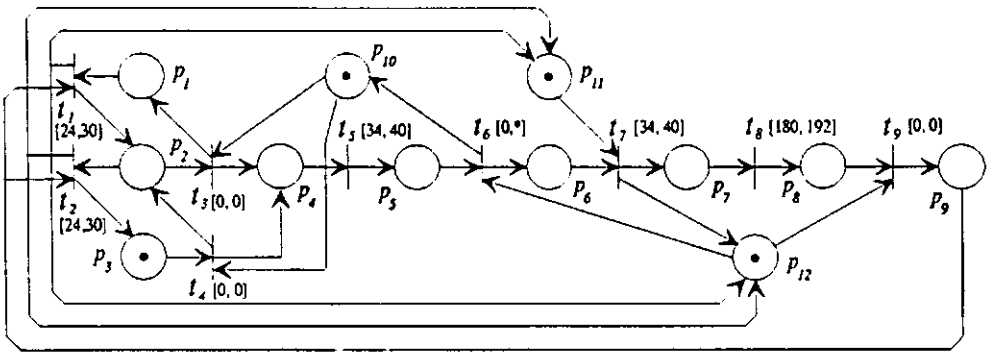


Figura 5.3: Modelo em rede de Petri temporal da célula de manufatura.

Tomemos como exemplo a célula de manufatura da Figura 5.2. A célula é constituída de uma estação de fixação, um robô (r_1), uma máquina (m_1) e quatro esteiras (e_1, e_2, e_3, e_4).

Nela as peças a serem processadas chegam pela esteira e_1 e são fixas às bandejas disponíveis em e_2, na estação de fixação. As peças fixas às bandejas são depositadas na esteira e_3 de onde são transportadas para m_1 por r_1. Após o processamento, a peça é separada da bandeja por m_1. Em seguida, r_1 transporta a peça trabalhada para e_4 e depois a bandeja para e_2. Admite-se que sempre há peças na esteira e_1 e que há duas bandejas na célula. O modelo em rede *TPN* para esta célula é mostrado na Figura 5.3. A tabela 5.1 contém a descrição dos lugares e a tabela 5.2 contém a descrição das transições.

Lugar (com ficha)	Interpretação
p_1, p_2 e p_3	e_2 vazia, com uma bandeja e com duas bandejas
p_4	peça sendo fixa à bandeja
p_5	peça fixa à bandeja
p_6	r_1 transportando conjunto para m_1
p_7	m_1 processando
p_8	peça processada
p_9	r_1 descarregando m_1
p_{10}	estação de fixação livre
p_{11}	m_1 livre
p_{12}	r_1 livre

Tabela 5.1: Interpretação dos Lugares

Transição	Interpretação
t_1, t_2	Fim da descarga de m_1 por r_1
t_3, t_4	início da fixação de uma peça a uma bandeja
t_5	término da fixação da peça a bandeja (conjunto)
t_6	início do transporte do conjunto para m_1
t_7	fim do transporte do conjunto para m_1 e início do processamento
t_8	fim do processamento
t_9	início da descarga de m_1 por r_1

Tabela 5.2: Interpretação das Transições

Neste modelo, somente a transição t_6 é controlável e as demais são não controláveis. A execução de uma transição controlável pode ser atrasada pelo supervisor. Mais especificamente, o disparo de uma transição controlável depende da decisão do supervisor. Considera-se que a duração da operação modelada por uma transição controlável é conhecida precisamente. O "sinal indicativo" de uma transição controlável é o limite superior do intervalo, associado a ela, ser infinito. Isto significa que o disparo da transição habilitada

pode ser bloqueada indefinidamente e mesmo não executada. O limite inferior do intervalo tem o significado da limitação do *hardware* - o tempo necessário para realizar a decisão de execução. Neste exemplo, associaremos à transição controlável o intervalo [0,∞], o valor zero do limite inferior indica que o tempo de execução da decisão pelo supervisor é nulo. A Figura 5.3, mostra o modelo da célula de manufatura em rede de Petri temporal, nela o simbolo * representa o ∞.

Executando o $AAAC$ obtemos a árvore de alcançabilidade de classes da rede da Figura 5.3. Na Figura 5.4(a) temos o gráfico de classes que representa esta árvore, obtida executando-se o $AAAC$ até o segundo passo. As classes obtidas até este passo são as seguintes:

$$\text{Classe } C_0 \begin{cases} M = (001000000111)^T \\ D : 0 \leq \theta_4 \leq 0 \end{cases}$$

$$\text{Classe } C_3 \begin{cases} M = (010001000110)^T \\ D : \begin{matrix} 0 \leq \theta_3 \leq 0 \\ 34 \leq \theta_7 \leq 40 \end{matrix} \end{cases}$$

$$\text{Classe } C_6 \begin{cases} M = (100100100001)^T \\ D : \begin{matrix} 0 \leq \theta_5 \leq 6 \\ 180 \leq \theta_8 \leq 192 \end{matrix} \end{cases}$$

$$\text{Classe } C_7 \begin{cases} M = (100010100001)^T \\ D : \begin{matrix} 0 \leq \theta_6 \leq \infty \\ 180 \leq \theta_8 \leq 192 \end{matrix} \end{cases}$$

$$\text{Classe } C_8 \begin{cases} M = (100010100001)^T \\ D : \begin{matrix} 0 \leq \theta_6 \leq \infty \\ 174 \leq \theta_8 \leq 192 \end{matrix} \end{cases}$$

O passo três do $AAAC$ diz respeito a situações nas quais temos classes como a C_7 e a C_8, que têm a mesma marcação mas o domínio de disparo diferente. No entanto, ambas apresentam as mesmas classes subseqüentes. Esta característica faz com que o passo três do algoritmo considere C_7 igual a C_8. Desta forma executando-se o $AAAC$ até o terceiro passo obtemos a árvore de classes representada pelo gráfico de classes da Figura 5.4(b).

Como dito na seção 4.2, a inclusão de tempo no modelo pode inibir o disparo de algumas transições em alguns estados. Podemos observar isto, no nosso exemplo, nas classes C_3 e C_6, que embora possuam duas transições habilitadas pela marcação só há o disparo de uma transição devido a restrição temporal. Para uma melhor visualização desta situação, temos na Figura 5.5(a) o gráfico de estados que representa o espaço de estados obtido pelo uso

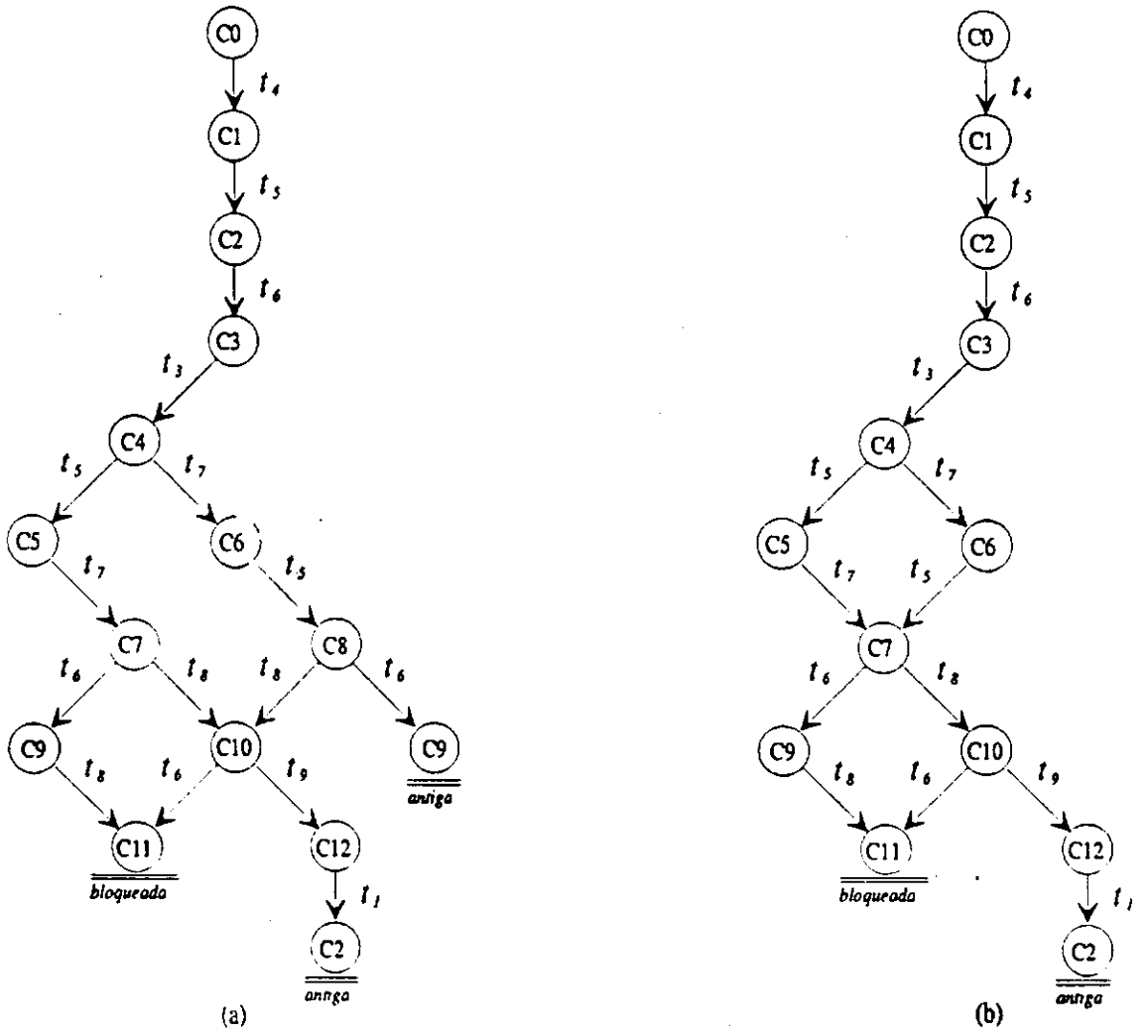


Figura 5.4: Gráficos de classes: (a) executando o $AAAC$ até o 2° passo, (b) executando o $AAAC$ até o 3° passo.

do $AMArA$ e na Figura 5.5(b) o espaço de estados obtido usando-se o $AAAC$. Por estas figuras podemos observar que o conjunto de seqüências de disparo no espaço de estados obtido pelo $AAAC$ é um subconjunto do conjunto de seqüências de disparo no espaço de estados obtido pelo uso do $AMArA$.

Devemos salientar que o $AAAC$ também pode obter o espaço de estados de um modelo em rede de Petri. Para isso, é suficiente que sejam associados intervalos de tempó iguais a $[0,\infty]$ a todas as transições do modelo.

Com o espaço de estados decorrente da execução deste algoritmo e com uma especificação funcional desejável, pode-se executar o $ACGS$ e obter as classes e as respectivas transições que precisam ser desabilitadas, *lista perigo da especificação*, para a realização da especificação. A especificação nesta abordagem é um dado de entrada constituído por um conjunto de pares, formados de classes e transições, que constituem o caminho da classe inicial C_0 à classe que representa o estado marcado do sistema. O $ACGS$ pode ser uti-

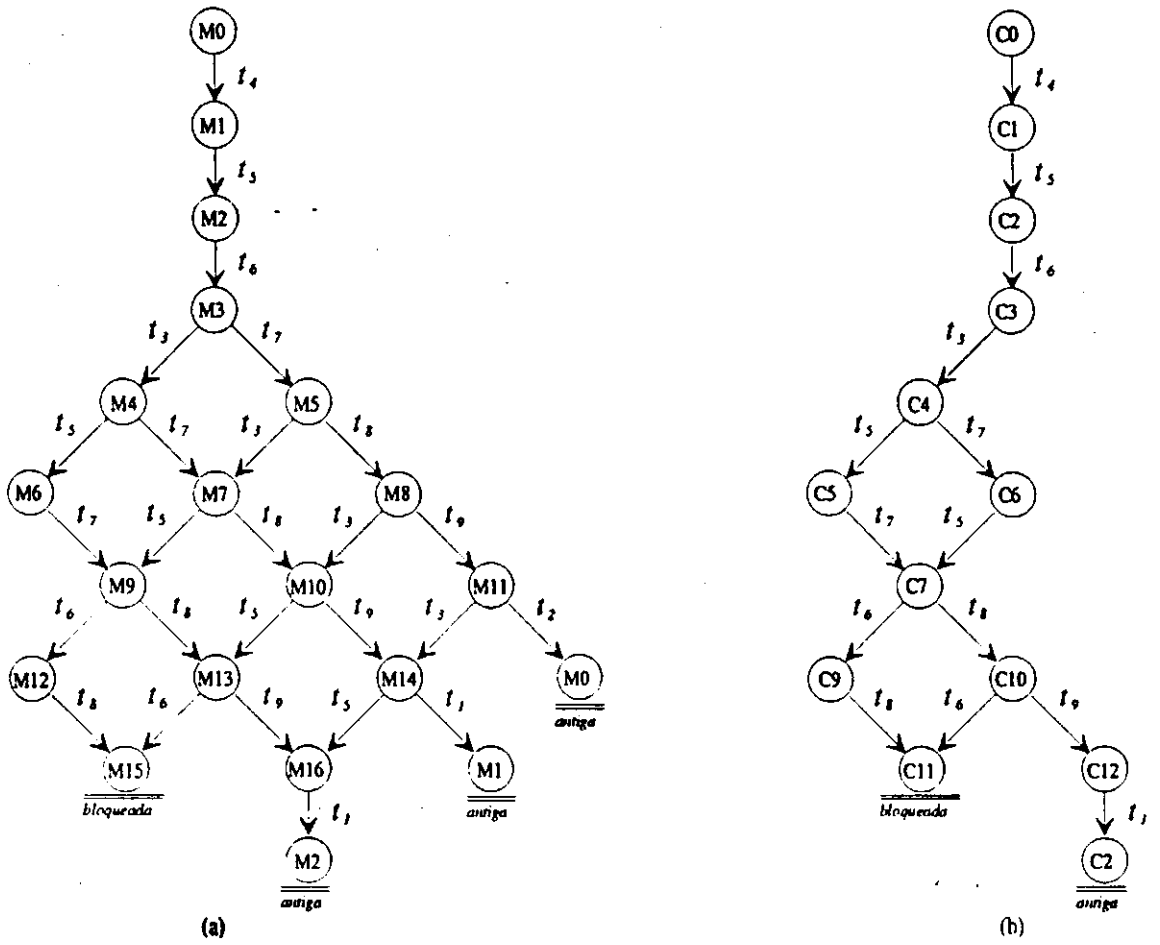


Figura 5.5: Gráficos de: (a) estados, (b) classes .

lizado para a síntese sem alterações nesta abordagem temporal, pois o espaço de estados gerado pelo $AAAC$ apresenta-se ao $ACGS$ como um gerador finito que modela a operação do sistema sob análise.

Rede de Petri Temporal Interpretada com Função de Habilitação de Transição

Na abordagem temporal proposta, o que se deseja da rede de Petri, que será utilizada como rede supervisora do sistema de manufatura, é que ela tenha a mesma estrutura da rede que modela o sistema, seja capaz de sincronizar o disparo de suas transições a eventos externos, associar ações aos lugares, temporizar ações, de realizar suas ações de controle através da restrição da ocorrência do disparo das transições e executar suas ações de controle em tempo real. Para se obter tal rede, combinou-se as características das redes de Petri Temporal, rede de Petri Interpretada, Rede de Petri com funções de Habilitação de transições e Redes de Petri em tempo real. Desta forma a rede de Petri supervisora, *Rede de Petri Temporal Interpretada com Função de Habilitação de Transições*, é definida a seguir.

Definição Uma rede de Petri temporal interpretada com funções de habilitação de

transições é uma quádrupla

$$RPTIFHT = (RP_T, \Phi, FX, Y), \text{ na qual}$$

- $RP_T = (P, T, I, O, IE, M_0)$ é uma estrutura de rede de Petri Temporal segura (ver capítulo 4);
- $\Phi = (\varphi_1, ..., \varphi_m) : R(RP_T, M_0) \rightarrow \{0, 1\}$ é a função de habilitação das transições (ver capítulo 3);
- $FX : P \rightarrow \{-, 0, 1, 2, ..., k_1\}$ e $FX(p_i) \neq FX(p_j), i \neq j$, é um mapeamento, associando todo lugar $p_i \in P$ com uma ação que é ativada assim que o lugar é marcado, onde k_1 é o número máximo de canais de saída do *dispositivo que implementa o supervisor* e "$-$" é um atributo que indica que nenhum canal está associado ao lugar.
- $Y : T \rightarrow \{-, 0, 1, 2, ..., k_2\}$ é um mapeamento, associando toda transição $t_i \in T$ a uma condição de disparo externa, k_2 é o número máximo de canais de entrada do *dispositivo que implementa o supervisor* e "$-$" é um atributo que indica que nenhum canal está associado a transição.

O comportamento dinâmico de uma *RPTIFHT* é descrito pela seguinte regra de disparo das transições:

1. Uma transição t é dita estar habilitada para disparar no instante de tempo τ, relativo ao tempo absoluto no qual ela se torna habilitada pela marcação segundo a regra de habilitação das redes de Petri (seção 3.1), se $Y(t)$ tem conteúdo igual a 1 e se a função φ associada à transição é verdadeira.
2. A transição dispara respeitando seus limites do *intervalo estático de disparo*, não podendo disparar antes do seu limite inferior do intervalo estático de disparo e devendo disparar antes ou até o seu limite superior do intervalo estático de disparo, a menos que outra transição dispare antes e desabilite-a.
3. Como resultado de um disparo o estado da rede muda e o novo estado da rede pode ser obtido segundo a regra de disparo entre estados, seção 4.3.3.

Note que a regra de disparo de uma transição em uma *RPTIFHT*, é igual a de uma transição em uma *TPN*, acrescida da condição imposta pela função de habilitação de transição e do mapeamento Y.

5.1.2 Etapa de Implementação do Supervisor

Com os resultados do *ACGS* e os mapeamentos dos canais de entrada e saída, do dispositivo que irá implementar o supervisor, para as transições e lugares da rede supervisora, podemos implementar um supervisor. O supervisor pode ser implementado tanto em um computador quanto em um *CLP*. Devido o uso de *CLPs* na automação industrial ser mais comum do que o uso de um *PC* escolheu-se implementar o supervisor em um *CLP*. Esta implementação se dá pela programação de um *CLP*, que é feita utilizando-se o Algoritmo para a Elaboração do Programa para um Controlador Lógico Programável (*AAAC*).

Controlador Lógico Programável

Define-se como *CLP* um equipamento eletrônico digital que tem como objetivo implementar funções específicas de controle e monitoração sobre variáveis de uma máquina ou processo por intermédio de módulos de entrada e saída [35, 11, 36, 30]. Os *CLPs* são evoluções dos controladores digitais de processos, contando com muito mais recursos que eles. Nele, todas as funções disponíveis devem poder ser programadas em uma memória interna e o *hardware* deve ser universal, de forma a permitir sua aplicação tanto em processos de produção contínuos quanto discretos.

Os *CLPs* encontram grande aplicação em quase todos os setores industriais envolvendo controle de processos, automação da manufatura, integração de sistemas de automatização, linhas de fabricação e montagem, automação predial, controle de subestações de energia, onde quer que sejam requeridas funções de controle, seqüenciamento e intertravamento de ações e supervisão do andamento de alguma atividade.

Os *CLPs* atuais são baseados em microprocessadores ou microcontroladores. Basicamente, eles são compostos por dois elementos principais: uma *CPU* (Unidade Central de Processamento) e interfaces para as variáveis de entrada e saída. As variáveis de entrada, são sinais externos recebidos pelo *CLP*, os quais podem ser oriundos de fontes pertencentes ao processo sob controle ou de comandos gerados pelo operador. Tais sinais são gerados por dispositivos como sensores diversos, chaves ou botoeiras, dentre outros. As variáveis de saída são os dispositivos controlados por cada um dos canais de saída do *CLP*. Tais canais poderão servir para intervenção direta no processo sob controle por acionamento próprio, ou também poderão servir para sinalização de estado em painel sinótico. Podem ser citados como exemplos de variáveis de saída os contactores, válvulas, lâmpadas, displays, dentre outros. A Figura 5.6 mostra o diagrama de blocos de um *CLP* genérico.

O princípio fundamental de funcionamento do *CLP* é a execução por parte da *CPU* de um programa, conhecido como "executivo" e de responsabilidade do fabricante, que realiza ciclicamente as ações de leitura das entradas, execução do programa de controle

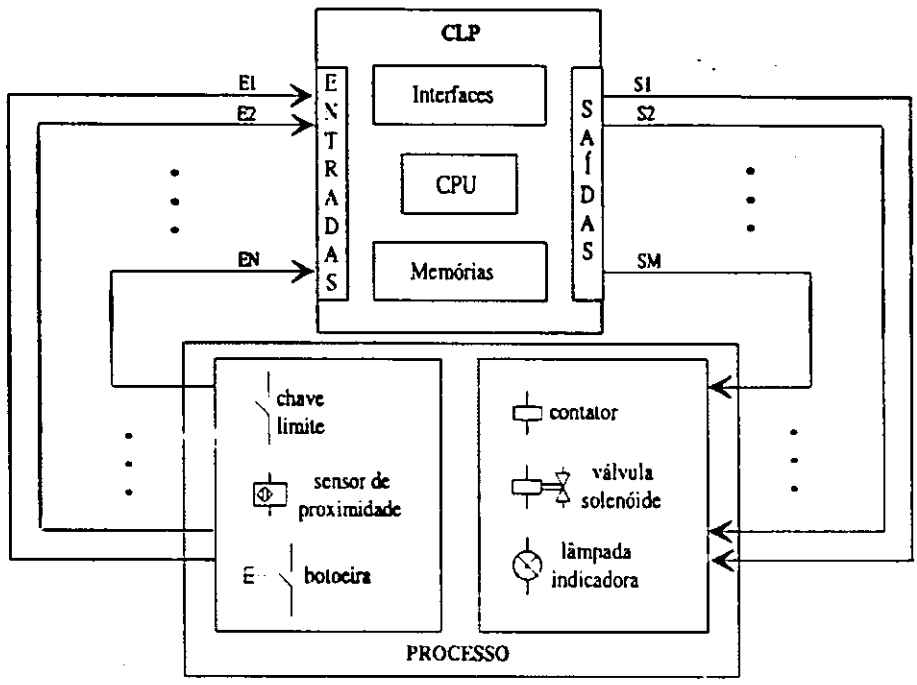


Figura 5.6: Configuração básica de um *CLP*.

do usuário e atualização das saídas, conforme ilustrado na Figura 5.7. Desta forma, ele atende a funções de seqüenciamento e intertravamentos[1] elétricos por meio de comparações, contagens, temporizações e controle *PID*, em conformidade com o programa do usuário. O programa do usuário é uma seqüência específica de instruções selecionadas de um conjunto de opções oferecidas pelo *CLP* em uso e, que irão efetuar as ações de controle desejadas, ativando ou não as memórias internas e os canais de saída do *CLP* a partir da monitoração do estado das mesmas memórias internas e/ou dos canais de entrada do *CLP*.

O tempo total para a execução dessas tarefas, chamado ciclo de varredura ou *scanning*, depende, dentre outros fatores, da velocidade e características do processador utilizado, do tamanho do programa de controle do usuário, além da quantidade e tipo de canais de entrada/saída. Como regra geral, tal tempo se encontra na faixa média de milissegundos (até microssegundos nos *CLP* de última geração).

Os *CLP* foram originalmente desenvolvidos com a intenção de substituir os painéis de controle a relés. Naquele contexto, uma linguagem de programação que fosse familiar à experiência dos técnicos e engenheiros, já acostumados com a lógica de relés, seria a escolha mais adequada ao desenvolvimento de programas para o *CLP*. Assim, desde então, os *CLPs* são primariamente programados em diagramas de relés (*ladder diagram*, que é um termo

[1]Intertravamentos são condições restritivas como a habilitação ou inibição de operação ou funcionamento de um equipamento. Entretanto, de forma mais concreta, podem ser considerados como funções que não permitem qualquer tipo de mudança de estado ou de ação até que outros estados ou ações estejam completadas. Existem, em princípio, os seguintes tipos de intertravamento: o de partida, o de funcionamento, o temporizado, o de não simultaneidade, o de seqüência e o de processo [30].

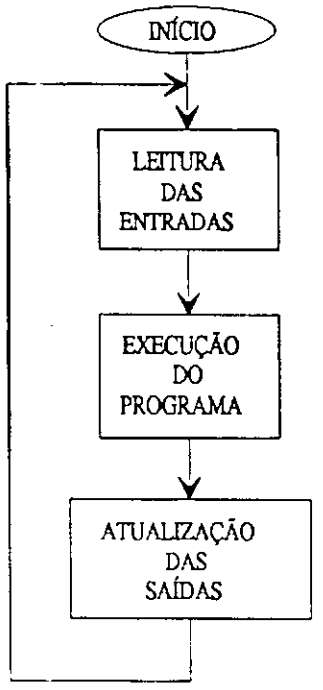


Figura 5.7: Ciclo de processamento de um *CLP*.

inglês para a descrição de circuitos de relés), independente de seu porte.

Muitas linguagens foram desenvolvidas para o *CLP*, desde a sua criação. Estas, podem ser do tipo textuais ou gráficas, basicamente. Dentre estes tipos as três linguagens mais difundidas de programação são:

- Programação a partir de operadores lógicos (ou *Instruction List - IL*): nesta, a lógica de comando é representada na forma de operadores booleanos do tipo AND, OR, XOR, NOT, etc. As linguagens de programação deste tipo se assemelham bastante com linguagens de máquina de microprocessadores (assembly). Se a lógica foi originalmente implementada com relés (situação freqüentemente encontrada na prática) é preciso obter a função lógica associada, a partir da qual é gerado o programa.

- Programação baseada no Diagrama de Escada (ou *Ladder Diagramm - LD*): esta linguagem foi criada para facilitar a implantação no *CLP* de lógicas originalmente realizadas com relés. A programação é realizada de forma gráfico-interativa e o programa resultante tem a forma dos chamados *diagramas de escada* ou *diagramas de relé*.

- Programação em *GRAFCET* (ou *Sequential Flow Chart - SFC*): a linguagem *GRAFCET* (GRAFo de Comando Etapa-Transição) representa um método moderno de programação baseado na técnica de redes de Petri, que a torna mais adequada que as anteriores para a implementação de lógicas complexas. A linguagem permite simultaneamente a programação e a verificação da correção lógica do programa (validação).

Os programadores de *CLP* encontram grandes dificuldades em programar diferentes tipos de *CLPs* devido as diferenças de linguagens e sintaxe de programação existentes entre os fabricantes. Com o propósito de estabelecer um padrão do modelo pelo qual os *softwares* de programação pudessem processar seus comandos, manipular suas variáveis e sua própria estrutura de apresentação, foi criado o comitê internacional IEC (International Electrotechnical Committee). Por este comitê foi elaborada a norma internacional IEC 1131. Nesta norma, em seu quesito terceiro, a IEC 1131-3, são definidos os padrões que definem as especificações mínimas das linguagens a serem respeitadas e as regras para as expansões futuras. Mais alguns detalhes sobre a IEC 1131 e as linguagens de programação para *CLP* são apresentados no Apêndice A.

Algoritmo para a Elaboração do Programa para um Controlador Lógico Programável - *AEPCLP*

O *AEPCLP* é o algoritmo utilizado para transformar a rede supervisora, obtida na etapa de síntese do supervisor, em um programa para *CLP* que possui estrutura igual a mostrada na Figura 5.8. O programa é na verdade um executivo de redes *RPTIFHT* constituído de duas partes: a parte de atualização dos estados e controle, *PAEC*, e a parte de atualização das funções de habilitação das transições, *PAFHT*.

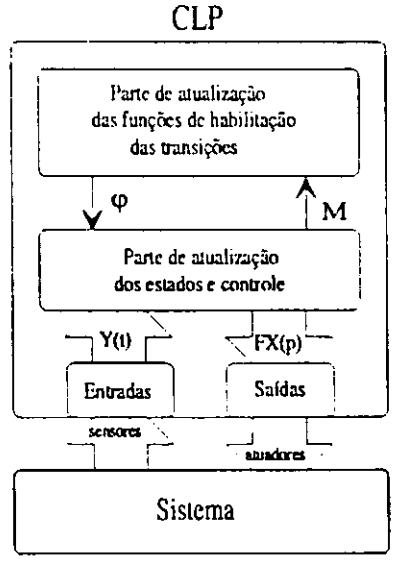


Figura 5.8: Estrutura do controlador.

A *PAEC* constitui o núcleo do executivo. Nesta parte são reproduzidas todas as relações existentes entre as transições e lugares da rede supervisora, e destes com os canais de entrada e de saída do *CLP*. Aqui, cada um dos lugares da rede é representado por uma variável interna com as quais se gera a informação de estado, que é equivalente ao estado do sistema. As funções de habilitação de transições também são representadas por variáveis internas.

A cada disparo de uma transição um novo estado interno é gerado e este é utilizado pela *PAFHT* para atualizar as funções de habilitação de transição.

A seguir será apresentado o algoritmo *AEPCLP*. Para um melhor entendimento do algoritmo, os itens do mesmo serão ilustrados pelas Figuras 5.10 e 5.11. Nestas figuras trechos de redes são convertidos em diagramas de escada para uma melhor compreensão do leitor, lembrando que neste trabalho a implementação do algoritmo gera o programa para o *CLP* em lista de instruções. Os símbolos gráficos do diagrama de escada, utilizados nas figuras 5.10 e 5.11, são apresentados na Figura 5.9.

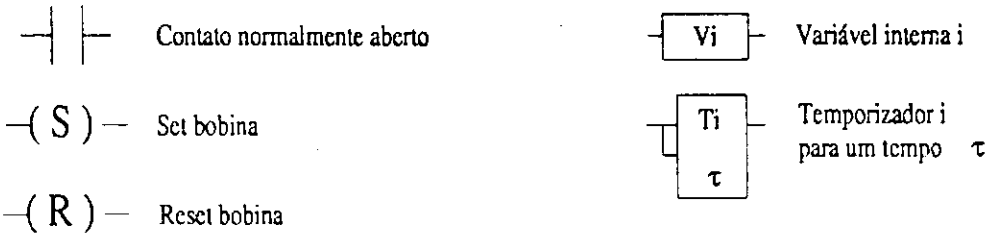


Figura 5.9: Símbolos gráficos do Diagrama de Escada (*LD*).

Algoritmo 2 Algoritmo para a Elaboração do programa para o *CLP*:

Início

1. Deve-se criar váriaveis internas para representar cada um dos lugares da rede supervisora e cada uma das funções de habilitação de transições (as controláveis). O valor inicial de cada uma das variáveis, que representam os lugares, corresponde a quantidade de fichas em cada lugar na marcação inicial, e o das funções é 1;

2. Para construir a parte de atualização de estados e controle deve-se construir uma lógica para cada transição da rede supervisora, da seguinte maneira:

   - Os lugares de entrada são associados em lógica *AND*. Eles são representados pelas variáveis internas quando não forem associados as saídas do *CLP*, ou são representados por saídas do *CLP* em caso contrário, conforme Figura 5.10(a);
   - Se a transição for uma transição controlável, agregar à lógica dos lugares de entrada a variável interna que representa a função de habilitação desta transição, conforme Figura 5.10(b);
   - Se à transição estiver associado um canal de entrada do *CLP*, agregar o canal de entrada à lógica dos lugares de entrada desta transição, conforme Figura 5.11(a);
   - Se à transição estiver associado um tempo de um temporizador interno ao *CLP*, a lógica dos lugares de entrada da transição constituirá a lógica de habilitação do temporizador que representará esta transição, conforme Figura 5.11(b);

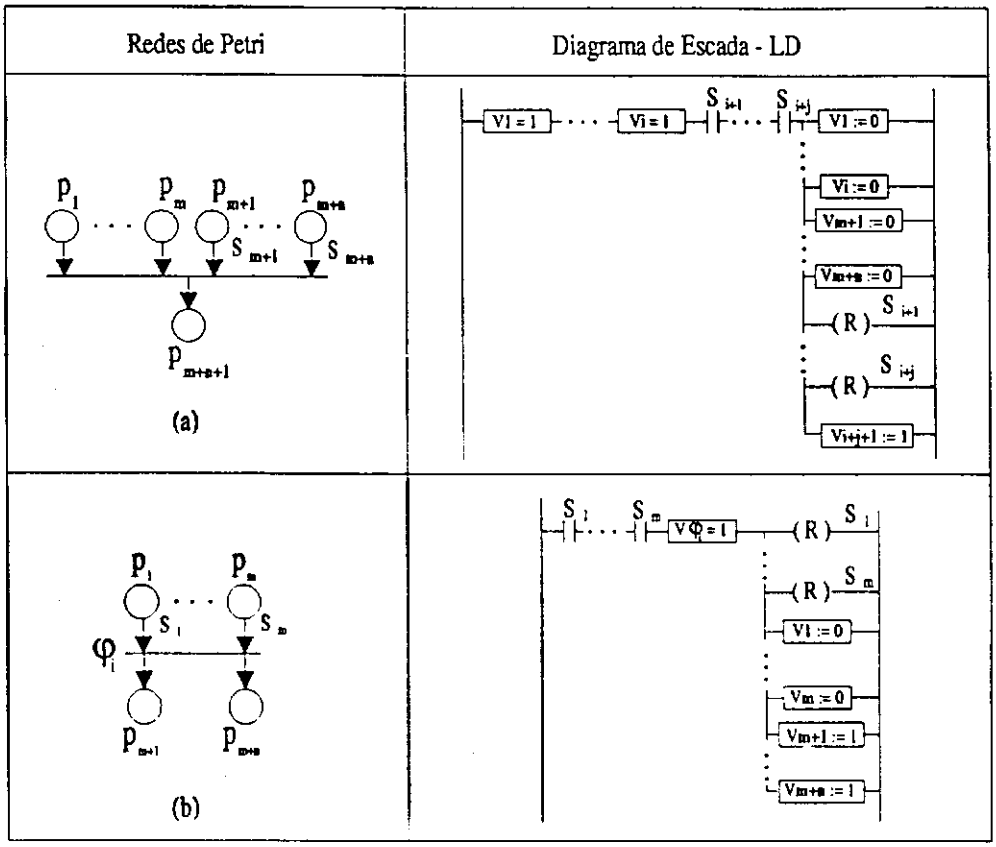


Figura 5.10: Exemplo de representação de rede de Petri *TPN* em *LD*.

- Os lugares de saída da transição, assim como os lugares de entrada, são associados em lógica *OR*. Eles são representados pelas variáveis internas e por saídas (quando forem associados às saídas do *CLP*), conforme Figura 5.10(a) e (b) e Figura 5.11(a) e (b). Às variáveis que representam os lugares de entrada é atribuído o valor zero, e às variáveis que representam os lugares de saída o valor um.
- Ao final de cada lógica, fazer uma chamada pela subrotina de atualização das funções de habilitação de transições.

3. A parte de atualização das funções de habilitação é uma lógica que reconhece o estado atual pela composição das variáveis internas que representam os lugares. Uma vez reconhecido o estado, as funções de habilitação das transições são desabilitadas segundo a *lista perigo da especificação.*

Fim.

A seguir será apresentado um exemplo de síntese de supervisores para uma célula de manufatura, usando esta abordagem.

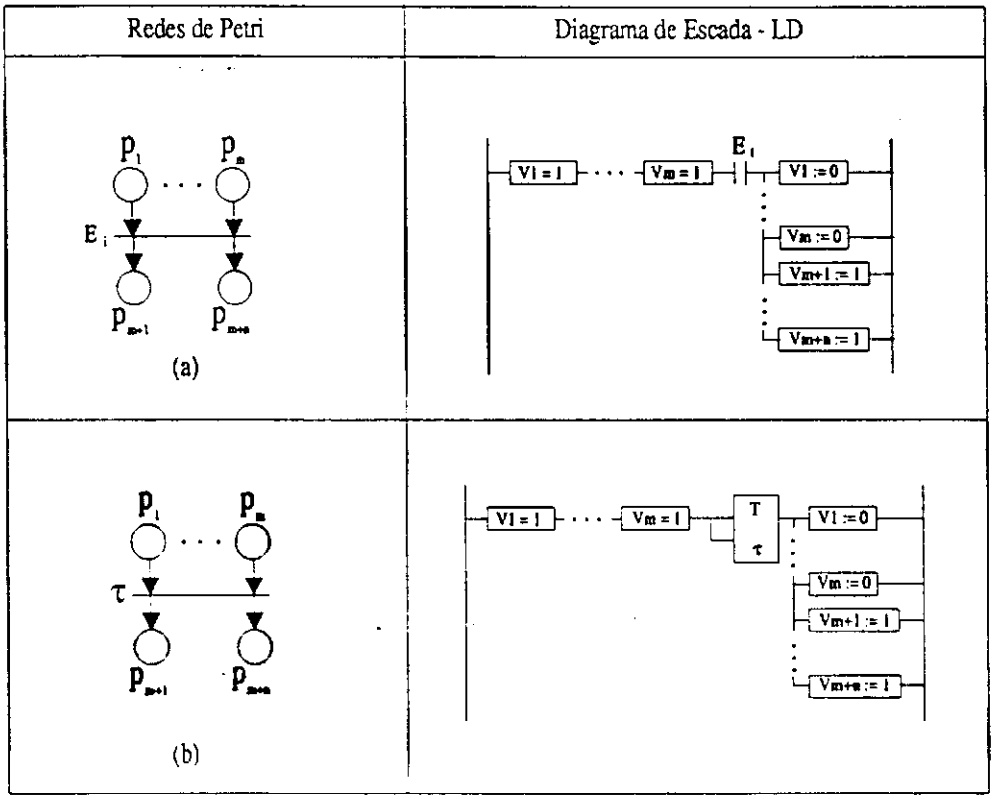


Figura 5.11: Exemplo de representação derede de Petri *TPN* em *LD*.

5.1.3 Supervisor para uma célula de manufatura

Tomemos a célula de manufatura da Figura 5.12 como exemplo. A célula consiste de duas máquinas diferentes (m_1 e m_2), dois robôs (r_1 e r_2), um depósito (d) e duas esteiras (e_1 e e_2). Todas as peças que chegam por e_1 são carregadas por r_1 para m_1. As peças processadas em m_1, são descarregadas por r_1 para d. As peças de d, são carregadas por r_2 para m_2, onde são processadas. As peças processadas em m_2 são descarregadas para e_2 por r_2. A cada uma das máquinas estão associados dois sensores, não indicados na figura, para indicar que a máquina está carregada e o término do processamento pela mesma (s_2, s_3, s_5 e s_6). Há também dois sensores, um ligado a e_1 (s_1), para indicar que há peças na esteira, e outro (s_4) ao depósito para indicar que há peças neste. Os eventos α_1, β_1, β_2, α_2, β_3, α_3, β_4, β_5, α_4, β_6 e α_5 estão associados às transições t_1, t_2, t_3, t_4, t_5, t_6 t_7, t_8, t_9, t_{10} e t_{11}, nesta ordem.

Na Figura 5.13 temos o modelo em rede de Petri temporal desta célula. Na parte (a) da figura, temos a indicação das transições e lugares enquanto na parte (b) os nomes das transições são substituídos pelos seus respectivos intervalos de tempo. A tabela 5.3 contém a descrição dos lugares enquanto a tabela 5.4 contém a descrição das transições. Para a marcação indicada na figura, o sistema encontra-se inicialmente com as máquinas e rôbos livres e o depósito sem peça. Admite-se que sempre há peças em e_1.

Seguindo o procedimento de síntese do supervisor, primeiramente executamos o $AAAC$

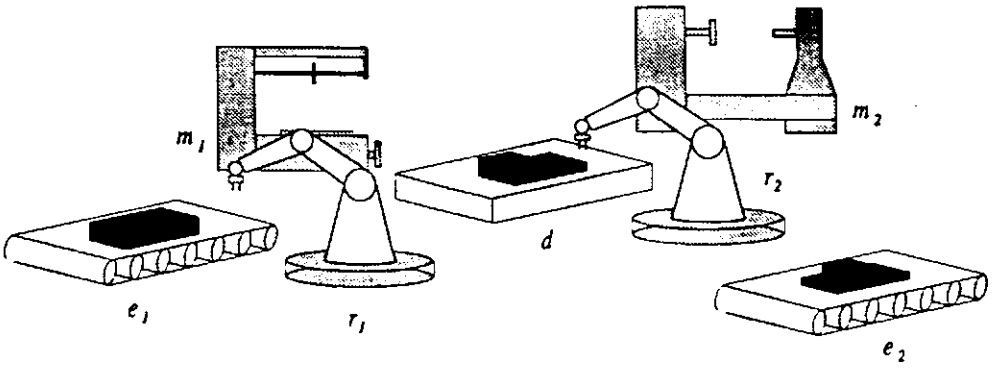


Figura 5.12: Célula de manufatura.

Lugar (com ficha)	Interpretação
p_1	r_1 movendo peça de e_1 para m_1
$p_2(p_7)$	peça sendo processada em $m_1(m_2)$
$p_3(p_8)$	peça pronta em $m_1(m_2)$
p_4	r_1 movendo peça de m_1 para d
p_5	peça no depósito
p_6	r_2 movendo peça de d para m_2
p_9	r_2 movendo peça de m_2 para e_2
$p_{10}(p_{12})$	$m_1(m_2)$ livre
p_{11}	depósito vazio
$p_{13}(p_{15})$	$r_1(r_2)$ livre
p_{14}	nenhum robô acessando o depósito

Tabela 5.3: Interpretação dos lugares

para obtermos a árvore de alcançabilidade de classes: até o segundo passo obtemos uma árvore com 278 classes, e após a execução do terceiro obtemos uma representação com 122 classes.

Agora, com este espaço de estados e uma especificação funcional, podemos executar o *ACGS*. Uma especificação funcional desejável para esta célula pode ser dada pela seqüência $\alpha_1\beta_1\beta_2\alpha_2\beta_3(\alpha_5\beta_1\beta_4\beta_2\beta_5\alpha_2\alpha_1\alpha_8\alpha_2\alpha_6\beta_6)^*$ (disparo da seqüência de transições t_1 t_2 t_3 t_4 t_5 t_{11} t_2 t_7 t_3 t_8 t_4 t_5 t_6 t_{10} t_{11} t_2 t_7 t_3 t_8 t_4...), significando que após o processamento da primeira peça em m_1 e o seu depósito em d, os robôs começarão a carregar as máquinas ao mesmo tempo e irão descarregá-las segundo a ordem do término do processamento, que neste caso é primeiro m_1 seguido de m_2. Assume-se que o alfabeto de eventos da célula é $\Sigma = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\}$ no qual $\Sigma_c = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$ e $\Sigma_u = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\}$. Com o espaço de estado de classes e a especificação funcional,

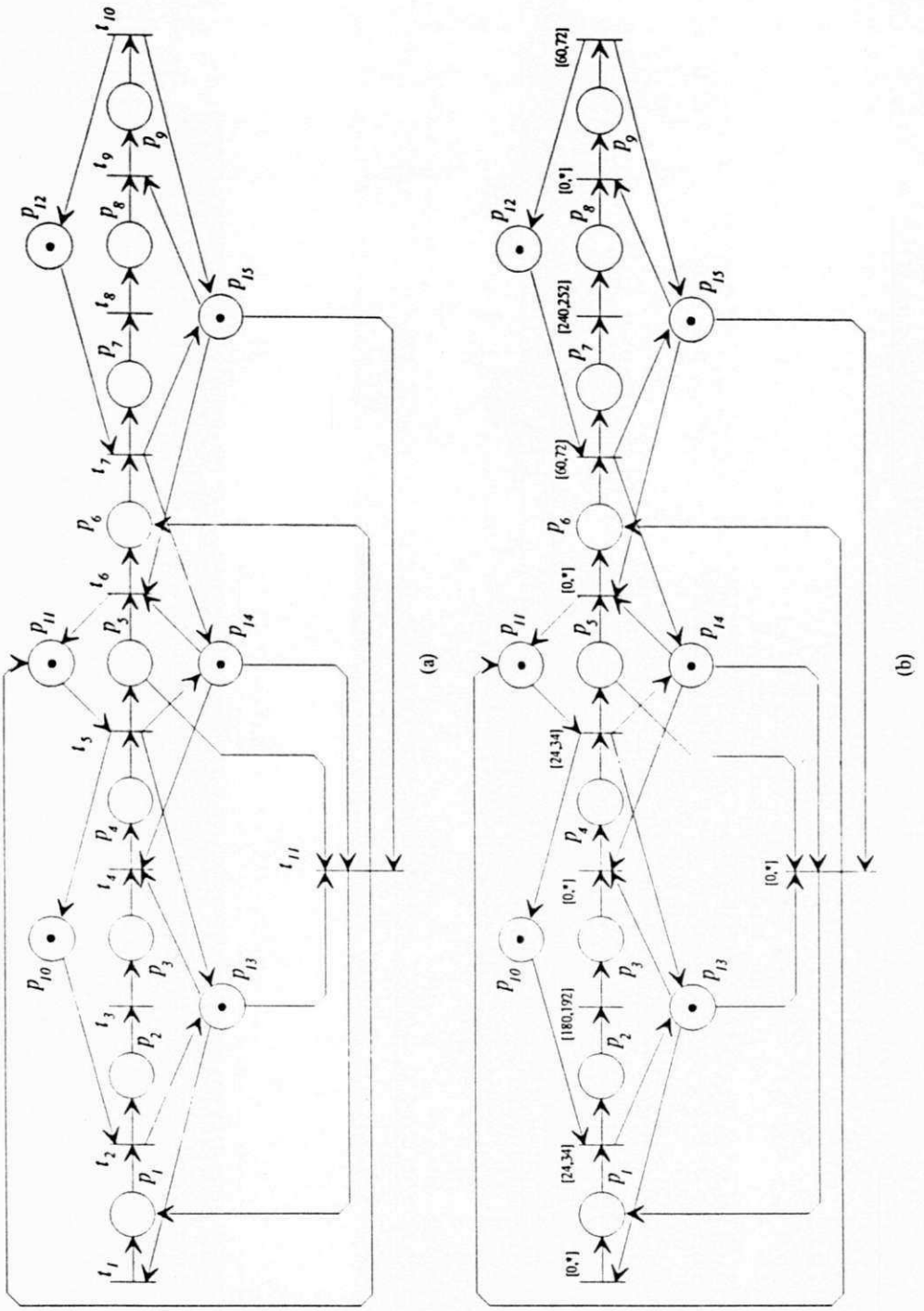


Figura 5.13: Rede *TPN* da célula de manufatura. (a) sem indicação dos intervalos de tempo, (b) com indicação dos intervalos de tempo.

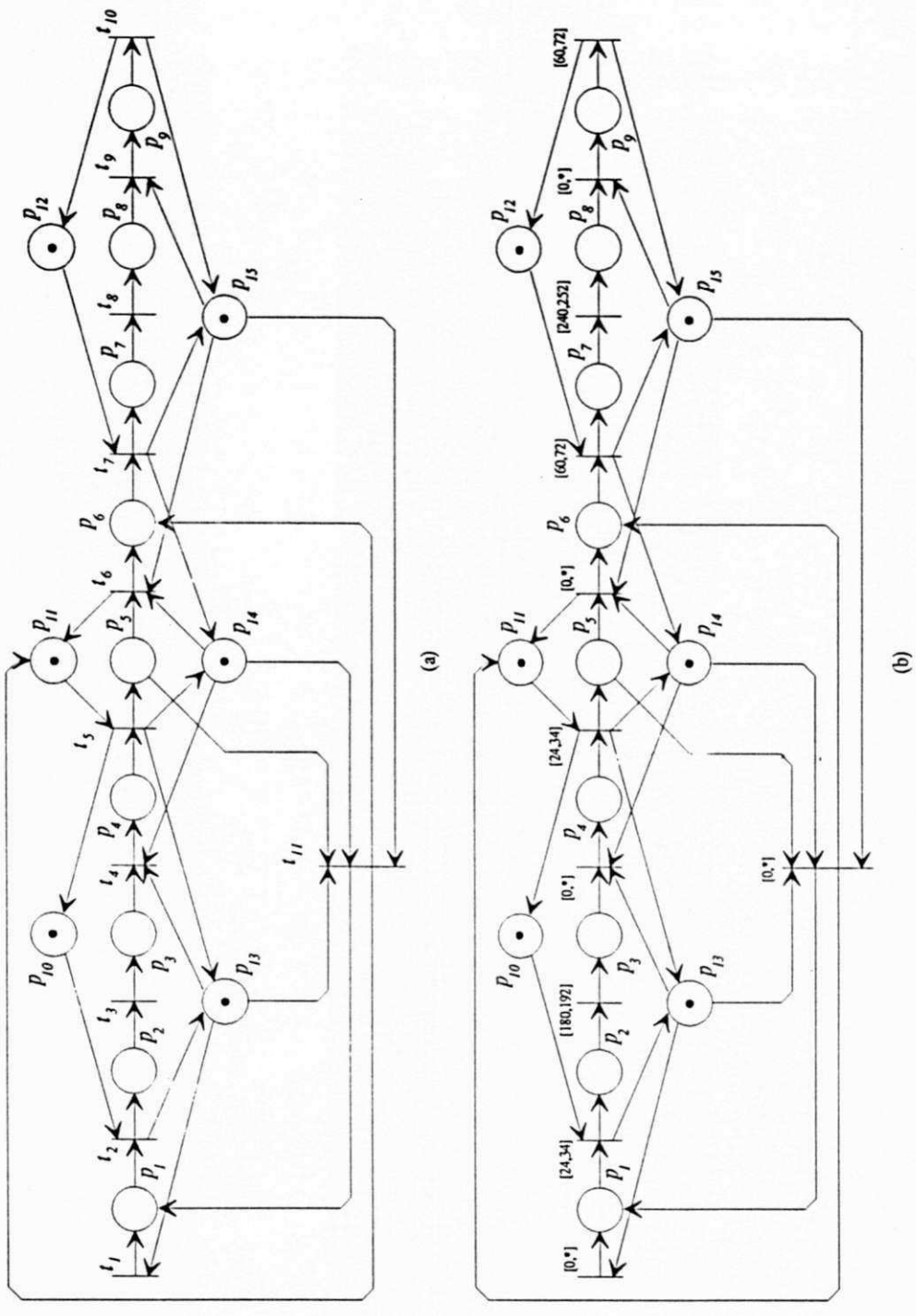


Figura 5.13: Rede *TPN* da célula de manufatura. (a) sem indicação dos intervalos de tempo, (b) com indicação dos intervalos de tempo.

Transição	Interpretação
t_1	início do transporte de uma peça de e_1 para m_1
t_2	fim do transporte e início do processamento por m_1
t_3	fim do processamento por m_1
t_4	início do transporte de uma peça de m_1 para d
t_5	fim do transporte de uma peça de m_1 para d
t_6	início do transporte de uma peça de d para m_2
t_7	fim do transporte e início do processamento por m_2
t_8	fim do processamento por m_2
t_9	início do transporte de uma peça de m_2 para e_2
t_{10}	fim do transporte de uma peça de m_2 para e_2
t_{11}	início do transporte de peças para as máq. simultaneamente

Tabela 5.4: Interpretação das Transições

executa-se o $ACGS$ e obtém-se a seguinte *lista_perigo*:

$$
\begin{aligned}
C_2 &\rightarrow t_1/\alpha_1;\\
C_4 &\rightarrow t_1/\alpha_1;\\
C_7 &\rightarrow t_1/\alpha_1; C_7 \rightarrow t_6/\alpha_3;\\
C_{15} &\rightarrow t_1/\alpha_1;\\
C_{25} &\rightarrow t_1/\alpha_1;\\
C_{38} &\rightarrow t_1/\alpha_1; C_{38} \rightarrow t_4/\alpha_2;\\
C_{49} &\rightarrow t_1/\alpha_1; C_{38} \rightarrow t_9/\alpha_4;\\
C_{59} &\rightarrow t_9/\alpha_4;\\
C_{69} &\rightarrow t_1/\alpha_1; C_{69} \rightarrow t_6/\alpha_3; C_{69} \rightarrow t_{11}/\alpha_5;\\
C_{83} &\rightarrow t_1/\alpha_1;
\end{aligned}
$$

De forma similar a notação usada para estados, $C_i \rightarrow t_j/e_k$, significa que o supervisor deve evitar o disparo da transição t_j (ocorrência do evento e_k) na classe C_i. Assim, para que a especificação seja executada, basta que α_1 não ocorra nas classes C_2, C_4, C_7, C_{15}, C_{25}, C_{38}, C_{49}, C_{69} e C_{83}, α_2 não ocorra na classe C_{38}, α_3 não ocorra nas classes C_7 e C_{69}, α_4 não ocorra nas classes C_{49} e C_{59}, e α_5 na classe C_{69}, o que é garantido pelas funções de habilitação associadas a t_1, t_4, t_6, t_9 e t_{11}.

Cabe observar que se esta especificação fosse feita a partir do espaço de estados construído a partir do modelo não temporizado do sistema, ela necessitaria ser melhor elaborada para ser realizada. A Figura 5.14, mostra duas especificações descritas como autômatos: na

Transições	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
Entrada $Y(t_i)$	1	2	3	–	4	–	5	6	–	–	–
Tipo	s	s	s	n	s	n	s	s	n	n	n

Tabela 5.5: Transições versus mapeamento de entrada do *CLP* e o tipo de transição

Lugares	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
Saída $FX(p_i)$	1	–	–	2	–	3	–	–	4	–	–	–	–	–	–
Tipo	s	n	n	s	n	s	n	n	s	n	n	n	n	n	n

Tabela 5.6: Lugares versus mapeamento de saída do *CLP* e o tipo de lugar

parte (a), temos a especificação feita para o sistema modelado sem se considerar o tempo e na parte (b) para o sistema modelado considerando-se o tempo. Observamos que pela inclusão do tempo alguns caminhos, antes possíveis, agora deixam de existir. Por exemplo, no estado M_{10} é possível a ocorrência do eventos β_1 e β_4 o que já não ocorre na classe C_{10}, pois não é possível que duas máquinas que possuem tempo de processamento diferentes e comecem o trabalho ao mesmo tempo terminem ao mesmo tempo ou a mais lenta termine primeiro que a mais rápida.

Com a rede supervisora, pode-se implementar um supervisor, para a célula da Figura 5.12, em forma de um programa para *CLP* utilizando-se o *AEPCLP*. Para tanto é necessário que se tenha os mapeamentos de entrada, $Y(t_i)$, e saída, $FX(p_i)$, do *CLP* para as transições e lugares da rede que supervisiona o sistema. Os mapeamentos, para este exemplo, são dados nas tabelas 5.5 e 5.6.

Seguindo o algoritmo *AEPCLP*, vemos que para este exemplo precisamos criar quinze variáveis internas ao *CLP* para representar cada um dos lugares da rede e atribuir a elas o valor da marcação inicial, isto pode ser visto na Figura 5.15(a) na qual as variáveis de %MW1 a %MW15 representam cada um dos lugares da rede e os valores a elas atribuidos a marcação inicial. Na Figura 5.15(b), temos três lógicas que representam as lógicas para as transições t_1, t_2 e t_3, referentes aos passos dois e três do algoritmo. Os operandos %I0.i e %Q0.j representam as entradas 0.i e saídas 0.j. A lógica de reconhecimento do estado e desabilitação das funções de habilitação das transições, segundo a *lista perigo da especificação*, consiste de uma subrotina demarcada pelos rótulos SR0: e RET, é mostrada na Figura 5.15(c), representando o passo quatro do algoritmo.

Os trechos de programas apresentados na Figura 5.15, foram feitos em lista de instruções usando a sintaxe do *PLC07* da Telemecanique, pois o programa que realiza o procedimento de síntese e implementação do supervisor gera um programa para o *PLC07*. Mais detalhes sobre o conjunto de instruções do *PLC07* utilizado para a elaboração dos programas podem ser vistos no Apendice B.

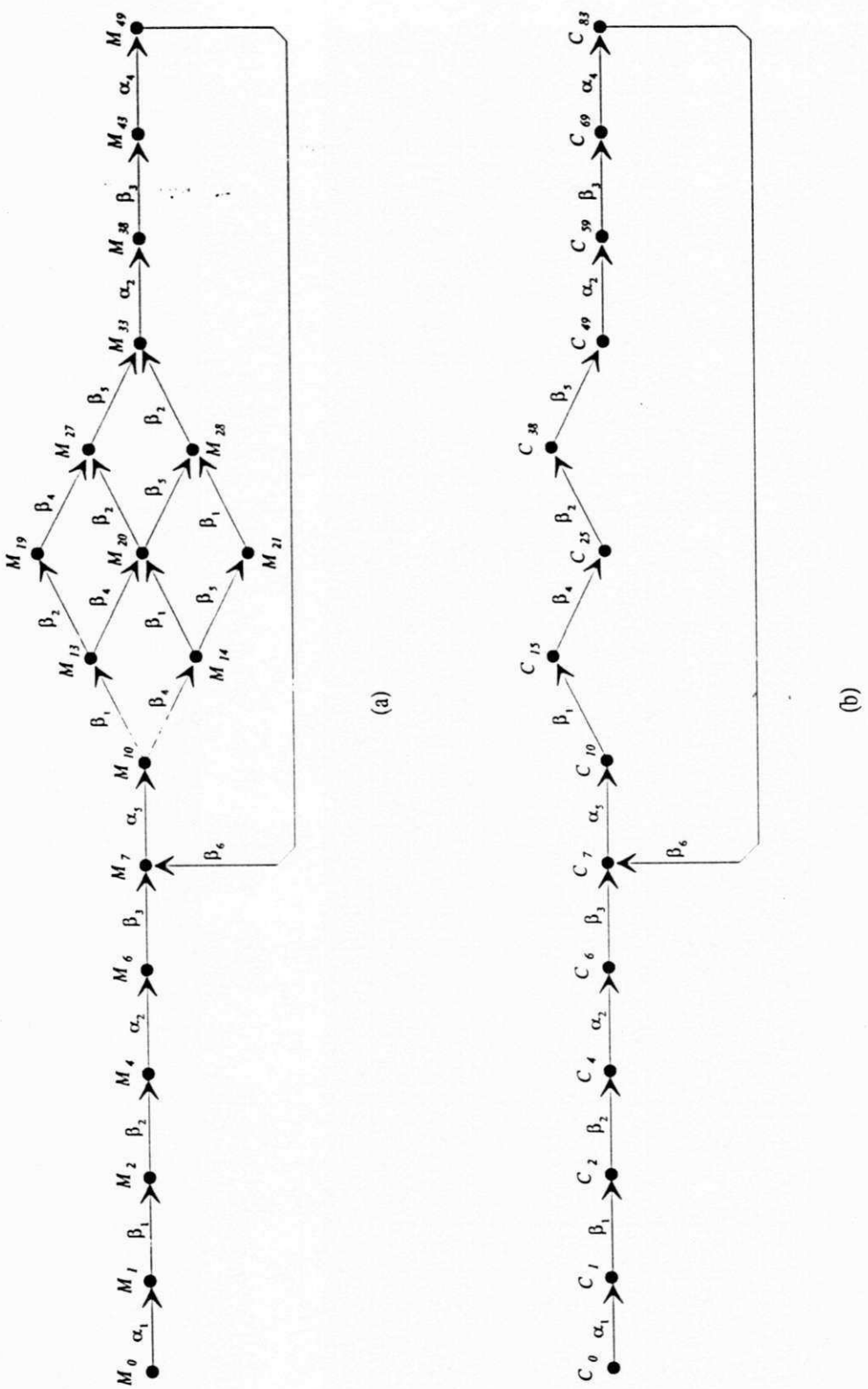


Figura 5.14: Especificações. (a) para uma análise sem tempo; (b) para uma análise com tempo.

```
LD  [ %MW0 = 1 ]
JMPC %L0
LD  1
[ %MW1 := 0 ]
[ %MW2 := 0 ]
[ %MW3 := 0 ]
        :
[ %MW13 := 1 ]
[ %MW14 := 1 ]
[ %MW15 := 1 ]
[ %MW0 := 1 ]
SR0
```

(a)

```
%L0:
LD  [ %MW13 = 1 ]
AND  %I0.1
AND  [ %MW16 = 1 ]
S  %Q0.1
[ %MW1 := 1 ]
[ %MW13 := 0 ]
SR0
LD  %Q0.1
AND  [ %MW10 = 1 ]
AND  %I0.2
[ %MW2 := 1 ]
[ %MW13 := 1 ]
R  %Q0.1
[ %MW1 := 0 ]
[ %MW10 := 0 ]
SR0
LD  [ %MW2 = 1 ]
AND  %I0.3
[ %MW3 := 1 ]
[ %MW2 := 0 ]
SR0
```

(b)

```
SR0:
LD  1
[ %MW16 := 1 ]
[ %MW17 := 1 ]
[ %MW18 := 1 ]
[ %MW19 := 1 ]
[ %MW20 := 1 ]
[ %MW21 := 0 ]
LD  [ %MW1 = 1 ]
[ %MW21 := %MW21 + 1 ]
LD  [ %MW2 = 1 ]
[ %MW21 := %MW21 + 2 ]
LD  [ %MW3 = 1 ]
[ %MW21 := %MW21 + 4 ]
        :
LD  [ %MW21 = 31746 ]
[ %MW16 := 0 ]
LD  [ %MW21 = 31248 ]
[ %MW16 := 0 ]
[ %MW18 := 0 ]
LD  [ %MW21 = 7202 ]
[ %MW16 := 0 ]
LD  [ %MW21 = 29762 ]
[ %MW16 := 0 ]
LD  [ %MW21 = 29764 ]
[ %MW17 := 0 ]
[ %MW16 := 0 ]
LD  [ %MW21 = 29828 ]
[ %MW19 := 0 ]
[ %MW16 := 0 ]
LD  [ %MW21 = 17544 ]
[ %MW19 := 0 ]
LD  [ %MW21 = 29328 ]
[ %MW20 := 0 ]
[ %MW18 := 0 ]
[ %MW16 := 0 ]
LD  [ %MW21 = 13072 ]
[ %MW16 := 0 ]
LD  [ %MW21 = 31748 ]
[ %MW16 := 0 ]
RET
```

(c)

Figura 5.15: Implementação do supervisor no *CLP* (Lista de Instruções).

5.2 O programa de síntese e implementação do supervisor

O procedimento de síntese e implementação do supervisor, descrito pelo diagrama de blocos da Figura 5.1, foi implementado em linguagem C. Ele é um programa interativo e funciona de acordo com o fluxograma da Figura 5.16.

Quando o programa começa, ele solicita o nome do arquivo de dados, arquivo com extensão p (*arquivo.p*), o qual contém as seguintes informações:

1. a quantidade de lugares e transições da rede que modela o sistema;
2. as matrizes de incidência de entrada, I, e de saída, O;
3. para cada transição as informações de
   (a) nome,
   (b) tipo,
      i. c - controlável e
      ii. u - não controlável
   (c) tipo
      i. s - sensor, indicando que há um canal de entrada do *CLP* associado à transição,
      ii. t - temporizador, indicando que há um temporizador do *CLP* associado à transição e
      iii. n - normal, não há qualquer associação à transição,
   (d) canal de entrada do *CLP*,
   (e) intervalos associados a cada transição;
4. para cada lugar as informações de
   (a) nome,
   (b) tipo
      i. s - saída, indicando que há um canal de saída do *CLP* associado ao lugar, e
      ii. n - normal, não há qualquer associação ao lugar,
   (c) canal de saída do *CLP*.

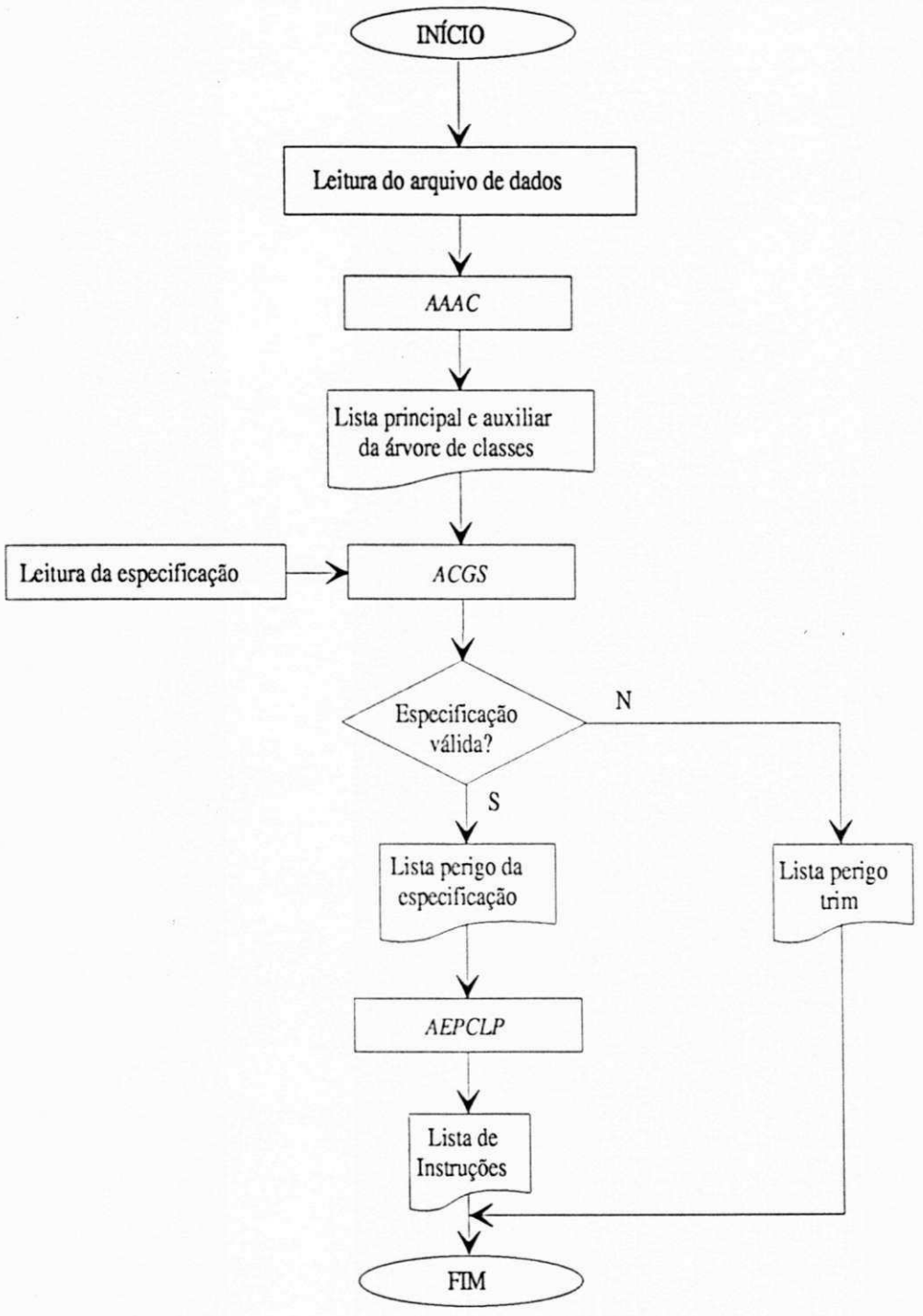


Figura 5.16: Fluxograma do programa que implementa a síntese do supervisor.

Durante a leitura de um arquivo de dados, os dados das transições e lugares são armazenados em estruturas por duas rotinas: *inicializacao_das_transicoes* e *inicializacao_dos_lugares*. Tomemos como exemplo a rede da Figura 5.3. O arquivo de dados para esta rede, EXF-ARQ.P, é mostrado na Figura 5.17. Neste arquivo estão todos os dados necessários para a execusão do $AAAC$ e os mapeamentos de entrada e saídas do CLP para as transições e lugares da rede supervisora.

```
12 9                                  } Quantidade de lugares e transições

100000001000 |                  | 010000000011 |
010000001000 |                  | 001000000011 |
010000000100 |                  | 100100000000 |
001000000100 |                  | 010100000000 |
000100000000 | } Matriz I       | 000010000000 | } Matriz O
000010000001 |                  | 000001000100 |
000001000010 |                  | 000000100001 |
000000100000 |                  | 000000010000 |
000000010001 |                  | 000000001000 |

24 30  24 30  0 0  0 0  34 40  } Intervalo de tempo
0 *  34 40  180 192  0 0       } de cada transição

1  u  s  1 |                  | mw1   0  n  0 |
2  u  s  1 |                  | mw2   0  n  0 |
3  u  n  0 |                  | mw3   1  n  0 |
4  u  n  0 |                  | mw4   0  n  0 |
5  u  s  2 |                  | mw5   0  n  0 |
6  c  n  0 | } Informações a   | mw6   0  s  4 | } Informações a
7  u  n  0 |   respeito das    | mw7   0  n  0 |   respeito dos
8  u  s  3 |   transições      | mw8   0  n  0 |   lugares
9  u  n  0 |                  | mw9   0  s  5 |
           |                  | mw10  1  n  0 |
           |                  | mw11  1  n  0 |
           |                  | mw12  1  n  0 |
```

Figura 5.17: Exemplo do arquivo de dados para a rede da Fig. 5.3.

Após a leitura do arquivo de dados, é executada a rotina que implementa o $AAAC$, *criar_arvore*. Esta rotina por sua vez faz chamada à subrotinas que computam cada uma das classes (ou estados, quando todos os intervalos de tempo forem [0,*]). Ao final da execução da rotina obtém-se como resultado duas listas encadeadas: a primeira com todas as classes que aparecem pela primeira vez durante o cálculo das classes e a segunda com todas as classes *idênticas*, independente do seu tipo. A composição destas listas forma a

árvore de alcançabilidade de classes que também é armazenada em um arquivo de extensão *p*. Tomemos como exemplo o arquivo EXF-ARV.P, mostrado na Figura 5.18, que é a árvore de classes para a rede da Figura 5.3. Esta árvore é obtida pela execução completa do algoritmo *AAAC*.

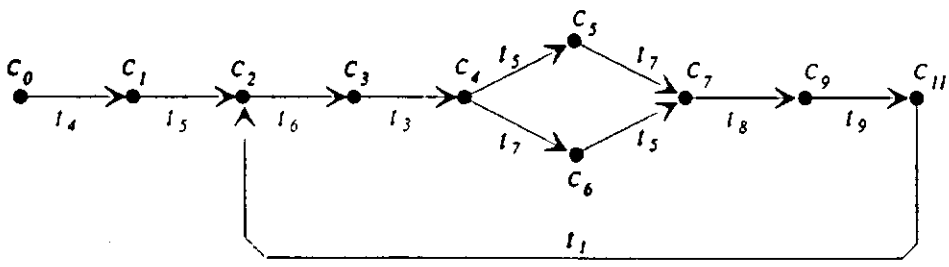
```
Pai C0 :0 /0 /1 /0 /0 /0 /0 /0 /0 /0 /1 /1 /1 / n    [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0]
  C1 :0 /1 /0 /1 /0 /0 /0 /0 /0 /0 /1 /1 / n T4      [0,0] [0,0] [0,0] [0,0] [34,40] [0,0] [0,0] [0,0] [0,0]
Pai C1 :0 /1 /0 /1 /0 /0 /0 /0 /0 /0 /1 /1 / n       [0,0] [0,0] [0,0] [0,0] [34,40] [0,0] [0,0] [0,0] [0,0]
  C2 :0 /1 /0 /0 /1 /0 /0 /0 /0 /0 /1 /1 / n T5      [0,0] [0,0] [0,0] [0,0] [0,0] [0,*] [0,0] [0,0] [0,0]
Pai C2 :0 /1 /0 /0 /1 /0 /0 /0 /0 /0 /1 /1 / n       [0,0] [0,0] [0,0] [0,0] [0,0] [0,*] [0,0] [0,0] [0,0]
  C3 :0 /1 /0 /0 /0 /1 /0 /0 /0 /1 /1 /0 / n T6      [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [34,40] [0,0] [0,0]
Pai C3 :0 /1 /0 /0 /0 /1 /0 /0 /0 /1 /1 /0 / n       [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [34,40] [0,0] [0,0]
  C4 :1 /0 /0 /1 /0 /1 /0 /0 /0 /0 /1 /0 / n T3      [0,0] [0,0] [0,0] [0,0] [34,40] [0,0] [34,40] [0,0] [0,0]
Pai C4 :1 /0 /0 /1 /0 /1 /0 /0 /0 /0 /1 /0 / n       [0,0] [0,0] [0,0] [0,0] [34,40] [0,0] [34,40] [0,0] [0,0]
  C5 :1 /0 /0 /0 /1 /1 /0 /0 /0 /0 /1 /0 / n T5      [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,6] [0,0] [0,0]
  C6 :1 /0 /0 /1 /0 /0 /1 /0 /0 /0 /0 /1 / n T7      [0,0] [0,0] [0,0] [0,0] [0,6] [0,0] [0,0] [180,192] [0,0]
Pai C5 :1 /0 /0 /0 /1 /1 /0 /0 /0 /0 /1 /0 / n       [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,6] [0,0] [0,0]
  C7 :1 /0 /0 /0 /1 /0 /1 /0 /0 /0 /0 /1 / n T7      [0,0] [0,0] [0,0] [0,0] [0,0] [0,*] [0,0] [180,192] [0,0]
Pai C6 :1 /0 /0 /1 /0 /0 /1 /0 /0 /0 /0 /1 / n       [0,0] [0,0] [0,0] [0,0] [0,6] [0,0] [0,0] [180,192] [0,0]
  C7 :1 /0 /0 /0 /1 /0 /1 /0 /0 /0 /0 /1 / a T5      [0,0] [0,0] [0,0] [0,0] [0,0] [0,*] [0,0] [180,192] [0,0]
Pai C7 :1 /0 /0 /0 /1 /0 /1 /0 /0 /0 /0 /1 / n       [0,0] [0,0] [0,0] [0,0] [0,0] [0,*] [0,0] [180,192] [0,0]
  C8 :1 /0 /0 /0 /0 /1 /1 /0 /0 /1 /0 /0 / n T6      [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,192] [0,0]
  C9 :1 /0 /0 /0 /1 /0 /0 /1 /0 /0 /0 /1 / n T8      [0,0] [0,0] [0,0] [0,0] [0,0] [0,*] [0,0] [0,0] [0,0]
  C8 :1 /0 /0 /0 /0 /1 /1 /0 /0 /1 /0 /0 / a T6      [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,192] [0,0]
Pai C8 :1 /0 /0 /0 /0 /1 /1 /0 /0 /1 /0 /0 / n       [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,192] [0,0]
  C10 :1 /0 /0 /0 /0 /1 /0 /1 /0 /1 /0 /0 / b T8     [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0]
Pai C9 :1 /0 /0 /0 /1 /0 /0 /1 /0 /0 /0 /1 / n       [0,0] [0,0] [0,0] [0,0] [0,0] [0,*] [0,0] [0,0] [0,0]
  C11 :1 /0 /0 /0 /1 /0 /0 /0 /1 /0 /0 /0 / n T9     [24,30] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0]
  C10 :1 /0 /0 /0 /0 /1 /0 /1 /0 /1 /0 /0 / b T6     [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0]
Pai C11 :1 /0 /0 /0 /1 /0 /0 /0 /1 /0 /0 /0 / n      [24,30] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0] [0,0]
  C2 :0 /1 /0 /0 /1 /0 /0 /0 /0 /0 /1 /1 / a T1      [0,0] [0,0] [0,0] [0,0] [0,0] [0,*] [0,0] [0,0] [0,0]
```

Figura 5.18: Arquivo da árvore de alcançabilidade de classes da rede da Fig. 5.3.

No arquivo EXF-ARV.P temos as classes denominadas PAI, que constituem os nós da árvore. Nestas classes temos a informação do nome da classe, da marcação, do tipo (a - *antiga*, b - *bloqueada*, p - *não-permitida*) e dos intervalos de disparo das transições. Os intervalos [0,0] servem tanto para indicar o intervalo de habilitação da transição quanto para indicar que a transição está desabilitada nesta classe. Há também, as classes que ficam entre as classes denominadas PAI, que são as classes subseqüentes das classes PAI. Estas classes contêm as informações do nome da classe, da marcação, do tipo, da transição que liga a classe PAI a ela e intervalos de disparo das transições.

Ao término da gravação das listas que compõem a árvore, em arquivo, é executada a rotina, *le_especificacao*, responsável pela leitura do arquivo da especificação funcional desejada. Esta especificação é armazenada em uma lista na qual estão as informações do nome das classes e as respectivas transições que podem ocorrer nestas classes. Por exemplo,

a seguinte especificação, mostrada na Figura 5.19, para o sistema representado pela rede da Figura 5.3. Na parte (a) temos a representação da especificação em autômato e na parte (b) em arquivo.p, a coluna da direita representa o nome das classes e a da esquerda o nome das transições. A palavra *end* no arquivo.p indica o fim da especificação. Adota-se que as especificações serão sempre cíclicas, desta forma o estado para o qual a transição é chamada de *end* não é usada pelo *ACGS* na hora de computar as transições a serem desabilitadas.



(a)

```
0  4
1  5
2  6
3  3
4  5
4  7
5  7
6  5
7  8
9  9
11 1
2  end
```

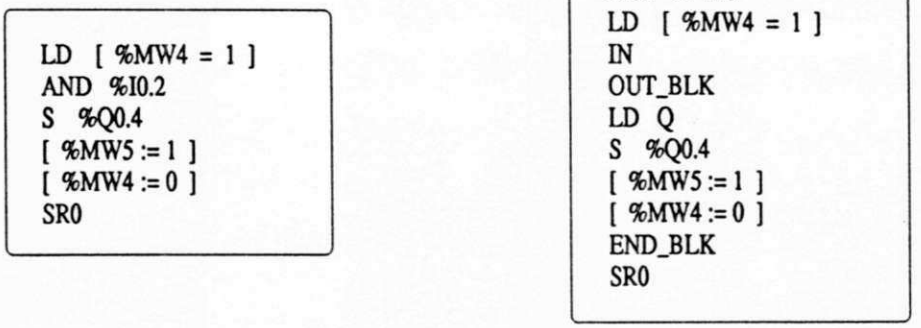
(b)

Figura 5.19: Especificação para o sistema da rede da Fig. 5.3. (a) em autômato e (b) em arquivo.p.

Após a leitura do arquivo da especificação, é executada a rotina *acgs*, que tem como dados de entrada as lista geradas na rotina *criar_arvore* e a gerada na *le_especificacao*. A rotina gera duas listas: a *lista_bloq* e a *lista_perigo*. Se a especificação for válida para o sistema o *ACGS* gera uma *lista_perigo* para que a especificação seja atendida, em caso contrário a *lista_perigo* gerada garantirá que o gerador resultante será *trim*.

Com a *lista_perigo* da especificação pode-se executar a rotina que implementa o *AEPCLP*, *plc_program*. Esta rotina irá elaborar o programa que implementa o supervisor em lista de intruções e gravá-lo em um arquivo de extensão TXT, similar ao da Figura 5.15. Neste trabalho o algoritmo *AEPCLP* foi implementado para gerar o programa na forma de lista de instruções (*Instruction List - IL*) para o *PLC07* da Telemecanique. Caso a especificação não seja válida o *AEPCLP* não será executado.

A denominação de uma transição como temporizador pode ser útil na hora em que há a quebra de um sensor no sistema e não há um disponível para substituição. Imaginemos a seguinte situação: a célula da Figura 5.2 está trabalhando sob o controle do *CLP* (supervisor) e em um determinado tempo o sensor que indica o término da fixação de uma peça a bandeja, associado a t_5 da rede da Figura 5.3, começa apresentar defeito, prejudicando o desempenho da célula. Nesta situação, presupondo-se que os tempos utilizados no modelo são confiáveis, pode-se substituir a informação do sensor pela de um temporizador interno



(a) (b)

Figura 5.20: Mudança de tipo da transição: (a) transição do tipo sensor; (b) transição do tipo temporizador.

ao *CLP*, que começa a sua contagem sob as mesmas condições que se encontrava o sensor. Para tanto, precisariamos mudar o intervalo associado à t_5 para $[40, 40]$, trocar o atributo de t_5 para "t" e executar o procedimento para determinação do supervisor. A Figura 5.20, mostra as linhas de código que mudariam no programa do *CLP* para a solução do problema apresentado. Para este *CLP* seria necessário utilizar o *software* de programação do fabricante para atribuir o tempo do temporizador, pois isto não é possível por meio de programação por *IL*.

Capítulo 6

Conclusão e Trabalhos Futuros

Neste trabalho foi apresentada uma abordagem temporal para a síntese e implementação de supervisores para sistemas a eventos discretos, utilizando controle supervisório e redes de Petri temporais (*TPNs*). A abordagem é dividida em duas etapas: a de síntese e a de implementação do supervisor.

A etapa de síntese foi fundamentada na abordagem de síntese de supervisores de Barroso. Nesta etapa foram introduzidos o *Algoritmo da Árvore de Alcançabilidade de Classes - AAAC*, utilizado para a obtenção do espaço de estados considerando o tempo associado às transições da rede que modela o sistema (para o qual se deseja obter um supervisor), e uma nova classe de rede de Petri para modelar o supervisor, *Rede de Petri Temporal Interpretada com Funções de Habilitação de Transição - RPTIFHT*.

Na etapa de implementação foi introduzido o *Algoritmo para a Elaboração do Programa para o CLP - AEPCLP*, que especifica como converter a rede supervisora obtida na etapa de síntese em um programa em lista de instruções para um controlador lógico programável, *CLP*.

Utilizou-se uma rede de Petri temporal para modelar o sistema pois desejava-se uma representação que permitisse a caracterização das propriedades temporais do sistema, o que não é possível fazer com as redes de Petri clássicas. Ela foi escolhida dentre algumas extensões, apresentadas no capítulo 4, pois a associação de tempo a uma transição pareceu ser mais natural do que a um lugar, e porque um intervalo de tempo também representa uma tolerância no tempo de execução das tarefas, que variam de acordo com o uso das ferramentas, por exemplo. Além disso, redes que associam tempo a lugares podem ser convertidas para redes que associam tempo a transições. Constatou-se que a inclusão de tempo no modelo da planta pode tanto alterar a ordem relativa de ocorrência dos eventos quanto inibir a ocorrência em alguns estados, conforme havia dito Freedman.

Devido a inclusão de tempo no modelo do sistema foi necessário a elaboração de um algoritmo para a enumeração dos estados do sistema, que considerasse esta característica.

Este algoritmo foi obtido utilizando-se os conceitos, do método enumerativo para análise de redes de Petri temporais de Berthomieu e Diaz, com os do algoritmo modificado da árvore de alcançabilidade de Barroso, resultando no algoritmo *AAAC*. Este algoritmo é capaz de enumerar tanto o espaço de estados de um modelo em *TPN* quanto de um modelo em rede de Petri.

Embora o espaço de estados tenha sido construído considerando-se o tempo associado as transições do modelo, o *Algoritmo para a Construção do Gerador da Suprema Linguagem Controlável - ACGS* - pôde ser utilizado para a síntese sem alterações nesta abordagem temporal, pois o espaço de estados gerado pelo *AAAC* apresenta-se ao *ACGS* como um gerador finito que modela a operação do sistema sob análise.

A rede *RPTIFHT* foi introduzida pois precisava-se de uma rede para modelar o supervisor que possuísse as seguintes características: tivesse a mesma estrutura da rede que modela o sistema; evoluísse sincronamente com os eventos do sistema; associasse ações a lugares; temporizasse ações e realizasse sua ação de controle em tempo real por meio da restrição da ocorrência do disparo de suas transições. Assim, para se obter tal rede, combinou-se algumas características das extensões de redes apresentadas no capítulo 4.

A implementação do supervisor poderia ter sido feita tanto em um *PC* quanto em um *CLP*. No entanto, optou-se pela implementação em *CLP*, pois este é um equipamento bastante utilizado na automação de sistemas de manufatura. Assim o algoritmo *AEPCLP* especifica como deve ser elaborado o programa em lista de instruções para o *CLP* que desempenhará as tarefas do supervisor.

Esta abordagem foi implementada em linguagem de programação C. O programa realiza as etapas de síntese e implementação do supervisor tendo como dados de entrada dois arquivos nos quais temos as informações da rede que modela o sistema (rede *TPN* segura), dos mapeamentos dos canais de entrada e saída do *CLP* para a rede supervisora e da especificação funcional desejada. O programa fornece como saída um arquivo que contém o espaço de estados do sistema, um arquivo que contém a *lista perigo da especificação* e um que contém o programa, em *lista de instruções*, para o *CLP*.

As principais contribuições do presente trabalho foram a extensão da abordagem de síntese de supervisores de Barroso, possibilitando o uso de modelos temporais, a elaboração do algoritmo para a enumeração do espaço de estados de modelos temporais, a elaboração do algoritmo que especifica como formar programas para *CLP*, e a construção de um programa que realiza tanto a etapa de síntese quanto a etapa de implementação do supervisor em um *CLP*.

Em face dos resultados alcançados, consideramos atingidos os objetivos inicialmente propostos.

6.1 Trabalhos Futuros

O trabalho tratou de uma abordagem de síntese e implementação de supervisores para sistemas a eventos discretos a partir de um modelo em rede de Petri temporal. Para torná-lo mais abrangente poder-se-ia associá-lo a ferramentas, já existentes para redes de Petri, que possibilitassem análises de desempenho e simulações, ALPHA/Sim, Artifex, CodeSign, Design/CPN e ELSIR por exemplo, obtendo-se uma boa ferramenta de síntese de supervisores. Mais detalhes sobre estas ferramentas podem ser obtidos no endereço eletrônico http://www.daimi.aau.dk/PetriNets/tools/quick.html.

A técnica utilizada para a enumeração do espaço de estados também pode ser alvo de melhorias. Técnicas poderiam ser desenvolvidas para possibilitar uma melhor construção do espaço de estado do sistema, evitando a explosão da representação.

Considerou-se neste trabalho que a especificação funcional desejada seria um dado, uma seqüência de classes e transições, e que ela existiria no espaço de estado do sistema. Poderia-se automatizar o programa proposto pelo acréscimo de uma ferramenta, que a partir do espaço de estados e da descrição do comportamento desejado (por exemplo, por fórmulas em lógica temporal [9]), validasse a descrição e a transformasse numa seqüência de classes e transições.

O algoritmo sugerido para a elaboração do programa de implementação do supervisor, possui uma estrutura de programação que implementa um jogador de rede de Petri. Trabalhos poderiam ser realizados para propor outras formas de implementação do programa.

Trabalhos podem ser realizados para aumentar a quantidade de linguagens de programação utilizadas para implementar o supervisor no *CLP*, como por diagramas de relés e o *GRAFCET*, e linguagens que permitam a implementação em *PCs*, como a linguagem de programação C.

Apêndice A

A Norma Internacional IEC 1131-3

Dois dos grandes problemas enfrentados pelos programadores de *CLP* são a quantidade de linguagens e sintaxe de programação existentes para os *CLP* encontrados no mercado. Isto ocorre devido os fabricantes de *CLP* não seguirem uma normalização de programação e cada um estabelecer sua própria forma de programação.

Com o propósito de estabelecer um padrão do modo pelo qual os *softwares* de programação pudessem processar seus comandos, manipular suas variáveis e sua própria estrutura de apresentação, foi criado o Comit Eletrotécnico Internacional, IEC (International Electrotechnical Committee). Organizado para promover e criar um modelo formal de padronização mundial, surge a norma internacional IEC 1131. Esta norma é constituida por cinco itens: visão geral; equipamento; linguagens de programação; manuais/guias do usuário e comunicação. Destaque para a tentativa de uniformização das linguagens de programação existentes e disponíveis, por meio de inúmeros fabricantes atuantes no mercado mundial. Foi exatamente em função do estabelecimento da norma, em seu quesito terceiro, que se estabeleceu e se definiu a IEC 1131-3.

Na IEC 1131-3, os padrões são definidos de modo flexível para estabelecer as especificações mínimas a serem respeitadas e as regras para as expansões futuras. A definição das especificações das linguagens são baseadas em estruturas de linguagem tais como Pascal, ADA, etc, de modo a garantir a sua portabilidade para equipamentos de diferentes fabricantes. A IEC 1131 reconhece dois grandes blocos que representam o tipo de linguagem de programação utilizada, cada qual contendo um conjunto de linguagens correntes. São eles:

Linguagens Gráficas

Diagrama de Funções Seqüenciais (*SFC* - Sequential Function Chart) ⇒ *GRAFCET.*

Diagrama em Ladder (*LD* - Ladder Diagram) ⇒ *Diagrama de Escada.*

Diagrama de Blocos de Funções (*FBD* - Function Block Diagram) ⇒ *Diagrama de Blocos.*

Linguagens Textuais

Lista de Instruções (*IL* - Instruction List) ⇒ *mnemônicos booleanos.*

Texto Estruturado (*ST* - Structured Text) ⇒ *parâmetros idiomáticos.*

A.1 Texto Estruturado (*ST*)

Texto estruturado (ST) é uma linguagem de auto nível que tem algumas similaridades com a programação Pascal. A principal característica é a possibilidade de estruturação de programas com processamentos numéricos, operações de comparação, comandos de IF, THEN, ELSE, CASE, REPEAT ...UNTIL, RETURN and EXIT e EXIT.

Há muitos operadores aritméticos disponíveis. A talela A.1 mostra uma lista de operadores aritmeticos. Eles são mostrados em sua ordem de precedência. O operador do topo da tabela é o de maior precedência.

```
CASE Var1 OF
1: Var2:=5;
2: IF Tval=87 THEN
       Var1:=35;
       Var2:=12;
   ELSE
       Var1:=54;
       Var2:=17;
   END_IF;
3: REPEAT
      Count:=Count +1;
      Pack:=5;
      Pump1:=OFF;
   UNTIL Count =5
   END_REPEAT
4...7:  Var2:=5; Alarm:=ON;
END_CASE;
```

Figura A.1: Programa exemplo de *ST*.

A.2 Diagramas de Blocos de Função (*FBD*)

É uma forma de representar combinações de blocos AND, OR, etc. A programação FDB é usada quando a aplicação envolve o fluxo de sinais entre os blocos de controle. O bloco de função sempre tem as entradas mostradas no lado esquerdo e saídas do lado direito. A Figura A.2 utiliza funções booleanas. Os blocos correspondem a funções padrões tais como booleanas, comparação, string, seleção, temporização, matemáticas e funções numéricas

Operador	Descrição
(....)	Parênteses usado como grupo para precedência
Function(....)	Usado para listar e avaliar parâmetros da função
**	Exponenciação
-	Negação
NOT	Complemento Booleano
*	Multiplicação
/	Divisão
MOD	Módulo
+	Adição
-	Subtração
<,>,<=,>=	Comparação
=	Igualdade
<>	Desigualdade
AND,&	AND Booleano
XOR	OR exclusivo Booleano
OR	OR Booleano

Tabela A.1: Operadores aritméticos para uso em texto estruturado

bem como blocos de funções criadas pelo usuário em *FBD*. Os blocos também podem ser combinados para permitir a representação de funções de alto nível. Quando somente funções lógicas são incluídas, é também chamado de diagrama de circuito lógico.

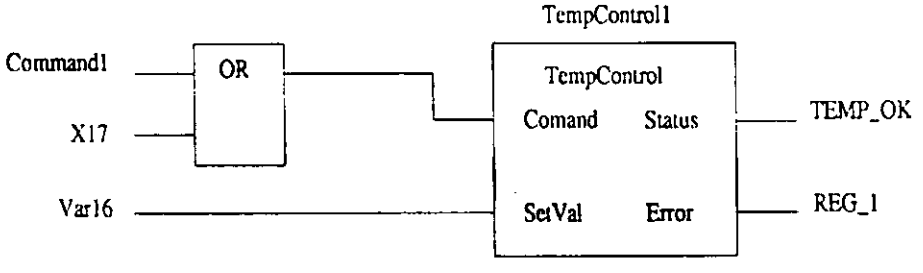


Figura A.2: Programa exemplo de *FBD*.

A tabela A.2 mostra uma lista de operadores que são reservados para o uso com blocos de função padrão. Os estados padrões das implementações podem prover uma facilidade tal que a ordem na qual as funções são avaliadas na rede pode ser definida. O padrão não define o método a ser usado. Isto significa que diferentes companhias que vendem *software* de programação em *FBD* podem definir seus próprios métodos de determinação como isto poderia ser avaliado. Um método seria uma lista que especificasse a ordem de avaliação.

Operador	Tipo do Bloco	Parametros
CD LD PV	Contador decrescente CTD	CD: contagem decrescente; LD: carregar; PV: valor de preset da contagem.
CU, R, PV	Contador crescente CTU	CU: contagem crescente; R: reset PV: valor de preset da contagem.
CU, CD, R, LD, PV	Contador Crescente e Decrescente	O mesmo para os contadores crescentes e decrescente.
IN, PT	Temporizador de Pulso TP	IN: início da temporização; PT: ajuste do tempo do pulso.
IN, PT	Temporizador para atraso de ligamento TON	IN: início do atraso; PT: ajuste do tempo de atraso.
IN, PT	Temporizador para atraso de desligamento TOF	IN: início do atraso; PT: ajuste do tempo de atraso.
CLK	Detetor de borda de subida R-Trig	Relógio do bloco de função, acionado pela borda de subida
CLK	Detetor de borda de descida F-Trig	Relógio do bloco de função, acionado pela borda de descida
S1,R	Biestável SR	Biestável Set e Reset
S,R1	Biestável RS	Biestável Reset e Set

Tabela A.2: Uma lista de operadores que são reservados para uso com blocos de função padrão

A.3 Diagrama de Relés (*LD*)

O diagrama de relés foi uma escolha lógica como uma linguagem de programação padrão devido ao seu grande uso. O diagrama de relé pode também ser usado em combinação com todos os outros métodos de programação que a IEC 1131-3 especifica. A Figura A.3 mostra os contatos que são especificados na IEC 1131-3. A Figura A.4 mostra as bobinas que são especificadas na IEC 1131-3. O *LD* também pode ser usado com outras linguagens da IEC 1131-3. A Figura A.5 mostra um exemplo do uso de *LD* com blocos de função os contatos que são especificados na IEC 1131-3.

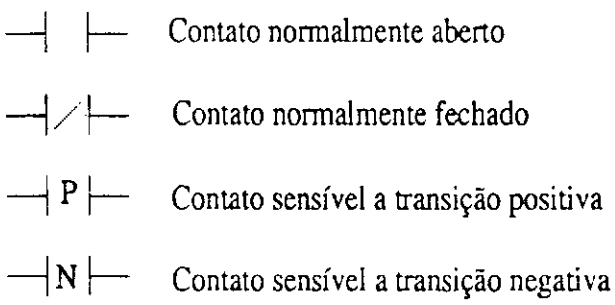


Figura A.3: Contatos especificados no IEC 1131-3.

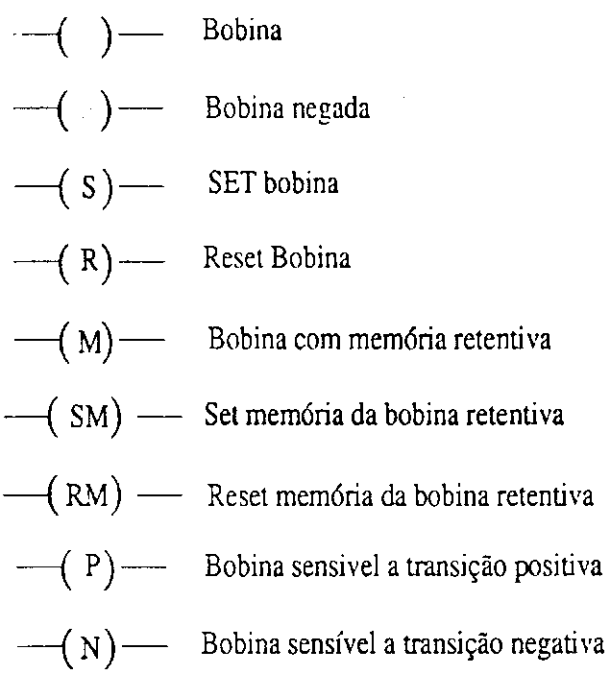


Figura A.4: Bobinas especificadas no IEC 1131-3.

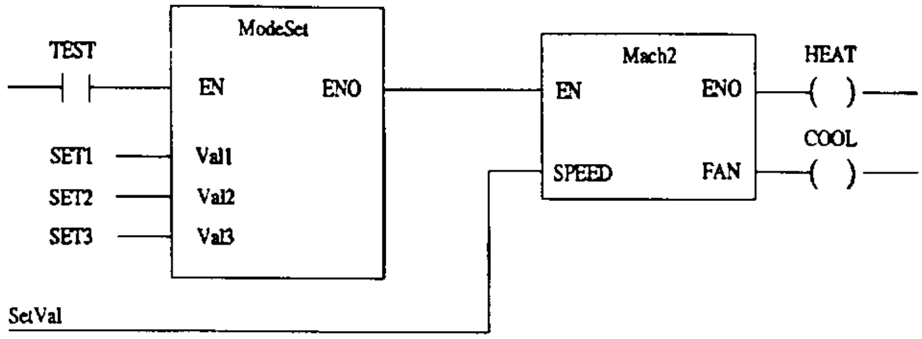


Figura A.5: Uso do *LD* combinado à blocos de função.

A.4 Lista de Instruções

Lista de instruções (*IL*) é uma linguagem de programação de baixo nível. Baixo nível tipicamente significa que não é amigável. Quanto mais alto for o nível da linguagem, mais fácil é o uso dela. A programação por lista de instruções é muito parecida com a programação em *assembly*. A Figura A.6 mostra um exemplo de um programa em lista de instrução.

```
           Operador  Operando   Comentário
              |         |            |
           LD        Temp       (*Carregar Temperatura*)
           LT        90         (*Testar se Temperatura é < 90*)
           JMPCN     Temp_Hi    (*Salte para Temp_Hi se Temp não é <90*)
           LD        Count      (*Carregar Count*)
           DIV       6          (*Dividir por 6*)
           ST        CASES      (*Armazena resultado da divisão na variável Cases*)
           LD        0          (*Carregar 0*)
           ST        Y12        (*Desligar a saída Y12*)
           ST        Y23        (*Desligar a saída Y23*)
Rótulo     LD        1          (*Carregar 1 *)
  |        ST        Y5         (*Armazenar na saída 5*)
Temp_Hi:   LD        1          (*Carregar 1*)
           ST        ALARM      (*Armazenar 1 na variável ALARM*)
```

Figura A.6: Programa exemplo em lista de instruções.

Programas *IL* podem ser usados por programadores experientes para desenvolver códigos mais rápidos e eficientes. A linguagem *IL* é considerada por muitos como a linguagem base para a IEC 1131. Ela é a linguagem para a qual todas as outras linguagens podem ser convertidas. O IEC não especifica qual é a linguagem base, no entanto. A Tab. A.3 mostra um exemplo de instruções aritméticas e booleanas que podem ser usadas em programas *IL*. A Tab. A.4 mostra as instruções de comparação que estão disponíveis na programação em

Operador	Operando	Propósito
ADD	Qualquer tipo	Adicionar
SUB	Qualquer tipo	Subtrair
MUL	Qualquer tipo	Multiplicar
DIV	Qualquer tipo	Dividir
LD	Qualquer tipo	Carregar
ST	Qualquer tipo	Armazenar
S	Booleano	Ajustar operando para verdadeiro
R	Booleano	Ajustar operando para falso
AND / &	Booleano	E Booleano
OR	Booleano	OU Booleano
XOR	Booleano	OU Exclusivo Booleano

Tabela A.3: Instruções aritméticas e Booleanas disponíveis na programação IL

IL.

Vamos comparar o programa em *IL* da Figura A.6 com o feito em texto estruturado.

```
IF Temp < 90 THEN
    CASES := Count / 6;
    Y12 := 0;
    Y23 := 0;
    Y5 := 1;
END_IF
ALARM :=1;
```

Funções e blocos de Funções podem ser chamados pelo uso do operador CAL. Um exemplo é mostrado abaixo.

```
CAL TEMP1 (SETPT := 85, CYC := 5)
```

Isto executaria o bloco de função chamado TEMP1. Este bloco de função atribuiria 85 ao parâmetro de entrada SETPT e 5 a CYC. Essas são as entrads requeridas pelo bloco de função TEMP1. Por este exemplo podemos observar a linguagem *IL* combinada a outras linguagens especificadas pela IEC 1131-3.

Há uma outra maneira de carregar parâmetros e fazer chamada por um bloco de função. No exemplo mostrado abaixo os valores são carregados e então armazenados como parâmetros para a função antes que a função seja chamada. A função LD é usada para carregar o

Operador	Operando	Propósito
GE	Qualquer tipo	Maior que ou igual a
GT	Qualquer tipo	Maior que
EQ	Qualquer tipo	Igual
NE	Qualquer tipo	Não igual
LE	Qualquer tipo	Menor ou igual a
LT	Qualquer tipo	Menor que
CAL	Nome	Chamar bloco de função
JMP	Rótulo	Saltar para Rótulo
RET		Retorno de uma função ou bloco de função
)		Execute a última operação diferida

Tabela A.4: Instruções de comparação disponíveis na programação *IL*

valor e então a ST é usada para armazenar o valor no parâmetro. O primeiro ST armazena 85 no parâmetro de entrada SETPT para o bloco de função (TEMP1.SETPT). O mesmo método é então usado para carregar 5 no parâmetro CYC. O bloco de função é então chamado com o operador CAL.

```
LD 85.0
ST TEMP1.SETPT
LD 5
ST TEMP1.CYC
CAL TEMP1
```

A Figura A.7 mostra um exemplo de um diagrama de um bloco de função para o início de uma seqüência. O equivalente lógico é mostrado abaixo.

```
LD Command_1
OR X17
ST StartRS.S
Load X12 Str\art.R1
LD StartRS.Q1
ST Start
```

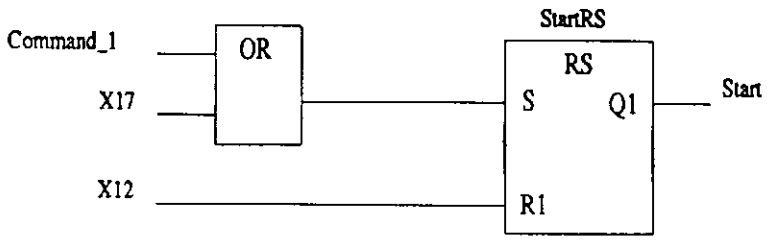


Figura A.7: Exemplo do uso de um OR.

A.5 *GRAFCET (SFC)*

A linguagem de programação *GRAFCET* (GRAFo de Comando Etapa-Transição) é uma adaptação da teoria de redes de Petri e permite uma visualização dos estados pelos quais o sistema a comandar deve passar para realizar uma dada operação. Esta linguagem também é conhecida como "Sequential Function Chart" (*SFC*). Um *GRAFCET* representa gráficamente o comportamento da parte de comando de um sistema automatizado. Ele é constituído por uma simbologia gráfica com *arcos orientados* que interligam *etapas* e *transições* por uma interpretação das variáveis de entrada e saída da parte de comando caracterizadas como *receptividades* e *ações*; e por regras de evolução que definem o comportamento dinâmico dos elementos comandados. A Figura A.8 mostra os elementos que constituem um *GRAFCET*.

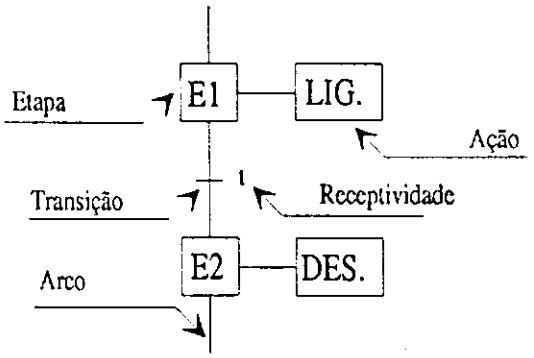


Figura A.8: Elementos de um *GRAFCET*.

A.5.1 Etapas

Uma *etapa* significa um estado no qual o comportamento do circuito de comando não se altera frente a suas entrads e saídas. As *etapas* são representadas graficamente por um quadrilátero, e devem ser identificadas com números, seguidos ou não por abreviaturas.

Em um determinado instante, uma etapa pode estar ativa ou inativa. O conjunto de etapas ativas em um determinado instante mostra, então, a situação em que o sistema se encontra.

A.5.2 Transição

É representada por um traço perpendicular aos arcos orientados e significa a possibilidade de evolução do *GRAFCET* de uma situação para a outra. Uma transição pode, em um dado instante, encontrar-se válida ou não, sendo que ela é válida quando todas as etapas imediatamente precedentes e ligadas a ela estiverem ativas. A passagem de uma situação à seguinte, portanto, só é possível com a validade de uma transição, momento este em que se diz que ocorre a transição.

A.5.3 Arcos Orientados

Indicam a sequencialização do *GRAFCET* pela interligação de uma etapa a uma transição e desta a outra etapa sucessivamente. A interpretação de sentido normal é de cima para baixo, sendo que em casos diferentes deste, é recomendável a indicação com flechas para a orientação de sentido.

A.5.4 Ação

As ações representam os efeitos que devem ser obtidos sobre o mecanismo controlado em uma determinada situação, ou seja, "o que deve ser feito". Em outros casos, pode também representar uma ordem de comando que especifica o "como deve ser feito". Multiplas ações podem ser associadas a uma etapa, cada uma delas é representada graficamente no interior de um retângulo e só são realizadas quando a etapa correspondente estiver ativa. A Figura A.9 mostra o formato geral para uma ação. O bloco de ações é mostrado a direita da etapa. A primeira parte do bloco de ação pode conter um qualificador. O qualificador controla como a ação é realizada. O qualificador neste caso é um N. Um N significa que a ação será executada enquanto a etapa associada estiver ativa. Qualificadores adicionais são mostrados na tabela A.5. A segunda parte de um bloco de ação é a ação a ser realizada. Neste caso a ação é StartSeq. A terceira parte do bloco de ação é opcional. O usuário pode usar uma variável na terceira parte do bloco de ações. A variável é chamada variável indicadora. A variável irá indicar quando a execução da ação termina. Neste exemplo a variável é chamada Stat. Neste exemplo quando StarSeq terminar sua execução Stat será ajustado para 1.

A.5.5 Receptividade

Receptividade é a funçaõ lógica combinacional associada a cada transição.

Quando em estado lógico verdadeiro, uma receptividade irá ocasionar a ocorrência de uma transição válida. Uma receptividade pode então ser encarada como o elo de ligação

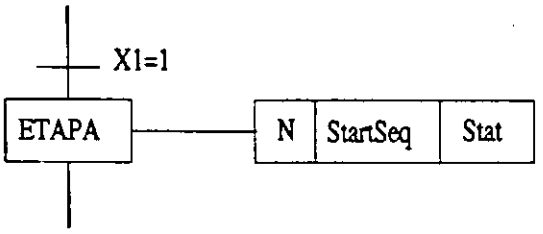


Figura A.9: Formato geral para uma ação.

Qualificador	Uso
N	Não armazenado, executa enquanto a etapa estiver ativa
None	Default, Não armazenado, executa enquanto a etapa estiver ativa
R	Reset uma ação armazenada
S	Set uma ação ativa
L	Terminar após um determinado período de tempo
D	Iniciar após um determinado período de tempo
P	Gerar um único pulso quando a etapa é desativada
SD	Tempo de atraso e armazenamento. A ação associada é ativada após um período de tempo, mesmo que a etapa associada seja desativada antes do fim deste período
DS	A ação é armazenar após um período de tempo. Se a etapa é desativada antes deste período a ação não é armazenada
SL	A ação é iniciada e executada por um dado período de tempo

Tabela A.5: Qualificadores que podem ser usados com ações

existente entre a lógica combinacional e a lógica seqüêncial.

Uma receptividade, na prática, pode representar variáveis lógicas tais como oriundas de sinais de entrada do sistema, variáveis internas de controle, resultado de comparações com contadores/temporizadores, informação sobre o estado de uma etapa (ativa ou inativa), ou ainda condicionada a uma determinada situação do *GRAFCET*.

Apêndice B

Conjunto de Instruções do PL7

Um programa feito utilizando lista de instruções para o *PL7* compreende uma série de instruções (até 1000 instruções) de vários tipos.

Cada linha do programa tem um número que é gerado automaticamente, um código de instrução e um operando do tipo *bit* ou palavra. Um exemplo de uma linha de instrução é mostrada na Figura B.1.

As instruções utilizadas na programação do PL7 quando da implementação do supervisor são: LD, S, R, AND, JMPC, SRn, SRn:,RET e bloco de instruções do temporizador.

LD: esta instrução é usada para carregar o acumulador com o conteúdo do operando;

S: esta instrução é usada para ajustar a saída para nível lógico 1;

R: esta instrução é utilizada para ajustar a saída para nível lógico 0;

AND: esta instrução realiza a lógica **AND** entre o conteúdo do acumulador e o operando especificado na linha de instrução, e troca o conteúdo do acumulador com o resultado da operação;

JMPC: A instrução **JMPC** provoca uma interrupção imediata na execução e o programa é continuado a partir de uma linha após a linha de programa contendo o rótulo %Li (i=0 a 15). **JMPC**, provoca um salto se o resultado da lógica precedente é 1. A Figura B.2 ilustra o funcionamento deste comando.

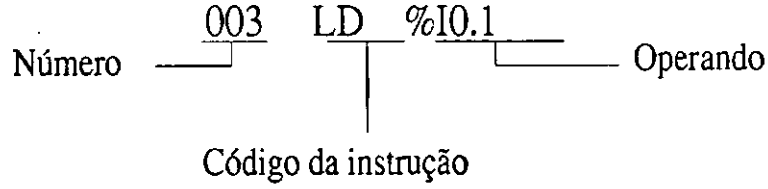


Figura B.1: Exemplo de uma linha de instrução em *IL*.

```
000  LD     %MW15
001   JMPC    %L8
002  LD     [%MW24>%MW12]
003  ST     %M15
004  %L8:
005  LD     %M12
006  AND    %M13
007  ST     %M2
```

Figura B.2: Exemplo do uso de **JMPC**.

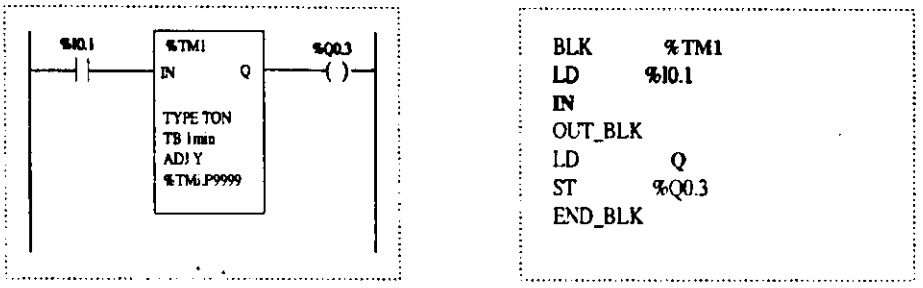


Figura B.3: Exemplo do bloco de instruções do temporizador.

SRn, SRn: e RET: A instrução **SRn** chama a subrotina referenciada pelo rótulo **SRn:** se o resultado da instrução booleana precedente for 1. A instrução **RET** é posta após a última linha de comando da subrotina forçando o retorno ao programa principal.

Bloco de instruções do temporizador: O bloco de instruções de um temporizador é mostrado na Figura B.3. Na Figura B.3(a) temos a representação de um temporizador como bloco de função em *LD*, e na Figura B.3(b) em *IL*. Nesta, **BLK** indica o início do bloco de instruções, **%TMi** (i=0 a 31) o número do temporizador, **OUT_BLK** indica início das condições de saída do bloco e **END_BLK** indica o fim do bloco.

Os operandos do tipo *bit* correspondem as entradas e saídas do *CLP*. O endereço de uma entrada (saída) é definido pelos seguintes caracteres:

%	I ou Q	0 ou 1	.	i
símbolo	I = entrada Q = saída	0 = *CLP* base ou *CLP peer* 1 = extensão I/O	ponto	i= N° da I/O

Os operandos do tipo palavra são variáveis internas ao *CLP* usadas para armazenar valores durante a operação. Elas são armazenadas na memória de dados. Palavras, %MW0

a %MW255, são lidas ou escritas diretamente pelo programa. Elas são usadas como palavras de trabalho.

Bibliografia

[1] D. Azzopardi e S. Lloyd. Reduction of Search Space for Scheduling of Multi-Product Batch Process Plant through Petri Net Modeling. *2nd IFAC/IFIP/IFORS Workshop on Intelligent Manufacturing Systems*, Viena, 13-15th, June 1994.

[2] D. Azzopardi e S. Lloyd. Scheduling and Simulation of Multi-Product Batch Process Plant Through Petri Net Modelling. *IEEE 4th International Conference on Advanced Factory Automation: Factory 2000*, University of York, UK, 3rd-5th, October 1994.

[3] Giovanni Cordeiro Barroso. *Uma Nova Abordagem para a Síntese de Supervisores de Sistemas a Eventos Discretos.* Tese de Doutorado, Universidade Federal da Paraíba, Brasil, 1996.

[4] Bernad Berthomieu e Michel Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259-273, March 1991.

[5] Christos G. Cassandras. *Discrete Event Systems Modeling and Performance Analysis.* Aksen Associates, 1993.

[6] Christos G. Cassandras e Stephen G. Strickland. Sample Path Properties of Timed Discrete Event Systems. *Proceedings of the IEEE*, 77(1):59-71, January 1989.

[7] Benedito Castrucci. *Elementos de teoria de conjuntos.* G.E.E.M. de São Paulo, 1967.

[8] Guy Cohen, Pierre Moller, Jean-Pierre Quadrat, e Michel Viot. Algebraic Tools for the Performance Evaluation of Discrete Event Systems. *Proceedings of the IEEE*, 77(1):39-58, January 1989.

[9] Eduard Montgomery Meira Costa. Contribuição ao Uso da Lógica Temporal na Especificação de Comportamentos de Sistemas a Eventos Discretos. Dissertação de Mestrado, Universidade Federal da Paraíba, Brasil, 1997.

[10] Luis Sergio Salles Costa e Heitor M. Caulliraux. *Manufatura Integrada por Computador Sistemas Integrados de Produção: Estratégia, Organização, Tecnologia e Recursos Humanos.* Editora Campus, 1995.

[11] Paulo R. da Silveira e Winderson E. Santos. *Automação e Controle Discreto.* Érica, 1998.

[12] René Dadid e Hassane Alla. *Petri Nets e Grafcet - Tools for Modeling Discrete Event Systems.* Prentice Hall, 1992.

[13] René David e Hassane Alla. Petri Nets for Modeling of Dynamic Systems - A Survey. *Automatica*, 30(2):175–202, February 1994.

[14] Edgard de Alencar Filho. *Teoria elementar dos conjuntos.* Editora Nobel S.A., 1967.

[15] Jorge Cesar Abrantes de Figueiredo. *Rede de Petri com Temporização Nebulosa.* Tese de Doutorado, Universidade Federal da Paraíba, Brasil, 1994.

[16] Alan A. Desrochers e Robert Y. Al-Jaar. *Applications of Petri Nets in Manufacturing Systems.* IEEE Press, 1995.

[17] S. Collart Dutilleul e J-P.Denat. P-Time Petri Nets and the Host Scheduling Problem. In *SMC'98 Conference Proceedings*, pp. 558–563. IEEE, 1998.

[18] Paul Freedeman. Time, Petri Nets, and Robotics. *IEEE Transactions on Robotics and Automation*, 7(4):417–433, August 1991.

[19] Georg Frey e Lothar Litz. Verification and Validation of control Algorithms by Coupling of Interpreted Petri Nets. In *SMC'98 Conference Proceedings*, pp. 7–12. IEEE, 1998.

[20] Peter W. Glynn. A GSPM Formalism for Discrete Event Systems. *Proceedings of the IEEE*, 77(1):14–23, January 1989.

[21] Keijo Heljanko. Model Checking the Branching Time Temporal Logic CTL. Relatório técnico, Helsinki University of Technology, 1997.

[22] Yu-Chi Ho. Dynamics of Discrete Event Systems. *Proceedings of the IEEE*, 77(1):3–6, January 1989.

[23] C. A. R. Hoare. *Communicating Sequential Process.* Prentice Hall, 1985.

[24] Stéphane Julia, Robert Valette, e José M. Fernandes. Scheduling Batch Systems Using A Token Player Algorithm. In *SMC'98 Conference Proceedings*, pp. 487–492. IEEE, 1998.

[25] Stéphane Julia, Robert Valette, Clarimundo M. M. Júnior, e José M. Fernandes. Escalonamento de sistemas de Produção Híbridos Usando-se um Jogador de Rede de Petri. *Proceedings of XII Brazilian Automatic Control Conference - XII CBA*, 4:1397–1402, September 14-18, Uberlândia, MG, Brazil 1998.

[26] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, e G. Franceschinis. *Modelling With Generalized Stochastic Petri Nets.* John Wiley and Sons Ltd, 1995.

[27] Philip M. Merlin. A Methodology for the Design and Implementacion of Communication Protocols. *IEEE Transactions on Communications*, 24(6):614–621, June 1976.

[28] Philip M. Merlin e David J. Farber. Recoverability of Communication Protocols - Implication of a Theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, September 1976.

[29] R. Milner. *A calculus of Communicating Systems.* Spring Verlag, 1980.

[30] Paulo Eigi Miyagi. *Controle Programável - Fundamentos do Controle de Sistemas a Eventos Discretos.* Editora Edgard Blucher Ltda, 1996.

[31] Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541-580, April 1989.

[32] Jean-Marie Proth e Xiolan Xie. *Petri Nets a Tool for Design and Management of Manufacturing Systems.* Wiley, 1996.

[33] Peter J. G. Ramadge e W. Murray Wonham. Supervisory Control of a Class of Discrete Processes. *SIAM J. Control and Optimization*, 25(1):206-230, 1987.

[34] Peter J. G. Ramadge e W. Murray Wonham. The Control of Discrete Event Systems. *Proceedings of the IEEE*, 77(1):81-98, January 1989.

[35] Marcelo Ricardo Stemmer. *Controladores Lógicos Programáveis.* Departamento de Engenharia Mecânica - UFSC, 1996.

[36] Jon Stenerson. *Fundamentals of Programmable Logic Controllers, Sensors and Communications.* Prentice Hall, 1999.

[37] Kurapati Venkatesh, MengChu Zhou, e Reggie J. Caudill. Comparing Ladder Logic Diagrams and Petri Nets for Sequence Controller Design Through a Discrete Manufacturing System. *IEEE Transactions on Industrial Eletronics*, 41(6):611–619, December 1994.

[38] Kurapati Venkatesh, MengChu Zhou, e Reggie J. CAudill. Discrete Event Control Design for Manufacturing Systems Via Ladder Logic Diagrams and Petri Nets: A Comparative Study. In *Petri Nets in Flexible and Agile Automation*, pp. 265–304. Kluwer Academic Publishers, 1995.

[39] N. Viswanadham e Y. Narahari. *Performance Modeling of Automated Manufacturing Systems.* Pretice Hall, 1992.

[40] Guido Wimmel. A BDD-based Model Checker for the PEP Tool. Relatório técnico, University of Neswcastle, 1997.

[41] W. Murray Wonham e Peter J. G. Ramadge. On the Supremal Controllable Sublanguage of A Given Language. *SIAM J. Control and Optimization*, 25(3):637–659, May 1987.

[42] MengChu Zhou e Frank Dicesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems.* Kluwer Academic Publishers, 1993.

[43] MengChu Zhou e Richard Zurawski. Introduction to Petri Nets in Flexible and Agile Automation. In *Petri Nets in Flexible and Agile Automation*, pp. 1–42. Kluwer Academic Publishers, 1995.

[44] Roberto M. Ziller. *A Abordagem Ramadge-Wonham no Controle de Sistemas a Eventos Discretos: Contribuições à Teoria.* Tese de Doutorado, Universidade Federal de Santa Catarina, 1993.

[45] Richard Zurawski. Systematic Construction of Funtional Abstractions of Petri Net Models of Flexible Manufacturing Systems. *IEEE Transactions on Industrial Eletronics*, 41(6):584–592, December 1994.

[46] Richard Zurawski e MengChu Zhou. Petri Nets and Industrial Applications: A Tutorial. *IEEE Transactions on Industrial Eletronics*, 41(6):567–583, December 1994.