Apostila
Redes de Petri e Sistemas a Eventos Discretos

Eduard Montgomery Meira Costa, DSc
UFBA

Introdução

Nos últimos anos houve um desenvolvimento significativo do estudo de sistemas dinâmicos a eventos discretos. Estes tipos de sistemas podem ser observados em muitas aplicações do cotidiano e, podem ser exemplificados pelas redes de computadores, pelos sistemas de manufatura, pelos sistemas de comunicação e pelos sistemas de tráfego aéreo e ferroviário.

De modo geral, pode-se dizer que promover a modernização industrial resulta na criação de sistemas dinâmicos a eventos discretos. De fato, esta modernização que está sempre associada à automação industrial, exige a instalação de tecnologia de base microeletrônica em vários níveis da cadeia produtiva. É justamente da interação dessa tecnologia com os sub-sistemas pré-existentes nos diversos níveis hierárquicos do processo industrial que emergem os sistemas dinâmicos a eventos discretos.

A modernização industrial progride a passos largos e, desse modo, há uma tendência clara de ampliação do nível de automação industrial e conseqüentemente uma proliferação dos sistemas dinâmicos a eventos discretos. Neste contexto, a investigação de ferramentas matemáticas adequadas para o estudo e projeto desse tipo de sistema adquire um elevado grau de importância. Este destaque é enfatizado pelo fato de que as ferramentas ditas convencionais de estudo (e.g., modelos formulados na forma de equações integro-diferenciais) não são suficientes para viabilizar o estudo deste novo tipo de sistema.

Os sistemas de automação construídos pelo homem são denominados de Sistemas Dinâmicos a Eventos Discretos (SEDs). Este tipo de sistema surge, de modo geral, quando realiza-se a modernização de um processo industrial substituindo, onde possível, a tecnologia analógica pela tecnologia digital. Estes sistemas modernizados não são representados através de equações diferenciais normalmente utilizadas para descrever os sistemas dinâmicos de variáveis contínuas (SDVCs). De fato, os SEDs exibem características específicas em sua evolução dinâmica que exigem outros paradigmas de representação. Uma primeira etapa no estudo de SEDs é localizar este tipo de sistema numa classificação genérica de sistemas com o intuito de destacar suas características peculiares.

Um SED é definido como um sistema cuja evolução dinâmica depende da ocorrência de eventos. Um evento pode ser identificado com uma ação proposital (e.g., ligar

um interruptor), uma ocorrência espontânea (e.g., perda de conexão com provedor de internet) ou o resultado da verificação de uma condição (e.g., a temperatura do reator excedeu o limite de segurança). Os eventos produzem mudanças de estado e, de modo geral, ocorrem em instantes de tempo irregulares.

Na Figura 0.1, é apresentado um diagrama que representa uma evolução dinâmica de um SED. Nesta figura os eventos são etiquetados com símbolos α, β e η e os estados designados por q_0, q_1, q_2 e q_3. O estado inicial é q_0 e, α, β e η, representam os eventos que mudam o estado do sistema para q_1, q_2 e q_3, respectivamente. Quando não há a ocorrência de nenhum evento, o sistema permanece no mesmo estado.

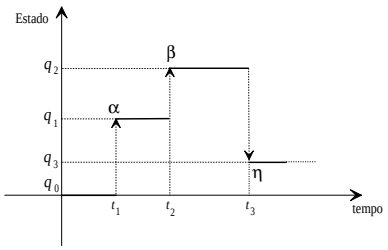

Figura 0.1: Evolução dinâmica de um sistema dinâmico a eventos discretos: ocorrências de eventos assíncronos e mudanças instantâneas de estado.

Uma diferença básica entre um SED e um SDVC é a maneira de interagir com o ambiente. No caso de um SDVC, as mudanças de estado decorrem da troca de energia com o ambiente. No caso de um SED, os eventos gerados no ambiente causam mudanças na sua configuração interna e determinam sua evolução dinâmica. Desse modo, um SED é definido por três características básicas, que são:

1. **Ciclo de funcionamento descrito através do encadeamento de eventos** que determinam as tarefas realizadas ou em realização;

2. **Ocorrência de eventos simultâneos**, isto é, vários eventos podem ocorrer ao mesmo tempo para alterar o estado do sistema;

3. **Necessidade de sincronização**, pois para que a evolução dinâmica seja correta, o início de certas atividades requer o término de outras.

A seguir são apresentados alguns exemplos típicos de SEDs, para um maior esclarecimento de suas características.

Exemplo 1 *Considere os dois computadores interligados, como mostrado na Figura 0.2. Este sistema pode ser considerado um SED quando se deseja estudar a comunicação entre os computadores e o ambiente, que é descrita por ações que podem ser caracterizadas como eventos. A interação com o ambiente em qualquer um dos computadores efetua-se através da apresentação de dados no vídeo, aquisição de dados através do teclado ou de um "scanner".*

- *As características desse sistema são típicas de um SED, desde que há eventos ocorrendo simultaneamente: um computador recebe dados pelo teclado, no mesmo instante em que o outro apresenta dados no vídeo. Seu funcionamento é descrito por seqüências de eventos: "apresentar dados no vídeo", "solicitar novos dados", "ler novos dados", "processar dados" e "apresentar novos dados no vídeo" é uma possível seqüência de eventos nesse sistema.*

- *Um computador só avalia os novos dados após sua leitura: isto implica em sincronização. Enquanto não é fornecida uma entrada solicitada, o computador se mantém em estado de espera e, ao pressionar a tecla <**ENTER**>, ou o botão do "mouse", o estado do sistema é mudado para o estado de processamento dos dados. A comunicação entre os computadores, é semelhante, desde que o envio de uma mensagem mantém um dos computadores num estado de espera.*

- *Deve existir um sincronismo entre os computadores para garantir que um dado enviado seja corretamente recebido. Desse modo, quando um deles começa a enviar dados, o outro imediatamente inicia um processo de leitura e vice-versa.*

- *Percebe-se também, que o estado do sistema muda instantaneamente com a ocorrência de qualquer um dos eventos citados.*

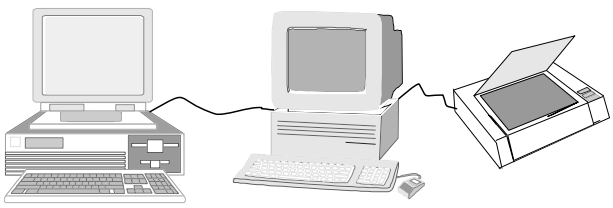

Figura 0.2: Redes de computadores podem ser consideradas SEDs.

Exemplo 2 *Na Figura 0.3, é apresentado um diagrama simplificado de uma estação ferroviária, que também pode ser considerada um SED. Alguns eventos desse sistema são: "abrir portas", "fechar portas", "aviso de partida", "partida do trem", "aviso de chegada" e "parada na estação". Assim, a evolução dinâmica do sistema é representada pelo encadeamento de eventos.*

- *A ocorrência de cada um destes eventos, muda o estado do SED abruptamente.*
- *A interação com o ambiente, efetua-se através de eventos do tipo: "entrada de passageiros", "saída de passageiros", "comunicação com a central" e "recepção de avisos".*
- *Vários eventos podem ocorrer simultaneamente, tais como: "aviso de chegada" e "abrir portas" ou "recebimento de aviso de estação ocupada" e "desaceleração para parada".*
- *Há sincronismo: Essa característica é vista na comunicação de um trem com a central que informa o estado da estação (livre ou ocupada). Neste caso, o trem deve responder e manter o movimento, caso a estação esteja livre, ou desacelerar e parar, caso a estação esteja ocupada. No segundo caso, o trem se mantém num estado de espera até receber um novo aviso de estação livre, e iniciar seu movimento, entrar na estação, parar, avisar da chegada e abrir as portas para saída de passageiros.*

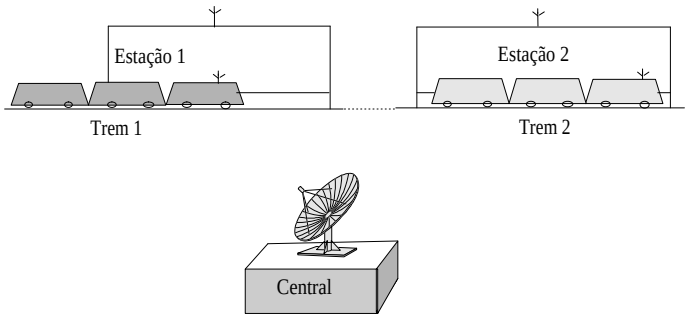

Figura 0.3: Sistema de supervisão de tráfego ferroviário.

Exemplo 3 *Um diagrama simplificado de um sistema de manufatura com recursos compartilhados composto por uma esteira, um braço robótico e uma máquina é mostrado na Figura 0.4. Este sistema também é um SED, pois seu comportamento pode ser descrito através de ações tais como "mover esteira", "colocar peça na máquina", "processar peça na máquina" e "retirar peça da máquina". Estes eventos em determinadas seqüências determinam tarefas.*

- *Alguns desses eventos podem ocorrer em paralelo, tais como: "mover esteira" e "processar peça na máquina", ou "mover esteira" e "retirar peça da máquina". Entretanto deve haver sincronismo tal como: "colocar peça na máquina" requer que a máquina tenha terminado o processamento da peça e que o evento "retirar peça da máquina" tenha ocorrido, ou "mover esteira" só pode ocorrer se não houver uma peça em sua extremidade, o que implica em dizer que a mesma só deve ser movimentada depois do evento "colocar peça na máquina".*

- *A cada ocorrência de um destes eventos, o estado do sistema é modificado. Assim, no estado "esteira com peça na extremidade" e "braço robótico livre", a ocorrência do evento "colocar peça na máquina", o estado muda para "esteira livre para movimentação" e "braço robótico ocupado".*

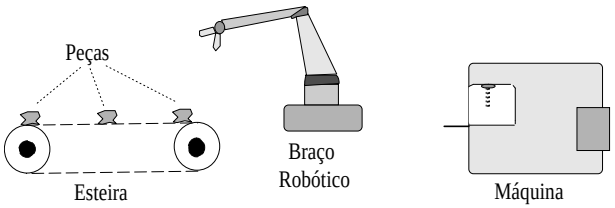

Figura 0.4: Um simples sistema de manufatura.

Considera-se que a quantidade de eventos diferentes de um SED é finita. Para o estudo de SEDs é necessária a definição de quais são os eventos e os estados relevantes, que em conjunto com o paradigma de modelagem escolhido proporcionam uma ferramenta matemática para analisar seu comportamento.

Quando é definido o paradigma de modelagem, elabora-se um modelo de um SED a partir do qual determina-se o espaço de estados. Neste espaço de estados, em muitos casos, torna-se necessário definir um ou mais estados para os quais o SED deva retornar

ao completar uma tarefa. Estes estados são conhecidos por estados recorrentes (*home state*). Em algumas aplicações, este estado pode ser o estado inicial do sistema. Quando isto acontece, denomina-se este processo de *reinicialização*.

O problema de controle de SEDs

Quando um SDVC não apresenta um comportamento dinâmico satisfatório projeta-se um controlador que interligado ao sistema numa configuração realimentada garante que o comportamento desejado seja efetivamente obtido. O projeto desse tipo de sistema de controle começa pela formulação de um problema de controle para o qual um controlador é a solução. Uma situação semelhante ocorre com os SEDs. Entretanto, nesse caso convenciona-se que a solução do problema de controle de um SED é um supervisor.

O problema de controle de SEDs é resolvido através da Teoria de Controle Supervisório (TCS) [1, 2, 3, 4, 5]. Uma característica importante da TCS é a separação explícita do sistema a ser controlado (*open loop dynamics*) do controlador propriamente dito (*feedback control*).

A formulação do problema de controle dos SEDs é dividida em três etapas:

1. **Modelagem**, na qual se utiliza um determinado paradigma que represente o SED matemática ou visualmente, e que permita determinar seus estados e sua evolução dinâmica. Se o modelo representa adequadamente o sistema, vários estudos e análises podem ser feitas utilizando-se o próprio modelo, sem o risco, custo ou inconveniência da manipulação direta do sistema real. Na modelagem de SEDs vários paradigmas podem ser considerados. Entretanto, até o presente momento, nenhum desses se tornou aceito universalmente. Os paradigmas mais utilizados na representação de SEDs são: Cadeias de Markov [6]; Teoria das filas [6]; Processos semi-Markovianos generalizados [6]; Álgebra de processos [7]; Álgebra de dióides [8, 9, 10, 11, 12], e suas formalizações específicas como a álgebra Max-Plus [13, 14, 15], álgebra Min-Plus [16, 17, 18, 19] e álgebra de caminhos [20]; Teoria de linguagens formais e autômatos (máquinas de estados finitos) [21, 22, 23, 24, 25]e Redes de Petri [26, 27, 28, 29, 30, 31, 32, 33].

2. **Especificação de comportamento**, onde se define qual a tarefa que o sistema

deve realizar. Essa tarefa é expressa em linguagem natural e formalizada numa linguagem de lógica matemática compatível com o paradigma de modelagem. Dois formalismos matemáticos podem ser utilizados para definir a especificação de comportamento. Estes formalismos matemáticos são: **Linguagens formais** [23, 22], em que é definido um alfabeto de símbolos que representam os eventos, e sua concatenação define palavras (*seqüências de símbolos*) ou seqüências de eventos, que representam tarefas a ser executadas; **Lógica temporal** [34, 35, 36, 37, 38, 39, 40], que utiliza proposições, operadores lógicos e operadores temporais para construção de fórmulas lógicas. A avaliação da veracidade ou falsidade da fórmula define se a tarefa é executada ou não. A lógica temporal é uma ferramenta que tem maior poder de expressão do que as linguagens formais, pois a veracidade ou falsidade de uma fórmula pode ser conseguida por vários caminhos diferentes.

3. **Síntese do supervisor**, em que, soluciona-se o problema de controle e define-se um supervisor. Este supervisor observa os eventos gerados pelo SED e define uma seqüência de ações que garante que a seqüência de eventos que define o comportamento especificado, será executada corretamente. Este ciclo representa a operação de um sistema de controle em malha fechada, como mostrado na Figura 0.5. Na síntese do supervisor considera-se que os eventos gerados pelo

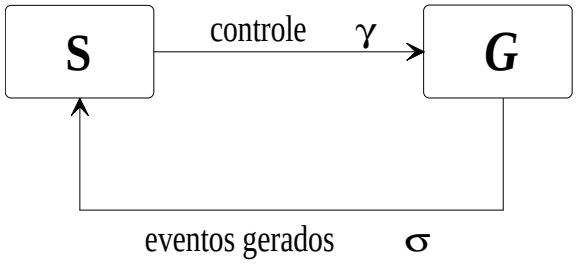

Figura 0.5: Controle supervisório em malha fechada: o gerador gera os eventos σ para o supervisor que determina a estratégia de controle que o controlador deve aplicar ao sistema para realizar a especificação desejada.

SED podem ser, numa primeira análise, classificados em duas classes distintas e disjuntas, as quais são designadas por: **Eventos controláveis:** Eventos cuja ocorrência pode ser alterada pela ação de controle, isto é, podem ser habilitados ou desabilitados em qualquer momento (por exemplo: o início de uma atividade e

a parada de uma esteira); **Eventos não-controláveis:** Eventos cuja ocorrência não pode ser alterada pela ação de controle, isto é, estão permanentemente habilitados em toda a evolução dinâmica do SED (por exemplo: a quebra de uma máquina, a falta de energia elétrica e o término do processamento de uma peça).

Observação: A formulação genérica do problema de controle no contexto da TCS visa a determinação do supervisor a partir do modelo do sistema controlador e da especificação de comportamento desejada para o SED. Uma dificuldade encontrada na síntese do supervisor, é que a tarefa especificada pode incluir ou não eventos do tipo não-controláveis. A ocorrência desses eventos pode levar o sistema a situações de bloqueio ou fazê-lo alcançar estados indesejáveis. Desse modo, a função do supervisor é assegurar que a seqüência de eventos gerada pela sua associação com o SED, implemente a especificação de comportamento desejada, ou seja, fazer com que o SED realize uma tarefa específica.

Exemplo 4 *Pode-se ilustrar como um exemplo típico de bloqueio, a situação da Figura 0.4, em que o braço robótico coloca uma peça na máquina para processamento e, de imediato pegue outra peça na esteira. Esta situação define um bloqueio no sistema, desde que o braço robótico fica com a peça sem poder devolvê-la para a esteira, nem colocá-la na máquina e nem retirar a peça processada da máquina. Por outro lado, é desejável que cada peça processada seja consumida. Caso contrário, a produção deve ser interrompida após ter sido armazenada uma determinada quantidade de peças processadas num depósito. Caso essas condições não sejam satisfeitas, haverá um aumento ilimitado de peças no depósito.*

Ferramentas para formalização do controle supervisório

Linguagens formais

As linguagens formais, são definidas a partir de um alfabeto que é um conjunto de símbolos (letras ou dígitos), representado geralmente pela letra grega maiúscula Σ. Por exemplo: $\Sigma = \{\alpha, \beta, \gamma, \delta\}$ e $\Sigma = \{a, b, c, d, e\}$.

Para um dado alfabeto, define-se uma palavra s como a concatenação de seus elementos, com ou sem repetição. Por exemplo, sendo $\Sigma = \{\alpha, \beta, \gamma, \delta\}$, então $\alpha\alpha, \beta, \alpha\beta, \gamma\delta$, $\gamma\alpha\delta, \gamma\delta\alpha\beta\beta, \beta\delta\delta\delta\alpha\alpha\gamma\delta$, são exemplos de palavras.

O comprimento de uma palavra s é igual ao número de símbolos que a compõe, isto é, é igual à sua cardinalidade, que é representada por $|s|$.

Na teoria de linguagens formais, a palavra vazia é a única palavra de comprimento nulo. Essa palavra é representada pelo símbolo ϵ. Desse modo, $\epsilon \notin \Sigma$, pois ϵ é uma palavra, e não um símbolo do alfabeto Σ.

Definem-se os conjuntos de palavras de mesmo comprimento $|s| = k$ como Σ^k.

Exemplo 5 *Se $\Sigma = \{\gamma, \zeta\}$ é um alfabeto, então*

$$\begin{aligned}
\Sigma^0 &= \{\epsilon\} \\
\Sigma^1 &= \{\gamma, \zeta\} \\
\Sigma^2 &= \{\gamma\gamma, \gamma\zeta, \zeta\gamma, \zeta\zeta\} \\
&\vdots
\end{aligned}$$

Note que sendo ϵ a única palavra de comprimento nulo, ela é também o único elemento de Σ^0.

A união dos conjuntos Σ^k gera os conjuntos:

- $\Sigma^+ = \bigcup_{k=1}^{\infty} \Sigma^k = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$, e
- $\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots = \Sigma^0 \cup \Sigma^+ = \{\epsilon\} \cup \Sigma^+$,

onde Σ^+ é interpretado como o conjunto de todas as palavras que podem ser formadas com os símbolos do alfabeto Σ. No caso de Σ^*, vê-se que este conjunto difere de Σ^+ apenas pela inclusão da palavra nula ϵ.

Uma linguagem, denotada por L, é definida como sendo um conjunto de palavras formadas através de uma concatenação específica de símbolos do alfabeto Σ. Logo, conclui-se que L é uma linguagem sobre o alfabeto dado, Σ, se e somente se $L \subseteq \Sigma^*$. Deve-se observar que a linguagem vazia $L = \{\}$, que não contém nenhuma palavra, é diferente da linguagem formada pela palavra nula $\Sigma^0 = \{\epsilon\}$, a qual contém um único elemento.

Também define-se a concatenação de palavras. Sendo s_1 e s_2 duas palavras formadas a partir dos símbolos de um alfabeto Σ, com

$$s_1 = \sigma_1\sigma_2\sigma_3...\sigma_k \text{ e}$$
$$s_2 = \sigma_{k+1}\sigma_{k+2}...\sigma_n,$$

sua concatenação, representada por s_1s_2, resulta numa nova palavra definida por

$$s = s_1s_2 = \sigma_1\sigma_2\sigma_3...\sigma_k\sigma_{k+1}\sigma_{k+2}...\sigma_n.$$

A concatenação de palavras pode ser estendida para definir a concatenação de linguagens. Se L_1 e L_2 são duas linguagens sobre um mesmo alfabeto, então L_1L_2, que é a linguagem concatenada, é definida por

$$L_1L_2 = \{s_1s_2 | s_1 \in L_1 \wedge s_2 \in L_2\}.$$

Define-se a parte inicial de uma palavra por prefixo dessa palavra. Assim, um prefixo de uma palavra s sobre um alfabeto Σ é qualquer palavra $s_1 \in \Sigma^*$ que pode ser completada com outra palavra $s_2 \in \Sigma^*$ para formar a palavra s. Ou seja, desde que s_1 concatenada com s_2, forma a palavra s, vê-se que s_1 é um prefixo de s. Desta maneira, tem-se que a palavra nula ϵ e a própria palavra s são prefixos da palavra s e assim, uma palavra s tem $|s| + 1$ prefixos. Denota-se por $Pref(s)$, o conjunto de todos os prefixos de uma palavra s.

O prefixo-fechamento, ou simplesmente fechamento da linguagem L, é uma linguagem associada a L, a qual é formada por suas palavras e por todos os seus prefixos. O prefixo-fechamento de L é denotado por $\overline{L}$ e é formalmente definido por

$$\overline{L} = \{s_1 : \exists s_2 \in \Sigma^* \wedge s_1s_2 \in L\},$$

de onde se vê que $L \subseteq \overline{L}$.

Exemplo 6 *Dada uma linguagem $L = \{\epsilon, \alpha\beta, \alpha\gamma\beta\}$ com palavras formadas a partir do alfabeto $\Sigma = \{\alpha, \beta, \gamma\}$, o fechamento de L é $\overline{L} = \{\epsilon, \alpha, \alpha\beta, \alpha\gamma, \alpha\gamma\beta\}$, e portanto $L \subset \overline{L}$.*

Exemplo 7 *Dada uma linguagem $L = \{\epsilon, \alpha, \alpha\alpha, \alpha\alpha\alpha, ...\}$ com palavras formadas a partir do alfabeto $\Sigma = \{\alpha\}$, o fechamento de L é $\overline{L} = \{\epsilon, \alpha, \alpha\alpha, \alpha\alpha\alpha, ...\}$ e, neste caso, $L = \overline{L}$.*

Define-se uma linguagem como prefixo-fechada, ou simplesmente fechada, se e somente se $L = \overline{L}$. Assim, no Exemplo 7, L é prefixo-fechada. Disto decorre que, se L é prefixo-fechada, então, para cada palavra s pertencente à linguagem L, tem-se que $Pref(s) \subseteq L$.

Outras operações definidas para as linguagens formais são: fechamento-*Kleene*, união, interseção e complemento.

O fechamento-*Kleene* representado por L^* é definido como a mesma operação de fechamento usual para o conjunto Σ (alfabeto). Só que neste caso, utilizam-se palavras, as quais podem ter comprimentos maiores que 1. Desta forma, um elemento de L^* é formado pela concatenação de um número finito de elementos de L. O mesmo é definido como

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

Esta operação é idempotente, isto é, $(L^*)^* = L^*$.

A união de linguagens é definida por

$$L_a \cup L_b := \{s_a, s_b \in \Sigma^* : (s_a, s_b) \wedge (s_a \in L_a) \wedge (s_b \in L_b)\},$$

a interseção é definida por

$$L_a \cap L_b := \{s_a, s_b \in \Sigma^* : (s_a = s_b) \wedge (s_a \in L_a) \wedge (s_b \in L_b)\}$$

e o complemento (L^c), é definido por

$$L^c := \{s \in \Sigma^* : s \notin L\}.$$

Exemplo 8 *Dadas* $L_1 = \{\beta, \alpha\beta, \alpha\alpha, \beta\alpha\}$ *e* $L_2 = \{\epsilon, \alpha, \beta\alpha, \alpha\beta\}$*, a união de* L_1 *e* L_2 *é*

$$L_1 \cup L_2 = \{\epsilon, \alpha, \beta, \alpha\beta, \alpha\alpha, \beta\alpha\}$$

e sua interseção é

$$L_1 \cap L_2 = \{\beta\alpha, \alpha\beta\}.$$

Exemplo 9 *Se* $\Sigma = \{\alpha\}$ *e* $L = \{\alpha\alpha, \alpha\alpha\alpha, \alpha\alpha\alpha\alpha, ...\}$ *seu complemento é*

$$L^c = \{\epsilon, \alpha\}.$$

Alguns tipos de linguagens podem ser representados de forma compacta através de expressões regulares. A construção de expressões regulares segue as seguintes regras básicas:

1. $\emptyset$ é uma expressão regular denotando o conjunto vazio, ϵ é uma expressão regular denotando o conjunto $\{\epsilon\}$, e σ também é uma expressão regular denotando o conjunto $\{\sigma\}$ para todo $\sigma \in \Sigma$.

2. Se r e s são expressões regulares, então rs, $(r+s)$, r^* e s^* também são expressões regulares.

3. Não há outras expressões regulares que possam ser construídas através da aplicação das regras 1 e 2 um número finito de vezes.

Logo, para representar de forma compacta linguagens complexas utiliza-se σ^* e s^* a repetição do símbolo σ e da palavra s por um número arbitrário de vezes e o símbolo $+$ representando o operador lógico "*ou*", indicando uma opção entre duas possibilidades.

Exemplo 10 *A linguagem*

$$L = \{\epsilon, \alpha, \alpha\beta, \alpha\beta\beta, \alpha\beta\beta\alpha, \alpha\beta\beta\alpha\beta, \alpha\beta\beta\alpha\beta\beta, ...\}$$

pode ser representada de forma compacta pela expressão regular

$$L = \left\{ \left(\alpha\beta^2\right)^* \left((\alpha + \alpha\beta) + \epsilon\right) \right\}.$$

Exemplo 11 *A expressão regular* $(\alpha\beta)^* + \gamma$ *representa a linguagem*

$$L = \{\epsilon, \gamma, \alpha\beta, \alpha\beta\alpha\beta, \alpha\beta\alpha\beta\alpha\beta, ...\}.$$

Definição 1 *Qualquer linguagem que pode ser denotada por uma expressão regular é denominada de* Linguagem Regular.

Esta Definição formaliza o conceito de linguagem regular, que é de fundamental importância para o estudo dos autômatos seqüenciais finitos [21, 22, 23, 24].

Autômatos

Um autômato é um dispositivo que pode reconhecer uma determinada linguagem e que também pode ser usado para representar certos tipos de sistemas dinâmicos. Um autômato têm estados que representam as condições operacionais do dispositivo ou as situações do sistema dinâmico. Se o número de estados é finito o autômato é denominado de autômato finito ou máquina de estado finito e se o número de estados é infinito o autômato é denominado de autômato infinito ou máquina de estado infinito.

Para reconhecer as palavras de uma determinada linguagem o autômato lê sequencialmente os símbolos e esta leitura produz, eventualmente, uma mudança de estado. Uma palavra é dita reconhecida se o estado final alcançado após a leitura do último símbolo pertence a um conjunto de *estados marcados*. O estado em que um autômato se encontra antes da leitura do primeiro símbolo é denominado de *estado inicial*. Desse modo, um autômato pode ser visto como uma entidade de controle que internamente tem uma variável que representa o estado do autômato. Logo, cada símbolo lido atualiza esta variável de acordo com uma função de transição que associa um novo estado a cada par (*símbolo*, *estado*) ou (*evento*, *estado*). Se a função de transição leva a mais de um estado quando certos símbolos são lidos, o autômato é denominado de autômato não determinístico. Por outro lado, se a função de transição leva a um único estado para cada símbolo lido, o autômato é dito determinístico.

Definição 2 *Um autômato determinístico finito, ou simplesmente um autômato é uma quíntupla* $A = (Q, \Sigma, \delta, q_0, Q_m)$*, onde:* Q *é um conjunto finito de estados* q*;* Σ *é o alfabeto ou conjunto de símbolos* σ*;* $\delta : \Sigma \times Q \to Q$ *é a função de transição de estados, onde*

$$\begin{aligned} &\delta(\epsilon, q) = q \qquad\qquad e \\ &\delta(\sigma, q) = q', \quad \text{para } q,\ q' \in Q \text{ e } \sigma \in \Sigma; \end{aligned}$$

$q_0 \in Q$ *é o estado inicial;* $Q_m \subseteq Q$ *é o conjunto de estados marcados.*

Observando a função de transição de estados δ, vemos que q' somente será um estado do autômato A, se o símbolo σ for uma entrada aceita por ele.

Graficamente, um autômato pode ser representado por um diagrama de transição de estados, que é um grafo direcionado, onde seus vértices são os estados e os arcos representam as funções de transição. Logo, um autômato definido por $A = (Q, \Sigma, \delta, q_0, Q_m)$

pode ser representado graficamente pelo grafo orientado $G = \{V, W\}$, no qual

$$V = Q,\ W = \{(q, q', \sigma) : q, q' \in Q \wedge \sigma \in \Sigma \wedge \delta(\sigma, q) = q',$$

os vértices ou estados são representados por círculos, os estados marcados, por círculos em linha dupla (dois círculos concêntricos) e o estado inicial é indicado por uma seta que não é saída de nenhum vértice.

Exemplo 12 *O autômato representado graficamente na Figura 0.6, é definido por:*

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

onde os círculos são estados, os círculos concêntricos são estados marcados, a seta que não é saída de nenhum vértice (*estado*) *é o estado inicial e os arcos indicam as funções de transição de estados.*

. *Nesta figura, observa-se que, do estado inicial* (*estado 0*)*, através do evento* α (*arco* α) *há a mudança para o estado marcado 1, que é representado pela função de transição* $\delta(\alpha, 0) = 1$. *Neste estado, a ocorrência do símbolo* γ *não produz mudança de estado* (*função de transição* $\delta(\gamma, 1) = 1$). *Contudo, no estado inicial, se houver a ocorrência do símbolo* β*, o autômato segue para o estado marcado 2* (*função de transição* $\delta(\beta, 0) = 2$) *e assim por diante.*

Observa-se que quanto maior o número de estados do autômato, mais difícil é sua visualização e sua compreensão através de sua representação gráfica.

Exemplo 13 *Considere um setor de uma fábrica que é constituído de um braço robótico, uma máquina que processa peças-brutas e um déposito para armazenar peças-trabalhadas* (buffer). *O braço robótico transporta as peças-brutas e as peças-trabalhadas. O buffer*

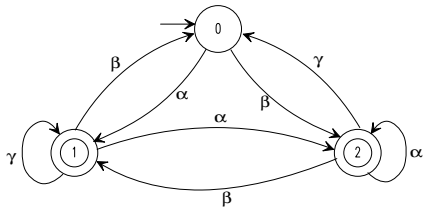

Figura 0.6: Autômato.

pode armazenar uma quantidade máxima de três peças-trabalhadas. O buffer de peças-brutas e a retirada de peças-trabalhadas do buffer *para outro setor da fábrica não são representados nesse exemplo. Os eventos desse sistema são: início de processamento de uma peça-bruta* (α)*, braço robótico retira uma peça-trabalhada na máquina* (β)*, braço robótico coloca uma peça-trabalhada no* buffer (γ) *e recarregamento da máquina com duas peças-brutas* (η)*. Desse modo, os estados do autômato são:*

estado	*máquina*	*braço*	buffer
1	*livre*	*livre*	*vazio*
2	*processando*	*livre*	*vazio*
3	*livre*	*carregando peça*	*vazio*
4	*processando*	*carregando peça*	*vazio*
5	*processando*	*livre*	*1 peça*
6	*livre*	*carregando peça*	*1 peça*
7	*processando*	*carregando peça*	*1 peça*
8	*processando*	*livre*	*2 peças*
9	*livre*	*carregando peça*	*2 peças*
10	*livre*	*livre*	*3 peças*

e o estado inicial é máquina carregada com três peças-brutas estando o braço robótico livre e o buffer *vazio. O estado marcado é o estado* 10*. Este autômato está representado graficamente na Figura 0.7.*

Neste segundo exemplo, a função de transição não é definida para todos os estados para evitar situações absurdas do tipo, *braço robótico retira peça-trabalhada do buffer e braço robótico coloca uma peça-trabalhada na máquina.* De modo geral, quando se utiliza um autômato para representar o comportamento de um SED, é necessário restringir a função de transição, como apresentado na próxima seção.

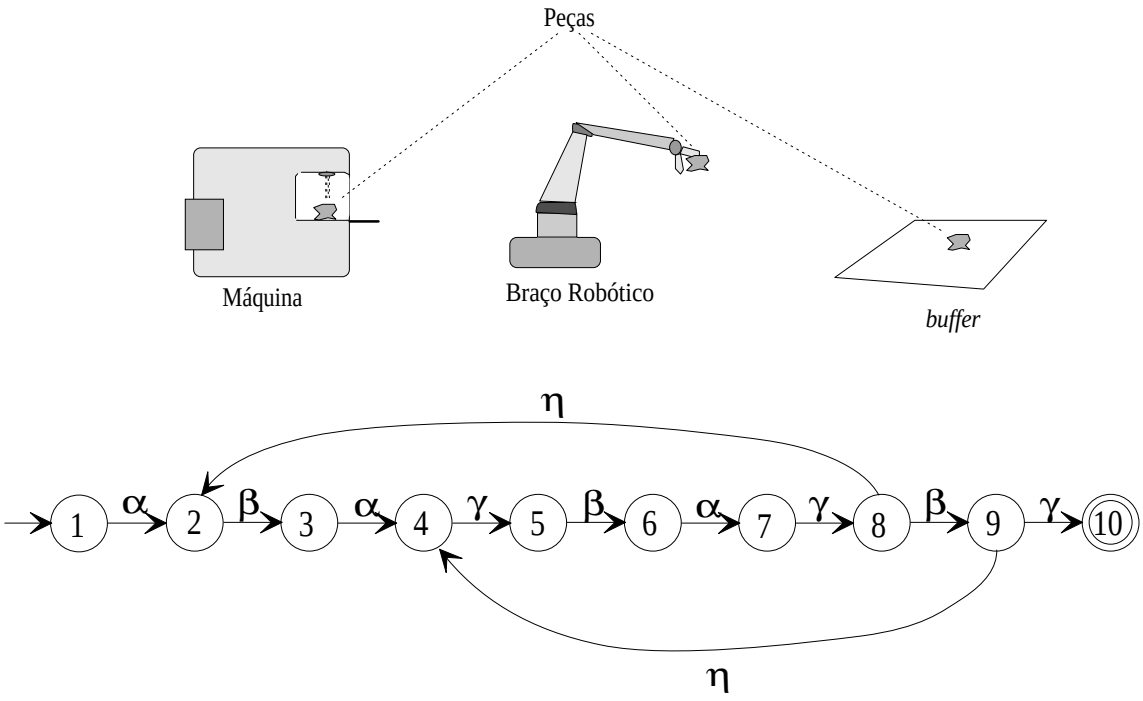

Figura 0.7: Sistema e autômato.

As seqüências de símbolos $\{\beta\alpha, \alpha\gamma\}$ e $\{\alpha\beta\alpha\gamma\beta\alpha\gamma\beta\gamma, \alpha\beta\alpha\gamma\beta\alpha\gamma\beta\eta\gamma\beta\alpha\gamma\beta\gamma\}$ dos exemplos anteriores constituem palavras definidas para o alfabeto do autômato. Isso sugere que a função de transição pode ser estendida e definida para processar seqüências de símbolos.

Definição 3 *Seja um autômato $A = (Q, \Sigma, \delta, q_0, Q_m)$ e seja uma função de transição estendida δ^*, que é a função $\delta^* : \Sigma^* \rightarrow Q$, de tal forma que:*

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

Desde que $\delta^*(\sigma, q) = \delta(\sigma, \delta^*(s, q)) = \delta(\sigma, q)$ para o caso em que $s = \epsilon$, pode-se utilizar δ ao invés de δ^*, por uma questão de conveniência.

Exemplo 14 *Considerando o Exemplo 13 e as palavras $s_1 = \beta\alpha, s_2 = \gamma\beta\alpha\gamma$, o autômato fica representado como visto na Figura 0.8, que simplifica sua representação em termos de estados intermediários (quando estes não têm importância na análise que se deseja realizar). Essa representação simplificada é denominada de grafo de transição.*

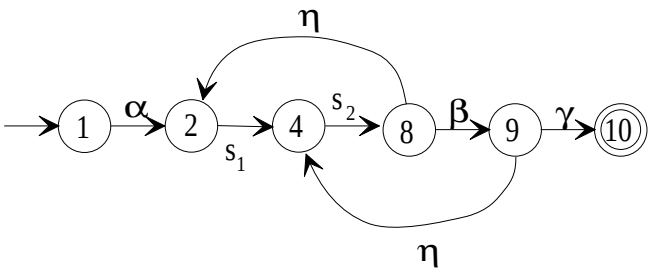

Figura 0.8: Grafo de transição.

Como se vê, conhecida a função de transição e o estado atual do autômato, é possível determinar seu estado após o processamento de um símbolo dado. Assim, processando uma palavra a partir de um dado estado, pode-se confirmar se o estado alcançado pertence ou não ao conjunto de estados marcados Q_m. A linguagem marcada ou reconhecida pelo autômato é definida por

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

ou seja, são palavras, que a partir do estado inicial q_0, levam o autômato a um estado marcado. Esta linguagem pode ser encontrada seguindo os arcos no grafo do mesmo.

Exemplo 15 *Seja $\Sigma = \{\alpha, \beta\}$ um conjunto de eventos (alfabeto) e as linguagens definidas por*

$$\mathcal{L}_\alpha = \{\epsilon, \alpha, \alpha\alpha, \beta\alpha, \alpha\alpha\alpha, \alpha\beta\alpha, \beta\alpha\alpha, \beta\beta\alpha, ...\}$$

e

$$\mathcal{L}_\beta = \{\epsilon, \beta, \beta\beta, \beta\alpha, \beta\beta\beta, \beta\alpha\beta, \alpha\beta\beta, \alpha\alpha\beta, ...\}$$

que consistem de todas as palavras formadas com símbolos de Σ sempre seguidas do evento α e β, respectivamente. Observe que $\mathcal{L}_\alpha \subset L_m(A)$ e $\mathcal{L}_\beta \subset L_m(A)$ sendo $L_m(A)$ a linguagem marcada pelo autômato $A = (Q, \Sigma, \delta, q_0, Q_m)$, apresentado na Figura 0.9, com $Q = \{0, 1\}$, $q_0 = 0$, $Q_m = \{0, 1\}$ e δ é definida através de $\delta(\alpha, 0) = 1$, $\delta(\beta, 0) = 0$, $\delta(\alpha, 1) = 1$, $\delta(\beta, 1) = 0$. Note que nesse caso, δ é uma função completa e portanto, a linguagem gerada pelo autômato A é $L(A) = \Sigma^*$.

Vários autômatos podem reconhecer uma mesma linguagem e podem ter uma mesma linguagem marcada L_m.

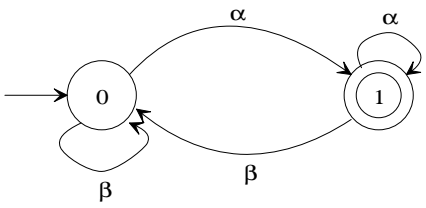

Figura 0.9: Autômato com linguagem marcada L_m=$\{\alpha, \alpha\alpha, \beta\alpha, \alpha\alpha\alpha, \alpha\beta\alpha, \beta\alpha\alpha, \beta\beta\alpha, ...\}$, consistindo de todas as palavras de α e β, seguidas por α, construída através dos símbolos do alfabeto Σ=$\{\alpha, \beta\}$.

Definição 4 *Considere dois autômatos A_1 e A_2. Diz-se que A_1 e A_2 são equivalentes se $L(A_1) = L(A_2)$ e $L_m(A_1) = L_m(A_2)$.*

Exemplo 16 *Os autômatos da Figura 0.10 têm a mesma linguagem gerada*

$$L(A) = \{\epsilon, \alpha, \alpha\alpha, \beta\alpha, \alpha\alpha\alpha, \alpha\beta\alpha, \beta\alpha\alpha, \beta\beta\alpha, ...\}$$

e a mesma linguagem marcada

$$L_m(A) = \{\alpha, \alpha\alpha, \beta\alpha, \alpha\alpha\alpha, \alpha\beta\alpha, \beta\alpha\alpha, \beta\beta\alpha, ...\}$$

do autômato do Exemplo 15. Logo, estes autômatos são equivalentes.

Uma outra consideração a ser feita com relação aos autômatos, é a condição de bloqueio. Diz-se que um autômato é bloqueável se $L_m(A) \subset L(A)$.

Exemplo 17 *O autômato visto na Figura 0.11 tem $L(A) = (\alpha\beta)^* \alpha(\beta + (\kappa + \epsilon))$ e $L_m(A) = (\alpha\beta)^* \alpha$. Este autômato é bloqueável, desde que $L_m(A) \subset L(A)$, Esta condição pode ser observada no estado 3, que é não marcado. Quando o autômato se encontra nesse estado, nele permanece, não podendo mais atingir nenhum estado marcado.*

Geradores

Um gerador é um autômato no qual a função de transição é definida para um subconjunto próprio de Σ^*. Os geradores são mais compatíveis com a representação de SEDs.

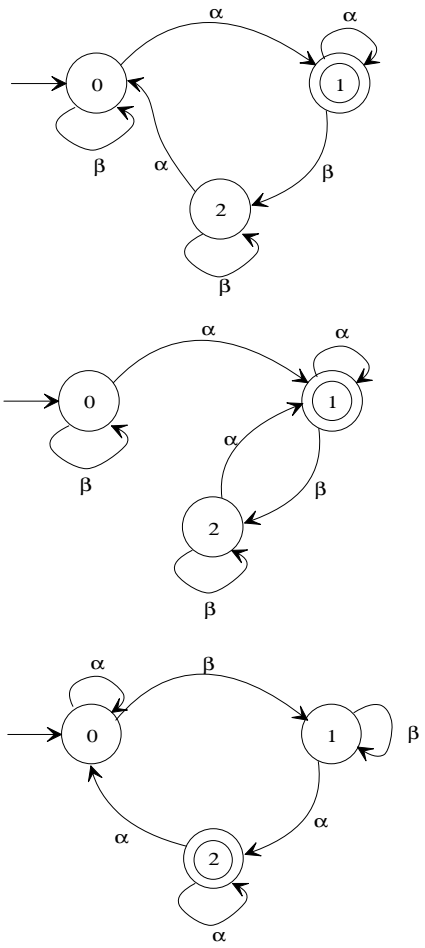

Figura 0.10: Autômatos com linguagem L=$\{\epsilon$, α, $\alpha\alpha$, $\beta\alpha$, $\alpha\alpha\alpha$, $\alpha\beta\alpha$, $\beta\alpha\alpha$, $\beta\beta\alpha$, ...$\}$ e linguagem marcada L_m=$\{\alpha$, $\alpha\alpha$, $\beta\alpha$, $\alpha\alpha\alpha$, $\alpha\beta\alpha$, $\beta\alpha\alpha$, $\beta\beta\alpha$, ...$\}$, construída através dos símbolos do alfabeto Σ=$\{\alpha$, $\beta\}$.

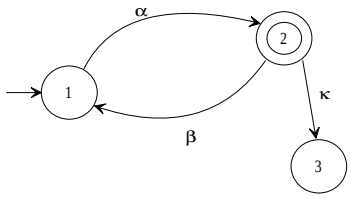

Figura 0.11: Autômato com bloqueio.

Definição 5 *Um gerador é uma quíntupla* $G = (Q, \Sigma, \delta, q_0, Q_m)$*, onde os elementos* Q, Σ, q_0, Q_m, *e* δ *têm a mesma definição do autômato 2, com* δ *definido como a função, geralmente parcial, de transição de estados.*

A diferença existente entre os autômatos e os geradores é que, no caso dos autômatos, a função de transição não pode ser parcial (definida apenas para um subconjunto de eventos para cada estado do gerador).

Semelhantemente aos autômatos, tem-se o conjunto de eventos, a função de transição estendida, sua linguagem e sua linguagem marcada, como com as seguintes definições:

Definição 6 *Para um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$*, associa-se a cada estado* $q \in Q$ *o conjunto de eventos definidos* $\Sigma(q)$*, definido por:*

$$\Sigma(q) = \{\sigma : \sigma \in \Sigma \wedge \delta(\sigma, q)!\} \tag{0.1}$$

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

Definição 7 *Seja um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$*. Sua função de transição estendida, denotada por* δ^**, é uma função*

$$\delta^* : \Sigma^* \times Q \to Q$$

tal que

$$\delta^*(\epsilon, q) = q \text{ e } \delta^*(s\sigma, q') = \delta(\sigma, \delta^*(s, q)) \tag{0.2}$$

$$\forall\ q \in Q, s \in \Sigma^*, q' = \delta^*(s, q) \text{ e } \delta^*(\sigma, q')! \tag{0.3}$$

Definição 8 *Seja um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$*. Sua linguagem gerada* $L(G)$ *é:*

$$L(G) = \{s : s \in \Sigma^* \wedge \delta(s, q_0)!\}. \tag{0.4}$$

A função de transição estendida é, de modo geral, representada por δ, ao invés de δ^* pelas mesmas razões dos autômatos, ou seja, por conveniência.

Também é observado que a linguagem gerada *L(G)* por um gerador é *prefixo-fechada*, isto é, $\forall G = (Q, \Sigma, \delta, q_0, Q_m), L(G) = \overline{L(G)}$.

Analogamente aos autômatos, a linguagem marcada do gerador é definida como a seguir:

Definição 9 *Dado um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$, *a linguagem marcada de* G, *denotada por* $L_m(G)$, *é* $L_m(G) = \{s : s \in \Sigma^* \wedge \delta(s, q_0) \in Q_m\}$.

Como o comportamento de um SED é caracterizado pela ocorrência de eventos, isto é, um SED gera palavras de comprimento crescente, à medida que evolui, e de acordo com o alfabeto gerado pelo SED, é possível encontrar seqüências de símbolos, isto é, palavras que não representam seqüências de eventos fisicamente possíveis. Assim, a linguagem gerada pelo sistema é um subconjunto próprio de Σ^*. Desta forma, esta linguagem gerada inclui, para cada palavra, todos os seus prefixos.

A linguagem *prefixo-fechada* representa o comportamento lógico de um SED, em que não ocorrem eventos simultâneos. A mesma é definida formalmente, da seguinte maneira:

Definição 10 *A linguagem* prefixo-fechada *que representa o comportamento lógico de um SED é denominada de* linguagem gerada do sistema;

Definição 11 *A linguagem que representa o conjunto de todas as tarefas que podem ser executadas por um SED, é denominada* linguagem marcada do sistema.

Considerando que L é a linguagem gerada por um SED e que L_m, seja sua linguagem marcada, de acordo com estas Definições, tem-se que a linguagem marcada L_m contém as palavras geradas pelo SED que também gera todos os seus prefixos, ou seja, um SED produz as palavras contidas em $\overline{L_m}$. Dessa forma, a linguagem marcada do SED não é necessariamente *prefixo-fechada*. Assim, é visto que $L_m \subseteq \overline{L_m} \subseteq L = \overline{L}$.

Para representar o comportamento de um SED através de um autômato, é necessário restringir sua função de transição, definindo-a apenas para alguns pares (*evento*, *estado*) do conjunto $\Sigma \times Q$. Essa restrição é inerente na definição do gerador e desse modo, é possível representar as linguagens gerada L e marcada L_m dos SEDs. Assim, um SED com linguagem gerada L e linguagem marcada L_m, é representado por um gerador, de tal forma que $L(G) = L$ e $L_m(G) = L_m$.

Semelhantemente ao autômatos, é necessário determinar se os estados do gerador são alcançáveis e se ele reconhece alguma palavra. Para isto, definem-se a acessibilidade e a coacessibilidade dos geradores.

Definição 12 *A componente acessível de um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$*, denotada por* $A_c(G)$ *é:*

$$A_c(G) = (Q_{ac}, \Sigma, \delta_{ac}, q_0, Q_{ac,m}), \quad onde$$
$$Q_{ac} = \{q : \exists s \in \Sigma^* \wedge \delta(s, q_0) = q\};$$
$$Q_{ac,m} = Q_{ac} \cap Q_m;$$
$$\delta_{ac} = \delta \mid (\Sigma \times Q_{ac}).$$

Nesta Definição, δ_{ac} é a função δ restrita ao domínio $\Sigma \times Q_{ac}$. Assim, um gerador G é dito acessível quando $G = A_c(G)$. A componente acessível de um gerador é a garantia da existência de palavras que o mesmo pode processar a partir do estado inicial, embora que nenhuma seja reconhecida, isto é, marcada.

Definição 13 *A componente coacessível de um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$*, denotada por* $Co(G)$ *é:*

$$Co(G) = (Q_{co}, \Sigma, \delta_{co}, q_0, Q_{co,m}), \quad onde$$
$$Q_{co} = \{q : q \in Q \wedge \exists\, s \in \Sigma^* \wedge \delta(s, q) = q_m\};$$
$$Q_{co,m} = Q_{co} \cap Q_m;$$
$$\delta_{co} = \delta \mid (\Sigma \times Q_{co}).$$

A componente coacessível de um gerador garante a existência de estados reconhecidos a partir de qualquer estado do autômato.

Definição 14 *Dado um gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$*, este será coacessível se e somente se, toda palavra em* $L(G)$ *for um prefixo de uma palavra em* $L_m(G)$*, isto é:*

$$L(G) \subseteq \overline{L_m(G)}. \tag{0.5}$$

Disto, vê-se que em um gerador coacessível, existe pelo menos uma seqüência de eventos que o leva a um estado marcado.

Os geradores que são, ao mesmo tempo, acessíveis e coacessíveis, são denominados ajustados ou *trim*.

Exemplo 18 *Na Figura 0.12(a), é visto um autômato G. Este gerador não é acessível devido ao fato de que o estado 6 não é alcançável a partir do estado 0. Na Figura 0.12(b), encontra-se um gerador semelhante mas sem o estado 6. Neste caso, o gerador*

é acessível pois todos os estados são alcançados. Por outro lado, para encontrar o gerador coacessível, é necessário identificar os estados de G que não são coacessíveis a partir do estado marcado 2. Estes estados são: 3, 4 e 5. Retirando esses estados e suas respectivas transições, o gerador resultante torna-se coacessível. Note que o estado 6 não é retirado, pois ele não é alcançável a partir do estado marcado 2. Este gerador coacessível é visto na Figura 0.12(c). Finalmente, o gerador trim*, está apresentado na Figura 0.12(d), pois a ordem em que as operações de acessibilidade e de coacessibilidade são tomadas não afetam o resultado final.*

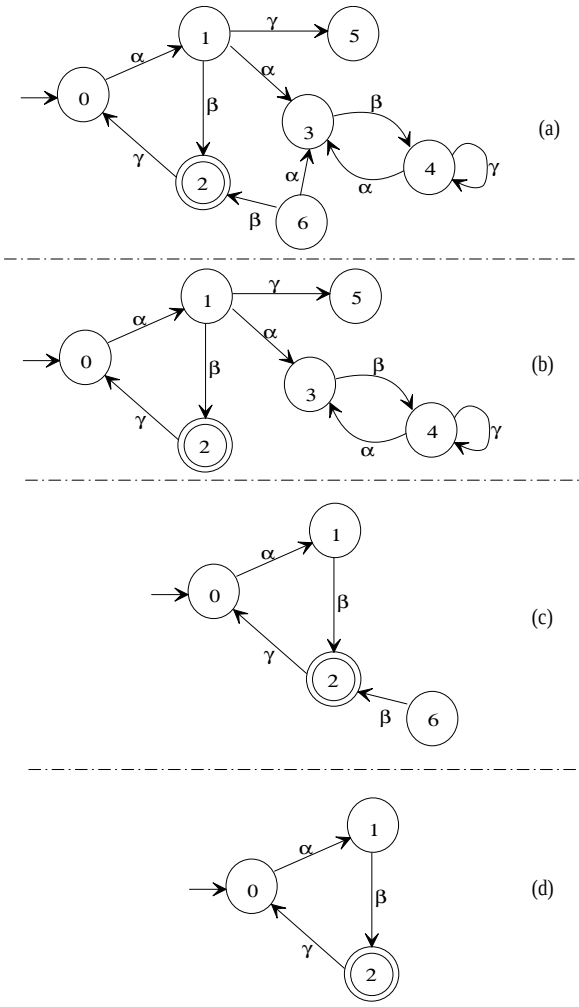

Figura 0.12: Gerador (a) e geradores acessível (b), coacessível (c) e *trim* (d).

Composição de autômatos

Uma das alternativas de modelagem de SEDs requer a decomposição do sistemas em sub-sistemas e, para cada sub-sistema é definido um autômato que representa seu comportamento dinâmico. No entanto, para analisar o sistema completo é necessário remontar essa decomposição para simplificar o estudo. Neste caso, é necessário compôr os autômatos que representam os sub-sistemas para obter um autômato do sistema. Há duas operações de composição que podem ser realizadas com autômatos finitos: a composição por produto, representada por $\times$ e a composição paralela, representada por $||$. A composição paralela é denominada de composição síncrona e a composição por produto é denominada de composição completamente síncrona.

Definição 15 *Sejam* $A_1 = (Q_1, \Sigma_1, \delta_1, q_{0_1}, Q_{m_1})$ *e* $A_2 = (Q_2, \Sigma_2, \delta_2, q_{0_2}, Q_{m_2})$ *dois autômatos. O autômato* $A_3 = A_1 \times A_2$ *resultante da composição por produto é definido por* $A_3 := (Q_3, \Sigma_3, \delta, q_{0_3}, Q_{m_3})$ *com* $Q_3 = Q_1 \times Q_2$, $\Sigma_3 = \Sigma_1 \cup \Sigma_2$, $q_{0_3} = (q_{0_1}, q_{0_2})$, $Q_{m_3} = Q_{m_1} \times Q_{m_2}$ *e*

$$\delta_3(\sigma,(q_{i_1},q_{i_2}))=\begin{cases}(\delta_1(\sigma,q_{i_1}),\delta_2(\sigma,q_{i_2}))=(q_{i'_1},q_{i'_2}) & \text{se } \exists\sigma,\delta_1(\sigma,q_{i_1})=q_{i'_1} \text{ e } \delta_2(\sigma,q_{i_2})=q_{i'_2}\\ (\delta_1(\sigma,q_{i_1}),q_{i_2})=(q_{i'_1},q_{i_2}) & \text{se } \exists\sigma,\delta_1(\sigma,q_{i_1})=q_{i'_1} \text{ apenas em } A_1\\ (q_{i_1},\delta_2(\sigma,q_{i_2}))=(q_{i_1},q_{i'_2}) & \text{se } \exists\sigma,\delta_2(\sigma,q_{i_2})=q_{i'_2} \text{ apenas em } A_2\\ \text{indefinido} & \text{caso contrário}\end{cases}$$

Exemplo 19 *O produto dos autômatos* A_1 *e* A_2*, apresentados da Figura 0.13*(a) *e 0.13*(b)*, respectivamente, onde os elementos de* Σ_1 *são os mesmos elementos de* Σ_2*, está apresentado na Figura 0.13*(c)*. Pode-se ver que nos dois autômatos* A_1 *e* A_2*, só se encontra um único arco* (α) *que sai do estado* x *para o estado* x*, em* A_1 *e que, equivalentemente, sai do estado* 0 *para o estado* 1 *no autômato* A_2*. Por outro lado, quando o autômato* A_2 *encontra-se no estado* 1*, só há um arco saindo do mesmo, que é equivalente no autômato* A_1*, que também é o arco* α*. Também, o estado* $(x, 0)$ *do autômato construído pelo produto de* A_1 *e* A_2*, não é marcado porque o estado inicial do autômato* A_2*, não é. Porém, o estado* $(x, 1)$ *do autômato construído pelo produto de* A_1 *e* A_2 *é marcado, pois em ambos os autômatos, a função de transição* α*, leva de um estado marcado para outro estado marcado.*

Exemplo 20 *A composição entre os autômatos* A_1 *e* A_2*, apresentados da Figura*

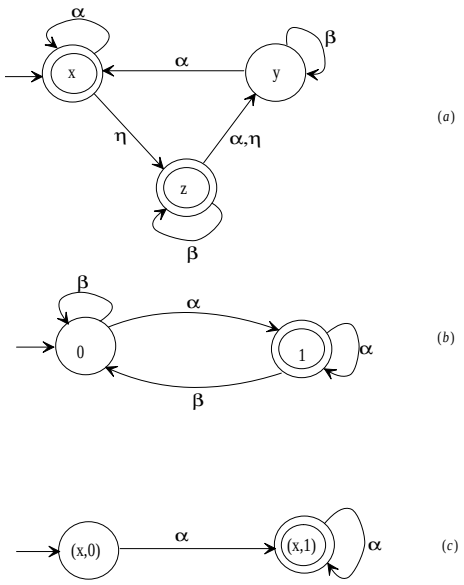

Figura 0.13: Autômato A_1 (a), autômato A_2 (b) e produto $A_1 \times A_2$ (c).

0.14(a) e 0.14(b), respectivamente, onde somente o elemento $\eta \in \Sigma_1 \cap \Sigma_2$*, está apresentado na Figura 0.14(c).*

Quando $\Sigma_1 \cap \Sigma_2 = \emptyset$, o procedimento para a construção da composição síncrona de dois autômatos A_1 e A_2 define um autômato que gera ambas linguagens $L(A_1)$ e $L(A_2)$, independentemente. Esta operação é conhecida como composição *shuffle*, e é formalizada de acordo com a Definição 15 de produto síncrono, onde a função de transição δ_3 só é definida para os dois últimos casos, ou seja,

$$\delta_3(\sigma,(q_{i_1},q_{i_2}))=\begin{cases} (\delta_1(\sigma,q_{i_1}),q_{i_2})=(q_{i'_1},q_{i_2}) & \text{se } \exists\sigma,\delta_1(\sigma,q_{i_1})=q_{i'_1} \text{ apenas em } A_1 \\ (q_{i_1},\delta_2(\sigma,q_{i_2}))=(q_{i_1},q_{i'_2}) & \text{se } \exists\sigma,\delta_2(\sigma,q_{i_2})=q_{i'_2} \text{ apenas em } A_2. \end{cases}$$

Exemplo 21 *A composição* $shuffle$ *dos autômatos* A_1 *e* A_2*, apresentados nas Figuras 0.15(a) e 0.15(b), respectivamente, onde todos os elementos de* Σ_1 *são diferentes dos elementos de* Σ_2*, está apresentado na Figura 0.15(c). Pode-se ver que esta composição se apresenta com os dois autômatos* A_1 *e* A_2 *produzindo suas linguagens independentemente.*

O produto de autômatos, é uma das operações utilizada na Teoria de Controle Supervisório para fundamentar a composição entre o supervisor e o gerador, determinando assim, o acoplamento que define o sistema supervisionado.

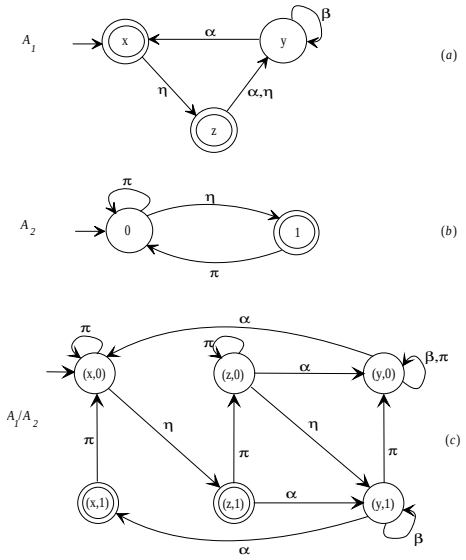

Figura 0.14: Autômato A_1 (a), autômato A_2 (b) e produto $A_1 \times A_2$ (c).

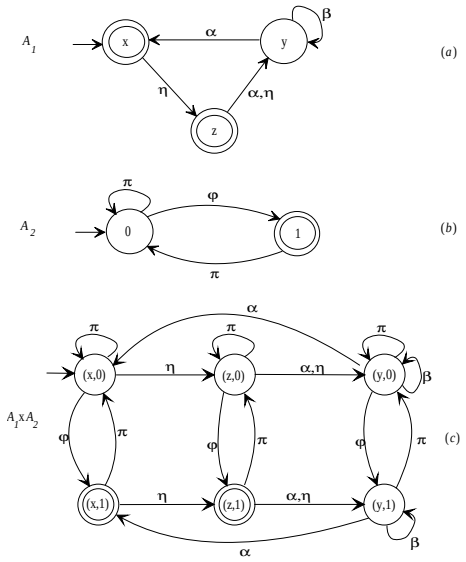

Figura 0.15: Autômato A_1 (a), autômato A_2 (b) e produto composição $shuffle$ de A_1 e A_2 (c).

Redes de Petri

Rede de Petri é uma ferramenta utilizada para modelar e analisar sistemas que apresentem concorrências e conflitos em suas evoluções dinâmicas. Graficamente, essas redes são representadas por grafos direcionados, bi-partidos e ponderados com uma marcação inicial. O termo bi-partido define que uma rede de Petri tem dois tipos de nós, que são denominados lugares e transições. Os arcos são direcionados sempre de um lugar para uma transição e de uma transição para um lugar. No primeiro caso, denominam-se lugares de entrada da transição e transições de saída do lugar, e vice-versa, no segundo caso.

A representação gráfica dos lugares em uma rede de Petri são círculos, e as transições são barras ou retângulos. Um arco com peso k representa k arcos paralelos ligando dois nós. A marcação de uma rede de Petri atribui a cada lugar, um número inteiro não negativo m, de onde se diz que o lugar com marcação m tem m fichas, as quais são representadas por pequenos círculos pretos. A marcação é designada por um vetor $M = [m_1, m_2, m_3, ..., m_k]^T$ onde k é o número de lugares da rede, e m_i o número de fichas no lugar i.

As redes de Petri podem ser de capacidade infinita (cada lugar pode conter um número ilimitado de fichas) ou de capacidade finita (número de fichas limitado para cada lugar).

Estas redes aqui tratadas, são também conhecidas por redes de Petri Lugar/Transição (RP L/Tr), as quais são definidas da seguinte maneira:

Definição 16 *Uma rede de Petri é uma sextupla* $RP = (P, T, A_r, K, W, M_0)$*, onde:*

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

Denomina-se estrutura de rede de Petri, uma rede sem marcação inicial e a mesma é denotada por $E_{rp} = (P, T, A_r, W)$. Assim, a notação de uma rede de Petri pode ser abreviada por $RP = \{E_{rp}, K, M_0\}$. Se a capacidade da rede é não limitada, então a mesma é denotada por $RP = \{E_{rp}, M_0\}$.

Uma rede de Petri também pode ser representada matricialmente, utilizando um vetor de marcação e duas matrizes *Pre* e *Post*, que são definidas por:

- $Pre(\bullet, t)$ e $Post(\bullet, t)$ são as colunas associadas a uma transição t e,
- $Pre(p, \bullet)$ e $Post(p, \bullet)$ são as linhas associadas a um lugar p.

Exemplo 22 *A rede de Petri apresentada na Figura 0.16, tem a seguinte representação matricial:*

$P = \{p_1, p_2, p_3, p_4\}$
$T = \{t_1, t_2, t_3\}$

Pre	t_1	t_2	t_3
p_1	1	0	0
p_2	0	1	0
p_3	0	0	1
p_4	1	0	0

Post	t_1	t_2	t_3
p_1	0	0	2
p_2	1	0	0
p_3	0	1	0
p_4	0	2	0

$M_0 = \begin{bmatrix} 1 & 0 & 0 & 2 \end{bmatrix}^T$.

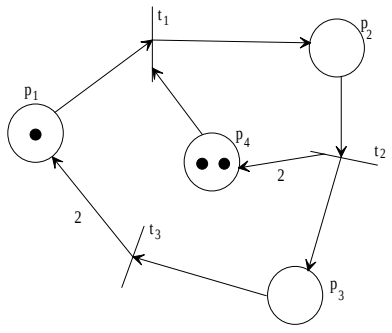

Figura 0.16: Rede de Petri utilizada no Exemplo das matrizes *Pre* e *Post*.

Seguindo as linhas da matriz *Pre*, tem-se os arcos que saem de cada lugar. As linhas da matriz *Post*, determinam os arcos que entram em cada lugar. A partir destas matrizes, é construída a rede de Petri utilizando o seguinte procedimento:

Definição 17 *O grafo* $G = \{P, T, \Gamma, V\}$ *associado à rede* $RP = \{P, T;\text{Pre},\text{Post}\}$ *é definido por:*

- $\forall p \in P$, $\Gamma(p) = \{t \in T | \text{Pre}(p,t) > 0\}$;
- $\forall t \in T$, $\Gamma(t) = \{p \in P | \text{Post}(p,t) > 0\}$;
- $\forall p \in P$, $\forall t \in T$, $V(p,t) = \text{Pre}(p,t)$ *e* $V(t,p) = \text{Post}(p,t)$.

Também pode-se utilizar a notação $t^\bullet = \Gamma(t)$ e $p^\bullet = \Gamma(p)$.

Exemplo 23 *A rede de Petri mostrada na Figura 0.16, tem seu grafo definido por:*

$$\Gamma(p_1) = \{t_1\}, \Gamma(p_2) = \{t_2\}, \Gamma(p_3) = \{t_3\}, \Gamma(p_4) = \{t_1\},$$
$$\Gamma(t_1) = \{p_2\}, \Gamma(t_2) = \{p_3, p_4\}, \Gamma(t_3) = \{p_1\}.$$

A evolução dinâmica de uma rede de Petri é determinada pelos disparos das transições habilitadas em cada marcação alcançada.

Definição 18 *A mudança da marcação da rede de Petri segue as seguintes regras de disparo das transições:*

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

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

2. *Uma transição habilitada pode ou não disparar;*

3. *O disparo de uma transição* $t \in T$*, habilitada na marcação* M*, é instantânea e resulta em uma nova marcação* M' *da rede de Petri dada pela equação:*

$$M'(p) = M(p) - W(p,t) + W(t,p), \forall p \in P; \tag{0.6}$$

4. *A ocorrência do disparo de t, que modifica a marcação M da rede para uma nova marcação M', é denotada por $M[t > M'$, ou $M' = \delta(t, M)$, para estabelecer uma analogia com a função de transição de estados dos autômatos.*

Exemplo 24 *Na Figura 0.17, é visto um exemplo de uma rede de Petri com marcação inicial $M_0 = [2\ 0\ 0]^T$, e sua nova marcação, após o disparo da transição t. Observa-se que após o disparo de t, a nova marcação é $M_1 = [1\ 2\ 1]^T$ e, que esta marcação está de acordo com a regra 3 de disparo das transições.*

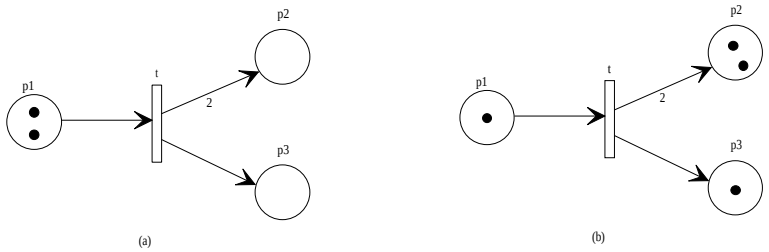

Figura 0.17: Exemplo de uma rede de Petri antes e após o disparo de sua transição.

Os disparos de transições em uma rede de Petri podem continuar indefinidamente, desde que haja, pelo menos uma transição habilitada em cada marcação atingida.

Definição 19 *$MA(E_{rp}, M_0)$ é o conjunto de marcações alcançáveis de uma rede de Petri a partir de M_0, tal que $M \in MA(E_{rp}, M_0)$ e, se $M_1 \in MA(E_{rp}, M_0)$ é para uma dada transição $t \in T$, $M_1[t > M_2$, então $M_2 \in MA(E_{rp}, M_0)$.*

Esta Definição indica que toda marcação alcançada a partir do disparo de uma transição t, também pertence ao conjunto de marcações alcançáveis da rede de Petri.

Quando é executada uma seqüência de disparos de transições, encontra-se uma seqüência de marcações atingidas. Com isto, é possível mapear todas as marcações alcançadas na rede de Petri. Esta estrutura de marcações alcançadas é denominada de Árvore de Alcançabilidade. Esta forma de análise permite avaliar mudanças de marcações seguindo uma determinada seqüência e é útil na verificação do comportamento da rede, pois permite determinar se existem bloqueios (i.e., quando não há nenhuma transição habilitada numa determinada marcação) ou outras situações não desejadas, como explosão de estados (i.e., quando o número de fichas num determinado lugar cresce sem limite).

Para construir a árvore de alcançabilidade, segue-se o seguinte algoritmo:

Algoritmo 1 *Algoritmo da Árvore de Alcançabilidade*

- *Início*

Passo 1 *Rotule a marcação inicial M_0 de raiz e etiquete-a com* nova*;*

Passo 2 *Enquanto houverem marcações* nova*, faça o seguinte:*

1. *Escolha uma nova marcação M;*
2. *Se M for idêntica a outra marcação no caminho da raiz até M, etiquete-a com* antiga *e vá para uma outra marcação;*
3. *Se nenhuma transição estiver habilitada em M, etiquete-a como* bloqueada*;*
4. *Enquanto existirem transições habilitadas em M, para cada transição habilitada, faça o seguinte:*

   i) *Obtenha a marcação M' que resulta do disparo da transição t em M;*

   ii) *Se no caminho da raiz até M' existir uma marcação M'' tal que $M'(p) \geq M''(p)$ para cada lugar p e $M' \neq M''$, então substitua $M'(p)$ por ω para cada lugar p tal que $M'(p) > M''(p)$;*

   iii) *Introduza M' como um nó, desenhe um arco com rótulo t, de M para M' e etiquete M' com* nova*;*

- *Fim*

Exemplo 25 *Considere a rede de Petri apresentada na Figura 0.18. Seguindo o algoritmo da árvore de alcançabilidade, constrói-se a árvore apresentada na Figura 0.19.*

A árvore de alcançabilidade é um dos métodos mais utilizados para análise de redes de Petri. Todavia, existem também os métodos baseados em propriedades da matriz de incidência e da equação de estado.

Esses métodos permitem descrever a evolução da rede utilizando uma notação matricial. Observando que as colunas da matriz *Pre* definem quantas fichas devem ser retiradas de cada lugar e, as colunas da matriz *Post*, quantas fichas devem ser colocadas em cada lugar, define-se a matriz de incidência como sendo uma matriz de inteiros $n \times m$ (n transições e m lugares) $A = [a_{ij}]$, com $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$, onde:

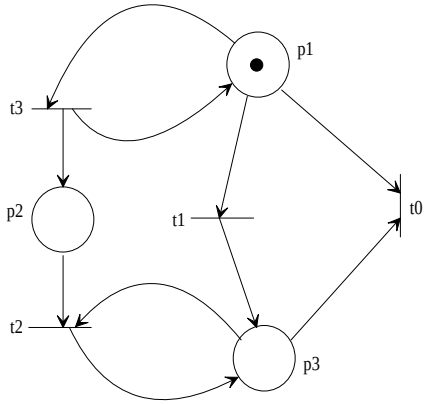

Figura 0.18: Rede de Petri utilizada para a construção da árvore de alcançabilidade.

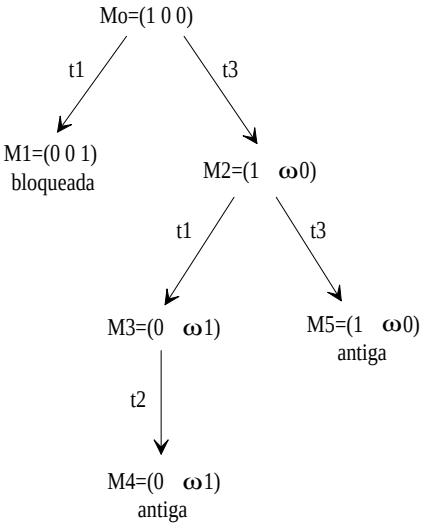

Figura 0.19: Árvore de alcançabilidade da rede de Petri.

- $a_{ij}^{+} = w(i,j)$ é o peso do arco que vai da transição i para o lugar de saída j e representa o número de fichas acrescentadas ao lugar j quando a transição i dispara;

- $a_{ij}^{-} = w(j,i)$ é o peso do arco que vai do lugar j para a transição de saída i e representa o número de fichas removidas do lugar j quando a transição i dispara.

Observa-se que, a transposta da matriz de incidência A é igual a subtração da matriz *Pre* da matriz *Post*, ou seja, $A^T = Post - Pre$.

A equação de estados determina uma nova marcação da rede quando há uma seqüência de disparos, através da matriz de incidência e do vetor de marcação. A mesma é dada por

$$M' = M + A^T x,$$

com $k = 1, 2, ...$ e x sendo um vetor coluna $n \times 1$, representando os disparos das transições.

Exemplo 26 *Para a rede de Petri ilustrada na Figura 0.20, sua nova marcação* $M = [1\ 2\ 0\ 6]^T$ *é alcançada a partir de* M_0*, que pode ser determinada pela equação de* $M' = M + A^T x$*, ou seja,*

$$M' = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix},$$

onde $x = [3\ 1\ 4]^T$ *representa a seqüência de disparos* $t_3t_1t_2t_3t_1t_3t_1t_3$*, ou seja, três disparos de* t_1*, um disparo de* t_2 *e quatro disparos de* t_3*.*

Uma seqüência de disparos, pode ser representada por uma linguagem formal, se as transições forem etiquetadas com símbolos de um alfabeto. Isto será visto na próxima seção.

Linguagens de redes de Petri

A linguagem gerada por uma rede de Petri é definida através dos disparos de transições que são etiquetadas com símbolos de um dado alfabeto. As transições da rede

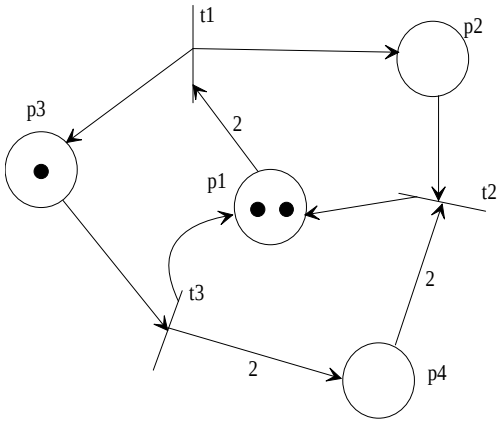

Figura 0.20: Rede de Petri.

são etiquetadas com os símbolos de um alfabeto, através da função de etiquetagem h definida como $h : T \rightarrow \Sigma$.

Exemplo 27 *Seja a rede de Petri mostrada na Figura 0.21. As transições estão etiquetadas por* $h(t_1) = \alpha$ *e* $h(t_2) = \beta$*. Logo, como se pode ver, a seqüência de disparos das transições é* $\tau = t_1t_2t_1t_2...$*, que através da função de etiquetagem* $h(\tau)$*, transforma-a na linguagem* $h(\tau) = \alpha\beta\alpha\beta...$*, que pode ser abreviada por* $(\alpha\beta)^*$*, que é a linguagem gerada pela rede de Petri.*

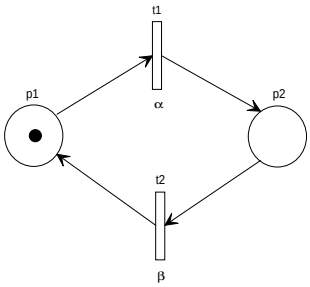

Figura 0.21: Exemplo de uma rede de Petri que gera a linguagem $(\alpha\beta)^*$.

Redes de Petri com função de habilitação de transição

Uma das formas de controlar a evolução de uma rede de Petri, é definir restrições para algumas de suas transições. Assim, uma determinada transição só estará habilitada se

todos os parâmetros de uma função associada a ela, forem satisfeitos. Dessa maneira, embora os lugares de entrada da transição tenham os números de fichas exigidos pelos pesos dos arcos, ela não estará habilitada se a marcação da rede não satisfizer à função de habilitação. A definição, desta rede é apresentada a seguir.

Definição 20 *Uma rede de Petri com função de habilitação de transição, é uma quádrupla* $RPFHT = (E_{rp}, l, K, M_0, \Phi)$*, em que:*

- $E_{rp} = (P, T, A_r, W)$ *é uma estrutura de rede de Petri;*
- $l : T \rightarrow \Sigma$*, é a função que etiqueta as transições, onde* Σ *é um alfabeto;*
- $K : P \rightarrow \mathbb{N} \cup \{\infty\}$ *é a função de capacidade;*
- M_0 *é a marcação inicial, como definido para as redes de Petri;*
- $\Phi = \{\phi_1, ..., \phi_m\} : MA(E_{rp}, M_0) \rightarrow \{0, 1\}$ *é a função de habilitação das transições, que mapeia o conjunto de marcações alcançáveis em 0 ou 1;*

Devido à função de habilitação, uma transição t_j com função de habilitação ϕ_j estará desabilitada numa marcação $M_i \in MA(E_{rp}, M_0)$ se $\phi_j(M_i) = 0$. Se $\phi_j(M_i) = 1$, então t_j estará habilitada e sujeitas às condições impostas pelas regras de disparo das transições, como em uma rede de Petri L/Tr. Assim, sua regra de disparo é definida como a seguir:

Definição 21 *O estado, ou marcação, de uma RPFHT varia de acordo com a seguinte regra de disparo das transições:*

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

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

2. *Uma transição habilitada pode ou não disparar;*

3. *O disparo de uma transição $t_j \in T$, habilitada na marcação M, é instantânea e resulta em uma nova marcação M' dada pela equação:*

$$M'(p) = M(p) - W(p, t_j) + W(t_j, p), \forall p \in P; \tag{0.7}$$

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

As RPFHTs têm funções lógicas associadas às suas transições, as quais criam condições específicas para habilitação, de acordo com o número de fichas em determinados lugares da rede (ou seja, de acordo com a marcação da rede), para sua evolução.

Para as RPFHTs, pode-se utilizar os mesmos métodos de análise das Redes de Petri L/Tr. Contudo, na construção da árvore de alcançabilidade, alguns ramos podem ser eliminados quando se consideram as funções de habilitação. No caso da equação de estado, é necessário avaliar cada passo, devido às funções das transições que impõem a impossibilidade de passagem por certas marcações.

Exemplo 28 *Na Figura 0.22 é visto um exemplo de RPFHT, onde a transição t tem a função de habilitação dada por*

$$\phi = [M(p_1) \geq 2 \wedge M(p_2) > 0 \wedge M(p_3) = 0].$$

Nela vê-se que pela função de habilitação ϕ, no caso da Figura 0.22(a), a transição t está habilitada podendo disparar, visto que todos os parâmetros são verdadeiros, o que torna a função $\phi = 1$. Após o disparo, Figura 0.22(b), os dois primeiros parâmetros da função são verdadeiros, contudo o terceiro $M(p_3) \neq 0$, o que torna a função $\phi = 0$, desabilitando a transição t.

A teoria de controle supervisório

A Teoria de Controle Supervisório (TCS) utiliza como paradigma de modelagem dos SEDs as linguagens formais e os autômatos. A idéia dessa teoria é de modelar o SED por um gerador e sua linguagem e, a partir daí, especificar um comportamento através

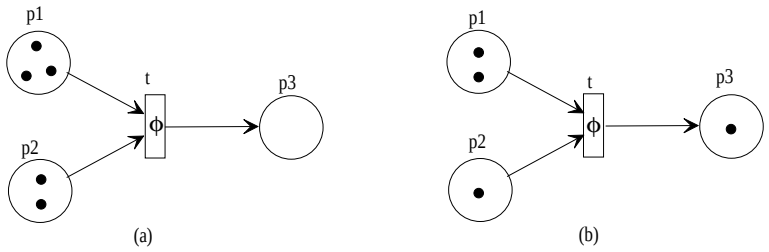

Figura 0.22: Exemplo de uma RPFHT.

de uma linguagem formal, avaliando as possibilidades desta fundamentar um outro autômato (supervisor) que possa realizar esta tarefa requerida, quando acoplado ao gerador que modela o SED. Esta teoria também é conhecida como modelo ou abordagem **R-W** (Ramadge e Wonham [1, 3, 4, 5]).

Para estudar a a TCS é necessário entender as noções de controle e síntese do supervisor, com suas condições de existência.

Os geradores no modelo R-W

Como visto anteriormente, os geradores representam um SED desde que sua linguagem gerada L e sua linguagem marcada L_m sejam tais que, $L(G) = L$ e $L_m(G) = L_m$. Contudo, dada uma linguagem $L \subseteq \Sigma^*$, nem sempre existe um gerador finito tal que $L(G) = L$. Uma das causas disto decorre da teoria de linguagens formais, que define que a classe de linguagens representáveis por autômatos determinísticos finitos é exatamente a classe de linguagens regulares. Como os resultados estabelecidos na abordagem **R-W** são formulados em termos da teoria de linguagens formais, o modelo **R-W** não é limitado à classe dos SEDs representáveis por geradores finitos, embora seus resultados úteis, geralmente o sejam. Devido a isto, na maioria dos casos, tratam-se de sistemas representáveis por geradores finitos.

Exemplo 29 *A Figura 0.23 mostra um gerador G que modela uma máquina com três estados: R (em repouso), A (em atividade) e M (em manutenção). Há quatro transições possíveis, identificadas pelos eventos do alfabeto $\Sigma = \{\alpha, \beta, \lambda, \mu\}$. O modelo permite concluir que a máquina parte do estado de repouso, de onde 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). 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:

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

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

$$L_m\left(G\right) = \left(\alpha\beta + \alpha\lambda\mu\right)^{*}.$$

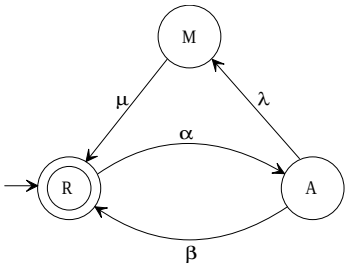

Figura 0.23: Gerador representando uma máquina com três estados.

Assim, vê-se que $L\left(G\right)$ é interpretada como uma representação do comportamento fisicamente possível do sistema e $L_m\left(G\right)$ como uma representação do conjunto de tarefas que o mesmo é capaz de executar. Logo, isto nos permite utilizar a coacessibilidade de um gerador como critério para a ausência de bloqueio, ou seja, num gerador coacessível $L\left(G\right) = \overline{L_m\left(G\right)}$, significando que toda palavra gerada é prefixo de alguma palavra marcada. Então, toda seqüência de eventos fisicamente possível tem pelo menos uma continuação que leva a uma tarefa completa. Assim, nesta interpretação, tem-se que um gerador coacessível e o sistema por ele representado são ditos *não bloqueantes*.

Geradores com Entradas de Controle

Na TCS, é considerado que em um dado gerador $G = (Q, \Sigma, \delta, q_0, Q_m)$ o alfabeto gerado Σ é dividido em dois subconjuntos: Σ_c e Σ_{uc}. O subconjunto de eventos $\Sigma_c \subseteq \Sigma$, cujos elementos são denominados eventos controláveis, são os eventos que podem ser inibidos ou impedidos de ocorrer através de um agente externo. Os eventos que

formam o conjunto Σ_{uc} são denominados eventos não controláveis, os quais estão sempre habilitados, não podendo sofrer ação de controle.

As relações válidas a estes conjuntos são:

$$\Sigma = \Sigma_{uc} \cup \Sigma_c \qquad \text{e} \qquad \Sigma_{uc} \cap \Sigma_c = \emptyset.$$

Com esta partição de Σ, podem-se descrever características de sistemas físicos. Assim, o início da operação de uma máquina e o envio de uma mensagem num sistema de comunicação são eventos controláveis, enquanto que uma pane, a perda de uma mensagem, ou o término de operação de uma máquina são 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. Desta forma, denomina-se *entrada de controle* o conjunto de eventos habilitados em um determinado estado. Logo, como uma especificação não deve tentar inibir eventos não controláveis, uma entrada de controle só é válida se o gerador contiver o conjunto de eventos não controláveis. Formalmente, o conjunto de entradas de controle é definido como:

Definição 22 *Dado um gerador $G = (\Sigma, Q, \delta, q_0, Q_m)$, cujo alfabeto é particionado em $\Sigma = \Sigma_c \cup \Sigma_{uc}$, o conjunto de entradas de controle associado a G é dado por:*

$$\Gamma_c = \{\gamma : \Sigma_{uc} \subseteq \gamma \subseteq \Sigma\}. \tag{0.8}$$

Dado o conjunto de entradas de controle Γ_c associado a um gerador G, este passa a ser visto como se suas funções de transição deixassem de ser definidas para os eventos inibidos pela entrada de controle aplicada em um determinado estado. Com isto pode-se definir um gerador controlado.

Definição 23 *Dado $\Gamma_c \subseteq 2^{\Sigma}$ como sendo o conjunto de entradas de controle, define-se um gerador controlado G_c como um par $\langle G, \Gamma_c \rangle$ onde G é um gerador com alfabeto Σ, particionado em eventos controláveis Σ_c e eventos não controláveis Σ_{uc}, equipado com um conjunto de entradas de controle Γ_c.*

Denomina-se *planta*, o modelo do sistema a ser controlado, semelhantemente à teoria clássica de controle. O comportamento do sistema na ausência de qualquer ação

de controle é definida como *linguagem da planta*, a qual representa o comportamento do sistema.

Como dito anteriormente, desde que seja definida uma entrada de controle γ a uma planta, esta irá se comportar como se os eventos inibidos fossem eliminados de sua estrutura de transição. Dessa forma, em um gerador cujo alfabeto é particionado em eventos controláveis e não controláveis, a função de transição deste gerador não está definida para os eventos inibidos por uma dada entrada de controle que seja aplicada ao gerador em um determinado instante. Logo, como este é o mecanismo de controle adotado pela TCS, necessita-se, então, chavear as entradas de controle para especificar, a partir da linguagem gerada, determinadas tarefas a serem realizadas. Então, usando a entrada de controle, a linguagem do gerador pode ser modificada.

Exemplo 30 *No gerador apresentado na Figura 0.24(a), a linguagem é*

$$L(G) = ((\alpha\beta + \eta)\nu + \epsilon)^* .$$

Considerando que α e η são eventos não controláveis e β e ν são eventos controláveis, mantendo a permanente desabilitação de β, a linguagem torna-se

$$L(G) = \alpha + (\eta\nu + \epsilon)^* ,$$

que é o gerador visto na Figura 0.24(b).

Este mecanismo de controle é quem define um chaveamento nas entradas de controle, determinando que a seqüência especificada que define a tarefa requerida, se possível, seja seguida. Com este mecanismo apresentado, torna-se necessário encontrar as condições necessárias e suficientes para a existência de um controlador que faça o chaveamente da planta de modo a satisfazer uma especificação de comportamento definida, bem como o procedimento de síntese para obter tal controlador. Este controlador para uma planta é denominado supervisor.

Supervisores e condições de existência

Com a formalização dos geradores controlados, deve-se determinar como chavear a entrada de controle em resposta à cadeia de eventos previamente gerada pelo sistema.

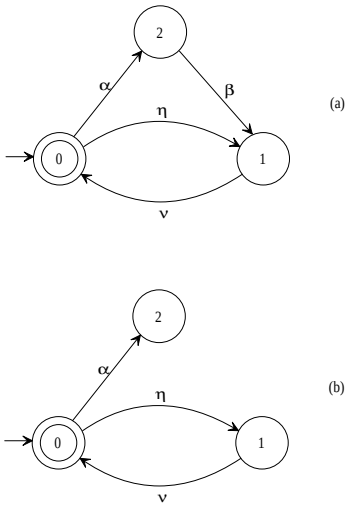

Figura 0.24: Gerador com todos os eventos habilitados e gerador com o evento β inibido.

Este chaveamento é feito pelo supervisor, que é o agente externo que determina a ação de controle a ser aplicada ao sistema.

Um supervisor é definido formalmente como:

Definição 24 *Um supervisor para um gerador controlado* $G_c = \langle G, \Gamma_c \rangle$ *é um par* $\langle \mathbf{S}, \Theta \rangle$*, composto de um gerador* $\mathbf{S} = (\Sigma, X, \xi, x_0, X_m)$ *e de um mapa de controle* Θ*, em que:*

- Σ *é o mesmo alfabeto de* 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;*
- $\Theta : X \to \Gamma_c$ *é uma função que associa a cada* $x \in X$ *uma entrada de controle* $\gamma \in \Gamma_c$.

Na Figura 0.25, está representado um sistema composto pelo gerador controlado G_c supervisionado pelo supervisor $\mathbf{S}$, em malha fechada. Nesta Figura pode-se ver que

o gerador controlado recebe a ação de controle do supervisor em resposta aos eventos gerados pela planta modelada por G. Desta forma, o gerador segue a especificação dada pelo supervisor.

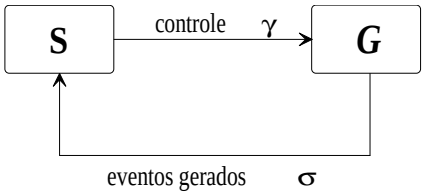

Figura 0.25: Supervisão de um SED.

De acordo com a definição de supervisor, tem-se que a ação de controle modifica a linguagem associada ao gerador, pois a mesma pode inibir seqüências de eventos que antes podiam ocorrer. Logo, sua linguagem, pode ser assim definida:

Definição 25 *Dados um gerador controlado G_c e um supervisor $\mathbf{S}$, a linguagem gerada pelo sistema supervisionado, denotada por $L(\mathbf{S}/G)$, é tal que*

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

onde Θ é o mapa de controle que associa a cada estado $x \in X$ uma entrada de controle $\gamma \in \Gamma_c$, a ser aplicada a G.

Por esta definição, observa-se que, somente os eventos que estão habilitados em cada estado, é que ocorrem sob supervisão. Isto determina a nova linguagem associada ao sistema.

Deve-se observar que, dada uma palavra s que pertença a $L(\mathbf{S}/G)$, ela também pertence a $L(G)$, o que determina que a linguagem do sistema supervisionado satisfaz

$$L(\mathbf{S}/G) \subseteq L(G). \tag{0.9}$$

Então, vê-se que $L(\mathbf{S}/G)$ é uma linguagem prefixo-fechada, ou seja,

$$L(\mathbf{S}/G) = \overline{L(\mathbf{S}/G)} \tag{0.10}$$

visto que uma palavra $s\sigma \in L(\mathbf{S}/G)$ somente se $s \in L(\mathbf{S}/G)$.

Define-se a linguagem controlada do sistema supervisionado como a seguir:

Definição 26 *Dados um gerador controlado G_c e um supervisor* $\mathbf{S}$*, a linguagem controlada do sistema sob supervisão, denotada por* $L_c(\mathbf{S}/G)$*, é definida como:*

$$L_c(\mathbf{S}/G) = L(\mathbf{S}/G) \cap L_m(G). \tag{0.11}$$

A linguagem controlada, $L_c(\mathbf{S}/G)$, é a parte da linguagem marcada original sob ação de controle que representa as tarefas que são completadas sob supervisão. Em outros termos, a linguagem controlada é simplesmente a parte da linguagem original que "sobrevive" sob a ação de controle.

Assim, é determinado que, se $L_m(G)$ representa as tarefas que podem ser completadas pela planta, então $L_c(\mathbf{S}/G)$ representa as tarefas que podem ser completadas sob supervisão. Logo, isto implica nas seguintes condições:

$$L_c(\mathbf{S}/G) \subseteq L(\mathbf{S}/G) \tag{0.12}$$

e

$$L_c(\mathbf{S}/G) \subseteq L_m(G) \tag{0.13}$$

Disto decorre que a linguagem marcada do sistema sob supervisão é

$$L_m(\mathbf{S}/G) = L_c(\mathbf{S}/G) \cap L_m(\mathbf{S}) \tag{0.14}$$

Assim, conclui-se que

$$L_m(\mathbf{S}/G) \subseteq L_c(\mathbf{S}/G) \subseteq L(\mathbf{S}/G) \subseteq L(G), \tag{0.15}$$

ou seja, a linguagem $L(\mathbf{S}/G)$, que é gerada pelo sistema composto pelo supervisor e pelo gerador controlado, $\mathbf{S}/G_c$, pode ser interpretada como o conjunto de todas as possíveis seqüências finitas de eventos que têm possibilidade de ocorrer no sistema.

Supervisores próprios

É preciso garantir que os eventos no supervisor $\mathbf{S}$ só devam ocorrer, quando eles também ocorrerem em G_c e estiverem habilitados por Θ. Para tanto, é necessário que, $\forall s \in \Sigma^*$, e $\sigma \in \Sigma$, as seguintes condições sejam verdadeiras:

$$\begin{aligned} &s \in L(\mathbf{S}/G), \\ &s\sigma \in L(G) \qquad \text{e} \\ &\sigma \in \Theta\left(\xi\left(s, x_0\right)\right). \end{aligned}$$

Essas condições implicam em que $\xi\,(s, x_0)!$, isto é, $\xi\,(s, x_0)$ é definido.

Esta é a condição para que o supervisor $\mathbf{S}$ seja dito completo em relação a um gerador controlado G_c. Isto garante que, se s é uma palavra que pode ocorrer no sistema supervisionado e o evento σ é uma continuação fisicamente possível desta palavra, e este evento σ está habilitado, então a palavra $s\sigma$ deve estar definida na função de transição do supervisor.

Torna-se necessário estabelecer duas restrições a serem satisfeitas pelas linguagens $L(\mathbf{S}/G)$, $L_c(\mathbf{S}/G)$ e $L_m(\mathbf{S}/G)$ para controlar um SED de maneira satisfatória. Estas restrições são definidas a seguir:

Definição 27 *Um supervisor* $\mathbf{S}$ *é dito não bloqueável se e somente se*

$$\overline{L_c(\mathbf{S}/G)} = L(\mathbf{S}/G) \tag{0.16}$$

Definição 28 *Um supervisor* $\mathbf{S}$ *é dito não rejeitável se e somente se*

$$\overline{L_m(\mathbf{S}/G)} = \overline{L_c(\mathbf{S}/G)}. \tag{0.17}$$

Destas duas definições, conclui-se que um supervisor é bloqueável se existir pelo menos uma palavra fisicamente possível em $L(\mathbf{S}/G)$ que não é prefixo de qualquer palavra em $L_c(\mathbf{S}/G)$, e portanto, através desta palavra o sistema nunca pode completar uma tarefa especificada. Por outro lado, um supervisor é rejeitável caso exista pelo menos uma palavra em $\overline{L_c(\mathbf{S}/G)}$ representando uma tarefa completada, contudo, que não pertence à linguagem marcada $L_m(\mathbf{S}/G)$. Neste caso, é possível atingir um estado em que nenhuma tarefa seja reconhecida, isto é, que não pertence à especificação. Estas situações indesejáveis são ilustradas no exemplo a seguir:

Exemplo 31 *Seja o gerador* $G = (Q, \Sigma, \delta, q_0, Q_m)$, *com* $\Sigma = \{\alpha, \beta, \lambda\}$ *mostrado na Figura 0.26*(a), *cuja linguagem marcada é* $L_m\,(G) = (\alpha\lambda^*\beta)^*$, *com* $\Sigma_c = \{\beta\}$. *Considere primeiramente o supervisor da Figura 0.26*(b). *Quando acoplado através do produto de autômatos ao gerador da Figura 0.26*(a), *o comportamento do sistema fica restrito ao representado pelo gerador da Figura 0.26*(c). *Vê-se que* $L\,(\mathbf{S}/G) = (\alpha\beta)^*\,(\epsilon + \alpha\lambda^*)$ *e que* $L_c\,(\mathbf{S}/G) = (\alpha\beta)^*$, *de modo que a condição de ausência de bloqueio não é satisfeita. Disto decorre que, nenhuma seqüência incluindo o evento* λ *leva a uma tarefa completada sob supervisão, pois o evento* β *é permanentemente inibido logo após a primeira*

ocorrência de λ. Por outro lado, considerando o supervisor da Figura 0.26(d) acoplado através do produto de autômatos ao gerador da Figura 0.26(a), cujo comportamento resultante é visto na Figura 0.26(e), tem-se que $L_c(\mathbf{S}/G) = L_m(G) = (\alpha\lambda^*\beta)^*$. Isto significa que todas as tarefas fisicamente possíveis podem ser completadas sob supervisão. Entretanto, $L_m(\mathbf{S}/G) = (\alpha\beta)^*$, que viola a condição de ausência de rejeição. Logo, a partir da primeira ocorrência do evento λ, todas as tarefas completadas deixam de ser marcadas.

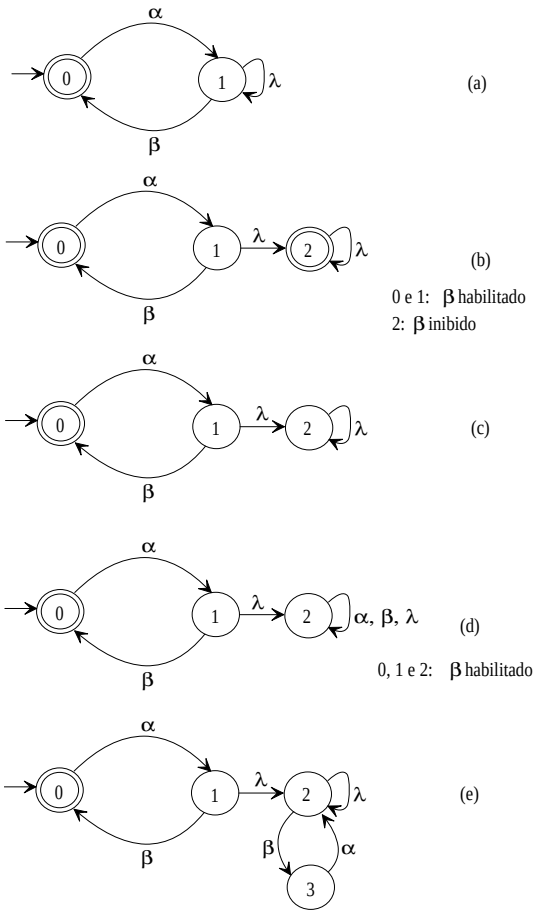

Figura 0.26: Gerador (a) e supervisores com bloqueio (b) e rejeição (d) e comportamentos resultantes (c) e (e), respectivamente.

Quando um supervisor é não bloqueável e não rejeitável, diz-se que ele é um supervisor completo.

Para a síntese do supervisor ser completa, determina-se que ele não deva apresentar rejeição nem bloqueio, o que leva a definição de supervisor próprio, que é a seguinte:

Definição 29 *Um supervisor próprio é um supervisor completo, isto é, não bloqueável e não rejeitável, de tal forma que*

$$\overline{L_m(\mathbf{S}/G)} = \overline{L_c(\mathbf{S}/G)} = L(\mathbf{S}/G). \tag{0.18}$$

Tendo isto em vista, torna-se necessário definir o problema de controle supervisório, e determinar a solução para o mesmo. Isto é visto a seguir.

Formulação e resolução do problema de controle supervisório

O problema principal da TCS, está definido em determinar mudanças no comportamento de um SED. As linguagens apresentadas anteriormente permitem a formulação de problemas abstratos de síntese de supervisores. De um modo geral, um problema desse tipo supõe que se represente o comportamento fisicamente possível do sistema e o comportamento desejado sob supervisão por linguagens, sendo o objetivo construir um supervisor para a planta tal que o comportamento do sistema em malha fechada se limite ao comportamento desejado. Para tanto, definem-se os seguintes conceitos:

Definição 30 *Sejam 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 31 *Sejam duas linguagens $K, L \subseteq \Sigma^*$ e um alfabeto $\Sigma = \Sigma_c \cup \Sigma_{uc}$. K é dita $L-$*controlável *se e somente se $\overline{K}\Sigma_{uc} \cap L \subseteq \overline{K}$.*

Estas Definições são os conceitos conhecidos por *fechamento* e *controlabilidade*, respectivamente, onde a linguagem K é definida como a linguagem da especificação de comportamento, ou o que se deseja que o SED realize, e a linguagem L é a linguagem gerada pelo SED.

Torna-se necessário determinar as condições de existência do supervisor para a realização de uma tarefa específica. É dada, então, a seguinte proposição:

Proposição 1 *Seja $\mathbf{S}$ um supervisor completo para G_c. Então $L(\mathbf{S}/G)$ é prefixo-fechada e $L(G)-$*controlável.

Também é necessário estabelecer as condições de existência de supervisores para os problemas formulados em termos de linguagens geradas e marcadas. Para isto, apresentam-se os teoremas a seguir:

Teorema 1 *Dados um gerador G tal que $L(G)$ represente seu comportamento fisicamente possível e uma linguagem especificada $K \subseteq L(G)$, existe um supervisor completo* $\mathbf{S}$ *tal que $L(\mathbf{S}/G) = K$ se e somente se K for prefixo-fechada e $L(G)$-controlável.*

Teorema 2 *Dados um gerador G tal que $L_m(G)$ represente as tarefas que podem ser completadas pelo sistema na ausência de qualquer ação de controle e uma linguagem especificada $K \subseteq L_m(G)$ então*

1. *Existe um supervisor completo* $\mathbf{S}$ *tal que $L_c(\mathbf{S}/\mathbf{G}) = K$ se e somente se K for $L_m(G)$-*fechada *e existir uma linguagem prefixo-fechada e $L(G)$-*controlável *K' tal que $K' \cap L_m(G) = K$.*

2. *Existe um supervisor não bloqueável* $\mathbf{S}$ *tal que $L_c(\mathbf{S}/G) = K$ se e somente se K for $L(G)$-*fechada *e $L(G)$-*controlável*;*

3. *O supervisor* $\mathbf{S}$ *será próprio se e somente se o gerador $S = (\Sigma, X, \xi, x_0, X_m)$ for tal que $X_m = X$.*

Para problemas formulados em termos de linguagens marcadas, se K satifizer as condições do Teorema 2, então o supervisor será tal que $L(\mathbf{S}/G) = \overline{K}$, visto que nesse caso,

$$L_c(\mathbf{S}/G) = \overline{K} \cap L_m(G) = K. \tag{0.19}$$

Estes resultados somente podem ser empregados quando a linguagem especificada K satisfaz as condições exigidas. Quando a linguagem K não satisfaz as condições exigidas à existência do supervisor, ou seja, a linguagem especificada K não é $L_m(G)$-*fechada* ou $L(G)$-*controlável*, é necessário encontrar uma sublinguagem $K^{\uparrow} \subseteq K$, a qual é sempre possível e satisfaz todas as condições restritivamente. Esta sublinguagem é conhecida por *Suprema Sublinguagem Controlável* $K^{\uparrow}$, ou $SupC(L)$, que soluciona o problema do supervisor, contudo restringe seus resultados ao mínimo tolerável. Dessa maneira, a $SupC(L)$ evita problemas na execução de uma tarefa especificada, eliminando as

possibilidades de ocorrências de bloqueios ou de execução de outras tarefas que não estejam especificadas.

Sendo assim, se $K^{\uparrow}$ solucionar de forma satisfatória o problema dado, isto é, se $K \supseteq A$, onde A é uma linguagem que representa o comportamento mais restrito que pode ser tolerado, $K^{\uparrow}$ pode ser utilizada em substituição à linguagem anteriormente especificada. A solução para este problema é um supervisor que implementa $K^{\uparrow}$. Logo, para o caso dos geradores de estado finitos, $K^{\uparrow}$ é sempre computável [3].

Considerando $K \subseteq \Sigma^*$ uma linguagem especificada, onde Σ^* representa o conjunto de todas as linguagens definidas a partir do alfabeto Σ e sendo $C(K)$ a família das linguagens controláveis de K, então $C(K)$ 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 sublinguagem controlável máxima $K^{\uparrow}$ tal que $K^{\uparrow} \subseteq K$. Assim, nota-se que $K^{\uparrow}$ pode ser a linguagem vazia.

Este problema pode ser enunciado da seguinte maneira: *Dados um gerador G, uma linguagem-alvo marcada $E \subseteq \Sigma^*$ e uma linguagem mínima admissível $L_A \subseteq E$, encontrar um supervisor próprio* $\mathbf{S}$ *tal que*

$$L_A \subseteq L_c(\mathbf{S}/G) \subseteq E. \tag{0.20}$$

Quando E é $L_m(G)$-*fechada* e $L(G)$-*controlável*, a existência de um supervisor tal que $L_c(S/G) = E$ é garantida, o que significa que o problema tem uma solução não restritiva, como visto anteriormente. Nos casos em que E não satisfaz estas condições, é possível obter uma solução minimamente restritiva, como dado pelo problema descrito anteriormente na equação (0.20).

Quando todos os estados do gerador são marcados, garante-se que qualquer supervisor obtido é não bloqueável.

O algoritmo a seguir soluciona o problema de controle supervisório, para o caso em que um gerador de estado finito G é descrito por L (comportamento em malha aberta) e K (comportamento desejado, representado pelo gerador H):

Algoritmo 2 Algoritmo para a Construção de $K^{\uparrow}$ *([3])*

- *Dados o gerador G trim e o gerador H, faça:*

1. *Construir a matriz de transições* $\mathbf{A}$ *do gerador* H*, onde*

$$\mathbf{A} = [a_{i,j}], a_{i,j} = \begin{cases} \sigma, & \text{se } \exists\, \sigma \text{ do estado } i \text{ para o estado } j; \\ - & \text{caso contrário;} \end{cases}$$

2. *Incluir ao lado direito da matriz de transições* $\mathbf{A}$ *o vetor coluna que representa*

$$\Sigma\left(H\left(x\right)\right) \cap \Sigma_{uc},$$

em que $H\left(x\right)$ *representa os eventos habilitados no estado* x *do gerador* H*;*

3. *Inclua ao lado direito da tabela o vetor coluna* $\Sigma\left(x\right)$*, representando os eventos habilitados no estado* x *do gerador* G*;*

4. *Para cada estado* x_i*, em* $\mathbf{A}$ *que não satisfaz* $\Sigma\left(H\left(x_i\right)\right) \cap \Sigma_{uc} \subset \Sigma\left(x_i\right)$*, remover a linha da tabela e a coluna da matriz* $\mathbf{A}$ *referente ao estado* x_i*;*

5. *Encontrar a componente coacessível do gerador resultante;*

6. *Itere este processo até que todos os estados* x_i *restantes satisfaçam*

$$\Sigma\left(H\left(x_i\right)\right) \cap \Sigma_{uc} \subset \Sigma\left(x_i\right).$$

Exemplo 32 *Sejam* $\Sigma = \{\alpha_1, \alpha_2, \beta\}$*,* $\Sigma_{uc} = \{\beta\}$*, e em notação de expressões regulares*

$$L\left(G\right) = \overline{\left(\alpha_1\beta^2 + \alpha_2\right)\beta^*},$$
$$L\left(H\right) = \overline{\alpha_1\beta^2} + \alpha_2\beta^*,$$

que são a linguagem do gerador trim visto na Figura 0.27 e a especificação de comportamento H *desejada para este gerador. Utilizando o algoritmo da* $\sup C\left(L\right)$ *apresentado*

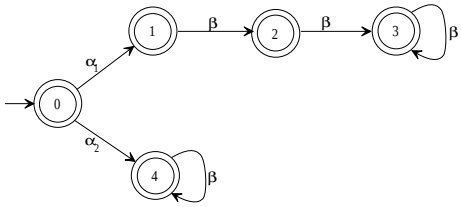

Figura 0.27: Autômato gerador de $L\left(G\right) = \overline{\left(\alpha_1\beta^2 + \alpha_2\right)\beta^*}$.

por Ramadge e Wonham [3], inclui-se separadamente à matriz de transição do gerador

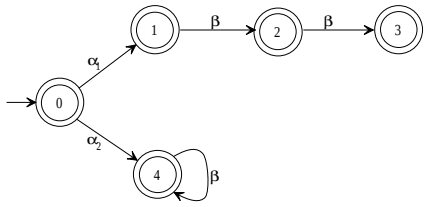

Figura 0.28: Especificação $L(H) = \overline{\alpha_1\beta^2} + \alpha_2\beta^*$.

da especificação de comportamento, visto na Figura 0.28, duas colunas: uma listando $\Sigma(H(x)) \cap \Sigma_{uc}$*, outra listando* $\Sigma(x)$*. Isto define a seguinte tabela:*

H_0 :

	1 2 3 4 5	$\Sigma(H(x)) \cap \Sigma_{uc}$	$\Sigma(x)$
1	α_1 α_2		$\alpha_1\alpha_2$
2	β	β	β
3	β	β	β
4		β	
5	β	β	β

Desde que $\Sigma(H(4)) \cap \Sigma_{uc} \not\subset \Sigma(4)$*, remove-se o estado* 4 *da tabela e encontra-se que o gerador resultante é trim. O resultado é um gerador para a linguagem* $L(H_1)$*, apresentado na tabela a seguir:*

H_1 :

	1 2 3 5	$\Sigma(H(x)) \cap \Sigma_{uc}$	$\Sigma(x)$
1	α_1 α_2		$\alpha_1\alpha_2$
2	β	β	β
3		β	
5	β	β	β

onde sua linguagem é $L(H_1) = \overline{\alpha_1\beta} + \alpha_2\beta^*$*. Iterando esse procedimento é produzida a seguinte seqüência de tabelas:*

H_2 :

	1 2 5	$\Sigma(H(x)) \cap \Sigma_{uc}$	$\Sigma(x)$
1	α_1 α_2		$\alpha_1\alpha_2$
2		β	
5	β	β	β

$L(H_2) = \overline{\alpha_1} + \alpha_2\beta^*$

$H_3:$

	1 5	$\Sigma(H(x)) \cap \Sigma_{uc}$	$\Sigma(x)$
1	α_2		$\alpha_1\alpha_2$
5	β	β	β

$L(H_3) = \alpha_2\beta^*$

em que, $L(H_3)$ *é a* $\sup C(L)$*, cujo supervisor está apresentado na Figura 0.29.*

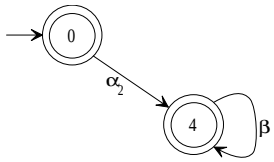

Figura 0.29: Supervisor para a $\sup C(L) = L(H_3) = \alpha_2\beta^*$.

Exemplo 33 *Considere o autômato apresentado na Figura 0.30 e a especificação de comportamento apresentada na Figura 0.31. A linguagem marcada do autômato é*

$$L_m(G) = (\alpha\beta)^* + (\alpha\kappa\eta)^*$$

e a linguagem marcada da especificação de comportamento é

$$L(H) = (\alpha + \kappa)^* \beta (\alpha + \kappa)^* \eta.$$

Pode-se ver que $L(H) \not\subset L_m(G)$*, pois* α^* *não existe em* $L_m(G)$*. Dessa forma, para*

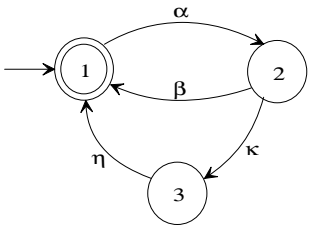

Figura 0.30: Autômato com linguagem $L_m(G) = (\alpha\beta)^* + (\alpha\kappa\eta)^*$.

calcular a $\sup C(L)$*, constrói-se a composição síncrona* $H||G$*, a qual é vista na Figura 0.32. Com a linguagem* $L(H') = L(H/G)$ *dessa composição, utiliza-se o algoritmo para construção da* $\sup C(L)$*, em que inclui-se separadamente à matriz de transição do gerador da especificação de comportamento e as duas colunas listando* $\Sigma(H'(x)) \cap \Sigma_{uc}$

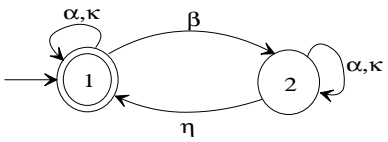

Figura 0.31: Especificação $L(H) = (\alpha + \kappa)^* \beta (\alpha + \kappa)^* \eta$.

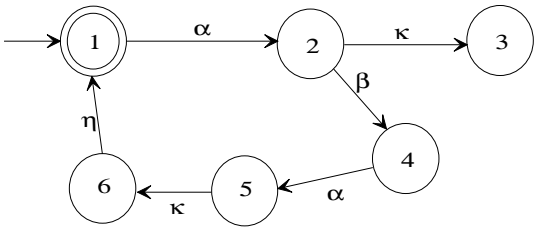

Figura 0.32: Composição síncrona $H||G$.

e $\Sigma(x)$*. Isto define a seguinte tabela:*

		1	2	3	4	5	6	$\Sigma(H'(x)) \cap \Sigma_{uc}$	$\Sigma(x)$
H_0':	1		α						α
	2			κ	β				$\kappa\beta$
	3							η	
	4					α			α
	5						κ		κ
	6	η						η	η

Desde que $\Sigma(H'(3)) \cap \Sigma_{uc} \not\subset \Sigma(3)$*, remove-se o estado* 3 *da tabela e encontra-se que o gerador resultante é trim. O resultado é um gerador para a linguagem* $L\left(H_1'\right)$*, apresentado na tabela a seguir:*

		1	2	4	5	6	$\Sigma(H'(x)) \cap \Sigma_{uc}$	$\Sigma(x)$
H_1':	1		α					α
	2			β				β
	4				α			α
	5					κ		κ
	6	η					η	η

onde sua linguagem é $L\left(H_1'\right) = \alpha\beta\alpha\kappa\eta$*. Como todas as outras linhas satisfazem*

$$\Sigma(H'(x)) \cap \Sigma_{uc} \subset \Sigma(x),$$

e H_1' *é coacessível, então,* $L\left(H_1'\right)$ *é a* $\sup C\left(L\right)$ *que é a linguagem do supervisor visto na Figura 0.33.*

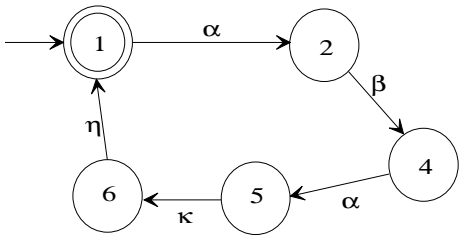

Figura 0.33: Supervisor para $\sup C\left(L\right) = L\left(H_1'\right) = \alpha\beta\alpha\kappa\eta$.

Redes de Petri e TCS

Uma outra abordagem para a síntese de supervisores é descrita por Barroso [41]. Nesta, as redes de Petri são utilizadas para a modelagem do gerador G e do supervisor $\mathbf{S}$, devido às vantagens apresentadas por estas, as quais permitem uma descrição mais detalhada do sistema além da síntese modular de sistemas compostos por subsistemas interagentes.

Na Figura 0.34 é apresentado o diagrama em blocos que mostra o uso das redes de Petri na modelagem, análise e controle de um SED. Nela, pode-se ver que a partir do modelo do sistema, sintetiza-se o controlador baseado em uma ou mais especificações de tarefas a serem realizadas pelo sistema sob supervisão. Assim, esta abordagem se torna mais flexível, visto que, dado o sistema modelado por uma rede de Petri, qualquer tarefa especificada não muda a estrutura do supervisor.

Nesta abordagem, a síntese do supervisor, a qual está apresentada na Figura 0.35, é realizada tendo como dados o modelo do sistema e seu comportamento requerido que é a especificação funcional. Como já citado anteriormente, a especificação funcional, determina uma tarefa física por meio de uma linguagem. No caso desta abordagem, a entrada da especificação de comportamento requerida para o algoritmo que constrói a $SupC(L)$, necessita do espaço de estados da rede de Petri que modela o sistema, o qual é a sua árvore de alcançabilidade. Através da especificação de comportamento e o espaço de estados da rede de Petri, a execução do algoritmo irá determinar se tal especificação é factível ou qual sua aproximação mais restritiva, isto é, a especificação possível ou a

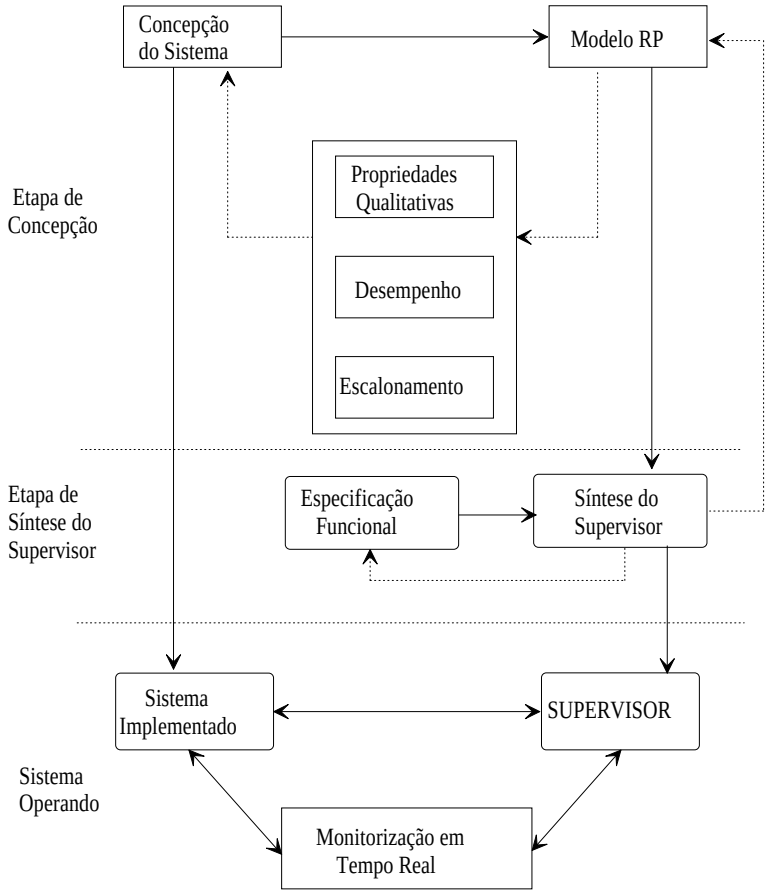

Figura 0.34: Utilização de redes de Petri e Teoria de Controle Supervisório para a concepção, análise e controle de um SED.

Suprema Sublinguagem Controlável. Com isso, procede-se à determinação das funções de transição que irão compor a rede de Petri supervisora, a qual é formalizada por uma RPFHT.

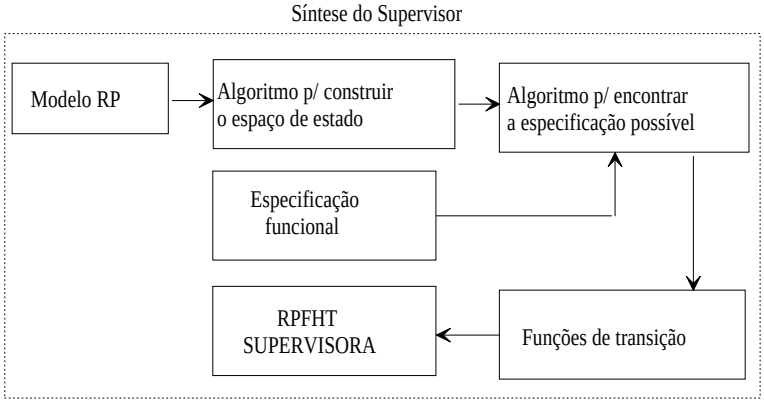

Figura 0.35: Diagrama em blocos do procedimento de síntese do supervisor.

Definida a especificação funcional, ou de comportamento desejado, procede-se às condições de existência do supervisor para um dado SED. Como já visto, a condição para a existência de um supervisor para um SED é o conceito de controlabilidade, em que um gerador controlado G_c gera uma linguagem, a qual é prefixo da linguagem do gerador não controlado G. Como tem-se que esses geradores estão modelados por redes de Petri, o gerador controlado G_c tem um conjunto próprio de todas as seqüências de transições que podem disparar e que sempre será um subconjunto do conjunto de seqüências de transições disparáveis do gerador não controlado G. Isto determina a possibilidade de execução síncrona da rede de Petri que modela o sistema juntamente com o supervisor, de acordo com as restrições impostas ao seu comportamento dinâmico. Essas restrições são determinadas pelas funções de habilitação de transições impostas às transições na RPFHT supervisora, de forma a evitar um comportamento não desejado. Assim, a rede supervisora segue uma seqüência de transições requerida, impedindo bloqueios ou comportamentos não permitidos ao sistema, fazendo com que o mesmo realize a tarefa especificada.

Os estudos baseados nessa abordagem incluem o *Algoritmo Modificado da Árvore de Alcançabilidade* (AMArA) e o *Algoritmo para a Contrução da Suprema Linguagem Controlável* (ACGS), que são os algoritmos responsáveis, respectivamente, pela construção da árvore de alcançabilidade associada à rede de Petri que modela o sistema

e pela construção da Suprema Linguagem Controlável ($SupC(L)$) que determinará as funções de habilitação das transições a serem aplicadas à RPFHT supervisora.

Definição do problema

Como visto anteriormente, o problema de controle supervisório é particionado em três etapas para sua resolução: modelagem, especificação de comportamento e síntese do supervisor. Aqui, são tratadas essas etapas separadamente, para a compreensão da solução do problema.

Modelagem de SEDs por redes de Petri

Para se modelar um SED é necessário, num primeiro momento, elaborar uma descrição textual que descreva detalhadamente o seu funcionamento. Juntamente com essa descrição é necessário definir um conjunto finito dos estados que o sistema pode alcançar, o qual deve ser suficiente para descrever o comportamento, as variáveis que se deseja estudar, e um conjunto de eventos, que descrevam todas as transições entre os estados deste sistema.

Na modelagem de SEDs por redes de Petri, considera-se que, a cada evento é associado a uma transição na rede, os estados do sistema são representados pelas marcações da rede e, os lugares são as partes que compõem o sistema [31, 33].

Na modelagem de SEDs utilizando redes de Petri considera-se que:

1. Um lugar pode ser interpretado como o estado de um recurso ou de uma atividade. Quando o lugar é interpretado como o estado de um recurso, o número inicial de fichas pode ser constante para representar que há uma quantidade fixa de recursos no sistema ou variável para representar a quantidade de tarefas realizadas no sistema.

2. Se um lugar é interpretado como o estado de um recurso, a presença de uma ou mais fichas nesse lugar indica que o recurso está disponível e a ausência de fichas indica que o recurso não é disponível. Por outro lado, se o lugar é interpretado como o estado de uma atividade, a presença da ficha indica que essa atividade está sendo realizada e a ausência da ficha indica que a atividade não está sendo realizada.

3. Uma transição pode ser interpretada tanto como o ínicio quanto o término de um processo ou de uma atividade.

Desse modo, para construir um modelo em rede de Petri para um SED, seguem-se os seguintes passos:

1. Identificar os recursos e atividades necessários ao funcionamento do SED;
2. Criar uma lista ordenada de atividades de acordo com as relações de precedência definidas da descrição textual do funcionamento do SED;
3. Para cada atividade da lista:
   (a) Criar e etiquetar um lugar para representar a condição da atividade;
   (b) Criar uma transição para representar o inicio da atividade com arcos direcionados para os lugares de saída;
   (c) Criar uma transição para representar o término da atividade com arcos direcionados para os lugares de entrada. De modo geral, a transição de término de uma atividade será a mesma transição de ínicio da próxima atividade na lista ordenada. Quando a rede for executada, uma ficha num lugar representa que a atividade está sendo executada e várias fichas indicarão sua execução na multiplicidade do número de fichas. O disparo de uma transição de inicialização representa o início do processo e o disparo de uma transição de finalização representa a complementação da atividade e pode também representar o início da próxima atividade;
4. Para cada atividade ordenada: se um determinado lugar não já tiver sido criado, crie-o e rotule o lugar para cada recurso que deve estar disponível para iniciar a atividade. Conecte todos os lugares de disponibilidade de recursos apropriados com arcos a cada transição de entrada para a inicialização da atividade. Crie arcos de saída para conectar às transições de finalização seguintes à atividade para algum lugar de recurso representando recursos que se tornem disponíveis (estão livres) na complementação da próxima atividade.
5. Especificar a marcação inicial para o sistema.

Seguindo estes passos, pode-se modelar um SED por uma rede de Petri.

Exemplo 34 *Considere um sistema de um cofre com um alarme, como apresentado na Figura 0.36. Este cofre tem um código formado por quatro dígitos (0 ou 1). A seqüência para abrir o cofre é formada por dois dígitos quaisquer (00, 01, 10, 11), seguidos por dois dígitos iguais (00 ou 11). A partir da entrada de cada dígito, o cofre fica a espera do próximo dígito. Ao término da entrada dos quatro dígitos, o sistema avalia o código que, se estiver correto, o cofre abre. Caso contrário, o alarme dispara, voltando o cofre ao estado inicial, após o alarme desligar. Quando o cofre está aberto, o mesmo pode ser fechado, voltando ao estado inicial. Assim, as atividades que são consideradas neste sistema, são:* cofre fechado (p_1)*, também considerado como* esperando o primeiro dígito da seqüência do código*;* esperando segundo dígito da seqüência do código (p_2)*;* esperando os dois últimos dígitos da seqüência do código (p_3)*;* avaliando o código (p_4)*;* alarme disparando (p_5) *e* cofre aberto (p_6)*. A seqüência das atividades é dada por*

$$p_1 \rightarrow \quad p_2 \rightarrow \quad p_3 \rightarrow \quad p_4 \begin{array}{l} \nearrow \\ \searrow \end{array} \begin{array}{l} p_5 \rightarrow p_1 \\ \\ p_6 \rightarrow p_1 \end{array}$$

Daí, para construir a rede de Petri que modela este sistema, inicialmente criam-se os lugares (p_1 a p_6), como visto na Figura 0.37. Depois criam-se as transições que definem as mudanças de atividades (Figura 0.38), colocam-se os arcos de entrada e de saída, de acordo com as inicializações e término de cada atividade e, definindo a marcação inicial como $M_0 = [1\ 0\ 0\ 0\ 0\ 0]^T$, isto é, considerando que o cofre esteja fechado inicialmente, constrói-se a rede de Petri apresentada na Figura 0.39, que é o modelo deste sistema. Observe que, quando o cofre está fechado, há uma ficha no lugar p_1, representando que o mesmo está esperando a entrada do primeiro dígito. Quando há a entrada deste primeiro dígito (representada pelo disparo da transição t_1), a ficha sai do lugar p_1, e vai para o lugar p_2, onde o cofre está esperando o segundo dígito da seqüência, e assim por diante. Também, deve-se observar que, este modelo é simples, desde que não é possível detectar qual o código que define a abertura do cofre. Há também, as restrições relativas ao tempo, desde que não se tem informação a respeito de quando o alarme deve parar, ou qual o tempo que o sistema deve esperar pela entrada de um novo

dígito. O primeiro caso se resolve utilizando um maior refinamento nas definições das atividades. Por outro lado, o segundo caso só é resolvido com a introdução de tempo nas redes de Petri, o que não é o caso tratado aqui. Para o problema colocado aqui, só será necessário a parte lógica das redes de Petri.

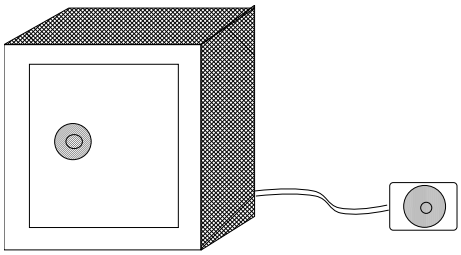

Figura 0.36: Cofre com alarme.

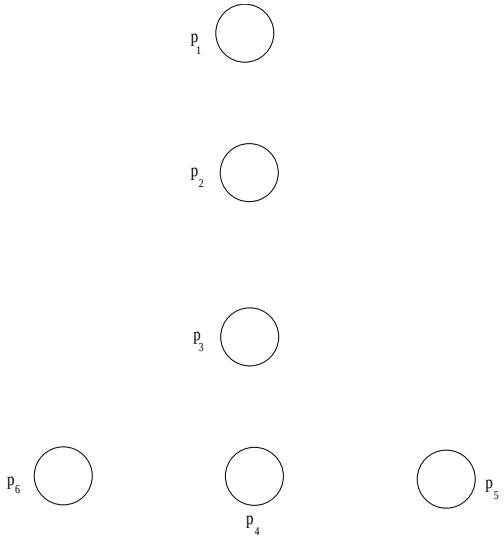

Figura 0.37: Primeiro passo na construção do modelo em rede de Petri para o sistema do cofre com alarme.

Exemplo 35 *Na Figura 0.40 está apresentado um simples sistema de manufatura consistindo de duas estações de processamento com máquinas, M_1 e M_2, um robô compartilhado, R, para descarga e um* buffer*, B, para armazenamento temporário de peças intermediárias. Cada peça é processada primeiro em M_1 e depois em M_2. As peças entram no sistema e na estação de processamento são automaticamente fixadas a uma*

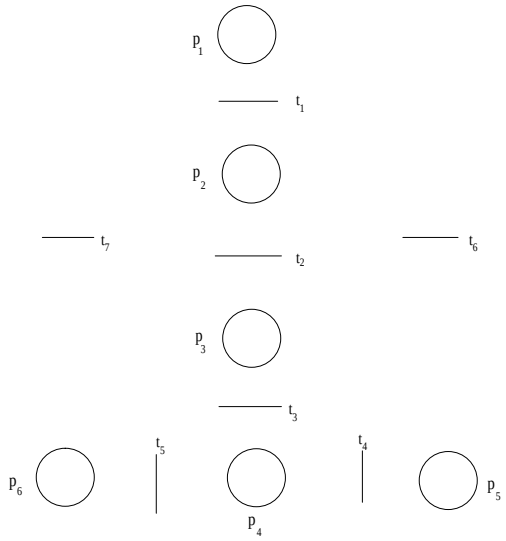

Figura 0.38: Segundo passo na construção do modelo em rede de Petri para o sistema do cofre com alarme.

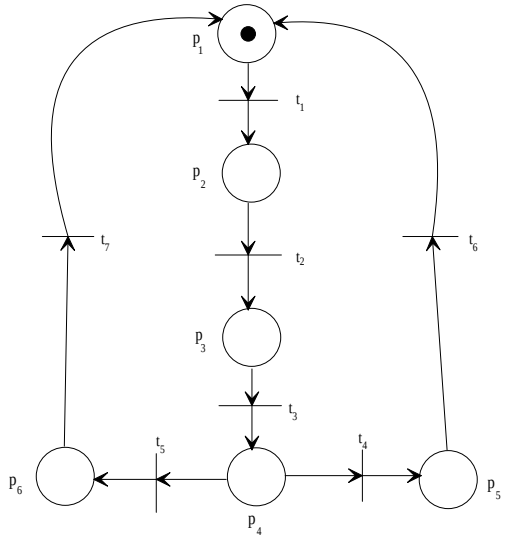

Figura 0.39: Modelo em rede de Petri para o sistema do cofre com alarme.

bandeja e carregadas na máquina M_1. Após o processamento, o robô R descarrega de M_1 a peça intermediária e a coloca no buffer B. *Logo a seguir, as peças intermediárias são automaticamente carregadas em M_2 e processadas. Quando M_2 encerra o processamento de uma peça, R descarrega o produto final e libera a bandeja para a primeira estação de trabalho. Assume-se que peças de entrada estão sempre disponíveis para serem processadas e que o produto final é sempre removido. Seguindo a metodologia de construção do modelo do sistema, tem-se no primeiro passo que as atividades requeridas são:* estações de processamento (fixação à bandeja, carga e processamento de peças), armazenamento e descarga. *Os recursos são M_1, M_2, R,* B, fixadores e peças. *No segundo passo, identifica-se a ordem das atividades como:*

M_1P : M_1carrega, fixa e processa a peça*;*
RU_1 : Rdescarrega uma peça intermediária no buffer*;*
$BS : B$ armazena uma peça intermediária*;*
M_2P : M_2carrega e processa uma peça intermediária*;*
RU_2 : Rdescarrega o produto final de M_2, libera a bandeja e retorna primeira estação de trabalho.

Seguindo o terceiro passo, é criado a rede mostrada na Figura 0.41(a). No quarto passo, tem-se que a atividade M_1P requer uma bandeja (representada pelo lugar PA) e a máquina M_1 livre (representada pelo lugar M_1l). Continuando no quarto passo, criam-se os lugares BA (buffer *livre*), *RA (representando o robô livre) e M_2l (representando a máquina M_2 livre). Os arcos são ligados às transições, como mostrado na Figura 0.41(b), satisfazendo os requerimentos do sistema. Por fim, coloca-se a marcação inicial: inicialmente, ambas as máquinas estão livres (uma ficha em M_1l e uma ficha em M_2l), há quatro bandejas disponíveis (quatro fichas no lugar PA) o robô está livre (uma ficha em RA) e há espaço livre no* buffer *para duas peças intermediárias (duas fichas no lugar BA). Assim, constrói-se a rede de Petri que modela o sistema de manufatura definido, como visto na Figura 0.42.*

Exemplo 36 *Na Figura 0.43 é apresentado um modelo em rede de Petri de uma célula de montagem de pistão. Observa-se neste modelo, que cada lugar na rede é designado com um rótulo, de forma a ser possível seguir a evolução dinâmica do SED. Deve-se observar também, que existe uma equivalência entre as redes de Petri e as máquinas*

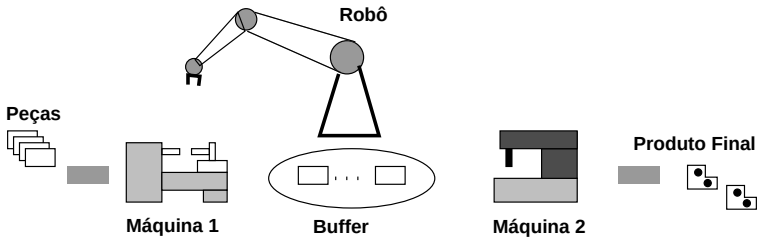

Figura 0.40: Simples sistema de manufatura com recursos compartilhados.

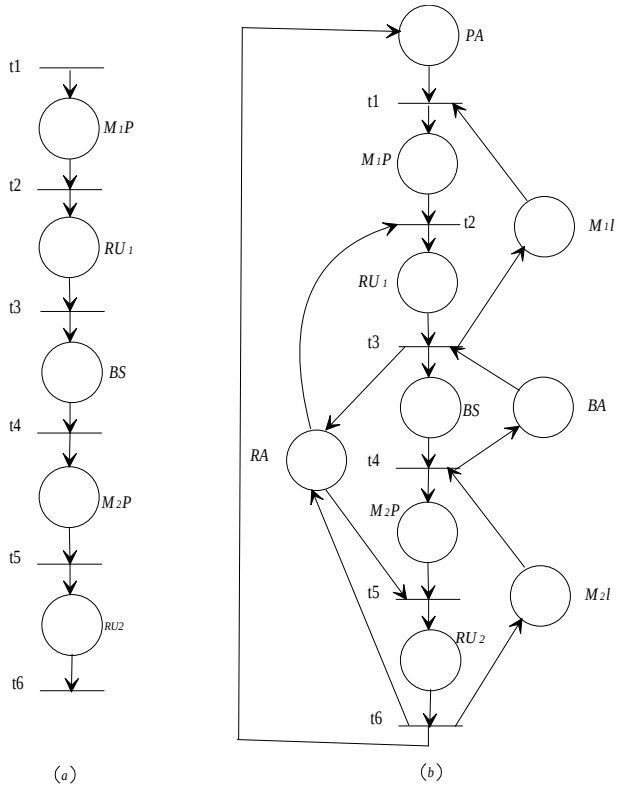

Figura 0.41: Criação da estrutura da rede de Petri que modela o SED.

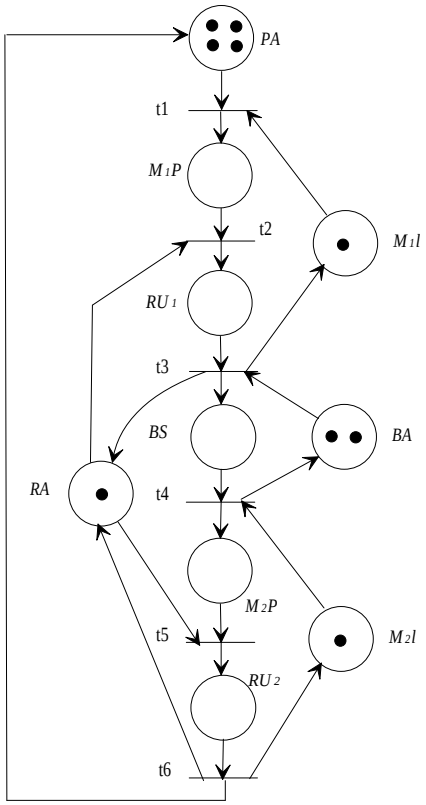

Figura 0.42: Rede de Petri que modela o sistema de manufatura com recursos compartilhados.

de estados finitos. Neste modelo, não existem bloqueios, e não é de se esperar que, é preciso analisar bem o sistema, antes de modelá-lo, para evitar situações indesejáveis. Na rede mostrada na Figura 0.44, é apresentado o modelo de um SED que bloqueia quando segue a seqüência de disparos de transições dada por

$$t_1t_2t_3t_4t_1t_2t_3t_1t_2t_3t_1t_2.$$

Nesta marcação alcançada, nenhuma transição está habilitada. Logo, exige-se que o modelo seja modificado para evitar este tipo de problema.

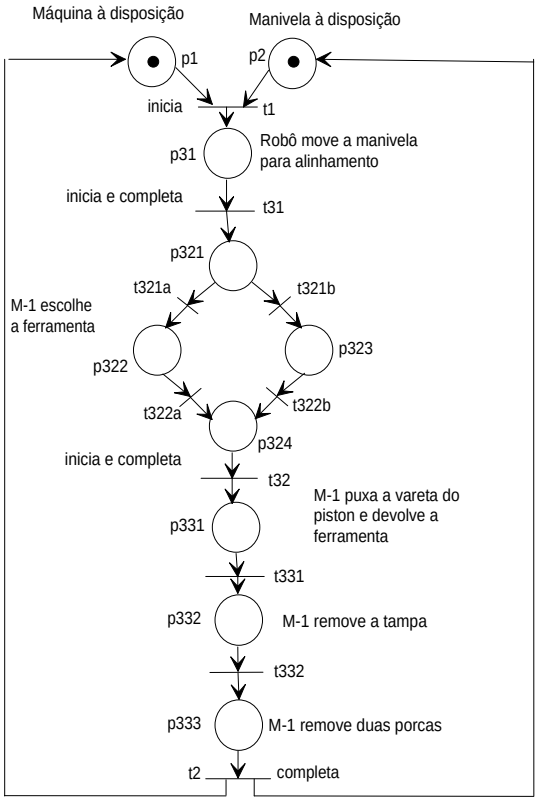

Figura 0.43: Célula de montagem de pistão modelado por uma rede de Petri.

Especificação de Comportamentos

A especificação de comportamento pode ser formalizada pelas linguagens formais e pela lógica temporal, como visto anteriormente. Basicamente, a especificação de comportamento é formulada igualmente à formalização dada quando utilizando os autômatos e

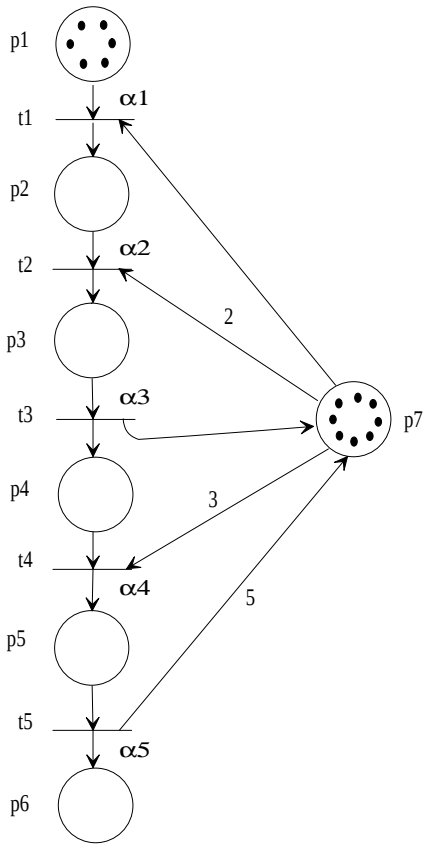

Figura 0.44: SED modelado por rede de Petri.

as linguagens formais. Isto porque a árvore de alcançabilidade de uma rede de Petri formaliza um grafo que é uma estrutura de um autômato. Neste caso, os estados marcados (para definir a linguagem marcada) são representados por marcações alcançáveis da rede que são desejadas e que representam tarefas completadas no sistema. Logo, a especificação de comportamento a ser definida deve satisfazer as mesmas condições apresentadas no modelo **RW**, em relação à linguagem da rede.

Exemplo 37 *Considere o sistema de transmissão e recepção de dados, que está apresentado na Figura 0.45, modelado por uma rede de Petri, que tem sua árvore de alcançabilidade apresentada na Figura 0.46. Neste sistema, o transmissor está sempre pronto para enviar uma mensagem que deve passar por um* buffer *antes de ser transmitida pelo canal, e o receptor deve enviar uma mensagem de retorno, confirmando o recebimento da mensagem. Neste sistema, uma nova mensagem só pode ser enviada, se houver o aviso de recepção. Formula-se, então, a especificação de comportamento definindo-a em termos de uma linguagem marcada. Assim, deseja-se que este sistema esteja sempre preparando novas mensagens para serem enviadas ao receptor, indiferentemente de sua transmissão. Isto implica na linguagem* $L_m = (t_1^* + t_1^* t_2 t_3 t_4)^*$. *Se é desejado que esta mensagem só seja preparada se o sistema receber uma mensagem de retorno, determina-se que após* t_1 *disparar, ela só deve disparar novamente se* t_4 *disparar, isto é,* $L_m = (t_1 t_2 t_3 t_4)^*$.

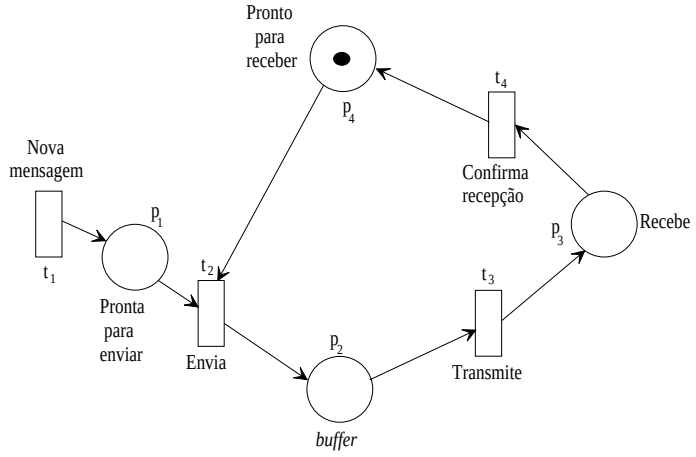

Figura 0.45: Sistema de Transmissão/Recepção modelado por rede de Petri.

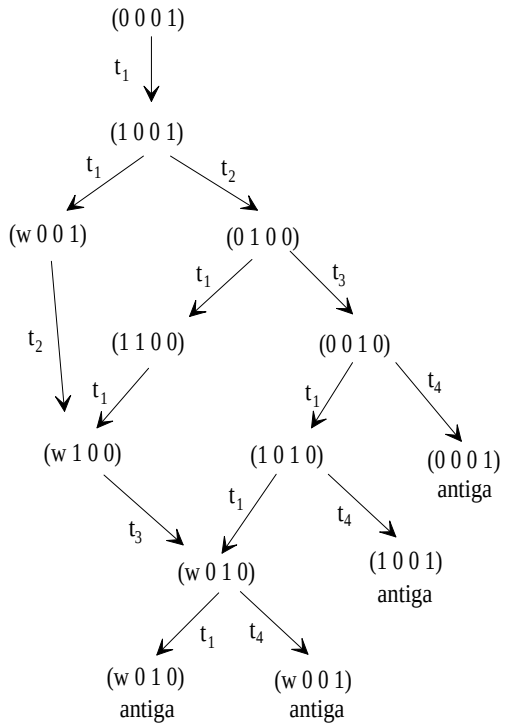

Figura 0.46: Árvore de alcançabilidade da rede de Petri, para o sistema Transmissão/Recepção.

Síntese do Supervisor por RPFHT

A síntese de supervisores de SEDs utilizando redes de Petri para modelagem dos sistemas, necessita dos algoritmos apresentados por Barroso [41], os quais em sua execução criam a árvore de alcançabilidade da rede de Petri (AMArA) e determina, a partir desta, juntamente com a especificação requerida, todas as funções que devem ser aplicadas às transições do supervisor modelado, para que o sistema não entre em *bloqueio* (ACGS) ou atinja estados não desejados.

Algoritmo Modificado da Árvore de Alcançabilidade

Este algoritmo - AMArA - tem uma modificação simples com respeito ao algoritmo original, que é um passo a mais na determinação de estados não permitidos (ramos da árvore com crescimento infinito). Assim, o AMArA lista todos os estados possíveis de serem alcançados pelo sistema, juntamente com as seqüências de transições disponíveis a partir de cada estado. Observa-se, então, que o mesmo é válido para redes com capacidade limitada ou finita.

Algoritmo 3 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 for idêntica a uma outra marcação já existente no caminho da raiz até M, etiquete-a como *antiga*;

c) Se nenhuma transição está habilitada em M, etiquete M como *bloqueada*;

d) Enquanto existirem transições habilitadas em M, faça o seguinte para cada transição habilitada:

i. Obtenha a marcação M' que resulta do disparo de t em M;

ii. Se a capacidade de algum lugar p é excedida na marcação M', então substitua $M'(p)$ por ω;

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.*

Exemplo 38 *O uso deste algoritmo está ilustrado para o caso do sistema apresentado na Figura 0.47, cujo modelo em rede de Petri é de um sistema produtor/consumidor. Neste caso, a utilização deste algoritmo, resulta na árvore de alcançabilidade apresentada na Figura 0.48, onde os estados mostrados nos vetores, são relativos ao vetor de marcação*

$$M = [p_1\ p_2\ p_3\ p_4\ p_5]^T .$$

É observado que, devido à seqüência $(\alpha_1\beta_1)^*$*, o sistema atinge uma marcação* não-permitida*, pois o lugar* p_5 *irá ter um aumento descontrolado no número de fichas. Esta seqüência deve ser evitada pela ação de controle externo ao sistema.*

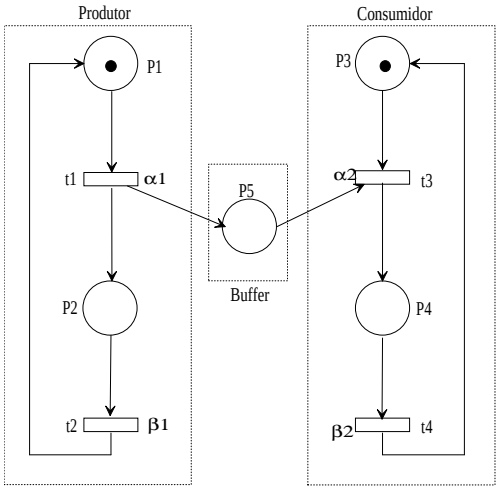

Figura 0.47: Modelo do sistema produtor/consumidor via rede de Petri.

Algoritmo para a Construção do Gerador da *SupC(L)*

Aqui tem-se um algoritmo que trabalha com a utilização dos dados referentes à árvore de alcançabilidade gerada pelo AMArA e da especificação do comportamento que se

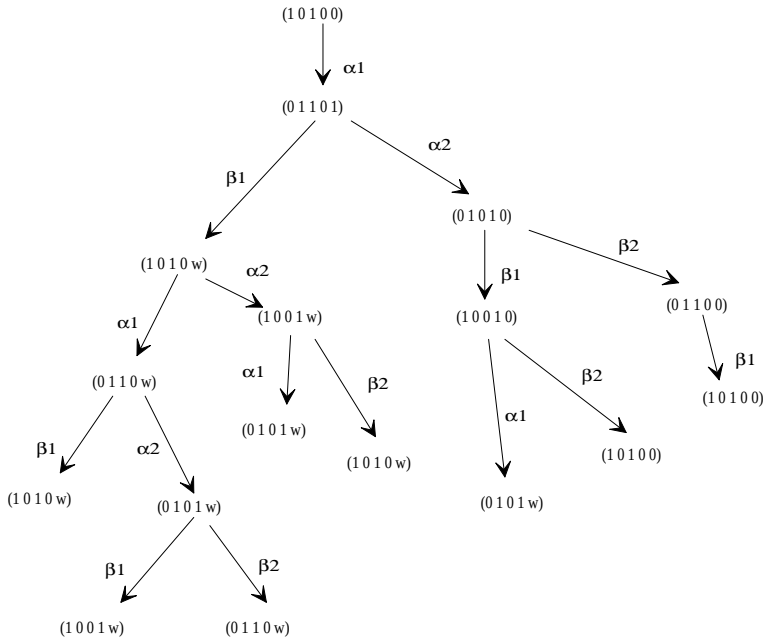

Figura 0.48: Árvore de alcançabilidade do sistema produtor/consumidor modelado por rede de Petri, com utilização do AMArA.

deseja para o sistema. Logo, este algoritmo só pode ser executado após a execução do AMArA e a entrada desta especificação. De posse destes dados, a execução do ACGS 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 necessitam ser impostas às respectivas transições para impedir que o sistema entre em bloqueio ou siga a especificação desejada, caso esta seja possível.

Algoritmo 4 ACGS

- *Início*

1. Criar uma lista dinâmica, *lista-bloc*, e incluir na mesma os estados ou marcações bloqueadas, incluindo as marcações do tipo *não-permitida*, onde:

$$M : M[t_j > 0, \text{ é uma marcação bloqueada e}$$
$$M : M(p_i) = w \text{ é uma marcação não-permitida;}$$

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

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

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

$$\exists \alpha \in \Sigma_{uc}, l(t_j) = \alpha \text{ e } M[t_j > M', M' \in \textit{lista-bloc;}$$

4. Criar uma lista, *lista-perigo*, com os estados antecessores dos elementos (estados) da *lista-bloc*, juntamente com o evento que os liga, desde que o antecessor não esteja na lista-bloc. Estes eventos deverão estar sempre desabilitados quando o sistema se encontrar nesses estados, ou seja:

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

5. Dada a especificação desejada para o sistema, encontre a suprema linguagem controlável - SupC(L);

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

$$\exists \beta \in \Sigma_c | l(t_j) = \beta \text{ e } M[t_j > M',\ \text{e } M \notin \textit{lista-perigo} \text{ e } M' \notin G(\operatorname{Sup} C(L)).$$

- *Fim.*

Neste algoritmo, o passo 5 se processa da seguinte forma:

Algoritmo 5 Passo 5 do ACGS

- *Dados o gerador* trim *e o gerador* $\boldsymbol{H}$ *(especificação), faça:*

1. Adicione à *lista-bloc* os estados que não satisfazem

$$\Sigma\left(\mathbf{H}\left(x\right)\right) \cap \Sigma_{uc} \subseteq \Sigma\left(x\right);$$

2. Para cada estado x_i, na *lista-bloc*, adicione à *lista-bloc*:

(a) os estados

$$x_j : (\exists \sigma_{uc} \in \Sigma_{uc})\, x_i = \xi\,(\sigma_{uc}, x_j)\,;$$

(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.

Vê-se que no passo 1, adiciona-se à *lista-bloc* todos os estados em que, um evento não controlável, que é fisicamente possível, não é definido na especificação; no passo 2 processa-se e incrementa-se a lista e, por fim, no passo 3, remove-se qualquer estado inacessível deixado pelos passos anteriores. Além do mais, a segunda parte do passo 2 preserva a coacessibilidade. No final destas operações, determina-se o gerador da suprema linguagem controlável para a especificação dada.

Exemplo 39 *Na rede mostrada na Figura 0.47, a utilização do ACGS, determina as funções de habilitação de transições que impedem que o sistema alcance estados não permitidos ou bloqueados. De acordo com uma especificação do tipo $\alpha_1\beta_1\alpha_2\beta_2$, estas funções seriam $\varphi_1 = \overline{p_5}$, $\varphi_2 = \varphi_3 = \varphi_4 = 1$. Nesta condição, a função de habilitação da transição p_1 significa que esta só pode disparar se não houverem fichas no lugar p_5, ou seja, $\varphi_1 = 1$ se $M(p_5) = 0$.*

Exemplos da síntese do supervisor de SEDs modelados por redes de Petri

Exemplo 40 *:*

Considere o sistema de manufatura com recursos compartilhados modelado por rede de Petri, apresentado na Figura 0.49, em que sua marcação inicial seja $M_0 = [6\ 0\ 0\ 0\ 0\ 0\ 8]^T$. Este sistema representa a produção de ítens utilizando recursos, os quais são representados pela marcação no lugar p_7. Deseja-se nele, determinar inicialmente sua árvore de alcançabilidade, para em seguida ser definida uma especificação de comportamento e determinar as funções de habilitação das transições que execute a mesma,

impedindo o sistema de cair num estado indesejável, como é o caso da seqüência de disparos das transições dada por

$$t_1t_2t_1t_2t_1t_1t_3t_3t_2t_3t_1,$$

que leva a rede a atingir a marcação $M^{'} = [1\ 2\ 0\ 3\ 0\ 0\ 0]^T$ em que nenhuma transição está habilitada.

A utilização do AMArA gera a árvore de alcançabilidade deste sistema, a qual guarda todas as seqüências de transições possíveis e todos os estados alcançados pelo mesmo, determinando para cada seqüência, qual leva a um estado bloqueado ou não. Esta árvore contém 162 estados e não é apresentada graficamente devido ao seu tamanho.

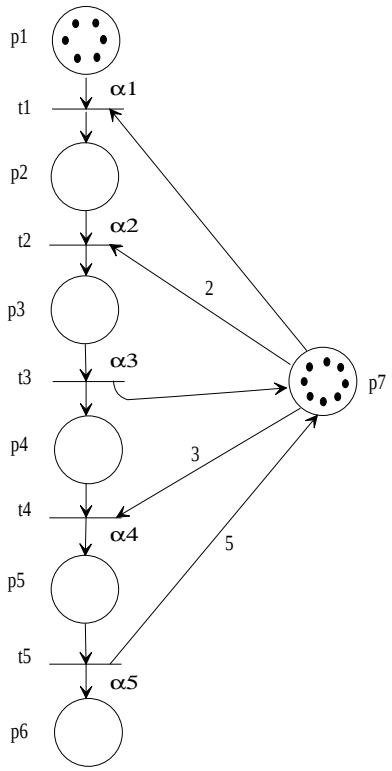

Figura 0.49: Modelo em rede de Petri para um sistema de manufatura com recursos compartilhados.

Como é desejado especificar o comportamento deste sistema, pode-se requerer que uma única peça seja processada por vez, definida pela linguagem:

$$L_m = (t_1t_2t_3t_4t_5)^*.$$

Para esta especificação, o ACGS retorna uma função para a transição t_1, dada por

$$\varphi_1 = \{M(p_7) = 8\},$$

a qual impede que seja iniciado novo processamento, enquanto o item não estiver completamente terminado. Neste caso, o sistema modelado por uma RPFHT, fica como mostrado na Figura 0.50.

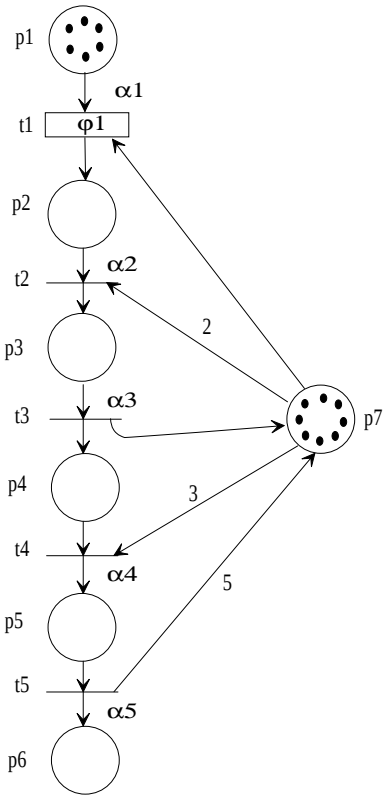

Figura 0.50: Modelo em RPFHT para um sistema de manufatura com recursos compartilhados, para realizar o processamento de uma única peça por vez.

Exemplo 41 *:*

Para este mesmo sistema, pode-se definir uma outra especificação de comportamento, de forma a ser feito o processamento paralelo de mais de uma peça, eliminando a ociosidade parcial citada. A partir da especificação

$$L_m = (t_1^2 t_2 (t_3 + t_2 t_3^2)(t_4 t_5)^2)^*$$

determina-se que, a transição t_1 pode disparar duas vezes seguidas, deixando o sistema trabalhando em paralelo. Isto leva a gerar as funções de habilitação de transições

$$\varphi_1 = \{M(p_7) > 6\}$$
$$\varphi_4 = \{M(p_7) \geq 3\},$$

que controlarão o sistema para impedir seus possíveis bloqueios e seguir a especificação dada. Este novo modelo, encontra-se na Figura 0.51. Observe que a especificação é uma sublinguagem da linguagem da rede de Petri, desde que após o disparo de t_4 pela primeira vez, não há mais recursos suficientes para repetir seu disparo, só sendo possível esta seqüência.

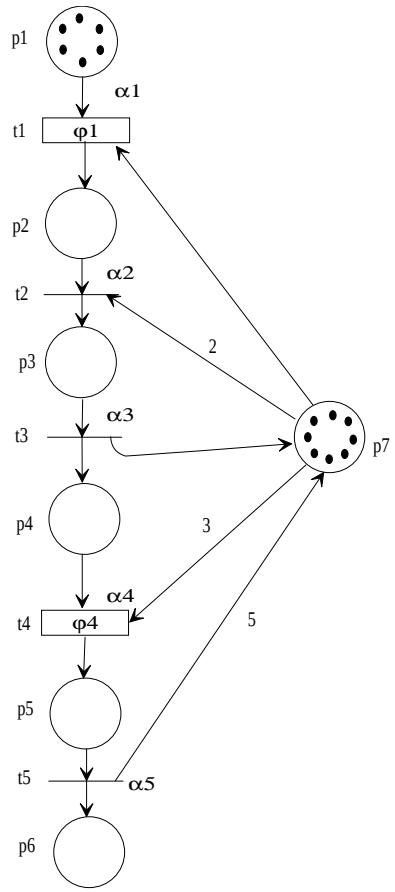

Figura 0.51: Modelo em RPFHT para um sistema de manufatura com recursos compartilhados, para realizar o processamento de duas peças em paralelo.

Bibliografia

[1] P.J.G. Ramadge and W.M. Wonham. Supervision of discrete event processes. *Procedings of 21st Conference on Decision and Control*, pages 1228–1229, 1982.

[2] P.J.G. Ramadge and W.M. Wonham. Modular feedback logic for discrete event systems. *SIAM Journal of Control and Optimization*, 25(5):1202–1218, 1987.

[3] P.J.G. Ramadge and W.M. Wonham. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, May 1987.

[4] P.J.G. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

[5] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[6] X.R. Cao and Y.C. Ho. Models of discrete event dynamic systems. *IEEE Control System Magazine*, 10(4):69–76, 1990.

[7] R. Milner. *A Calculus of Comunicating Systems*. Springer Verlag, New York, USA, 1980.

[8] S. Gaubert. *Théorie des Systèmes Linéaires dans les Dioïdes*. PhD thesis, École Nationale Supérieure des Mines de Paris, 1992.

[9] J.L. Boimond. Sur l'étude des systèmes à événements discrets dans l'algèbre des dioïdes: Identification, commande des graphes d'événements temporisés, représentation des graphes d'événements temporisés à paramètres variables. Technical report, l'Université d'Angers, 1999.

[10] S. Gaubert. On rational series in one variable over certain dioids. Technical report, Institut National de Recherche en Informatique et en Automatique, 1994.

[11] D. L'Her. *Modélisation du GRAFCET Temporisé et Vérification de Propriétés Temporelles*. PhD thesis, l'Université de Rennes 1, 1997.

[12] L. Libeaut. *Sur l'utilisation des Dioïdes pour la Commande des Systèmes a Événements Discrets*. PhD thesis, École Doctorale Sciences pour L'Ingenieur de Nantes, 1996.

[13] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronisation and Linearity. An Algebra for Discrete Event Systems*. John Wiley Sons, 1992.

[14] S. Gaubert. On the burnside problem for semigroups of matrices in the (max,+) algebra. *Semigroup Forum, Springer-Verlag New York Inc.*, 52:271–292, 1996.

[15] S. Gaubert and J. Mairesse. Task resource models and (max,+) automata. In J. Gunawardena Ed., editor, *Idempotency - Collection of the Isaac Newton Institute*, 1995.

[16] G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot. A linear system theoretic view of discrete event process and its use for performance evaluation in manufacturing. *IEEE Transactions on Automatic Control*, 30(3):210–220, 1985.

[17] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[18] R.C. Green. *Minimax Algebra*. Lectures Notes in Economics and Mathematical Systems. Springer-Verlag, 1979.

[19] J.C. Terrasson, S. Gaubert, and J. Gunawardena. Dynamics of min-max functions. Technical report, HP Laboratories Technical Report, 1997.

[20] S. Gaubert. Introduction aux systèmes dynamiques à Événements discrets. Notes de Cours, 1999.

[21] A. Gill. *Introduction to the Theory of Finite-State Machines*. McGraw-Hill Electronic Sciences Series. McGRAW-HILL Book Company, 1962.

[22] J.E. Hopcroft and J.D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley Publishing Company, 1969.

[23] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, USA, 1979.

[24] A. Salomaa. *Formal Languages*. ACM Monograph Series. Academic Press, Inc. New York, 1973.

[25] S. Pinchinat, H. Marchand, and M. Le Borgne. Symbolic abstractions of automata and their application to the supervisory control problem. Technical report, Institut National de Recherche en Informatique et en Automatique, 1999.

[26] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[27] J.L. Peterson. *Petri Net Theory and Modeling of Systems*. Prentice Hall, 1981.

[28] W. Reisig. *Petri Nets: an Introduction*. Springer-Verlag Berlin Heidelberg, 1985.

[29] W. Reisig. *A Primer in Petri Net Design*. Springer-Verlag, 1992.

[30] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, volume 1 : Basic Concepts. Springer-Verlag, 1992.

[31] F. DiCesare, G. Harhalakis, J.M. Proth, M. Silva, and F.B. Vernadat. *Practice of Petri Nets in Manufacturing*. Chapman and Hall, 1993.

[32] J.M. Proth and X. Xie. *Petri Nets: A Tool for Design and Management of Manufacturing Systems*. John Wiley - Sons Ltd, 1996.

[33] M.C. Zhou and F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers, 1993.

[34] A. Pnueli. The temporal semantics of concurrent programs. *18th Symposium on Foudations of Computer Science*, 1977.

[35] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, 1992.

[36] J.C. Bradfield. *Verifying Temporal Properties of Systems*. Progress in Theoretical Computer Science. Birkhäuser, 1992.

[37] A. Galton. *Temporal Logics and their Applications*. Academic Press, 1987.

[38] K. Heljanko. Model checking the branching time temporal logic ctl. Technical Report 45, Helsinki University of Technology - Otaniemi, Finland, May, 1997.

[39] J.S. Ostroff. *Temporal Logic for Real-Time Systems*. Research Studies Press Ltd., 1989.

[40] P. Racloz and D. Buchs. Symbolic proof of ctl formulae over petri nets. Technical report, C.U.I. University of Geneva, 1997.

[41] G.C. Barroso. *Uma Nova Abordagem para a Síntese de Supervisores de Sistemas a Eventos Discretos*. Tese de Doutorado. Universidade Federal da Paraíba, Campus II, Campina Grande, PB, Brasil, 1996.