V Simpósio Brasileiro de Automação Inteligente

Teoria de Controle Supervisório de Sistemas a Eventos Discretos

Prof. José Eduardo Ribeiro Cury
Universidade Federal de Santa Catarina
Departamento de Automação e Sistemas
cury@das.ufsc.br

Canela-RS, Novembro de 2001

Agradecimentos

Esta apostila foi elaborada a partir da minha experiência como professor, orientador e pesquisador na área de Sistemas a Eventos Discretos e Sistemas Híbridos, nos últimos dez anos, inicialmente no Departamento de Engenharia Elétrica e mais recentemente no Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina. Ela não poderia ter sido realizada sem o apoio direto ou indireto de todos os meus alunos de Graduação e Pós Graduação e orientados de Mestrado e Doutorado. Partes do documento devem inclusive ser creditadas a documentos que foram elaborados por estes. Neste sentido, agradeço a todos estes alunos, alguns hoje colegas de profissão, em particular ao Antonio, César, José Miguel, Max, Tati e Ziller, de quem "roubei"idéias, parágrafos e/ou figuras.

Sumário

Lista de Figuras

Capítulo 1

Introdução

A tecnologia moderna tem produzido, em escala crescente, sistemas com a finalidade de executar tarefas que, seja pela importância que adquirem em seu contexto, seja por sua complexidade e seu custo, justificam o esforço despendido na sua otimização e automação. Tais sistemas estão presentes em uma série de aplicações, incluindo por exemplo a automação da manufatura, a robótica, a supervisão de tráfego, a logística (canalização e armazenamento de produtos, organização e prestação de serviços), sistemas operacionais, redes de comunicação de computadores, concepção de software, gerenciamento de bases de dados e otimização de processos distribuídos. Tais sistemas têm em comum a maneira pela qual percebem as ocorrências no ambiente à sua volta, o que se dá pela recepção de estímulos, denominados eventos. São exemplos de eventos o início e o término de uma tarefa e a percepção de uma mudança de estado em um sensor. Estes eventos são, por sua natureza, instantâneos, o que lhes confere um caráter discreto no tempo. Sistemas com estas características são denominados sistemas a eventos discretos (SED), em oposição aos sistema de variáveis contínuas, tratados pela Teoria de Controle clássica. A natureza discreta dos SEDs faz com que os modelos matemáticos convencionais, baseados em equações diferenciais, não sejam adequados para tratá-los. Por outro lado, a sua importância faz com que seja altamente desejável encontrar soluções para problemas relacionados ao seu controle. Em razão disso, existe uma intensa atividade de pesquisa voltada à busca de modelos matemáticos adequados à sua representação, sem que se tenha conseguido até agora encontrar um modelo que seja matematicamente tão conciso e computacionalmente tão adequado como o são as equações diferenciais para os sistemas dinâmicos de variáveis contínuas. Não existe, por isso, consenso sobre qual seja o melhor modelo. Dentre os modelos existentes, destaca-se o proposto por Ramadge e Wonham, baseado em Teoria de

Linguagens e de Autômatos e denominado “modelo RW”. Este faz uma distinção clara entre o sistema a ser controlado, denominado planta, e a entidade que o controla, que recebe o nome de supervisor. A planta é um modelo que reflete o comportamento fisicamente possível do sistema, isto é, todas as ações que este é capaz de executar na ausência de qualquer ação de controle. Em geral, este comportamento inclui a capacidade de realizar determinadas atividades que produzam um resultado útil, sem contudo se limitar a esse comportamento desejado. Por exemplo, dois robôs trabalhando em uma célula de manufatura podem ter acesso a um depósito de uso comum, o que pode ser útil para passar peças de um ao outro. No entanto, cria-se com isso a possibilidade física de ocorrer um choque entre ambos, o que é, em geral, indesejável. O papel do supervisor no modelo RW é, então, o de exercer uma ação de controle restritiva sobre a planta, de modo a confinar seu comportamento àquele que corresponde a uma dada especificação. Uma vantagem desse modelo é a de permitir a síntese de supervisores, sendo estes obtidos de forma a restringir o comportamento da planta apenas o necessário para evitar que esta realize ações proibidas. Desta forma, pode-se verificar se uma dada especificação de comportamento pode ou não ser cumprida e, caso não possa, identificar a parte dessa especificação que pode ser implementada de forma minimamente restritiva. Um critério de aceitação pode então ser utilizado para determinar se, com a parte implementável da especificação, o sistema trabalha de maneira satisfatória. Neste documento serão apresentados os principais conceitos básicos da Teoria de Controle Supervisório, como introduzida por Ramadge e Wonham. Os conceitos de base da teoria de Linguagens e Autômatos serão apresentados, bem como os principais resultados básicos de síntese de supervisores para SEDs. A forma de apresentação procurará se adaptar a um curso que possa ser facilmente assimilado por alunos em nível de Graduação em cursos de Engenharia ou Ciências de Computação.

Capítulo 2

Sistemas a Eventos Discretos

2.1 Definição e Características

De um modo geral, um sistema é uma parte limitada do Universo que interage com o mundo externo através das fronteiras que o delimitam. Este conceito se aplica também aos sistemas tratados no presente documento, que apresentam ainda as características descritas a seguir. Os sistemas de interesse percebem as ocorrências no mundo externo através da recepção de estímulos, denominados eventos. São exemplos de eventos o início e o término de uma tarefa (mas não sua execução), a chegada de um cliente a uma fila ou a recepção de uma mensagem em um sistema de comunicação. A ocorrência de um evento causa, em geral, uma mudança interna no sistema, a qual pode ou não se manifestar a um observador externo. Além disso, uma mudança pode ser causada pela ocorrência de um evento interno ao próprio sistema, tal como o término de uma atividade ou o fim de uma temporização. Em qualquer caso, essas mudanças se caracterizam por serem abruptas e instantâneas: ao perceber um evento, o sistema reage imediatamente, acomodando-se em tempo nulo numa nova situação, onde permanece até que ocorra um novo evento. Desta forma, a simples passagem do tempo não é suficiente para garantir que o sistema evolua; para tanto, é necessário que ocorram eventos, sejam estes internos ou externos. Note ainda que a ocorrência desses eventos pode depender de fatores alheios ao sistema, de modo que este não tem, em geral, como prevê-los. O que se disse acima permite apresentar agora a seguinte definição:

Definição 2.1 *Sistema a eventos discretos (SED) é um sistema dinâmico que evolui de acordo com a ocorrência abrupta de eventos físicos, em intervalos de tempo em geral*

irregulares e desconhecidos.

Diz-se ainda que, entre a ocorrência de dois eventos consecutivos, o sistema permanece num determinado estado. A ocorrência de um evento causa então uma transição ou mudança de estado no sistema, de forma que sua evolução no tempo pode ser representada pela trajetória percorrida no seu espaço de estados, conforme ilustrado na figura 2.1.

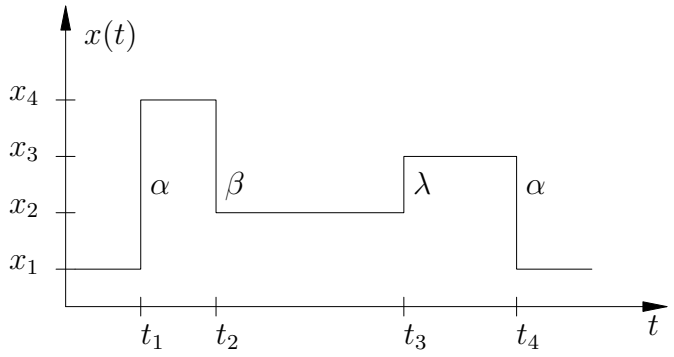

Figura 2.1: Trajetória típica de um sistema a eventos discretos

Nesta trajetória ocorrem eventos representados pelas letras α, β e λ. Vê-se que um mesmo evento pode ter efeitos diferentes, dependendo do estado em que ocorre. Por exemplo, se o sistema está no estado x_1 e ocorre o evento α, o próximo estado será x_4; se α ocorre em x_3, volta-se para x_1. A trajetória pode continuar indefinidamente, inclusive com a ocorrência de outros eventos, não representados na figura. Para todos os sistemas tratados, porém, assume-se que o número total de eventos diferentes que podem ocorrer é finito. O número de estados pode ser ilimitado no caso geral, embora a classe de sistemas com um número finito de estados seja um caso particular importante. Costuma-se distinguir um dos estados do sistema dos demais, o qual recebe o nome de estado inicial. Este é o estado em que o sistema se encontra antes de ocorrer o primeiro evento. Na prática, em geral é possível forçar um sistema a voltar a esse estado, antes de iniciar sua utilização para um determinado fim, processo denominado de reinicialização.

Os sistemas a eventos discretos, entendidos segundo a definição 2.1, contrastam com os sistemas dinâmicos a variáveis contínuas, descritos por equações diferenciais . É instrutivo comparar a trajetória típica de um SED, apresentada na figura 2.1, com a de um sistema dinâmico de variáveis contínuas, apresentada na figura 2.2.

O espaço de estados de um SED é limitado a um conjunto enumerável, ao passo que é contínuo e portanto infinito nos sistemas contínuos. Estes, em geral, mudam de estado a cada instante, tendo o seu comportamento descrito por uma função que relaciona o

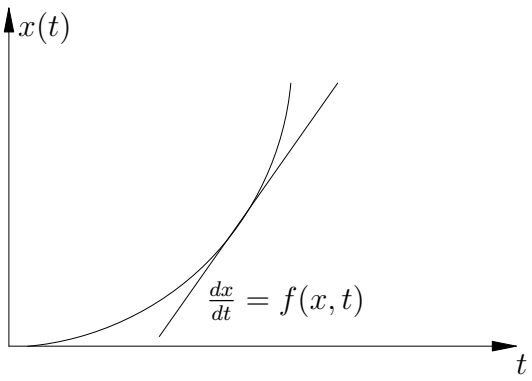

Figura 2.2: Trajetória típica de um sistema dinâmico com variáveis contínuas

estado (variável dependente) ao tempo (variável independente). Já os sistemas a eventos discretos podem permanecer um tempo arbitrariamente longo em um mesmo estado, sendo que sua trajetória pode ser dada por uma lista de eventos na forma $\{\sigma_1, \sigma_2, ...\}$, incluindo-se eventualmente os instantes de tempo em que tais eventos ocorrem, na forma $\{(\sigma_1, t_1), (\sigma_2, t_2), ...\}$. A quantidade de informação necessária depende dos objetivos da aplicação.

O acima exposto permite ver a tarefa de especificar o comportamento de um sistema a eventos discretos como sendo a de estabelecer seqüências ordenadas de eventos que levem à realização de determinados objetivos. Com uma formulação tão abrangente, não é surpreendente que tenha havido esforços em mais de uma área para abordar o problema. De fato a Teoria de Sistemas a Eventos Discretos é apresentada como pertencendo à área da Pesquisa Operacional, valendo-se ainda de resultados da Ciência da Computação (em particular da Inteligência Artificial e do Processamento de Linguagens), bem como da Teoria de Controle.

Foram desenvolvidos até o momento vários modelos para SEDs, sem que nenhum tivesse se afirmado como universal. Esses modelos refletem diferentes tipos de SEDs bem como diferentes objetivos na análise dos sistemas em estudo.

Os principais modelos utilizados para sistemas a eventos discretos são os seguintes:

- Redes de Petri com e sem temporização;
- Redes de Petri Controladas com e sem temporização;
- Cadeias de Markov;
- Teoria das Filas;

- Processos Semi-Markovianos Generalizados (GSMP) e Simulação;
- Álgebra de Processos;
- Álgebra Max-Plus;
- Lógica Temporal e Lógica Temporal de Tempo Real;
- Teoria de Linguagens e Autômatos (Ramadge-Wonham)

Dentre os modelos citados acima, dois apresentam uma característica particular: são dotados de procedimentos de síntese de controladores; são eles os modelos de Ramadge-Wonham (temporizados ou não), baseado na Teoria de Autômatos e/ou Linguagens, e o de Redes de Petri Controladas (temporizadas ou não). Pela características citada, estes modelos têm dado forte contribuição ao desenvolvimento da teoria de Controle de SEDs.

Os demais modelos citados servem sobretudo à análise de SEDs.

De todo modo, nenhum dos modelos serve atualmente como paradigma. Os SEDs formam uma área de pesquisa de intensa atividade e desafios.

2.2 Exemplos de Sistemas a Eventos Discretos

Nesta seção descreveremos alguns exemplos de SEDs. Iniciaremos por um sistema simples que serve como "módulo base"na representação de muitos SEDs de interesse.

I. Sistemas de Filas: Os Sistemas de Filas tem origem no seguinte fato comum intrínseco à maioria dos sistemas que projetamos e desenvolvemos: o uso de certos recursos exige espera. Os três elementos básicos de um sistema de filas são:

1. As entidades que devem esperar pelo uso de recursos. Costuma-se denominar estas entidades de *clientes*.
2. Os recursos pelos quais se espera. Como em geral estes recursos fornecem alguma forma de serviço aos clientes, são denominados *servidores*.
3. O espaço onde a espera se faz, denominado *fila*, ou em alguns casos, "buffers".

São exemplos de clientes:

- pessoas (esperando num Banco ou Parada de Ônibus);
- mensagens transmitidas através de um canal de comunicação;
- tarefas ou processos executados num sistema de computação;
- peças produzidas num sistema de manufatura;
- veículos numa rede rodoviária.

Do mesmo modo, são exemplos de servidores:

- pessoas (caixas de Banco ou Supermercado);
- canais de comunicação responsáveis pela transmissão de mensagens;
- processadores ou dispositivos periféricos em sistemas de computação;
- máquinas usadas na manufatura;
- vias em um sistema de tráfico.

Finalmente, são exemplos de filas:

- filas de bancos, supermercados, paradas de ônibus, etc.;
- buffers de chamadas telefônicas ativas, ou processos executáveis.

A motivação para o estudo de sistemas de filas está no fato de que em geral os recursos não são ilimitados. Isto gera problemas na alocação dos recursos e seus critérios conflitantes associados como: satisfação das necessidades dos clientes; utilização eficiente dos recursos, redução de custos. A figura 2.3 mostra um diagrama representativo de uma fila.

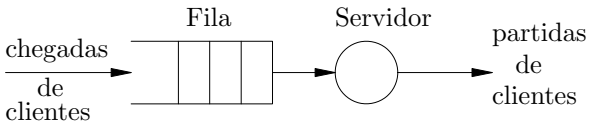

Figura 2.3: Filas

De modo a se especificar completamente as características de um sistema de filas deve-se ainda definir:

1. *processo de serviço*;
2. *capacidade da fila*;
3. *disciplina de atendimento*.

Visto como um SED, um sistema de filas tem $\Sigma = \{c, p\}$ onde c é o evento *chegada de cliente* e p o evento *partida de cliente*. Além disso, uma variável de estado natural é o número de clientes na fila, ou, comprimento da fila (por convenção, inclui o cliente em serviço). Portanto

$$X = \{0, 1, 2, ..., C + 1\}$$

onde C é a capacidade da fila.

Finalmente, os componentes de um sistema de filas como o descrito podem se conectar de diferentes formas, de modo a formar sistemas de redes de filas, onde os clientes fluem pelas filas e servidores da rede, de acordo com regras específicas, como pode ser visto na figura 2.4.

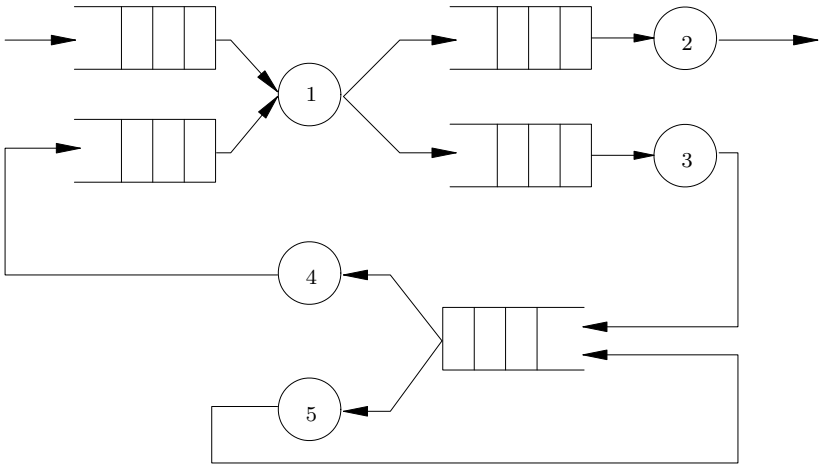

Figura 2.4: Redes de Filas

No exemplo, clientes deixando o servidor 1 devem seguir regras na escolha de uma das duas filas conectadas aos servidores 2 e 3; o servidor 1 deve adotar regras na seleção de uma das duas filas de entrada para receber o próximo cliente.

II. Sistemas de Computação: Num Sistema de Computação típico, tarefas ou processos são clientes competindo pela atenção de servidores como os vários processadores (CPU, impressoras, discos, ...). É muitas vezes conveniente representar tal sistema através de um modelo de rede de fila como o da figura 2.5.

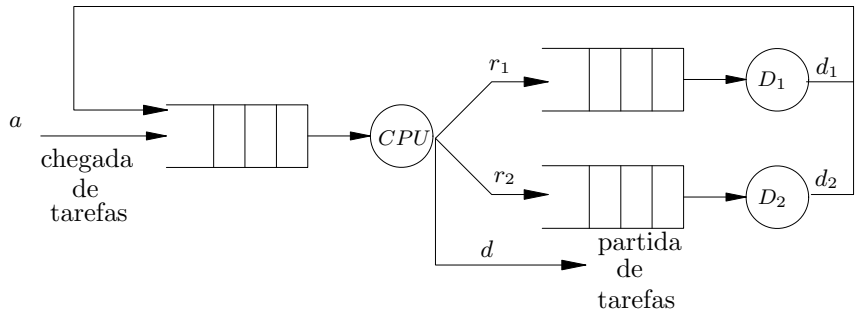

Figura 2.5: Sistema de Computação

No exemplo, tarefas chegam numa fila da CPU; uma vez servidas, estas partem ou requerem acesso a um de dois discos, para após retornar para mais um processamento pela CPU.

Tem-se:

$$\Sigma = \{a, d, r_1, r_2, d_1, d_2\}$$

onde

a corresponde à *chegada de uma tarefa ao sistema*;

d corresponde à *partida de uma tarefa do sistema*;

r_1, r_2 correspondem à *partida de tarefas da* CPU, roteadas aos discos D_1 e D_2 respectivamente;

d_1, d_2 correspondem à *partida de tarefas dos discos* D_1 *e* D_2 , respectivamente, retornando à fila da CPU.

Uma possível representação para o estado deste sistema é

$$x = (x_{CPU}, x_1, x_2)$$

correspondendo ao comprimento das três filas na CPU, disco 1 e disco 2.

O espaço de estados do sistema é

$$X = \{(x_{CPU}, x_1, x_2) : x_{CPU}, x_1, x_2 \geq 0\}$$

III. Sistemas de Comunicação: São sistemas muitas vezes descritos por modelos de filas com

- clientes: mensagens, chamadas;
- servidores: meios (canais) de comunicação, equipamentos de chaveamento (redes telefônicas);

Característica importante desses sistemas é a necessidade de mecanismos de controle que garantam o acesso aos servidores de modo eficiente e coerente. Esses mecanismos são muitas vezes chamados de protocolos, e podem ser bastante complexos. A validação e/ou projeto de protocolos são problemas interessantes de análise e síntese de controladores para SEDs.

Por exemplo, considere o caso de dois usuários A e B e um canal comum M, de capacidade 1 mensagem. O envio de duas mensagens (por A e B, p.e.) implica em sinal ininteligível (colisão).

Os estados possíveis do sistema são:

Para o canal M: I - vazio; T - transmitindo 1 mensagem; C - colisão;

Para cada usuário A ou B: I - repouso; T - transmitindo; W - aguardando com mensagem.

O espaço de estados pode então ser definido por

$$X = \{(x_M, x_A, x_B) : x_M \in \{I, T, C\} \text{ e } x_A, x_B \in \{I, T, W\}\}$$

e o conjunto de eventos que afetam o sistema é

$$\Sigma = \{\alpha_A, \alpha_B, \tau_A, \tau_B, \tau_M\}$$

onde

α_A, α_B correspondem à *chegada de mensagem a ser transmitida por A e B*, respectivamente;

τ_A, τ_B correspondem ao *envio de mensagem ao canal M por A e B*, respectivamente;

τ_M corresponde ao *término de uma transmissão pelo canal* (com 1 ou mais mensagens presentes).

Dois problemas podem se configurar neste exemplo:

1. possibilidade de colisão;

2. desconhecimento por cada usuário, do estado do outro usuário e/ou do estado do meio.

Esses problemas podem ser modelados como um problema de Controle Descentralizado, ou controle com restrição na estrutura de informação.

IV. Sistemas de Manufatura: São também sistemas muitas vezes convenientemente descritos por modelos de filas. Em geral, tem-se:

- clientes: peças ou ítens; "pallets";
- servidores: máquinas; dispositivos de manuseio e transporte (robôs, esteiras, ...);
- filas: armazéns; "buffers".

Considere por exemplo uma Linha de Transferência com duas Máquinas e um Armazém Intermediário de capacidade 2, conforme ilustrado na figura 2.6.

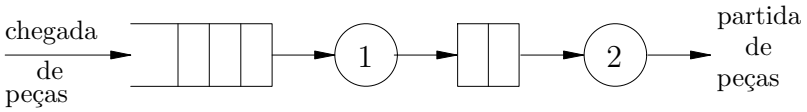

Figura 2.6: Linha de Transferência

Neste caso o conjunto de eventos que afetam o sistema pode ser descrito por

$$\Sigma = \{a, b, d_2\}$$

onde

a corresponde à *chegada da peça*;

b corresponde ao *término de serviço pela máquina 1*;

d_2 corresponde à *partida de peça da máquina 2.*

Considera-se que se a máquina termina um serviço e o armazém intermediário está cheio, esta entra em estado de bloqueio.

O espaço de estados do sistema pode ser representado por

$$X = \{(x_1, x_2) : x_1 \geq 0, x_2 \in \{0, 1, 2, 3, B\}\}$$

onde x_1 indica o estado do armazém de entrada da linha, e x_2 o estado do armazém intermediário; o estado onde $x_2 = B$ indica a situação de bloqueio, ou seja aquele estado onde o armazém intermediário está cheio e a máquina 1 tem uma peça terminada esperando para ser descarregada.

O espaço de estados X pode ainda ser representado por

$$X = \{(x_1, x_2) : x_1 \in \{I, W, B\}, x_2 \in \{I, W\}\}$$

onde x_i representa o estado da máquina i, com $x_1 = B$ indicando a situação de bloqueio do sistema.

V. Sistemas de Tráfego: Considere o exemplo da figura 2.7 que representa uma interseção semafórica. Tem-se 4 tipos de veículos segundo a direção que os mesmos podem tomar. Assim,

- $(1, 2)$: veículos de 1 para 2;
- $(1, 3)$: veículos de 1 para 3;
- $(2, 3)$: veículos de 2 para 3;
- $(3, 2)$: veículos de 3 para 2.

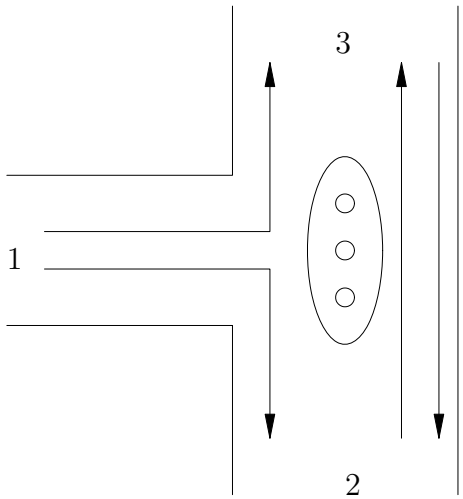

Figura 2.7: Sistema de Tráfego

Considera-se que o sinal verde libera os veículos $(2, 3)$ e $(3, 2)$; e que o sinal vermelho libera os veículos $(1, 2)$ e $(1, 3)$.

O conjunto de eventos que afetam o sistema é

$$\Sigma = \{a_{12}, a_{13}, a_{23}, a_{32}, d_{12}, d_{13}, d_{23}, d_{32}, g, r\}$$

onde

a_{ij} corresponde à chegada de veículo tipo (i, j);

d_{ij} corresponde à partida de veículo tipo (i, j);

g indica que o sinal torna-se verde;

r indica que o sinal torna-se vermelho.

O espaço de estados é

$$X = \{(x_{12}, x_{13}, x_{23}, x_{32}, y) : x_{ij} \geq 0, y \in \{G, R\}\}$$

onde

x_{ij} indica o número de veículos do tipo (i, j) em fila;

y indica o estado do sinal, verde (G) ou vermelho (R).

Capítulo 3

Um Problema Motivador

Neste capítulo será apresentado informalmente o problema de síntese de controladores para um SED, através de um exemplo de um sistema de manufatura. Pretende-se que este problema seja um agente motivador para os conceitos e metodologias relativos à Teoria de Controle Supervisório que serão abordados nos capítulos subseqüentes.

3.1 Linha de Produção de Cervejas

Na linha de produção de uma fábrica de cervejas, existe uma estação de envasilhamento baseada numa esteira com pallets fixados a cada metro e quatro máquinas dispostas em série de acordo com a figura 3.1. O funcionamento da estação de envasilhamento é comandado por um controlador lógico programável (CLP) que garante o envasilhamento de cada garrafa conforme a seguinte seqüência de passos:

1. o atuador avança, depositando uma garrafa vazia em P_1;
2. a esteira avança 1 metro;
3. a bomba enche a garrafa de cerveja;
4. a esteira avança 1 metro;
5. a garrafa é tampada;
6. a esteira avança 1 metro;

7. o robô retira a garrafa da esteira.

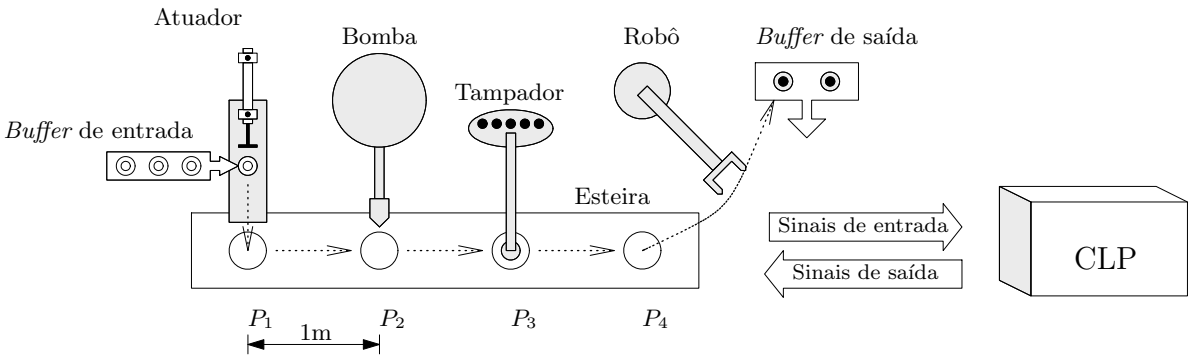

Figura 3.1: Estação de envasilhamento

A esteira foi inicialmente projetada para operar em seqüência apenas uma garrafa por vez, ou seja, o atuador só pode ser acionado novamente depois que o robô retirar a garrafa da esteira. Esta restrição na lógica de controle evita os problemas que podem ocorrer na operação de múltiplas garrafas em paralelo. Entretanto, esse modo de funcionamento é muito pouco eficiente, visto que o atuador, a bomba, o tampador e o robô passam a maior parte do tempo parados enquanto poderiam estar operando em paralelo. Um engenheiro de controle e automação industrial é contratado, então, para desenvolver uma lógica de controle que garanta uma maior eficiência da estação de envasilhamento. O técnico responsável pela manutenção da linha de produção tem bastante prática na programação de CLPs. Ele se dispõe a implementar o programa de controle caso lhe seja fornecida a lógica de funcionamento da estação, baseada nos sinais de entrada e saída do CLP apresentados a seguir.

- α_0: comando que inicia um avanço de 1 metro da esteira;
- β_0: sinal de final de operação da esteira. (Uma vez iniciada a operação, não pode ser evitado);
- α_1: sinal de comando do atuador pneumático, de início de depósito de uma garrafa no pallet da esteira situado na posição P_1;
- β_1: sinal de final de operação do atuador pneumático. (Uma vez iniciada a operação, não pode ser evitado);
- α_2: comando que inicia o enchimento da garrafa que estiver na posição P_2;

- β_2: sinal de final de operação da bomba automática. (Uma vez iniciada a operação, não pode ser evitado);
- α_3: comando que começa a tampar uma garrafa situada na posição P_3;
- β_3: sinal de final de operação do tampador automático. (Uma vez iniciada a operação, não pode ser evitado);
- α_4: comando que inicia a retirada de uma garrafa do pallet da esteira situado na posição P_4;
- β_4: Sinal de final de operação do robô. (Uma vez iniciada a operação, não pode ser evitado).

O programa deve ser tal que o sistema em malha fechada obedeça às seguintes restrições de coordenação:

1. não operar o atuador, a bomba, o tampador ou o robô enquanto a esteira estiver avançando;
2. não sobrepor garrafas em P_1;
3. não avançar a esteira sem que as garrafas em P_2, P_3 e P_4 tenham sido enchidas, tampadas ou retiradas, respectivamente;
4. não encher, tampar ou acionar o robô sem garrafas nas posições P_2, P_3 e P_4, respectivamente;
5. não encher ou tampar duas vezes a mesma garrafa;
6. não avançar a esteira à toa, ou seja, sem garrafa em alguma das posições.

O problema de controle que se coloca ao engenheiro pode então ser especificado como segue:

Problema 3.1 *Seja a* ***planta*** *de engarrafamento de cervejas, composta de um atuador pneumático, uma esteira, uma bomba de cerveja, um tampador, e um robô;* ***projetar uma lógica de controle*** *a ser implementada no CLP, de modo a permitir à esteira operar uma, duas, três ou quatro garrafas em paralelo; garantindo que o*

comportamento do sistema controlado obedeça a ***especificação*** *de funcionamento do sistema, de forma a evitar os problemas que podem ocorrer na operação de múltiplas garrafas em paralelo,* ***restringindo o sistema somente o necessário****, e garantindo* ***uma produção continuada*** *de cervejas.*

3.2 Considerações acerca da Resolução do Problema

O problema 3.1 tal como colocado, suscita algumas observações.

Primeiramente, se se deseja sintetizar uma lógica de controle é necessário que se conheça as características do sistema a controlar. Isto passa pela obtenção de um modelo para a planta, que possa ser obtido de forma sistemática e que seja adequado ao problema em questão. Neste sentido convém observar que a planta é em geral, como no caso do exemplo, composta de um conjunto de equipamentos ou sub-sistemas que devem trocar sinais com um elemento de controle de forma a que o comportamento coordenado destes equipamentos seja aquele desejado. A obtenção de um modelo para o sistema global a ser controlado pode então ser pensado como uma composição de modelos para seus sub-sistemas.

Do mesmo modo, a síntese de controle pressupõe um modelo adequado para as especificações de comportamento. Estas especificações são definidas por um conjunto de restrições que determinam como deve ser a operação coordenada dos equipamentos. Assim como para a planta a controlar, pode-se pensar que o modelo da especificação que define o comportamento global desejado para o sistema é o resultado da composição de um conjunto de modelos para cada especificação elementar definida sobre uma parte do sistema a controlar.

Ainda, o problema 3.1 estabelece que a lógica de controle deve ser ótima, no sentido de restringir o comportamento dos equipamentos que compõem a planta, somente o necessário para que não se violem as especificações. Além disso se deseja que o controle não leve o sistema a situações de bloqueio, ou seja, garanta o cumprimento de tarefas de modo perene.

Finalmente, espera-se que a lei de controle ótima não-bloqueante seja gerada automaticamente, ou seja, através de algoritmos bem definidos e que garantam as características consideradas acima.

O objetivo da teoria a ser desenvolvida nos capítulos que seguem é resolver problemas como o descrito acima. A colocação do problema justifica a metodologia a ser apresentada, como sendo composta das seguintes fases:

1. Obtenção de um modelo para a *planta* a ser controlada;
2. Obtenção de um modelo de representação das *especificações* a serem respeitadas;
3. *Síntese* de uma lógica de controle *não bloqueante* e *ótima*.

Por último, cabe observar que a natureza dos eventos envolvidos no problema descrito no exemplo é diversa. Enquanto uma parte destes eventos correspondem a sinais de comando que podem ou não serem ativados, uma outra parte dos eventos, por sua natureza, não podem ser evitados de ocorrer por qualquer ação de controle quando o estado da planta for tal que os habilite. Esta característica dos SEDs é fundamental no desenvolvimento da metodologia a ser apresentada.

3.3 Conclusão

Este capítulo apresentou de maneira informal, através de um exemplo de aplicação, o tipo de problema a ser tratado neste documento. Os capítulos que seguem apresentarão as bases conceituais da Teoria de Controle de Sistemas a Eventos Discretos.

Capítulo 4

Linguagens como modelos para SEDs

Neste capítulo serão introduzidos os elementos básicos da teoria de linguagens e será mostrado como o comportamento lógico de um SED pode ser modelado a partir de linguagens.

4.1 Notação e definições básicas

Uma *linguagem* L definida sobre um *alfabeto* Σ, é um conjunto de cadeias formadas por símbolos pertencentes a Σ.

Exemplo 4.1 *Considere o alfabeto* $\Sigma = \{\alpha, \beta, \gamma\}$. *Alguns exemplos de linguagens sobre* Σ *são:*

- $L_1 = \{\beta, \alpha, \alpha\beta\beta\}$
- $L_2 = \{$*Todas as possíveis cadeias formadas com 3 eventos iniciados pelo evento* $\alpha\}$

O conjunto de todas as possíveis cadeias finitas compostas com elementos de Σ é denotado Σ^*, incluindo a cadeia vazia, denotada por ε. Assim, uma linguagem sobre Σ é sempre um subconjunto (não necessariamente próprio) de Σ^*. Em particular $\emptyset$, Σ e Σ^* são linguagens.

Se $tuv = s$, com $t, u, v \in \Sigma^*$, então:

- t é chamado *prefixo* de s
- u é chamada uma *subcadeia* de s
- v é chamado *sufixo* de s

4.2 Operações sobre linguagens

Algumas operações podem ser executadas sobre linguagens. Algumas são usuais, como as operações sobre conjuntos; três outras operações serão aquí adicionadas.

1. *Concatenação*: Sejam duas linguagens $L_1, L_2 \subseteq \Sigma^*$, então a concatenação de L_1 e L_2, denotado L_1L_2, é definida por

$$L_1L_2 := \{s \in \Sigma^* : (s = s_1s_2)\ e\ (s_1 \in L_1)\ e\ (s_2 \in L_2)\}$$

   Em palavras, uma cadeia está em L_1L_2 se ela pode ser escrita como a concatenação de uma cadeia de L_1 com uma cadeia de L_2.

2. *Prefixo-Fechamento*: Seja uma linguagem $L \in \Sigma^*$, então, o prefixo-fechamento de L, denotado por $\overline{L}$, é definido por

$$\overline{L} := \{s \in \Sigma^* : \exists t \in \Sigma^*(st \in L)\}$$

   Em palavras, $\overline{L}$ consiste de todas as cadeias de Σ^* que são prefixos de L. Em geral, $L \subseteq \overline{L}$. L é dita prefixo-fechada se $L = \overline{L}$. Uma linguagem L é prefixo-fechada se qualquer prefixo de qualquer cadeia de L é também uma cadeia de L. Como veremos mais tarde linguagens *geradas* por sistemas físicos são exemplos de linguagens prefixo-fechadas.

3. *Fechamento-Kleene*: Seja uma linguagem $L \subseteq \Sigma^*$, então o fechamento Kleene de L, denotado por L^* é definido por

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

   Uma cadeia de L^* é formada pela concatenação de um número finito de cadeias de L, incluindo a cadeia vazia ε.

Exemplo 4.2 *Considere o alfabeto* $\Sigma = \{\alpha, \beta, \gamma\}$*, e as linguagens* $L_1 = \{\varepsilon, \alpha, \alpha\beta\beta\}$ *e* $L_2 = \{\gamma\}$ *definidas sobre* Σ*. Observe que tanto* L_1 *como* L_2 *não são prefixo-fechadas, pois* $\alpha\beta \notin L_1$ *e* $\varepsilon \notin L_2$*. Então:*

- $L_1 L_2 = \{\gamma, \alpha\gamma, \alpha\beta\beta\gamma\}$
- $\overline{L_1} = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta\beta\}$
- $\overline{L_2} = \{\varepsilon, \gamma\}$
- $L_1\overline{L_2} = \{\varepsilon, \alpha, \alpha\beta\beta, \gamma, \alpha\gamma, \alpha\beta\beta\gamma\}$
- $L_2^* = \{\varepsilon, \gamma, \gamma\gamma, \gamma\gamma\gamma, \cdots\}$

As seguintes observações são verdadeiras:

1. $\varepsilon \notin \emptyset$;
2. $\{\varepsilon\}$ é uma linguagem não vazia contendo somente a cadeia vazia. Veremos mais tarde que esta linguagem pode representar a situação de um sistema bloqueado em seu estado inicial;
3. Se $L = \emptyset$ então $\overline{L} = \emptyset$, e se $L \neq \emptyset$ então necessariamente $\varepsilon \in \overline{L}$;
4. $\emptyset^* = \{\varepsilon\}$ e $\{\varepsilon\}^* = \{\varepsilon\}$.

4.3 Representação de SEDs por linguagens

O comportamento de um sistema a eventos discretos (SED) pode ser descrito através de um par de linguagens. Para isto considera-se um alfabeto Σ como correspondendo ao conjunto de eventos que afetam o sistema. A evolução seqüencial do SED, ou seu *comportamento lógico*, pode então ser modelado através de uma dupla $D = (L, L_m)$.

No modelo D, $L \subseteq \Sigma^*$ é a linguagem que descreve o *comportamento gerado* pelo sistema, ou seja, o conjunto de todas as cadeias de eventos fisicamente possíveis de ocorrerem no sistema. Por sua vez $L_m \subseteq L$ é a linguagem que descreve o *comportamento marcado* do sistema, ou seja, o conjunto de cadeias em L que correspondem a tarefas completas que o sistema pode realizar.

Considerando a evolução seqüencial de um Sistema a Eventos Discretos e um alfabeto Σ correspondendo ao conjunto de eventos que afetam o sistema, pode-se afirmar que para que um sistema produza uma cadeia qualquer w, então o mesmo deve ter produzido anteriormente todos os seus prefixos. Portanto, o comportamento gerado de qualquer sistema a eventos discretos em que não ocorram eventos simultâneos, pode ser representado por uma linguagem prefixo fechada.

As observações acima podem ser sintetizadas formalmente nas seguintes propriedades de linguagens L e L_m que representam um SED:

1. $L \supset L_m$, ou seja, o comportamento gerado contém o comportamento marcado de um SED;
2. $L = \overline{L}$, ou seja, o comportamento gerado de um SED é prefixo-fechado.

Exemplo 4.3 *Como exemplo, considere o problema motivador 3.1 da Linha de Produção de Cervejas. Se se considera isoladamente a Esteira, pode-se identificar* $\Sigma = \{\alpha_0, \beta_0\}$ *como o conjunto de eventos associados ao equipamento. Neste caso a linguagem* L *que corresponde ao comportamento gerado pela Esteira consiste de todas as sequências de eventos que alternam os dois eventos considerados, iniciando com* α_0 *e finalizando com* α_0 *ou* β_0 *. Observe que* $\varepsilon \in L$ *correspondendo à situação da esteira em seu estado inicial. Por outro lado, se se considera como tarefa completa da esteira as cadeias que a levam ao estado de repouso, pode-se afirmar que* L_m *consiste de todas as cadeias de* L *que terminam com* β_0*, acrescida da cadeia* ε*. Assim,*

$$L = \{\varepsilon, \alpha_0, \alpha_0\beta_0, \alpha_0\beta_0\alpha_0, \alpha_0\beta_0\alpha_0\beta_0, \alpha_0\beta_0\alpha_0\beta_0\alpha_0, ...\}$$

e

$$L_m = \{\varepsilon, \alpha_0\beta_0, \alpha_0\beta_0\alpha_0\beta_0, \alpha_0\beta_0\alpha_0\beta_0\alpha_0\beta_0, ...\}$$

Exercício 4.1 *Considere o robô da figura 4.1. Este robô está inserido em um sistema de manufatura onde deve realizar as tarefas de retirar peças de uma esteira de entrada (evento b) e colocá-las em um de dois possíveis armazéns de saída (eventos* c_1 *e* c_2*). Considerando o robô como um SED, descreva as linguagens* L *e* L_m *do robô.*

Questão para reflexão: *Dado um SED* $= (L, L_m)$ *é sempre verdade que* $\overline{L_m} = L$*? Se sim justifique, caso contrário dê um contra-exemplo. Observe que a igualdade se verifica no caso do exemplo da esteira.*

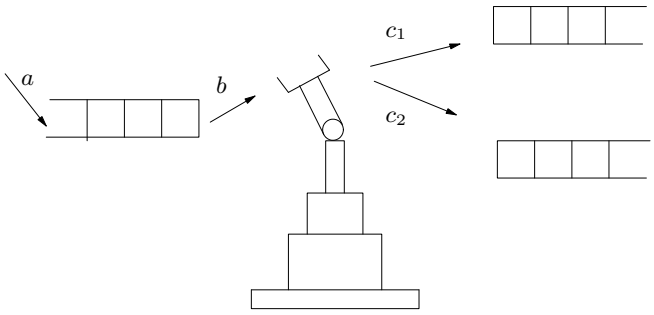

Figura 4.1: Sistema robô-esteira

4.4 Expressões Regulares

Como vimos acima, uma linguagem pode ser vista como uma maneira formal de descrever o comportamento de um SED. Ela pode especificar todas as possíveis seqüências de eventos que um SED pode gerar (L) ou o conjunto de tarefas que o sistema pode completar (L_m). No entanto, a descrição de uma linguagem como vista até agora, feita pela sua descrição em "linguagem natural"ou pela enumeração das cadeias que a definem, pode ser uma tarefa pouco prática. Seria conveniente que se pudesse dispor de uma forma de representação de linguagens que seja simples, concisa, clara e sem ambigüidade. Em outras palavras, é necessário utilizar estruturas compactas que possam representar estas linguagens. Neste documento serão apresentadas duas estruturas: as expressões regulares e os autômatos. Consideremos inicialmente as expressões regulares.

Para um alfabeto Σ dado, define-se recursivamente uma expressão regular da seguinte maneira:

1. (a) $\emptyset$ é uma expressão regular que representa a linguagem vazia,
   (b) ε é uma expressão regular denotando a linguagem $\{\varepsilon\}$,
   (c) σ é uma expressão regular denotando $\{\sigma\}$ $\forall e \in \Sigma$;
2. Se r e s são expressões regulares então rs, r^*, s^*, $(r + s)$ são expressões regulares;
3. Toda expressão regular é obtida pela aplicação das regras 1 e 2 um número finito de vezes.

Expressões regulares fornecem um meio de descrição de linguagens.

Exemplo 4.4 *Como ilustração, considerando o alfabeto $\Sigma_1 = \{a, b, g\}$, são exemplos de expressões regulares:*

- $(a+b)g^*$, *que representa a linguagem* $L = \{$*todas as cadeias que começam com a ou b e são seguidas por um número qualquer de* $g's\}$;
- $(ab)^*+g$, *que representa a linguagem* $L = \{$*todas as cadeias formadas por um número qualquer de* $ab's$ *ou uma ocorrência do elemento* $g\}$

Exemplo 4.5 *Se se retoma o exemplo da esteira da linha de produção de cervejas,* L *e* L_m *podem ser reescritos através de suas representações em expressões regulares respectivamente como*

$$L = (\alpha_0\beta_0)^*(\varepsilon + \alpha_0)$$

$$L_m = (\alpha_0\beta_0)^*$$

Exemplo 4.6 *Considere ainda como exemplo, um alfabeto* $\Sigma = \{a, b\}$*, composto por símbolos que representam eventos que correspondam ao acesso de duas tarefas diferentes a um mesmo recurso, por exemplo um processador. Pode-se encontrar expressões que representam uma restrição sobre o uso deste processador pelas tarefas. Deseja-se encontrar restrições que expressem justiça no uso do processador pelas tarefas. Possíveis expressões regulares para a especificação de justiça no uso do recurso são:*

- $A_1 = (ab)^*$, *expressa alternância no uso do recurso, porém privilegia o acesso a, no sentido de garantir a como primeiro acesso ao recurso;*
- $A_2 = (a+b)^*$, *não expressa justiça pois permite acesso irrestrito ao recurso;*
- $A_3 = (ab)^* + (ba)^*$, *expressa alternância, sem priorização no primeiro acesso.*

Exercício 4.2 *Para o exemplo de acesso a um recurso comum, existe uma restrição que expressa justiça de modo mais inteligente, ou seja, garante que em nenhum momento alguma tarefa faça mais do que 1 acesso a mais do que a outra. Encontre uma expressão regular para exprimir esta restrição de justiça.*

Exercício 4.3 *Encontre expressões regulares que representem as linguagens encontradas no problema do robô da figura 4.1.*

Questão para reflexão: *Toda linguagem é representável por uma expressão regular*?

4.5 Conclusão

Neste capítulo foram introduzidos conceitos relativos à teoria de linguagens e como se pode utilizar linguagens como meio de representar um SED. Neste sentido o modelo de representação de SEDs por *Linguagens* pode ser visto como um modelo *comportamental externo* do sistema uma vez que se baseia na descrição de suas trajetórias (sequências de eventos). Entretanto, a representação através de linguagens e expressões regulares é limitada computacionalmente. Para resolver este problema utiliza-se a representação de SEDs através de autômatos. Este fará objeto do próximo capítulo.

Capítulo 5

Autômatos como modelos para SEDs

Neste capítulo serão introduzidos os principais conceitos relacionados aos autômatos de estados finitos. Serão estabelecidas relações entre os autômatos e as linguagens e será considerada a questão de representação de um SED através de modelos de autômatos.

5.1 Autômatos Determinísticos de Estados Finitos

Um autômato determinístico de estados finitos (ADEF) é uma quíntupla $G = (X, \Sigma, f, x_0, X_m)$, onde:

- X é o conjunto finito de estados do autômato;
- Σ é o conjunto de símbolos (eventos) que definem o alfabeto;
- $f : X \times \Sigma \rightarrow X$ é a função de transição, possivelmente parcial, ou seja, não há necessidade da função ser definida para todo elemento de Σ em cada estado de X;
- x_0 é o estado inicial do autômato;
- X_m é o conjunto de estados marcados ou finais, $X_m \subseteq X$.

Um autômato pode ser representado graficamente como um grafo dirigido, onde os nós representam os estados e os arcos etiquetados representam as transições entre os estados. O estado inicial é identificado através de uma seta apontando para ele e os estados finais são representados com círculos duplos.

Exemplo 5.1 *A figura 5.1 é um exemplo de um autômato determinístico, cuja descrição formal correspondente é a seguinte:*

- $X = \{x, y, z\}$
- $\Sigma = \{a, b, g\}$
- *A função de transição do exemplo é:* $f(x, a) = x$, $f(x, g) = z$, $f(y, a) = x$, $f(y, b) = y$, $f(z, b) = z$ *e* $f(z, a) = f(z, g) = y$
- $x_0 = \{x\}$
- $X_m = \{x, z\}$

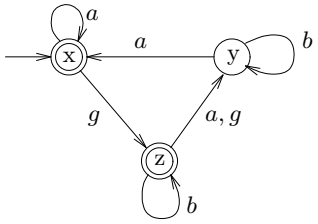

Figura 5.1: Autômato determinístico

Notação: $\Sigma_G(x)$ denota o conjunto ativo de eventos num estado $x \in X$, ou seja

$$\Sigma_G(x) = \{\sigma \in \Sigma : f(x, \sigma) \text{ é definida } \}.$$

No exemplo da figura 5.1, $\Sigma_G(x) = \{a, g\}$, $\Sigma(y) = \{a, b\}$ e $\Sigma_G(z) = \{a, b, g\}$.

O autômato G pode ser visto como um dispositivo que opera como segue. Inicia a partir do estado inicial x_0 e lá permanece até a ocorrência de um evento $\sigma \in \Sigma_G(x_0) \subseteq \Sigma$ que disparará a transição $f(x_0, \sigma) \in X$. Este processo continua baseado nas transições definidas em f.

A função f pode ser estendida do domínio $X \times \Sigma$ para o domínio $X \times \Sigma^*$, operando sobre cadeias, da seguinte maneira:

$$\begin{aligned} \hat{f}(x, \varepsilon) &:= x \\ \hat{f}(x, \sigma) &:= f(x, \sigma), \ \ \sigma \in \Sigma \\ \hat{f}(x, s\sigma) &:= f(\hat{f}(x, s), \sigma) \text{ para } s \in \Sigma^* \text{ e } \sigma \in \Sigma. \end{aligned}$$

Exemplo 5.2 *Aplicando a definição da função de transição estendida, $\hat{f}$, ao autômato da figura 5.1, tem-se o seguinte resultado:*

$$\begin{aligned}\hat{f}(y,\varepsilon) &= y\\ \hat{f}(x,gba) &= f(\hat{f}(x,gb),a) = f(f(\hat{f}(x,g),b),a) = f(f(z,b),a) = f(z,a) = y\end{aligned}$$

5.2 Linguagens representadas por autômatos

Um autômato G está associado a duas linguagens, a linguagem gerada $L(G)$ e a linguagem marcada $L_m(G)$.

A linguagem gerada por $G = (X, \Sigma, f, x_0, X_m)$ é:

$$L(G) := \{s \in \Sigma^* : \hat{f}(x_0, s) \text{ é definida}\}.$$

A linguagem marcada de G é:

$$L_m(G) := \{s \in L(G) : \hat{f}(x_0, s) \in X_m\}.$$

Em palavras, a linguagem $L(G)$ representa todas cadeias que podem ser seguidas no autômato, partindo do estado inicial. A linguagem $L_m(G)$ considera todas as cadeias que partindo do estado inicial chegam a um estado marcado.

Um SED pode ser modelado por um autômato G, onde $L(G)$ é o comportamento gerado pelo sistema e $L_m(G)$ é o comportamento marcado ou conjunto de tarefas completas do sistema.

Exercício 5.1 *Construa um autômato G tal que $L(G)$ e $L_m(G)$ correspondam aos comportamentos gerado e marcado do robô da figura 4.1.*

Exemplo 5.3 *A linguagem gerada do autômato da figura 5.2 é*

$$L(G) = [b^*aa^*b]^*(\varepsilon + b^*aa^*)$$

e sua linguagem marcada é

$$L_m(G) = [b^*aa^*b]^*b^*aa^*.$$

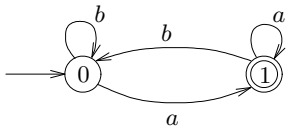

Figura 5.2: Exemplo de autômato

5.3 Linguagens Regulares e Autômatos de Estados Finitos

Iniciaremos esta seção retomando a questão deixada como reflexão, apresentada ao final da seção 4.4, ou seja, *"Toda linguagem é representável por uma expressão regular?"*. Para responder a esta questão consideremos o seguinte exemplo:

Exemplo 5.4 *O exemplo a ser analisado é o de uma fila de capacidade infinita. Clientes chegam e ficam na fila aguardando para usar o servidor. A figura 5.3 ilustra o problema. Como no exemplo* ***I*** *introduzido na seção 2.2, considera-se o conjunto de eventos* $\Sigma = \{c, p\}$ *onde* $\underline{c}$ *corresponde à chegada de cliente e* $\underline{p}$ *à partida de cliente do sistema.*

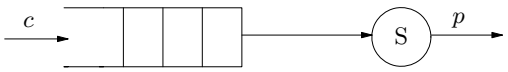

Figura 5.3: Problema fila infinita

A linguagem que corresponde ao comportamento gerado pelo sistema fila infinita é

$L = \{$*cadeias tais que quaisquer de seus prefixos não contenham mais p's do que c's*$\}$

Considerando como tarefas completas neste sistema de filas, as cadeias de eventos que deixam a fila vazia, pode-se observar que

$L_m = \{$*cadeias tais que o número de c's é igual ao número de p's e tais que quaisquer de seus prefixos não contenham mais p's do que c's*$\}$

O desafio de encontrar uma expressão regular para a linguagem descrita no exemplo da fila infinita é intransponível, ou seja, não existe tal expressão. De fato, o conjunto de linguagens para as quais existem expressões regulares que as representem constitui um conjunto particular de linguagens denominadas *Linguagens Regulares*.

Por outro lado, se se procura encontrar um autômato que tenha as linguagens L e L_m, respectivamente como linguagens gerada e marcada, não existe solução no domínio

dos autômatos de estados finitos. A figura 5.4 mostra um autômato que representa o comportamento da fila infinita.

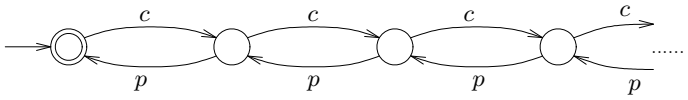

Figura 5.4: Autômato fila infinita

É possível perceber que o número de estados do autômato é infinito. De fato, o teorema a seguir estabelece uma relação entre o conjunto das Linguagens Regulares e o conjunto de Autômatos de Estados Finitos.

Teorema 5.1 ***(Teorema de Kleene)*** *Se L é regular, existe um autômato G com número finito de estados tal que $L_m(G) = L$. Se G tem um número finito de estados, então $L_m(G)$ é regular.*

Uma questão que se pode colocar é a seguinte: *Seja G um autômato com número infinito de estados, é sempre verdade que $L_m(G)$ é não regular?* A resposta a esta pergunta é **não**, pois pode existir um autômato finito que possua a mesma linguagem marcada que o autômato infinito dado.

A observação acima mostra que, em geral, para uma dada linguagem regular L, não existe um único autômato G tal que $L_m(G) = L$.

Autômatos G_1 e G_2 para os quais $L(G_1) = L(G_2)$ e $L_m(G_1) = L_m(G_2)$ são ditos serem *autômatos equivalentes*.

Por outro lado, dois autômatos são ditos serem *isomorfos* quando $G_1 = G_2$, a menos de uma renomeação de estados.

5.4 Acessibilidade e co-acessibilidade de um ADEF

De forma geral, um ADEF $G = (X, \Sigma, f, x_0, X_m)$ pode ter estados inacessíveis, isto é, estados que jamais podem ser alcançados a partir do estado inicial.

Formalmente, um estado $x \in X$ é dito ser *acessível* se $x = f(x_0, u)$ para algum $u \in \Sigma^*$.

G é dito ser *acessível* se x é acessível para todo $x \in X$.

A componente acessível, G_{ac}, de um autômato G é obtida pela eliminação de seus estados não acessíveis e das transições associadas a eles.

- $G_{ac} = (X_{ac}, \Sigma, f_{ac}, x_0, X_{m_{ac}})$
- X_{ac} é o conjunto de estados acessíveis de G
- $f_{ac} : f \mid \Sigma \times X_{ac}$
- $X_{m_{ac}} : X_{ac} \cap X_m$

Se o autômato é acessível então $G = G_{ac}$.

Por outro lado G é dito ser *co-acessível*, ou não bloqueante, se cada cadeia $u \in L(G)$ pode ser completada por algum $w \in \Sigma^*$ tal que $uw \in L_m(G)$, ou seja se cada cadeia $u \in L(G)$ for um prefixo de uma cadeia em $L_m(G)$.

Em outras palavras, um ADEF é co-acessível se, a partir de qualquer um de seus estados, existir ao menos um caminho que leve a um estado marcado.

A condição de co-acessibilidade de um ADEF pode ainda ser descrita pela equação.

$$L(G) = \overline{L_m(G)} \tag{5.1}$$

Assim como existe a componente acessível é possível identificar a componente co-acessível, G_{co}, de G, pela eliminação de estados não co-acessíveis (estados alcançados por cadeias que não podem ser completadas em tarefas) e transições associadas.

Quando um autômato é acessível e co-acessível ele é dito ser *trim*.

5.5 Bloqueio num SED

A equação (5.1) permite definir a idéia de ausência de bloqueio num sistema a eventos discretos.

Um sistema a eventos discretos com comportamento $L(G)$ e $L_m(G)$ é dito ser *não bloqueante*, **sse** satisfaz as condições da equação (5.1), isto é, $L(G) = \overline{L_m(G)}$.

Por outro lado um SED descrito por um autômato G que não satisfaz as condições da equação (5.1) será bloqueante. A condição de bloqueio ($L(G) \neq \overline{L_m(G)}$) corresponde à existência de cadeia(s) geradas pelo sistema ($u \in L(G)$), a partir da(s) qual(is) não se pode completar alguma tarefa no sistema ($u \notin \overline{L_m(G)}$).

Exemplo 5.5 *O gerador da Figura 5.5 modela um SED não bloqueante. De fato*

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

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

$$\overline{L_m(G)} = \varepsilon + \alpha + \beta + \alpha\alpha(\beta\alpha)^*(\varepsilon + \beta) = L(G)$$

G:

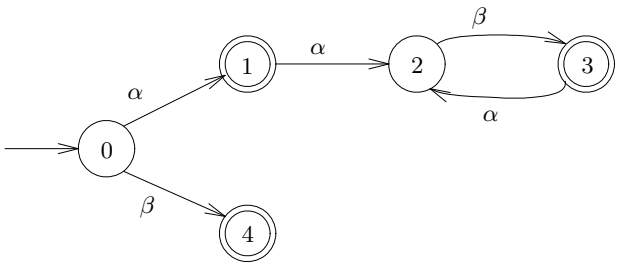

Figura 5.5: SED não bloqueante.

Observe que este exemplo satisfaz a condição de SED não bloqueante *($L(G) = \overline{L_m(G)}$) ainda que após o estado* 4 *nenhum evento esteja habilitado. De fato, o estado* 4 *corresponde a uma tarefa completa do sistema, o que caracteriza o não bloqueio. Este exemplo coloca em evidência a diferença entre a condição de bloqueio aqui definida e a condição conhecida como "deadlock".*

Exemplo 5.6 *Observe agora o autômato da figura 5.6. É possível perceber que neste autômato existem estados não-coacessíveis. De fato, a partir dos estados* 3, 4 *ou* 5, *o sistema não pode completar nenhuma tarefa, caracterizando a situação de bloqueio. Tal situação indica que o sistema está em "deadlock"(estado* 5*) ou "livelock"(estados* 3 *e* 4*).*

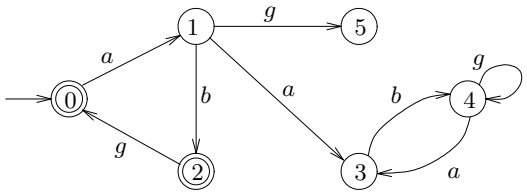

Figura 5.6: SED com bloqueio

No autômato da figura 5.6, de fato é possível identificar várias cadeias de G que o levam ao bloqueio, dentre elas:

- $ag \in L(G)$ *e* $ag \notin \overline{L_m(G)}$;
- $aa(bg^*a)^* \in L(G)$ *e* $aa(bg^*a)^* \notin \overline{L_m(G)}$;
- $aaba \in L(G)$ *e* $aaba \notin \overline{L_m(G)}$;
- $(abg)^*ag \in L(G)$ *e* $(abg)^*ag \notin \overline{L_m(G)}$.

Exercício 5.2 *Considere um sistema como o da figura 5.7, onde dois usuários $US1$ e $US2$ compartilham dois recursos $R1$ e $R2$(podem representar por exemplo duas máquinas que fazem acesso a duas ferramentas para a realização de uma operação). A operação básica do usuário $US1$ é definida pela sequência de eventos $a_{11}a_{12}c_1$ que indicam seu acesso aos recursos $R1$ (evento a_{11}) e $R2$ (evento a_{12}), nesta ordem, seguido da devolução de ambos os recursos (evento c_1). Já o usuário $US2$ tem como operação básica a sequência $a_{22}a_{21}c_2$, indicando seu acesso aos recursos $R2$ (evento a_{22}) e $R1$ (evento a_{21}), nesta ordem, seguido da devolução de ambos (evento c_2). Considere que cada o sistema opera sob uma lógica de controle que atua sobre cada usuário evitando o acesso a um recurso não disponível no momento.*

1. *Modelar por um autômato, o comportamento global do sistema sob a lógica de controle descrita;*
2. *Verifique se existe bloqueio;*
3. *Se a resposta for sim proponha uma nova lógica de controle que solucione o problema de bloqueio.*
4. *Represente o comportamento resultante através de um novo autômato.*

Exercício 5.3 *Para o conjunto de eventos $\Sigma = \{a, b\}$, encontrar autômatos não-bloqueantes G, tais que*

1. $L_m(G) = (a + b)^*ab$;
2. $L_m(G) = a^*bb^*a + a^*a$.

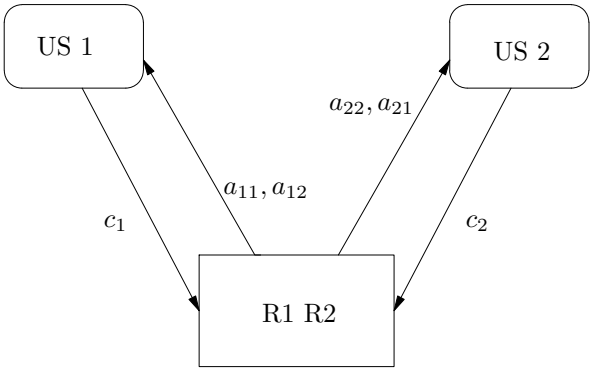

Figura 5.7: Sistema usuário-recurso

5.6 Autômatos não determinísticos

Considere a linguagem $(a + b)^*ab$ do exercício 5.3. Encontrar um ADEF para esta linguagem pode não ser uma tarefa tão simples, apesar da aparente simplicidade da linguagem em questão. De fato, embora o autômato da figura 5.8 pareça ser um candidato natural a solução ao exercício, ele não satisfaz as condições da definição de ADEF, já que a ocorrência de um evento num estado (evento a no estado 0) causa a transição para mais de um novo estado (estados 0 ou 1). Autômatos com a característica do autômato da figura 5.8 correspondem a uma nova classe de autômatos denominada *Autômatos Não Determinísticos de Estado Finito* (ANDEF).

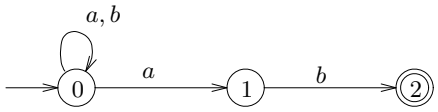

Figura 5.8: Autômato para $L_m(G) = (a + b)^*ab$

Formalmente, um ANDEF é uma quíntupla $G = (X, \Sigma, f, x_0, X_m)$, onde X, Σ, x_0, X_m são definidos como anteriormente, e f é uma função de transição

$$f : X \times \Sigma \rightarrow 2^X$$

onde 2^X é o conjunto de subconjuntos de X, também chamado conjunto das partes de X.

ANDEFs podem ser uma alternativa na modelagem de SEDs, pela simplicidade de obtenção dos mesmos (vide caso do exemplo) ou como meio de exprimir a falta de informações sobre o sistema que se está modelando. Uma questão que se coloca é sobre o poder de expressão de ANDEFs em relação aos ADEFs, ou seja,

Questão: *É possível que um ANDEF reconheça linguagens que não são reconhecíveis por algum ADEF?*

O teorema seguinte responde à questão acima:

Teorema 5.2 *Toda linguagem representável por um autômato não determinístico de estados finitos (ANDEF) possui uma representação por um autômato determinístico de estados finitos (ADEF).*

Corolário 5.1 *A todo autômato não determinístico corresponde um autômato determínistico equivalente, ou seja, que reconhece a mesma linguagem.*

A seção seguinte introduz um algorítmo que calcula, para um dado, ANDEF, um ADEF equivalente, ou seja, tal que suas linguagens geradas e marcadas sejam iguais.

5.7 Determinização de um ANDEF

O algoritmo para determinização de um autômato não determinístico será apresentado a seguir. A partir de um ANDEF $G = (X, \Sigma, x_0, f, X_m)$ constrói-se um ADEF $G^D = (X^D, \Sigma, x_0^D, f^D, X_m^D)$, onde:

- $X^D = 2^X$
- $x_0^D = X_0$
- $f^D(x^D, \sigma) = \bigcup_{x \in X} f(x, \sigma)$
- $X_m^D = \{x^D \in X^D \mid x^D \cap X_m \neq \emptyset\}$

Aplicando o algoritmo ao exemplo da figura 5.8, encontra-se o autômato da figura 5.9. O autômato G^D fica definido da seguinte forma:

- $X^D = 2^X = \{\phi, \{0\}, \{1\}, \{2\}, \{1,2\}, \{0,1\}, \{0,2\}, \{0,1,2\}\}$;
- $x_0^D = \{0\}$

- Função de transição f^D: $f^D(\{0\}, a) = \{0, 1\}$, $f^D(\{0\}, b) = \{0\}$, $f^D(\{0, 1\}, a) = \{0, 1\}$, $f^D(\{0, 1\}, b) = \{0, 2\}$, $f^D(\{0, 2\}, a) = \{0, 1\}$ e $f^D(\{0, 2\}, b) = \{0\}$ (aqui já foram excluídos os estados não alcançáveis);
- $X_m^D = \{0, 2\}$

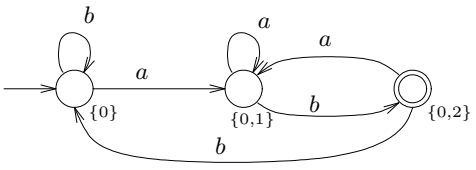

Figura 5.9: Autômato determinístico

Pode-se verificar a equivalência de G e G^D.

5.8 Minimização de autômatos

As seções anteriores mostraram a não unicidade de autômatos associados a linguagens regulares dadas. Não existe portanto um modelo único para representar um dado SED. No entanto, é certamente desejável, por razões computacionais, ou mesmo para maior legibilidade do modelo, que se trabalhe com autômatos que tenham o menor número possível de estados. É portanto importante que se procure meios de encontrar em um dado autômato, estados que sejam de certo modo redundantes (ou equivalentes) e que possam ser agregados em um único estado.

Tendo um autômato $G = (X, \Sigma, f, x_0, X_m)$ é possível fazer algumas observações:

1. Se $x \in X_m$ e $y \notin X_m$, x e y não podem ser equivalentes;
2. Se $f(x, \sigma) = f(y, \sigma)$ para todo $\sigma \in \Sigma$ então x e y são equivalentes.

O algoritmo a seguir identifica estados equivalentes em um dado ADEF G. A agregação destes estados permite obter um ADEF equivalente ao original, com número mínimo de estados. Este autômato é unico, a menos de possível isomorfismo.

Ao final, os pares de estados não marcados pelo algorítmo definem uma partição do espaço de estados X em classes de estados equivalentes.

Algorítmo para identificação de estados equivalentes

Marcar (x, y) para todo $x \in X_m$, $y \notin X_m$
Para todo par (x, y) não marcado anteriormente:
se $(f(x, \sigma), f(y, \sigma))$ está marcado para algum $\sigma \in \Sigma$ ou não está definido para uma só das componentes **então**
  Marcar (x, y)
  Marcar todos os pares não marcados (w, z) na lista de (x, y).
  Repetir para cada par (w, z) até que nenhuma marcação seja possível.
fim se
se $(f(x, \sigma), f(y, \sigma))$ não está marcado para nenhum $\sigma \in \Sigma$ **então**

  se $f(x, \sigma) \neq f(y, \sigma)$ **então**
    acrescentar (x, y) à lista de $(f(x, \sigma), f(y, \sigma))$
  fim se
fim se

Exemplo 5.7 *O autômato da figura 5.10 detecta cadeias terminadas com a seqüência de dígitos* 123*. Ele pode ser visto como um dispositivo que lê dígitos em sequência e sinaliza cada vez que a sequência* 123 *é detectada. As linguagens gerada e marcada do autômato são, respectivamente* $L = (1 + 2 + 3)^*$ *e* $L_m = (1 + 2 + 3)^*123$*. A aplicação do algoritmo acima permite identificar os estados* 0*,* 2 *e* $(1, 3)$ *como estados equivalentes. A agregação destes estados leva ao autômato mínimo equivalente mostrado na figura 5.11.*

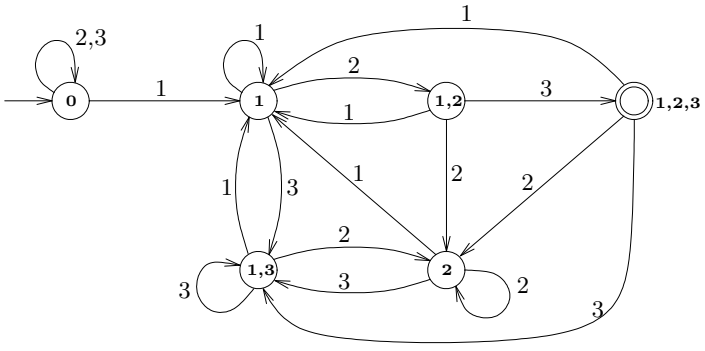

Figura 5.10: Autômato que detecta seqüência 123

5.9 Composição de Autômatos

A modelagem de um SED por ADEFs, pode ser em princípio abordada de duas maneiras: uma abordagem global e uma abordagem local.

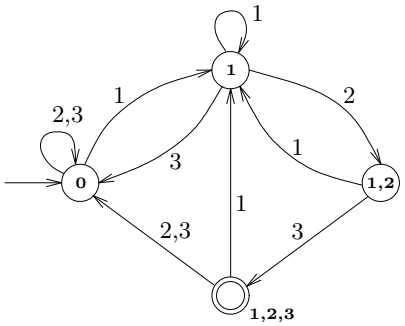

Figura 5.11: Autômato mínimo que detecta seqüência 123

Na abordagem global o sistema é analisado como um todo e procura-se por um ADEF que represente todas as seqüências possíveis de eventos que ele pode gerar e tarefas que pode completar. Para sistemas de maior porte, esta pode ser uma tarefa de grande complexidade. Além disso, qualquer alteração no sistema, por exemplo, pela inclusão ou retirada de equipamentos, ou modificação em sua lógica de controle, requer a reconstrução do modelo como um todo.

Exemplo 5.8 *Voltemos ao exemplo da figura 5.7, abordado no exercício 5.2. Considerando o comportamento do sistema sob a lógica de controle inicialmente descrita no exercício, pode-se construir, a partir da análise global do sistema, o modelo ADEF conforme mostrado na figura 5.12.*

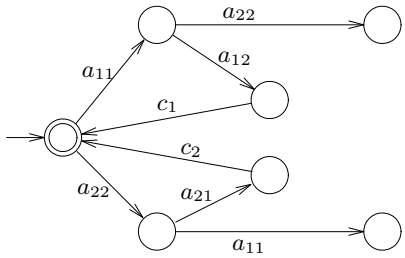

Figura 5.12: Autômato sistema usuário-recurso

Por outro lado, como já discutimos, em geral um SED pode ser visto como a composição de *subsistemas*, atuando em geral sob ações de um controlador que impõe *restrições de coordenação* ao sistema. A abordagem local de modelagem parte do princípio de que se pode construir modelos locais para cada sub-sistema e restrição de coordenação, e que se pode compor os mesmos para obter um modelo do sistema global. Uma abordagem de modelagem localizada, sugere maior facilidade na obtenção de modelos de sistemas

de grande porte. Além disso, permite pressupor que alterações num subsistema ou em alguma restrição somente exigirão uma mudança no modelo específico correspondente.

Exemplo 5.9 *Voltando ao exercício 5.2 (usuário de recursos) pode-se aplicar a idéia da modelagem local aqui introduzida. Constrói-se modelos para os usuários e para a restrição de coordenação separadamente. Os autômatos das figuras 5.13 e 5.14 são modelos para o comportamento isolado dos usuários $US1$ e $US2$, respectivamente.*

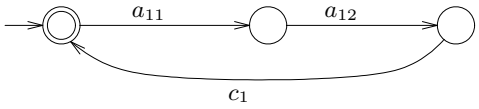

Figura 5.13: Modelo usuário US1

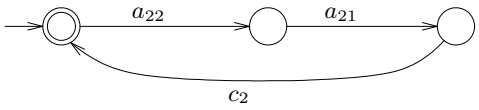

Figura 5.14: Modelo usuário US2

A restrição de coordenação implícita na lógica inicialmente implementada pode ser traduzida por: "Se o recurso $R1$ estiver em uso por algum dos usuários (ocorrência de a_{11} ou a_{21}), o mesmo somente poderá ser acessado após ser devolvido (ocorrência de c_1 ou c_2); idem para o recurso $R2$". Estas restrições podem ser modeladas pelos autômatos das figuras 5.15 (recurso R1) e 5.16 (recurso R2)

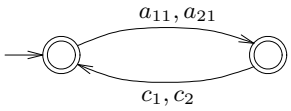

Figura 5.15: Restrição recurso R1

Supondo que se possa abordar a modelagem localmente, o que seria necessário modificar se fosse introduzido mais um recurso do tipo R1 no sistema? Bastaria modificar o autômato que representa a restrição de coordenação do recurso R1, como mostra a figura 5.17.

A aplicabilidade da abordagem local para modelagem de SEDs por ADEF é garantida pela operação de Composição de Autômatos, como definida a seguir.

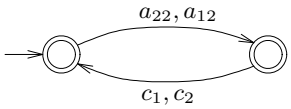

Figura 5.16: Restrição recurso R2

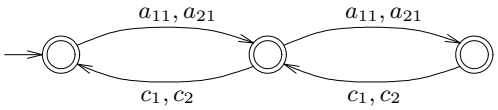

Figura 5.17: Restrição recurso R1 modificada

Dados dois autômatos $G_1 = (X_1, \Sigma_1, f_1, x_{0_1}, X_{m_1})$ e $G_2 = (X_2, \Sigma_2, f_2, x_{0_2}, X_{m_2})$, define-se a *composição síncrona* $G_1 \| G_2$ como:

$$G_1 \| G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\|2}, (x_{0_1}, x_{0_2}), X_{m_1} \times X_{m_2})$$

onde:

$$f_{1\|2} : (X_1 \times X_2) \times (\Sigma_1 \cup \Sigma_2) \rightarrow (X_1 \times X_2)$$

Ou seja:

$$f_{1\|2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)) & \text{se } \sigma \in \Sigma_1 \cap \Sigma_2 \text{ e } \sigma \in \Sigma_1(x_1) \cup \Sigma_2(x_2) \\ (f_1(x_1, \sigma), x_2) & \text{se } \sigma \in \Sigma_1 \text{ e } \sigma \notin \Sigma_2 \text{ e } \sigma \in \Sigma_1(x_1) \\ (x_1, f_2(x_2, \sigma)) & \text{se } \sigma \in \Sigma_2 \text{ e } \sigma \notin \Sigma_1 \text{ e } \sigma \in \Sigma_2(x_2) \\ \text{indefinida} & \text{caso contrário} \end{cases}$$

Um evento comum a Σ_1 e Σ_2 só pode ser executado sincronamente nos dois autômatos; os demais ocorrem assincronamente, ou seja, de modo independente em cada autômato. Se os alfabetos são iguais $\Sigma_1 = \Sigma_2$, a composição é completamente síncrona, isto é, todos os eventos estão sincronizados. No caso oposto, $\Sigma_1 \cap \Sigma_2 = \emptyset$, não existe nenhuma sincronização entre os eventos dos dois autômatos.

É possível ressaltar algumas propriedades da composição síncrona:

- $G_1 \| G_2 = G_2 \| G_1$

- $(G_1 \| G_2) \| G_3 = G_1 \| (G_2 \| G_3)$
- A definição pode ser estendida para n autômatos.

Exercício 5.4 *Efetue a composição dos autômatos do exemplo 5.9. Compare o resultado obtido com o autômato da figura 5.12. Obtenha o modelo do sistema para o caso do recurso* $R1$ *duplicado.*

5.10 Conclusão

Este capítulo introduziu um modelo de estados para SEDs, baseado nos Autômatos de Estados Finitos. Os conceitos introduzidos permitem desenvolver uma abordagem para a modelagem destes sistemas. No capítulo que segue será desenvolvida uma metodologia para síntese de leis de controle para SEDs, baseada na base conceitual até aqui introduzida.

Capítulo 6

Controle Supervisório de SEDs

O objetivo deste capítulo é formular, formalmente, o Problema de Controle Supervisório de SEDs, e apresentar sua resolução.

6.1 Introdução

Como visto no capítulo 3, o sistema a ser controlado ou *planta* corresponde, em geral a um conjunto de sub-sistemas (equipamentos) arranjados segundo uma distribuição espacial dada. Estes sub-sistemas, vistos isoladamente, têm cada um, um comportamento básico original que, quando atuando em conjunto com os demais sub-sistemas, deve ser restringido de forma a cumprir com a função coordenada a ser executada pelo sistema global. A composição dos comportamentos de cada sub-sistema isolado pode então ser identificado com a *planta* G, com comportamentos gerado e marcado $L(G)$ e $L_m(G)$, respectivamente. Assume-se aqui que G é modelado por um ADEF. A notação G então será usada indistintamente para se referenciar à planta ou a seu modelo ADEF.

O conjunto de restrições de coordenação define uma *especificação* E a ser obedecida, e pode ser interpretado como segue. As linguagens $L(G)$ e $L_m(G)$ contem cadeias indesejáveis de eventos por violarem alguma condição que se deseja impor ao sistema. Pode ser o caso de estados proibidos em G, por provocarem bloqueio ou por serem inadmissíveis como o caso de uma colisão de um robô com um AGV, em um sistema de manufatura. Pode ainda ser o caso de cadeias que violam um ordenamento desejado para os eventos, como por exemplo no caso de justiça no acesso a recursos.

De modo a fazer com que os sub-sistemas atuem de forma coordenada, introduz-se um agente de controle denominado *supervisor*, denotado por S. Em nosso paradigma, considera-se que o supervisor S interage com a planta G, numa estrutura de malha fechada (figura 6.1) onde S observa os eventos ocorridos em G e define que eventos, dentre os fisicamente possíveis de ocorrerem no estado atual, são permitidos de ocorrerem a seguir. Mais precisamente, S tem uma ação desabilitadora de eventos e, neste sentido diz-se que S é um controle de natureza *permissiva*. O conjunto de eventos habilitados num dado instante pelo supervisor define uma entrada de controle. Esta é atualizada a cada nova ocorrência de evento observada em G.

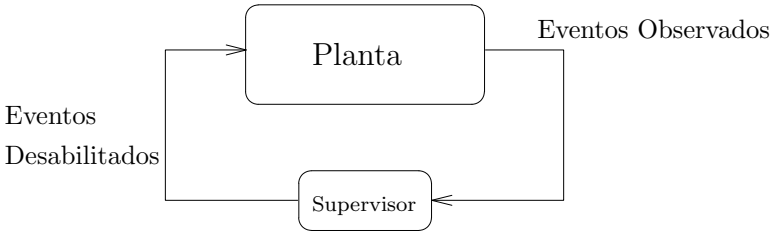

Figura 6.1: SED em malha fechada.

Considera-se ainda que o conjunto de eventos que afetam a planta G é particionado num conjunto de eventos desabilitáveis, ou controláveis, e um conjunto de eventos cuja natureza não permite a desabilitação.

Finalmente, será considerado aqui que o supervisor pode desmarcar estados marcados da planta G, ou seja, não reconhecer como tarefa do sistema em malha fechada, uma cadeia que corresponde a uma tarefa do sistema em malha aberta.

A seguir, o problema de controle será reapresentado, agora formalmente, juntamente com a noção de supervisor.

6.2 O problema de controle

Dada uma planta representada por um ADEF G com comportamento dado pelo par de linguagens $(L(G), L_m(G))$ definidas sobre um conjunto de eventos Σ, define-se uma estrutura de controle Γ para G, pelo particionamento de Σ em

$$\Sigma = \Sigma_c \dot{\cup} \Sigma_u$$

onde

Σ_c é o conjunto de *eventos controláveis*, que podem ser inibidos de ocorrer no sistema, e

Σ_u é o conjunto de *eventos não controláveis*, que não podem ser inibidos de ocorrer no sistema.

Uma opção de controle $\gamma \in \Gamma$ aplicada ao sistema contém o conjunto ativo de eventos habilitado a ocorrer no sistema. Com a definição da controlabilidade de eventos, a estrutura de controle toma a forma

$$\Gamma = \{\gamma \in 2^{\Sigma} : \gamma \supseteq \Sigma_u\}$$

onde a condição $\gamma \supseteq \Sigma_u$, indica simplesmente que os eventos não controláveis não podem ser desabilitados.

Um supervisor pode ser representado por um autômato S, definido sobre o mesmo alfabeto Σ, cujas mudanças de estado são ditadas pela ocorrência de eventos na planta G. A ação de controle de S, definida para cada estado do autômato, é desabilitar em G os eventos que não possam ocorrer em S após uma cadeia de eventos observada. O funcionamento do sistema controlado S/G pode ser descrito pelo SED resultante da composição síncrona de S e G, isto é, $S\|G$. De fato, na composição síncrona $S\|G$ somente as transições permitidas tanto no sistema controlado G, como no supervisor S são permitidas.

O comportamento do sistema em malha fechada é então dado por

$$L(S/G) = L(S\|G) \quad \text{e} \quad L_m(S/G) = L_m(S\|G)$$

O Problema de Controle Supervisório pode ser formalmente apresentado como segue.

Problema 6.1 (PCS) *Dada uma planta G, com comportamento $(L(G), L_m(G))$ e estrutura de controle Γ, definidos sobre o conjunto de eventos Σ; e especificações definidas por $A \subseteq E \subseteq \Sigma^*$; encontre um supervisor não bloqueante S para G tal que*

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

No problema em questão, as especificações A e E definem limites superior e inferior

para o comportamento do sistema em malha fechada. Observe que a exigência de não bloqueio imposta ao supervisor pode ser expressa pela condição

$$L(f/D) = \overline{L_m(f/D)}.$$

6.3 Controlabilidade e Solução do PCS

Essencial para a solução do problema de controle supervisório é o conceito de controlabilidade de linguagens. Dada uma planta G, com comportamento $(L(G), L_m(G))$ e estrutura de controle Γ, definidos sobre o conjunto de eventos Σ e a linguagem $E \subseteq L(G)$, E é dita ser *controlável* com respeito a G, ou simplesmente controlável, se

$$\overline{E}\Sigma_u \cap L \subseteq \overline{E}.$$

O seguinte resultado é fundamental para a solução do problema de controle supervisório como estabelecido anteriormente.

Proposição 6.1 *Dada uma planta G, com comportamento $(L(G), L_m(G))$ e estrutura de controle Γ, definidos sobre o conjunto de eventos Σ e $K \subseteq L_m(G)$, existe um supervisor não bloqueante S para G tal que*

$$L_m(S/G) = K,$$

se e somente se K for controlável.

Para uma linguagem $E \subseteq \Sigma^*$, seja $C(E)$ o conjunto de linguagens controláveis contidas em E. Pode-se provar que $C(E)$ possui um elemento supremo $\sup C(E)$. O seguinte teorema fornece a solução para o problema de controle supervisório.

Teorema 6.1 *O problema de controle supervisório possui solução se e somente se*

$$\sup C(E) \supseteq A.$$

No caso das condições do teorema 6.1 serem satisfeitas, $\sup C(E)$ representa o comportamento menos restritivo possível de se implementar por supervisão no siste-

ma G, satisfazendo as especificações A e E. Assim, o supervisor ótimo S é tal que $L_m(S/G) = \sup C(E)$.

6.4 Conclusão

Este capítulo apresentou formalmente o PCS e sua solução. O conceito de linguagem controlável e a existência do elemento supremo do conjunto de linguagens controláveis contidos numa dada linguagem que representa a especificação de controle, são chaves na resolução do problema. No capítulo seguinte será mostrado como se aplicam estes resultados na síntese de supervisores não bloqueantes ótimos.

Capítulo 7

Metodologia para a síntese de supervisores ótimos

A metodologia básica para a síntese de um supervisor ótimo que resolve o problema 6.1 foi inicialmente proposta por Ramadge e Wonham (RW), e se baseia em três passos, como introduzido no capítulo 3. Estes passos serão desenvolvidos a seguir, à luz dos resultados apresentados no capítulo 6.

Para isso, retomamos os passos tais como introduzidos no capítulo 3.

1. Obtenção de um modelo para a *planta* a ser controlada;
2. Obtenção de um modelo de representação das *especificações* a serem respeitadas;
3. *Síntese* de uma lógica de controle *não bloqueante* e *ótima*.

7.1 Obtenção de um modelo para a planta

Este passo pode ser implementado, aplicando-se a abordagem local para modelagem de SEDs, como apresentada na seção 5.9. Assim, a planta a ser controlada pode ser obtida como segue.

1. Identificar o conjunto de sub-sistemas ou equipamentos envolvidos no sistema a controlar;

2. Construir o modelo básico ADEF G_i, de cada equipamento i envolvido no sistema;

3. Obter o modelo da planta a ser controlada, através da composição síncrona de todos os ADEF G_i.

4. Definir a estrutura de controle Γ, pela identificação dos conjuntos de eventos controláveis e não controláveis Σ_c e Σ_u, que afetam o sistema.

OBS: Representa-se um evento controlável nos diagramas dos autômatos, por um pequeno corte nos arcos correspondentes às transições deste evento.

Exemplo 7.1 (Estação de Envasilhamento) *Considere a estação de envasilhamento apresentada no capítulo 3. Neste exemplo, pode-se identificar os seguintes sub-sistemas:*

- G_0*: Esteira;*
- G_1*: Atuador Pneumático;*
- G_2*: Bomba;*
- G_3*: Tampador;*
- G_4*: Robô.*

No nível de abstração que nos interessa, pode-se modelar cada um destes equipamentos pelos autômatos da figura 7.1.

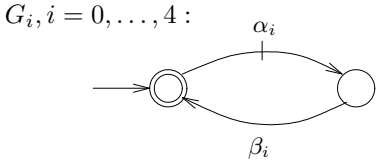

Figura 7.1: Autômatos para $G_i, i = 0, \ldots, 4$.

Neste caso é natural a identificação dos eventos controláveis e não controláveis como sendo

$$\Sigma_c = \{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4\},$$

correspondendo ao conjunto de comandos de início de operação dos equipamentos e

$$\Sigma_u = \{\beta_0, \beta_1, \beta_2, \beta_3, \beta_4\},$$

correspondendo aos sinais de sensores, de final de operação de cada equipamento.

O modelo da planta global G (não mostrado aqui), com 32 estados, é obtido pela composição dos autômatos $G_i, i = 0, ...4$.

Exemplo 7.2 (Pequena Fábrica) *Considere agora o exemplo de uma Linha de Transferência composta de duas máquinas M_1 e M_2 e um armazém intermediário de capacidade 1 (figura 7.2).*

Figura 7.2: Linha de transferência

Considera-se que o comportamento básico de cada máquina, sem considerar a possibilidade de quebra, é o seguinte: ao receber um sinal de comando α_i, a máquina M_i carrega uma peça e realiza uma operação sobre a mesma; ao finalizar a operação (sinal β_i) a máquina descarrega automaticamente a peça. O modelo de cada equipamento (M_1 e M_2) é então aquele descrito na figura 7.3. No caso

$$\Sigma_c = \{\alpha_1, \alpha_2\}, \quad \Sigma_u = \{\beta_1, \beta_2\}.$$

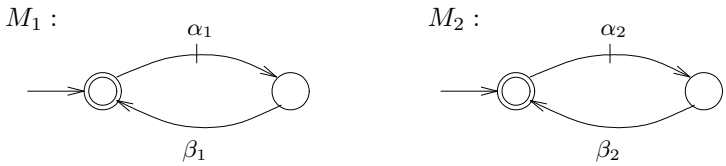

Figura 7.3: Modelo das máquinas M_1 e M_2

O modelo da planta G, obtido pela composição dos modelos de cada máquina, é aquele mostrado na figura 7.4.

7.2 Obtenção de um modelo para a especificação

A obtenção da linguagem $E \subset L_m(G)$ que restringe o comportamento do sistema a um comportamento que atenda aos objetivos de projeto, também pode feita seguindo uma abordagem local de modelagem. De fato, em geral as restrições de coordenação para o sistema são múltiplas e a modelagem de cada uma é mais natural e simples de ser

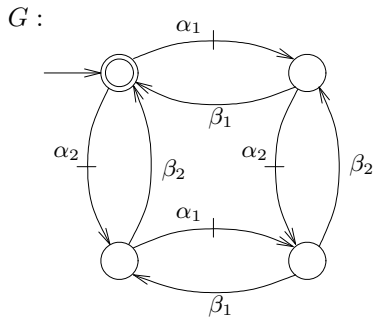

Figura 7.4: Modelo da planta.

feita separadamente. Os passos para a obtenção da especificação global E, seguindo uma abordagem local, são os seguintes.

1. Construir autômatos E_j para cada restrição de coordenação j, do sistema a ser controlado;
2. Realizar a composição dos autômatos construídos no passo 1; compor o autômato resultante com o autômato da planta G, gerando o autômato R;
3. Atualizar R pela eliminação, caso houver, de estados considerados proibidos;
4. Atualizar R pelo cálculo de sua componente co-acessível, ou não bloqueante.

A especificação global E é dada por

$$E = L_m(R).$$

Algumas considerações podem ser feitas acerca de cada passo do procedimento acima.

O passo 1 corresponde à identificação e modelagem das restrições de coordenação a serem impostas ao sistema. Em geral, cada uma delas diz respeito a um sub-conjunto dos sub-sistemas ou equipamentos do sistema. Além disso elas podem ser muitas vezes descritas por um conjunto pequeno de eventos do sistema, e não necessitam considerar o comportamento global de cada equipamento. Por exemplo quando se modela a justiça no acesso de tarefas a um recurso (exemplo 4.6), considera-se tão somente os eventos a e b, sem necessidade de levar em conta o comportamento global de cada tarefa.

A composição das restrições de coordenação, feita no passo 2, leva a uma restrição global de coordenação. Para que esta restrição seja reescrita levando em conta o compor-

tamento da planta, faz-se necessário sua composição com G. O autômato R resultante é tal que

$$L_m(R) \subset L_m(G).$$

O autômato R pode conter estados proibidos (indicando por exemplo uma colisão) que devem então ser eliminados, como no passo 3. Após este passo pode-se afirmar que

$$L_m(R) = E.$$

Finalmente, o passo 4 reflete a idéia de que não se deseja que a especificação inclua situações de bloqueio. Ao final deste passo, tem-se

$$L_m(R) = E \subset L_m(G) \quad e \quad L(R) = \overline{L_m(R)},$$

correspondendo ao modelo final para a especificação E.

Exemplo 7.3 *Retomando-se o exemplo 7.2 da pequena fábrica, considere a restrição de coordenação que impede que haja "overflow"(M_1 descarrega peça no armazém já cheio) ou "underflow"(M_2 inicia processo de carga com o armazém vazio). Esta restrição pode ser modelada pelo autômato E_1 da figura 7.5, que expressa a idéia de que se deve alternar β_1 e α_2, iniciando-se pelo primeiro.*

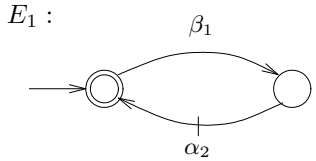

Figura 7.5: Especificação de não overflow e não underflow do buffer.

Na figura 7.5, o fato de se marcar o estado inicial indica que somente se deseja considerar tarefa completa no sistema controlado, as cadeias que correspondam a situações em que o armazém está vazio. Observe que esta condição não é garantida pela planta G da figura 7.4.

O autômato R que representa a especificação E, obtido pela composição do autômato E_1 com G, é mostrado na figura 7.6.

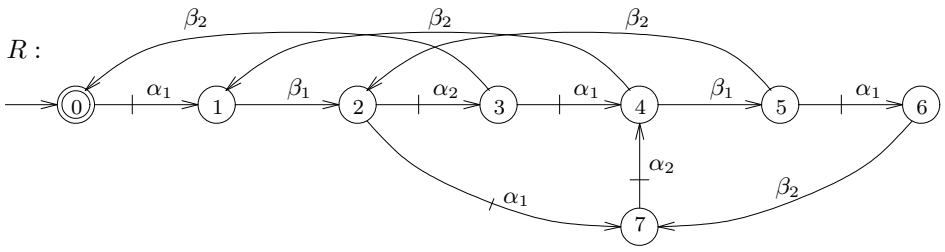

Figura 7.6: Autômato para E.

7.3 Síntese do supervisor não bloqueante ótimo

O último passo na resolução do PCS é a síntese do supervisor S que implementa a lógica não bloqueante ótima, no sentido de menos restritiva possível. Caso $E \subset L_m(G)$ seja controlável, a proposição 6.1 garante que existe um supervisor não bloqueante tal que $L_m(S/G)$. No entanto, nem sempre E atende a condição de controlabilidade, o que torna necessário que se calcule a linguagem controlável que mais se aproxime de E, e que não contenha cadeias indesejáveis de eventos. Esta linguagem é dada por $\sup C(E)$ e representa a lógica ótima de supervisão.

O cálculo de $\sup C(E)$ baseia-se num procedimento iterativo que identifica "maus estados"no autômato R que modela E, obtido no segundo passo da metodologia, como descrito na seção 7.2.

Algorítmo de cálculo de $\sup C(E)$

Dados $G = (X, \Sigma, f, x_0, X_m)$ e $R = (Q, \Sigma, f_R, q_0, Q_m)$, obtidos como descrito nas seções 7.1 e 7.2, tais que $L_m(R) = E \subset L_m(G)$;

1. Identificar maus estados em R; caso não existam faça $S = R$, fim.
2. Caso existam, atualizar R por eliminação dos maus estados;
3. Calcular a componente trim de R e voltar ao passo 1.

7.3.1 Definição de maus estados

É possível observar que o autômato R é obtido inicialmente pela composição de G com os autômatos que definem as restrições de coordenação (seção 7.2). Isso implica

que cada estado $q \in Q$ de R pode ser visto como um estado composto $(x, .)$ que aponta univocamente para um estado x de G, tal que se s é uma seqüência de eventos tais que

$$f_R(q_0, s) = q$$

então:

$$x = f(x_0, s)$$

As figuras 7.7, 7.8 ilustram idéia acima.

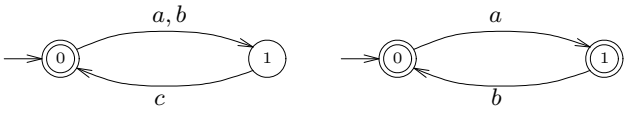

Figura 7.7: Autômatos G e $C = \|E_i$

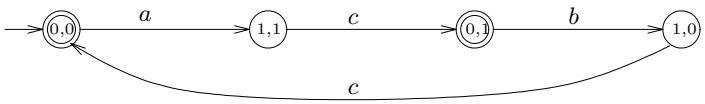

Figura 7.8: Autômato $R = G\|C$

Um estado $q = (x, .)$ de R é mau estado se existe $\sigma \in \Sigma_u$ tal que $\sigma \in \Sigma_G(x)$ e $\sigma \notin \Sigma_R(q)$.

7.4 Implementação / Realização do supervisor

O autômato $S = (\Omega, \Sigma, f_S, \omega_0, \Omega_m)$ obtido no algoritmo descrito anteriormente é tal que $L_m(S) = \sup C(E)$. Na realidade S pode ser usado para gerar um programa de controle que implementa a supervisão desejada, através, por exemplo, de um programa "jogador de autômatos", da seguinte forma:

> Na ocorrência de um evento, o jogador de autômatos atualiza o estado de S, segundo sua função de transição e, no estado destino ω, os eventos $\sigma \notin \Sigma_S(\omega)$ são desabilitados na planta. O comportamento resultante em malha fechada é $L_m(S)$ $(\overline{L_m(S)} = L(S))$.

Na verdade, S não é, em geral, o único autômato que pode ser usado como supervisor. De fato, qualquer autômato S' tal que $L_m(S'\|G) = L_m(S)$ e $L(S'\|G) = L(S)$ pode também ser usado como autômato para implementar a supervisão.

Exercício 7.1 *Considere o autômato G da figura 7.9. Aplique o algoritmo da metodologia apresentada quando a restrição de coordenação é alternar os eventos b e d, considerados controláveis. Neste caso G já é a planta livre. Discuta a ação de controle. Este é o supervisor mais simples que implementa uma ação de controle?*

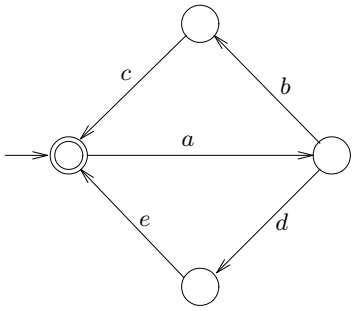

Figura 7.9: Autômato G (exercício 7.1)

Exemplo 7.4 *Retomando o exemplo 7.2, da "pequena fábrica", a partir de G (figura 7.4) e R (figura 7.3), pode-se calular S como na figura 7.10.*

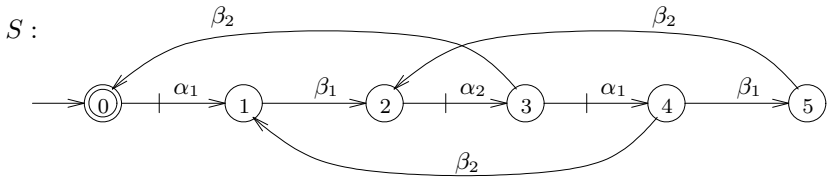

Figura 7.10: Máxima linguagem controlável.

Este autômato é diferente do autômato R que representa E, indicando a não controlabilidade desta linguagem. S mostra, por exemplo, a ação de controle que impede a máquina M_1 de iniciar uma operação quando o armazém estiver cheio (evento α_1 no estado 2 do supervisor). Observe que este evento não está desabilitado na especificação E (estado 2 de R). Esta ação é necessária para impedir de forma indireta a ocorrência do evento não controlável β_1.

7.5 Exemplos

Esta seção apresenta alguns exemplos do uso da metodologia desenvolvida.

Exemplo 7.5 *Considere novamente o exemplo da "pequena fábrica"mostrada na figura 2.6. No entanto, considere agora que se deseja construir um supervisor que possa reagir automaticamente possíveis quebras nas máquinas. Para isso se supões que o agente de supervisão tem acesso à informação de quebra, embora, obviamente não possa desabilitá-las. Neste caso os modelos de comportamento individual das máquinas devem ser alterados para incluir o estado de quebra. A figura 7.11 mostra modelos de autômatos para o comportamento das máquinas, que podem estar em repouso, trabalhando ou quebradas. O modelos inclui os eventos não controláveis de quebra das máquinas (λ_1 e λ_2), bem como os eventos de reparo das mesmas (μ_1 e μ_2), estes controláveis. Do modelo se pode observar que se assume que as máquinas só quebram quando em operação, e que, quando reparadas elas voltam ao estado de repouso, com o descarte das peças.*

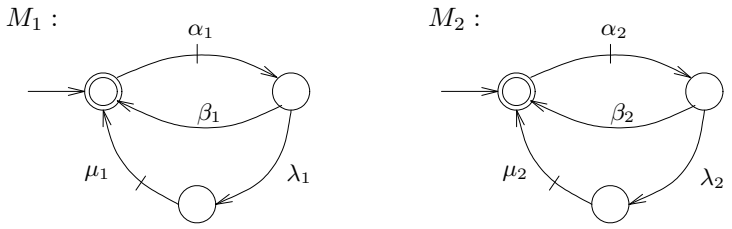

Figura 7.11: Modelo das máquinas com quebra M_1 e M_2

Deve-se projetar um supervisor para atender as seguintes restrições de coordenação:

1. *Impedir "overflow"e "underflow"do armazém;*
2. *A prioridade de reparo é da máquina M_2, ou seja, caso M_1 e M_2 encontrem-se ambas quebradas, M_2 deve ser reparada primeiro.*

A figura 7.12 mostra autômatos que modelam essas restrições. O autômato $R = E_1 \| E_2 \| G$ não é mostrado.

Finalmente, a figura 7.13 mostra um autômato para S, que implementa o comportamento $\sup C(E)$.

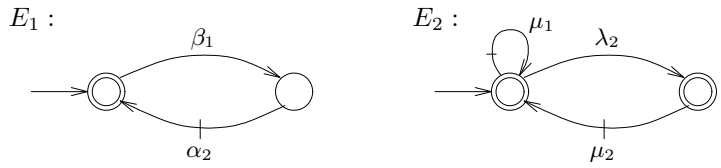

Figura 7.12: Não overflow e underflow do armazém, e prioridade de reparo de M_2.

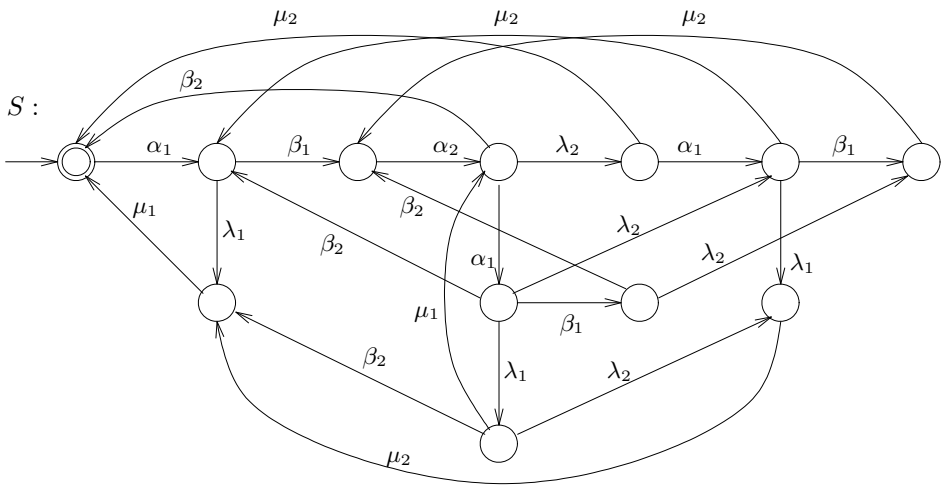

Figura 7.13: Máxima linguagem controlável.

Exemplo 7.6 (Linha de Transferência com Retrabalho) *Considere agora a linha de transferência da figura 7.14, onde existem duas máquinas, M_1 e M_2, e uma unidade de teste TU, separadas por armazéns B_1 e B_2. A unidade de teste TU realiza um teste de controle de qualidade sobre as peças (evento 5) e, segundo o resultado seja "peça boa"(evento 60) ou "peça ruim"(evento 80), descarrega a peça (evento 62) ou retorna a mesma ao armazém B_1, para retrabalho em M_2. A figura 7.15 mostra modelos de autômatos para cada equipamento do sistema.*

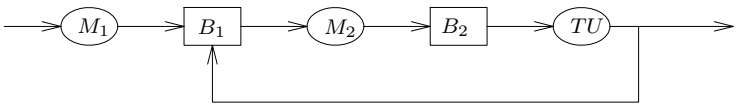

Figura 7.14: Linha de transferência

As restrições de coordenação são dadas pelo "não overflow"ou "não underflow"dos armazéns B_1 e B_2, considerados ambos de capacidade 1. Modelos para estas restrições são apresentados na figura 7.16. O autômato R não é mostrado.

A figura 7.17 mostra a solução do problema.

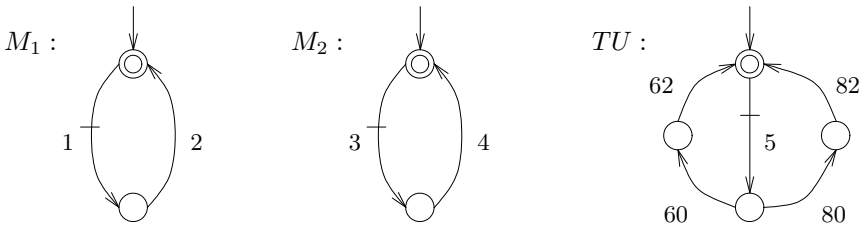

Figura 7.15: Modelo dos componentes do sistema.

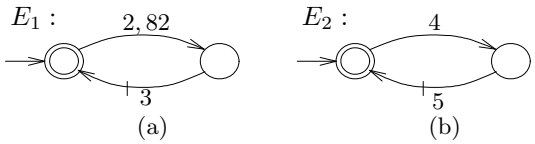

Figura 7.16: Especificação de não overflow e não underflow dos armazéns: (a) B_1 e (b) B_2.

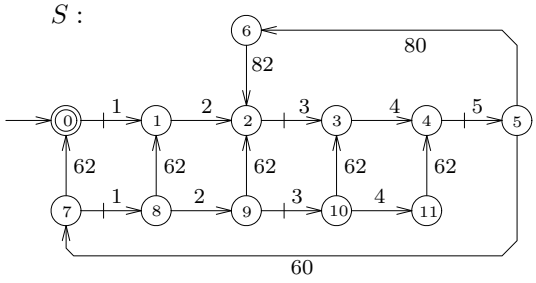

Figura 7.17: Lei de controle ótima para a linha com retrabalho.

Exemplo 7.7 *Considere agora o sistema mostrado na figura 7.18, formado por duas máquinas, um AGV e uma esteira. As máquinas depositam peças no AGV, e ele leva de uma máquina para outra ou coloca na esteira.*

Os eventos considerados não controláveis são

$$\Sigma_u = \{b, d\},$$

os eventos de descarga da peça no AGV.

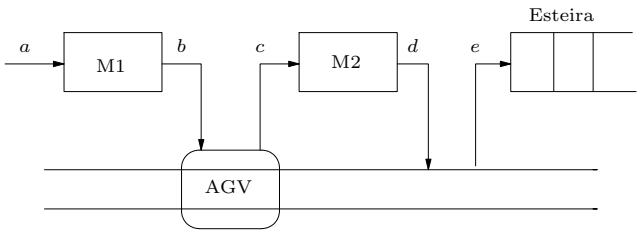

Figura 7.18: Sistema AGV

Os modelos de autômatos para as máquinas e para o AGV são mostrados nas figuras 7.19 e 7.20, respectivamente.

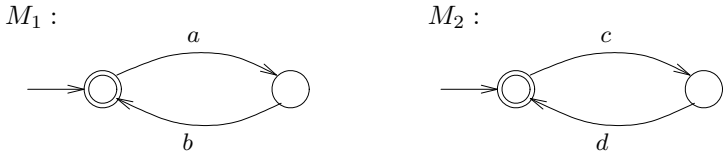

Figura 7.19: Modelo das máquinas M_1 e M_2

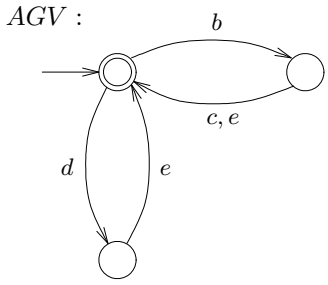

Figura 7.20: Modelo do AGV

A figura 7.21 mostra o modelo da composição entre M_1 e M_2, e a figura 7.22 mostra o modelo da planta $G = M_1 \| M_2 \| AGV$.

As restrições de coordenação são:

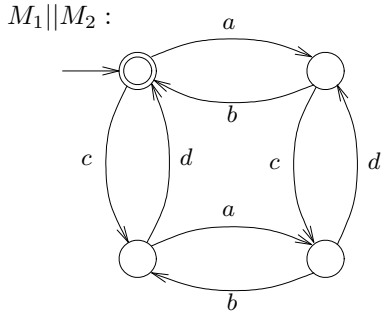

Figura 7.21: Modelo de $M_1||M_2$

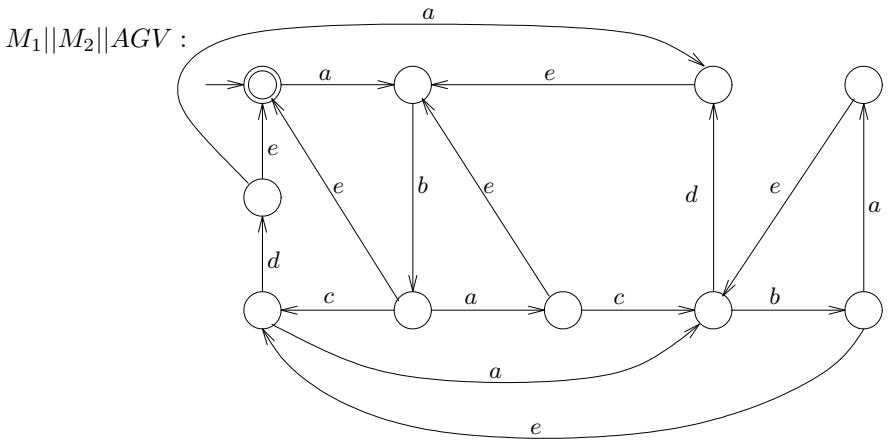

Figura 7.22: Modelo de $M_1||M_2||AGV$

1. *As peças devem ser processadas na ordem M_1M_2;*

2. *Evitar o bloqueio.*

A figura 7.23 mostra um modelo para a primeira restrição, proibindo o evento $\underline{e}$ quando o AGV é carregado com uma peça de M_1 (evento $\underline{b}$). A segunda restrição é naturalmente resolvida pelo procedimento de síntese.

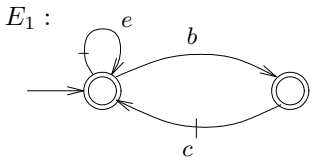

Figura 7.23: Restrição E_1

A especificação final E, modelada por R é mostrada na figura 7.24.

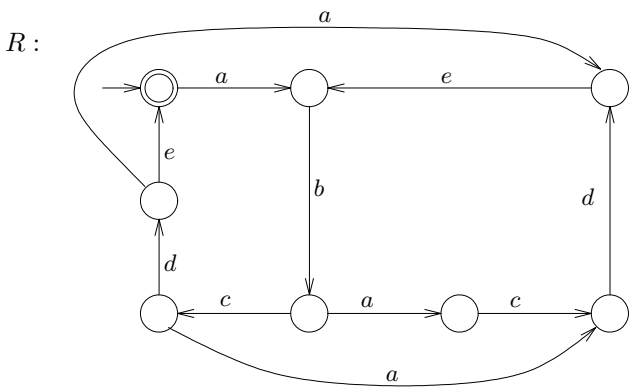

Figura 7.24: Autômato para E

A figura 7.25 ilustra a lógica de controle ótima, obtida após o cálculo de $\sup C(E)$.

Exercício 7.2 *Encontre modelos para as restrições de coordenação do problema da estação de envasilhamento de cerveja.*

7.6 Considerações sobre a resolução do PCS

A metodologia de resolução do PCS, como introduzida neste capítulo permite abordar o problema de síntese de controladores para SEDs de modo formal e sistemático. No entanto, alguns aspectos da metodologia considerada dificultam a sua utilização prática, especialmente em aplicações de grande porte. Estes aspectos serão discutidos a seguir.

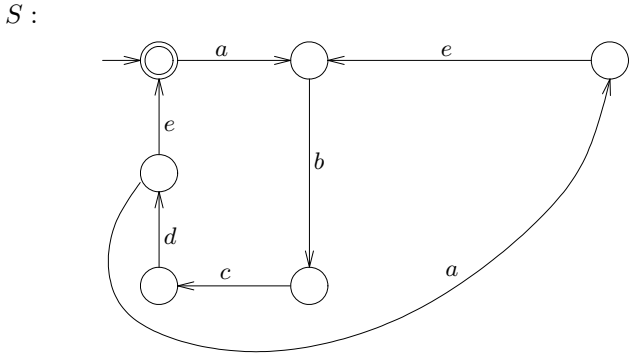

Figura 7.25: Autômato para $\sup C(E)$.

7.6.1 Complexidade Computacional

A síntese de supervisores compreende basicamente os seguintes procedimentos computacionais: composição síncrona de autômatos para a obtenção da planta G e do autômato R que modela a especificação; cálculo do autômato S que implementa a lógica de controle dada por $\sup C(E)$. A complexidade computacional deste conjunto de procedimentos é polinomial no produto do número de estados de G e R. Isto significa que o número de operações realizadas pode ser escrito como um polinômio a variável que representa este produto. Isto garante que o tempo de processamento, sendo proporcional ao número de operações, não explode (caso de complexidade exponencial). No entanto, o número de estados de G ou R cresce exponencialmente com o número de sub-sistemas ou de restrições de coordenação, o que coloca problemas de tempo de processamento em sistemas de porte maior. Algumas extensões da abordagem clássica aqui apresentada têm sido desenvolvidas na literatura para lidar com este problema. Dentre estas pode-se citar o Controle Modular, Controle de Sistemas com Simetria, Controle Hierárquico, etc.

7.6.2 Legibilidade/estruturação

Outro problema que advém da abordagem apresentada neste documento é o fato de que a solução do problema de síntese é dada na forma de um supervisor único, representado por um autômato que pode ter um número grande de estados. Isto gera um problema de legibilidade da lógica resultante, ou seja, o engenheiro não tendo condição de "interpretar"esta lógica, tem em geral, resistência em aceitá-la. Além disso, a implementação de um programa de controle que implemente este supervisor monolítico, é pouco estruturada. Uma solução para estes problemas é a abordagem modular de síntese, que explora

o caráter modular da planta e da especificação, explodindo-o em problemas menores. O resultado é um conjunto de supervisores de pequeno porte que atuam conjuntamente. Esta forma de resolver o PCS leva a lógicas elementares de maior legibilidade e permitem uma melhor estruturação do programa, dividido em módulos de controle. Além disso, estes módulos muitas vezes podem ser implementados de forma distribuída, em diferentes processadores atuando sobre partes da planta.

7.6.3 Ferramentas

A viabilidade do uso da metodologia de síntese proposta passa forçosamente pela disponibilidade de ferramentas computacionais que a implementem. Algumas destas estão hoje disponíveis porém, em geral, são desenvolvidas no meio acadêmico, e não têm as boas características de um produto, no que diz respeito a suas interfaces e capacidade de lidar com problemas de porte. O desenvolvimento de ferramentas computacionais "comerciais"é de fundamental importância para a disseminação e aplicação da metodologia aqui apresentada.

O anexo A apresenta o pacote GRAIL, que implementa várias funções relacionadas ao PCS.

7.7 Conclusão

Este capítulo apresentou um desenvolvimento dos passos que constituem a metodologia de síntese de supervisores, na sua forma básica. Extensões desta metodologia foram e estão sendo desenvolvidas para lidar com aspectos particulares envolvidos no controle de SEDs. A observação parcial de eventos, modelos temporizados, modelos estocásticos, controle modular, controle descentralizado, controle hierárquico, controle robusto, sistemas com simetria, controle supervisório de plantas de dinâmica híbrida, dentre outras, são algumas destas extensões.

Capítulo 8

Conclusão

Este documento foi gerado com o propósito de ser um texto básico para a Teoria de Controle de Sistemas a Eventos Discretos, possível de ser utilizado em uma disciplina de graduação em engenharia. Neste sentido, o documento priorizou aspectos do uso da teoria, sem aprofundar muito os aspectos formais, tanto os da teoria de Linguagens e Autômatos, base para a metodologia apresentada, como os da própria teoria de controle.

8.1 Referências

As referências listadas abaixo abrangem diferentes aspectos da área, divididos como segue.

- Teoria de linguagens e autômatos: (Hopcroft & Ullmann 1979)
- Sistemas a eventos discretos: (Cassandras 1992)(Cassandras & Lafortune 1999)
- Teoria básica de controle de SEDs: (Kumar & Garg 1995)(Ramadge & Wonham 1987b)(Ramadge & Wonham 1987a)(Ramadge & Wonham 1989)(Vaz & Wonham 1986)(Wonham & Ramadge 1987)(Wonham 1999)(Ziller 1993)(Ziller & Cury 1994)
- Aplicações e implementação: (Balemi, Hoffmann, Gyugyi, Wong-Toi & Franklin 1993)(Brandin 1996)(de Queiroz & Cury 2001a)(de Queiroz & Cury 2001b)(de Queiroz, Santos & Cury 2001)(Ernberg, Fredlund, Hansson, Jonsson, Orava & Pehrson 1991)(Hubbard & Caines 1998a)(Hubbard & Caines 1998b)(Lauzon, Ma, Mills &

Benhabib 1996)(Leduc, Brandin & Wonham 2001)(Martins & Cury 1997)(Martins 1999)(Tittus & Lennarston 1998)(Torrico 1999)

- Ferramentas computacionais: (Raymond & Derick 1995)(Rudie 1988)

- Extensões da teoria: (Antsaklis 2000)(Brandin & Wonham 1994)(Brave & Heymann 1993)(Caines, Gupta & Chen 1997)(Caines & Wei 1995)(Cury, Krogh & Niinomi 1998)(Cury & Krogh 1999)(Cury, Torrico & Cunha 2001)(da Cunha & Cury 2000)(da Cunha & Cury 2002)(de Queiroz & Cury 2000a)(de Queiroz & Cury 2000b)(de Queiroz 2000)(Gohari-Moghadam & Wonham 1998)(Gohari & Wonham 2000)(González 2000)(González, Cunha, Cury & Krogh 2001)(González & Cury 2001)(Li, Lin & Lin 1998)(Li, Lin & Lin 1999)(Lin & Wonham 1990)(Lin 1993)(Pappas, Lafferriere & Sastry 2000)(Raisch & O'Young 1998)(Ramadge & Wonham 1987b)(Rudie & Wonham 1992)(Sathaye & Krogh 1998)(Thistle & Wonham 1994a)(Thistle & Wonham 1994b)(Torrico & Cury 2001b)(Torrico & Cury 2001a)(Wong, Thistle, Malhame & Hoang 1995)(Wong & Wonham 1996a)(Wong & Wonham 1996b)(Wong & Wonham 1998)(Wong, Thistle, Malhame & Hoang 1998)(Wong 1998)(Wong, Thistle, Malhame & Hoang 2000)(Wong & Wonham 2000)

Apêndice A

Resumo Ferramenta Grail

Tatiana Renata Garcia
tati@lcmi.ufsc.br

A.1 Introdução

A ferramenta Grail é um ambiente de computação simbólico para máquinas de estado finitas, expressões regulares e linguagens finitas. A ferramenta foi elaborada com a intenção de ser usada em ensino, pesquisa e extensão.

A.2 FM — Máquinas de estado finitas

No Grail o formato de especificação de uma FM consiste de uma lista de instruções armazenada em um arquivo ASCII. A FM da figura A.1 possui a seguinte especificação no Grail:

```
(START) |- 0
0  a  1
1  b  2
1  -| (FINAL)
2  -| (FINAL)
```

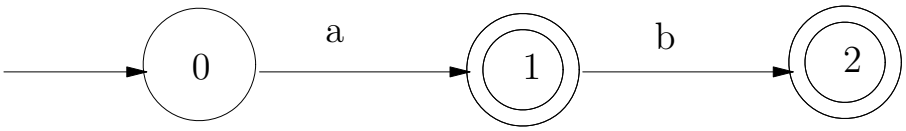

Figura A.1: Máquina de estados finitos

`iscomp`	testa se FM é completa
`isdeterm`	testa se FM é determinística
`isomorph`	testa se FM é isomorfa
`isuniv`	testa se FM é universal

Tabela A.1: Predicados do Grail

O Grail oferece alguns predicados e vários filtros para trabalhar com FM. A tabela A.1 mostra os predicados e a tabela A.2 mostra os filtros.

A.3 Exemplo

A seguir é apresentado um problema e como resolvê-lo utilizando o Grail.

Suponha um sistema constituído de duas máquinas e um buffer, como na figura A.2. Os eventos $\Sigma_c = \{a_1, a_2\}$ indicam início de operação e depósito de peça no buffer e $\Sigma_u = \{b_1, b_2\}$ indicam fim de operação. As máquinas devem ser modeladas sem a possibilidade de quebra.

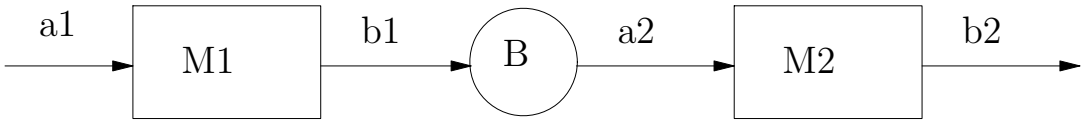

Figura A.2: Pequena fábrica

O autômato que representa a máquina M_1 está na figura A.3 e no Grail possui o seguinte formato:

```
(START) |- 0
0 a_1 1
1 b_1 0
0 -| (FINAL)
```

A máquina M_2 é mostrada na figura A.4 e possui o seguinte formato no Grail:

`fmalpha`	tira o alfabeto de uma FM
`fmcat`	concatena duas FMs
`fmcment`	complementa uma FM
`fmcomp`	completa uma FM
`fmcondat`	informa dados de controle sobre FM
`fmcross`	intersecciona duas FMs
`fmdeterm`	torna FM determinística
`fmenum`	enumera palavras reconhecidas pela FM
`fmexec`	dada uma cadeia executa a FM
`fmloop`	faz o self-loop de eventos da primeira FM na segunda FM
`fmmark`	marca todos os estados da FM
`fmmin`	minimiza a FM
`fmminrev`	minimiza a FM (outro método)
`fmplus`	faz o plus de uma FM
`fmproj`	faz a projeção de uma FM
`fmreach`	retira a componente acessível de uma FM
`fmremove`	elimina estados de uma FM
`fmrenum`	renumera os estados de uma FM
`fmreverse`	encontra o reverso de uma FM
`fmsort`	sorteia as instruções para os estados
`fmstar`	faz o fechamento Kleene de uma FM
`fmstats`	obtêm informações sobre a FM
`fmsupc`	encontra a máxima linguagem controlável
`fmsync`	faz o produto síncrono de duas FMs
`fmtodot`	converte uma FM para o formato .dot
`fmtovcg`	converte uma FM para o formato .vcg
`fmtrim`	encontra a componente trim de uma FM
`fmunion`	encontra a união de duas FMs

Tabela A.2: Filtros do Grail

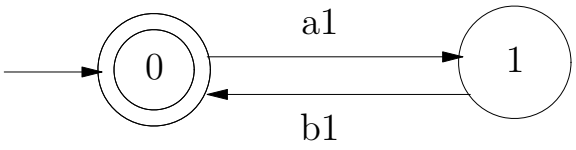

Figura A.3: Modelo máquina 1

```
(START) |- 0
0 a_2 1
1 b_2 0
0 -| (FINAL)
```

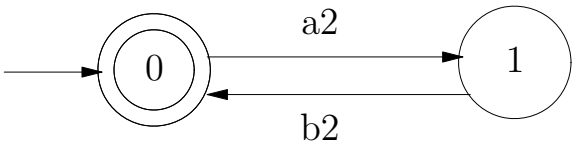

Figura A.4: Modelo máquina 2

A restrição de coordenação, ou especficação, para este sistema consiste em evitar overflow e underflow no buffer. O autômato que modela esta restrição está na figura A.5 e sua representação no Grail é a seguinte:

```
(START) |- 0
0 b_1 1
1 a_2 0
0 -| (FINAL)
```

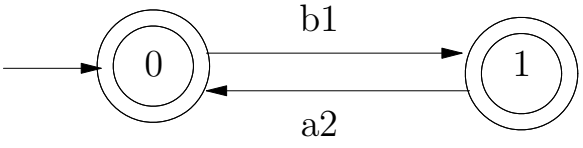

Figura A.5: Modelo restrição

A parte de modelagem do exemplo está concluída com cada modelo armazenado em um arquivo, agora podem-se aplicar os filtros do Grail para encontrar o supervisor minimamente restritivo.

1. Construir a planta livre, através da composição síncrona das máquinas M_1 e M_2:

   ```
   fmsync m1 m2 > planta
   ```

Normalmente joga-se o resultado da função num outro arquivo (`> planta`);

2. Realizar a composição da planta com a restrição:

   `fmsync planta restrição > s`

3. Encontrar a componente co-acessível de `s`:

   `fmtrim s > strim`

4. Criar um arquivo de um único estado com `self-loop` dos eventos não controláveis (arquivo `n-cont`, por exemplo):

   `(START) |- 0 0 b_1 0 0 b_2 0 0 -| (FINAL)`

5. Encontrar o supervisor minimamente restritivo:

   `fmsupc planta strim ncon > supervisor`

   onde `strim` é a especificação que a planta deve obedecer.

Para visualizar os resultados é possível utilizar a ferramenta Graphviz, já que o Grail possui uma função que converte o arquivo ASCII com o autômato para o formato da ferramenta (.dot). Para converter um arquivo basta usar o seguinte comando:

```
fmtodot nomearq > nomearq.dot
```

Uma observação deve ser feita sobre a numeração dos estados nos autômatos gerados pelo Grail. Ao executar a função `m3 = fmsync m1 m2`, n_3 é o número de um estado de `m3` e deseja-se poder determinar a numeração de estados equivalentes para `m1` e `m2`, isto é, o par (n_1, n_2) equivalente aos estados de `m1` e `m2` correspondentes aos estados de `m3`.

O primeiro passo para encontrar o par (n_1, n_2) é determinar max_1, o máximo número de estado para `m1` e depois realizar a seguinte divisão:

$$\frac{n_3}{max_1 + 1} = n_1$$

e o resto da divisão indica n_2.

A.4 Utilização do Grail

As ferramentas Grail e Graphviz são utilizadas através de uma linha de comando, e no LCMI estão instaladas na máquina `Kleene`. Para utilizá-las, é necessário ajustar o PATH:

```
set path="%path%";c:\sed\grail\bin;c:\sed\graphviz\bin;
```

Após acertar o PATH pode-se chamar todas as funções do Grail através da linha de comando. Para utilizar o Graphviz basta chamar a função `dotty`, como mostra o exemplo:

```
dotty strim.dot
```

As ferramentas podem ser obtidas nos seguintes endereços:

`ftp://ftp.lcmi.ufsc.br/pub/Windows/programacao/sed/`

`http://www.research.att.com/sw/tools/graphviz/`

Referências Bibliográficas

Antsaklis, P. J. (2000). Special issue hybrid systems: Theory and applications, *Proceedings of the IEEE* **88**(7).

Balemi, S., Hoffmann, G. J., Gyugyi, P., Wong-Toi, H. & Franklin, G. F. (1993). Supervisory control of a rapid thermal multiprocessor, *IEEE Transactions on Automatic Control* **38**(7): 1040–1059.

Brandin, B. A. (1996). The real-time supervisory control of an experimental manufacturing cell, *IEEE Transactions on Robotics and Automation* **12**(1): 1–14.

Brandin, B. A. & Wonham, W. M. (1994). Supervisory control of timed discrete-event systems, *IEEE Transactions on Automatic Control* **39**(2): 329–342.

Brave, Y. & Heymann, M. (1993). Control of discrete event systems modeled as hierarchical machines, *IEEE Transactions on Automatic Control* **38**(12): 1803–1819.

Caines, P. E., Gupta, V. & Chen, G. (1997). The hierarchical control of st-finite state machines, *Systems and Control Letters* **32**(6): 185–192.

Caines, P. E. & Wei, Y. J. (1995). The hierarchical lattices of a finite machine, *System and Control Letters* **30**(3): 257–263.

Cassandras, C. G. (1992). *Discrete Event Systems Modeling, Perfomance and Analysis*, 1 ed., Aksen Associates Incorporated Publishers, New Jersey.

Cassandras, C. G. & Lafortune, S. (1999). *Introduction to Discrete Event Systems*, 2 ed., Kluwer Academic Publishers, Massachussets.

Cury, J. E. R. & Krogh, B. H. (1999). Synthesizing supervisory controllers for hybrid systems, *Journal of the Society of Instrument and Control Engineers* **38**(3): 161–168.

Cury, J. E. R., Krogh, B. H. & Niinomi, T. (1998). Synthesis of supervisory controllers for hybrid systems based on approximating automata, *IEEE Transactions on Automatic Control* **43**(4): 564–568.

Cury, J. E. R., Torrico, C. R. C. & Cunha, A. E. C. (2001). A new approach for supervisory control of discrete event systems, *Proceedings of the European Control Conference*, Porto – Portugal.

da Cunha, A. E. C. & Cury, J. E. R. (2000). Uma metodologia de redução de modelos para sistemas a eventos discretos para síntese de supervisores, *Anais do XIII Congresso Brasileiro de Automática*, Florianópolis, SC, p. 2257–2262.

da Cunha, A. E. C. & Cury, J. E. R. (2002). Hierarchically consistent controlled discrete event systems. submetido ao IFAC2002.

de Queiroz, M. H. (2000). *Controle Modular Local para Sistemas de Grande Porte*, Dissertação (mestrado), Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC.

de Queiroz, M. H. & Cury, J. E. R. (2000a). Modular control of composed systems, *Proceedings of the American Control Conference (ACC)*, Chicago – USA.

de Queiroz, M. H. & Cury, J. E. R. (2000b). Modular supervisory control of large scale discrete event systems, *Proceedings of the Workshop on Discrete Event Systems (WODES)*, Ghent – Belgium.

de Queiroz, M. H. & Cury, J. E. R. (2001a). Controle supervisório modular de sistemas de manufatura. Aceito para publicação na Revista da SBA.

de Queiroz, M. H. & Cury, J. E. R. (2001b). Synthesis of readable supervisory control programs. Submetido ao IFAC 2002.

de Queiroz, M. H., Santos, E. A. P. & Cury, J. E. R. (2001). Síntese modular do controle supervisório em diagrama escada para uma célula de manufatura, *Proceedings of the Simpósio Brasileiro de Automação Inteligente (SBAI)*, Canela - Brasil.

Ernberg, P., Fredlund, L.-A., Hansson, H., Jonsson, B., Orava, F. & Pehrson, B. (1991). Guidelines for specification and verification of communication protocols, *Relatório técnico*, Swedwish Institute of Computer Science.

Gohari-Moghadam, P. & Wonham, W. M. (1998). A linguistic framework for controlled hierarchical des, *Proceedings of the Fourth Workshop on Discrete Event Systems*, Cagliari, Italy, p. 207–212.

Gohari, P. & Wonham, W. M. (2000). Reduced supervisors for timed discrete-event systems, *In:* R. Boel & G. Stremersch (ed.), *Discrete Event Systems: Analysis and Control*, Kluwer Academic Publishers, Ghent, Belgium, p. 119–130.

González, J. M. E., Cunha, A. E. C., Cury, J. E. R. & Krogh, B. H. (2001). Supervision of event driven hybrid systems: Modeling and synthesis, *In Proceedings of Hybrid Systems: Computation and Control*, LNCS, Italy.

González, J. M. E. & Cury, J. E. R. (2001). Exploiting simmetry in the synthesis of supervisors for discrete event systems. To appear in the IEEE Transactions on Automatic Control.

González, J. M. E. (2000). *Aspectos de Síntese de Supervisores para Sistemas a Eventos Discretos e Sistemas Híbridos*, Tese (doutorado), Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, Brasil.

Hopcroft, J. E. & Ullmann, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*, 1 ed., Addison Wesley Publishing Company, Reading.

Hubbard, P. & Caines, P. E. (1998a). A state aggregation approach to hierarchical supervisory control with applications to a transfer-line example, *Proceedings of the Fourth Workshop on Discrete Event Systems*, Cagliari, Italy, p. 2–7.

Hubbard, P. & Caines, P. E. (1998b). Trace-dc hierarchical supervisory control with applications to transfer-lines, *Proceedings on the 37^{th} IEEE Conference on Decision and Control*, Tampa, Florida, p. 3293–3298.

Kumar, R. & Garg, V. K. (1995). *Modeling and Control of Logical Discrete Event Systems*, 1 ed., Kluwer Academic Publishers, Boston.

Lauzon, S. C., Ma, A. K. L., Mills, J. K. & Benhabib, B. (1996). Application of discrete-event-system theory to flexible manufacturing, *IEEE Control Systems Magazine* p. 41–48.

Leduc, R. J., Brandin, B. A. & Wonham, W. M. (2001). Hierarchical interface-based non-blocking verification. A aparecer em conferência no Canadá.

Li, Y., Lin, F. & Lin, Z. H. (1998). A generalized framework for supervisory control of discrete event systems, *International Journal of Intelligent Control and Systems* **2**(1): 139–159.

Li, Y., Lin, F. & Lin, Z. H. (1999). Supervisory control of probabilistic discrete event systems with recovery, *IEEE Transactions on Automatic Control* **44**(10): 1971–1974.

Lin, F. (1993). Robust and adaptative supervisory control of discrete event systems, *IEEE Transactions on Automatic Control* **38**(12): 1848–1852.

Lin, F. & Wonham, W. M. (1990). Descentralized control and coordination of discrete-event systems with partial observation, *IEEE Transactions on Automatic Control* **35**(12): 1330–1337.

Martins, E. D. (1999). *Uma Arquitetura Física para a Implementação do Controle Supervisório de Sistemas a Eventos Discretos*, Dissertação (mestrado), Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC.

Martins, E. D. & Cury, J. E. R. (1997). Uma arquitetura para implementação de controle supervisório de sistemas a eventos discretos, *Anais 3º Simpósio Brasileiro de Automação Inteligente*, Vitória, ES, p. 184–189.

Pappas, G. J., Lafferriere, G. & Sastry, S. (2000). Hierarchically consistent control systems, *IEEE Transactions on Automatic Control* **45**(6): 1144–1160.

Raisch, J. & O'Young, S. D. (1998). Discrete approximation and supervisory control of continuous systems, *IEEE Trans. on Automatic Control* **43**(4): 569–573.

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

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

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

Raymond, D. & Derick, W. (1995). Grail: A c++ library for automata and expressions, *Journal of Symbolic Computation* **11**.

Rudie, K. G. (1988). *SOFTWARE FOR THE CONTROL OF DISCRETE EVENT SYSTEMS: A Complexity Study*, Dissertação (mestrado), Systems Control Group, Department of Electrical & Computer Engineering, University of Toronto, Toronto, Canada.

Rudie, K. & Wonham, W. M. (1992). Think globally, act locally: Decentralizaed supervisory control, *IEEE Transactions on Automatic Control* **37**(11): 1692–1708.

Sathaye, A. S. & Krogh, B. H. (1998). Supervisor synthesis for real-time discrete event systems, *Discrete Event Dynamic Systems: Theory and Applications* **8**(1): 5–35.

Thistle, J. G. & Wonham, W. M. (1994a). Control of infinite behavior of finite automata, *SIAM J. Control and Optimization* **32**(4): 1075–1097.

Thistle, J. G. & Wonham, W. M. (1994b). Supervision of infinite behavior of discrete-event systems, *SIAM J. Control and Optimization* **32**(4): 1098–1113.

Tittus, M. & Lennarston, B. (1998). Hierarchical supervisory control for batch processes, *Proceedings on the 37th IEEE Conference on Decision and Control*, Tampa, Florida, p. 3251–3255.

Torrico, C. C. & Cury, J. E. R. (2001a). Controle supervisório hierárquico de sistemas a eventos discretos: Uma abordagem baseada na agregação de estados, *Proceedings of the Simpósio Brasileiro de Automação Inteligente (SBAI)*, Canela - Brasil.

Torrico, C. C. & Cury, J. E. R. (2001b). Hierarchical supervisory control of discrete event systems based on state aggregation. Submitted to the 2002 IFAC World Congress.

Torrico, C. R. C. (1999). *Implementação de Controle Supervisório de Sistemas a Eventos Discretos Aplicado a Processos de Manufatura*, Dissertação (mestrado), Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC.

Vaz, A. F. & Wonham, W. M. (1986). On supervisor reduction in discrete-event systems, *International Journal of Control* **44**(2): 475–491.

Wong, K. C. (1998). On the complexity of projections of discrete-event systems, *Proceedings of the Fourth Workshop on Discrete Event Systems*, Cagliari, Italy, p. 201–206.

Wong, K. C., Thistle, J. G., Malhame, R. P. & Hoang, H.-H. (1995). Conflict resolution in modular control with feature interaction, *Proceedings of The 34th Conference of Decision and Control*, New Orleans, LA, p. 416–421.

Wong, K. C., Thistle, J. G., Malhame, R. P. & Hoang, H.-H. (1998). Supervisory control of distributed systems: Conflict resolution, *Proceedings on the 37th IEEE Conference on Decision and Control*, Tampa, Florida, p. 3275–3280.

Wong, K. C., Thistle, J. G., Malhame, R. P. & Hoang, H.-H. (2000). Supervisory control of distributed systems: Conflict resolution. A aparecer na revista Discrete Event Systems.

Wong, K. C. & Wonham, W. M. (1996a). Hierarchical control of discrete-event systems, *Discrete Event Dynamic Systems* **6**(3): 241–273.

Wong, K. C. & Wonham, W. M. (1996b). Hierarchical control of timed discrete-event systems, *Discrete Event Dynamic Systems* **6**(3): 275–306.

Wong, K. C. & Wonham, W. M. (1998). Modular control and coordination of discrete-event systems, *Discrete Event Dynamic Systems* **8**(3): 247–297.

Wong, K. C. & Wonham, W. M. (2000). On the computation of observers in discrete-event systems, *2000 Conference on Information Sciences and Systems*, Princenton University, p. 1–1.

Wonham, W. M. (1999). *Notes on Control of Discrete-Event Systems*, Systems Control Group, Department of Electrical & Computer Engineering, University of Toronto, Toronto, Canada.

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

Ziller, R. M. (1993). *A Abordagem Ramadge-Wonham no Controle de Sistemas a Eventos Discretos: Contribuições à Teoria*, Dissertação (mestrado), Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC.

Ziller, R. M. & Cury, J. E. R. (1994). *Lecture Notes in Control and Information Sciences 199*, SPRINGER VERLAG, capítulo On the Supremal Lm-closed and the Supremal Lm-closed and L-controllable Sublanguages of a Given Language.