UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Implementação de Controle Supervisório de Sistemas a Eventos Discretos Aplicado a Processos de Manufatura.

Dissertação submetida à Universidade Federal de Santa Catarina
como requisito parcial à obtenção do grau de

Mestre em Engenharia Elétrica

por

César Rafael Claure Torrico

Florianópolis, 15 de Março de 1999.

Implementação de Controle Supervisório de Sistemas a Eventos Discretos Aplicado a Processos de Manufatura.

César Rafael Claure Torrico

Esta dissertação foi julgada adequada para a obtenção do título de **Mestre em Engenharia** na especialidade **Engenharia Elétrica**, área de concentração **Controle, Automação e Informática Industrial**, e aprovada em sua forma final pelo curso de Pós-Graduação.

Florianópolis, 15 de Março de 1998.

Prof. José Eduardo Ribeiro Cury, Dr. d'Etat.
orientador

Prof. Ildemar Cassana Decker, D. Sc.
Coordenador do curso de Pós-Graduação em Engenharia Elétrica
da Universidade Federal de Santa Catarina.

Banca Examinadora

Prof. José E. Ribeiro Cury, Dr. d'Etat.
orientador

Prof. Jean-Marie Farines, Dr. Ing.

Prof. Marcelo R. Stemmer, Dr.

Prof. Roberto M. Ziller, M. Eng.

El caminante hace su camino al caminar...

(Antonio Machado)

A minha Mãe...

Rosa

Agradecimentos

Inicialmente quero agradecer a minha mãe Rosa, por seu carinho, compreensão e apoio.

Agradeço a meu orientador José Eduardo R. Cury, pelo ensinamento e incentivo ao longo do trabalho.

Aos membros da banca examinadora, que contribuíram opinando e sugerindo.

Sincero agradecimento para meu assessor Walter Cossio C, por ter me incentivado em seguir pelo caminho do mestrado.

Também agradeço a José Miguel Eyzell e Emerson Raposo, pela colaboração e sugestões.

A CAPES pelo financiamento econômico.

A Fili pelo apoio moral.

Finalmente agradeço a toda minha turma: Karina Barbosa, Marcos Mendes, Carlos Venturo, Jorge Ricardo, Sandro Batistella, Pierre A, Ronei Mascarenhas, Ivana Capanema, Carlos Rocha, Amarilys; pelo companheirismo e os bons momentos nestes dois anos.

Resumo

O presente trabalho consiste na implementação de um controlador de sistemas a eventos discretos a partir de uma arquitetura proposta num trabalho de dissertação de mestrado desenvolvido anteriormente.

Durante a etapa do desenvolvimento do trabalho são estudados os sistemas a eventos discretos em geral, encarando o problema de controle supervisório, com o cálculo da máxima sub-linguagem controlável do comportamento desejado. A idéia do controle supervisório para controlar sistemas a eventos discretos, originalmente proposta por Ramadge-Wonham é expressa em termos da observação e inibição de eventos, realizadas por um supervisor minimamente restritivo. Este modelo é estendido introduzindo-se a noção dos eventos forçáveis estabelecido inicialmente por Golaszewski-Ramadge, o qual permite conseguir uma maior expressividade na modelagem. Como contribuição a esta teoria é desenvolvido neste trabalho um algoritmo para o cálculo da máxima sub-linguagem controlável levando em conta eventos forçáveis.

Fundamentados na teoria estudada anteriormente foram desenvolvidas três ferramentas computacionais que permitem modelar e controlar Sistemas a Eventos Discretos: o CONDES 3.0 *"com eventos forçáveis"* dá suporte à modelagem de SED's e à sintese da lei de controle para o sistema em questão; o CONCEL permite a organização e mapeamento dos dados de um nível físico a um nível supervisório; o GERENCIADOR ou controlador interage diretamente com o nível físico do sistema de manufatura em tempo real.

Como aplicação, o sistema de controle é testado em uma célula de manufatura do Laboratório de Automação Industrial (LAI) do Departamento de Automação e Sistemas (DAS) da Universidade Federal de Santa Catarina (UFSC).

Resumen

El presente trabajo consiste en implementar un controlador de sistemas a eventos discretos a partir de una arquitectura propuesta en un trabajo de disertación de maestría desarrollado anteriormente.

Durante la etapa de desarrollo del trabajo son estudiados los sistemas a eventos discretos en general, encarando el problema de control supervisório, con el cálculo del máximo sub-lenguaje controlable del comportamiento deseado. La estructura del control supervisório para modelar sistemas a eventos discretos, originalmente propuesto por Ramadge-Wonham es expresado em terminos de observación e inibición de eventos, realizados por un supervisor mínimamente restrictivo. Este modelo es extendido introduciendo la noción de los eventos forzábles planteado inicialmente por Golaszewski-Ramadge, el cual permite conseguir una mayor expresividad en el modelaje. Como contribución a esta teoría es desarrollado en este trabajo un algorítmo para el cálculo del máximo sub-lenguaje controlable tomando en cuenta eventos forzables.

Fundamentados en la teoría estudiada anteriormente se desenvolvieron tres herramientas computacionales que permiten controlar y modelar Sistemas a Eventos Discretos : CONDES 3.0 *"con eventos forzables"* da soporte al modelaje de SED's y a la sintesis de la ley de control para el sistema en cuestión; CONCEL permite la organización y mapeamiento de los datos de un nivel fisico a un nivel de supervision; GERENCIADOR o controlador interactua directamente con el nivel físico del sistema de manufactura en tiempo real.

Como aplicacion el sistema es probado a una célula de manufactura del Laboratório de Automatización Industrial (LAI) del Departamento de Automatización y Sistemas (DAS) de la Universidad Federal de Santa Catarina (UFSC).

Abstract

The present work consists of the controller implementation for discrete-event systems from an architecture proposed in a previously developed work.

During the stage of the development of this work discrete-event systems process were studied, facing the problem of supervisory control, with the computation of the maximal controllable sub-language of a desired behavior. The structure of the supervisory control for modeling the discrete-event systems, originally considered by Ramadge-Wonham, is expressed in terms of observation and inhibition of events, made directly by a minimally restrictive supervisor. Extending this approach, the notion of forced events, established initially by Golaszewski-Ramadge, is introduced allowing to obtain a bigger modeling expressivity. As a contribution to this theory an algorithm of the computation maximum controllable language is developed taking into account the forced events.

Based on the previously studied theory, three computational tools were developed for modeling and control of Discrete-Event Systems: The CONDES3.0 *"with forced events "* gives support to modeling of SED's and synthesis of control law for the system in question; the CONCEL allows the data organization and mapping events from the physical level to the supervisory level; and the GERENCIADOR, or controller, interacts directly with the physical level of a manufacturing system in real time.

The application of the control system was tested in a manufacturing cell of the Laboratory (LAI) of Industrial Automation of the Automation and Systems Department (DAS) of the Federal University of Santa Catarina (UFSC).

Sumário

Capítulo 1

Introdução

O controle dos sistemas a eventos discretos (SED) é uma área de pesquisa de vitalidade atual, estimulada pela diversidade de possíveis aplicações, como é o caso dos sistemas de manufatura, sistemas de tráfego, sistemas de gerência de base de dados, protocolos de comunicação, entre outros.

De um ponto de vista formal, um Sistema a Eventos Discretos (SED) define-se como um sistema dinâmico a estado discreto que evolui conforme a ocorrência assíncrona de certas transições discretas, chamadas *eventos*. Por exemplo, um evento poderia ser a chegada de uma peça numa fila, uma tarefa completada ou a pane de uma máquina num sistema de manufatura; a transmissão de uma mensagem numa rede de comunicação, envio de um comando, etc.

As características principais dos SED's são: a) o espaço de estados é discreto; b) o mecanismo de transição de estados é dirigido por eventos. Estas propriedades contrastam com os sistemas a variáveis contínuas, que se caracterizam pela continuidade das variáveis de estado e cujo mecanismo de transição é dirigido pelo tempo. Nesses últimos podem ser usados modelos de equações diferenciais, e o tempo é uma variável independente natural. A não-linearidade dos SED's está inerente nas descontinuidades das transições de estado.

A idéia do controle supervisório para modelar sistemas a eventos discretos, originalmente proposto por Ramadge-Wonham [RW89] é expressa em termos da observação e inibição de eventos controláveis, realizadas por um "supervisor" minimamente restritivo sobre um gerador espontâneo de eventos chamado "planta". A *planta* ou sistema a controlar, reflete o comportamento fisicamente possível do sistema. i.e., todas as seqüências

de eventos que o sistema é capaz de gerar na ausência de controle. A tarefa do *supervisor* ou elemento controlador consiste em confinar o comportamento da planta a certa especificação desejada, através de ações de controle, baseado no comportamento passado da planta.

Como extensão à teoria clássica de Ramadge-Wonham surgiram outros estudos na procura de melhorar alguns problemas que foram aparecendo na modelagem de SED's. Tal é o caso de Golaszewsky-Ramadge [GR87] que introduziram a noção dos eventos forçáveis conseguindo assim um menor nível de abstração ou mesmo uma maior expressividade na modelagem. De forma contrária aos eventos controláveis/não-controláveis, a característica principal dos eventos forçáveis reside em que eles não são gerados pela planta e não são espontâneos, mas representam ações enviadas do supervisor à planta.

Continuando com esta pesquisa, Martins e Cury [MC97] estudam a natureza dos eventos no sentido de definir uma metodologia para a implementação de controladores. Num sistema de manufatura, os eventos no nível físico basicamente estão representados por comandos específicos dos atuadores (i.e, trancar ou recolher um atuador) e a leitura de sensores (i.e. sensibilização de algum sensor S_i quando uma peça começa a passar à sua frente). Por outro lado, há os eventos do nível supervisório, que, de uma forma ou outra, estão relacionados com os eventos do nível físico. Neste sentido em [MC97] propõe-se um procedimento para a modelagem do problema de controle, através do mapeamento de eventos do nível supervisório em eventos do nível da planta, como segue: os eventos forçáveis são mapeados em comandos enviados a atuadores do nível físico; os eventos não-controláveis, na sensibilização ou dessensibilização de um sensor; os eventos controláveis também são mapeados na sensibilização ou dessensibilização de um sensor, mas neste caso o significado da habilitação ou desabilitação do evento são mapeados em comandos enviados a atuadores (ex. seja um evento controlável *"passar"* do nível supervisório, que significa a passagem de uma peça por uma estação de espera. A ocorrência do evento *passar* estará representada pela sensibilização de um sensor S_i na estação, mas a resposta da ação de controle para habilitação deste evento é mapeada ao recolhimento de um atuador A_x, ou, caso contrário; sua desabilitação é mapeada na extensão do atuador A_x). Com o estudo desenvolvido até aqui, em [MC97] são propostas uma metodologia e uma arquitetura para implementação de controle supervisório realizável na prática.

1.1 Objetivos do trabalho

Com base na breve discussão acima, os principais objetivos para este trabalho podem ser apresentados como:

1. desenvolver ferramentas computacionais para modelagem e controle de sistemas a eventos discretos de modo a permitir a implementação de controladores de acordo com a arquitetura proposta em [MC97];
2. desenvolver algoritmos para resolver o problema de controle supervisório em relação ao cálculo da máxima sub-linguagem controlável do comportamento desejado quando eventos forçáveis são considerados na modelagem;
3. aplicar as ferramentas e algoritmos desenvolvidos em um problema real, no caso, à célula de manufatura do Laboratório de Automação Industrial (LAI), situado no Departamento de Automação e Sistemas (DAS) da Universidade Federal de Santa Catarina (UFSC).

1.2 Organização do trabalho

O capítulo 2 apresenta os conceitos básicos da teoria de linguagens e autômatos, necessários para representar sistemas a eventos discretos e abordar a teoria de controle supervisório.

O capítulo 3 apresenta a teoria de controle supervisório em sua forma clássica, tal como encontrado na literatura [RW87] [RW89]. Estuda-se uma extensão desta teoria tratando os eventos forçáveis, introduzidos por Golaszewsky-Ramadge [GR87]. Para ambas situações, os autores citados estabelecem as condições de existência de supervisores e como aporte deste trabalho é desenvolvido um algoritmo para o cálculo da máxima sub-linguagem controlável quando na modelagem são introduzidos eventos forçáveis.

O capítulo 4 apresenta a implementação do controle supervisório baseada na arquitetura para implementação de controle proposta por Martins e Cury [MC97].

O capítulo 5 apresenta uma aplicação da implementação desenvolvida no capítulo anterior à célula de manufatura do Laboratório de Automação Industrial (LAI), comparando o uso ou não dos eventos forçáveis na modelagem.

No capítulo 6 são apresentadas a conclusão global do trabalho em função dos objetivos estabelecidos e sugestões para futuros trabalhos.

O apêndice A apresenta o manual da ferramenta CONDES 3.0. Esta ferramenta é uma renovação do CONDES 2.0 desenvolvido inicialmente por um grupo de pesquisa no Laboratório de Automação Industrial (LAI) da Universidade Federal de Santa Catarina (UFSC). Com o CONDES, o usuário poderá modelar sistemas a eventos discretos através das funções implementadas, quais sejam: Produto síncrono, Interseção de linguagens, Cálculo da máxima sub-linguagem controlável, etc. O aporte do CONDES 3.0 em relação à anterior versão é a inclusão dos eventos forçáveis na modelagem e a reformulação da interface com o usuário, visando torná-la mais amigável.

No apêndice B é descrito o manual do CONCEL. Esta é uma ferramenta de suporte, que ajuda a preparar os dados de entrada para um sistema Gerenciador. O manual mostra a forma de se fazer o mapeamento dos eventos do nível físico com os eventos do nível supervisório.

No apêndice C é descrito o manual do GERENCIADOR. O Gerenciador é um sistema de controle que interage diretamente em tempo real com sistemas de manufatura por meio da porta paralela do computador.

Finalmente no apêndice D descrevem-se os algoritmos computacionais usados para as funções implementadas no CONDES 3.0.

Capítulo 2

Linguagens e Autômatos

Neste capítulo apresentam-se os conceitos básicos da teoria de linguagens, autômatos e geradores, que serão requeridos nos próximos capítulos, para o entendimento do modelo clássico Ramadge-Wonham (RW) e o modelo complementar adicionando eventos forçáveis, necessários para o controle de sistemas a eventos discretos.

Para maior detalhe em relação à teoria de linguagens e autômatos, consultar [HU79] [RW87] [CL89], onde o leitor encontrará a origem do material apresentado neste capítulo.

2.1 Linguagens

Um Sistema a Eventos Discretos (SED) pode ser descrito pelo conjunto de todas as possíveis seqüências de eventos a ocorrer. Este conjunto pode ser modelado por uma linguagem. As definições a seguir introduzem a noção de linguagens.

Def. 2.1 Σ é um *alfabeto* **sse** Σ é um conjunto finito, não vazio, de símbolos distintos.

Um elemento de um *alfabeto* algumas vezes é chamado de *letra*. Alguns exemplos familiares de alfabetos são: as 23 letras do alfabeto português e o conjunto de caracteres ASCII.

Baseados na definição de *alfabeto* podemos definir entidades compostas chamadas *palavras* ou *cadeias*, a seqüência finita de símbolos de um alfabeto.

Def. 2.2 Para um alfabeto Σ, e um número natural n, a seqüência de símbolos $\sigma_1\sigma_2\sigma_3.....\sigma_n$ é uma *palavra* ou *cadeia* sobre o alfabeto Σ de comprimento n onde para cada $i = 1, 2, 3.....n, \sigma_i \in \Sigma$.

Uma *palavra* sobre um alfabeto é uma cadeia ordenada de símbolos, onde cada símbolo na cadeia é um elemento do alfabeto. Um exemplo óbvio de palavras são aquelas que você está lendo neste momento.

Def. 2.3 Para um alfabeto Σ e uma palavra $w = \sigma_1\sigma_2...\sigma_n$ sobre Σ, $|w|$ denota o comprimento de w. P.e., $|\sigma_1\sigma_2...\sigma_n| = n$.

Def. 2.4 A palavra nula ou vazia ε é a única palavra de comprimento nulo.

Cabe notar que $\varepsilon \notin \Sigma$, por ser ε uma palavra e não um símbolo

Conjuntos de palavras são estruturas fundamentais na teoria de linguagens.

Def. 2.5 Dado um alfabeto Σ e um número inteiro não negativo $k \in \mathbb{N}$, denota-se: $\Sigma^k = \{x | x \text{ é uma palavra sobre } \Sigma \text{ e } |x| = k\}$.

Exe. 2.1 Dado $\Sigma = \{0, 1\}$:

$\Sigma^0 = \{\varepsilon\}$
$\Sigma^1 = \{0, 1\}$
$\Sigma^2 = \{00, 01, 10, 11\}$
$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

Note que ε é o único elemento de Σ^0, o conjunto de todas as palavras contendo zero letras.

Def. 2.6 Dado um alfabeto Σ definem-se:
$\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ...$
e
$\Sigma^+ = \bigcup_{k=1}^{\infty} \Sigma^k = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ...$

Σ^* é o conjunto de todas as palavras que podem ser formadas a partir das letras de um alfabeto Σ.

Σ^+ é o conjunto de todas as palavras "não vazias" que podem ser formadas a partir de Σ. A relação entre Σ^* e Σ^+ é: $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

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

Def. 2.7 Dado um alfabeto Σ, L é uma *linguagem* sobre Σ **sse** $L \subseteq \Sigma^*$.

Segundo a definição anterior, tanto Σ^* quanto ϕ são linguagens. Note que a linguagem vazia, $\phi = \{\}$, é diferente da linguagem formada pela palavra nula $\Sigma^0 = \{\varepsilon\}$:

- Σ^0 contém uma palavra de zero letras.
- ϕ contém zero palavras.

2.1.1 Operações com palavras

É possível juntar duas palavras para formar uma palavra composta; este processo é chamado de *concatenação*.

Def. 2.8 Dado um alfabeto Σ, sejam $x = \alpha_1\alpha_2...\alpha_n$ e $y = \beta_1\beta_2...\beta_n$ cadeias onde cada $\alpha_i \in \Sigma$ e cada $\beta_i \in \Sigma$. A concatenação de cadeias x e y denotado por xy é a justaposição de x e y, ou seja $xy = \alpha_1\alpha_2...\alpha_n\beta_1\beta_2...\beta_n$.

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

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

Exe. 2.2 Dado um alfabeto $\Sigma = \{\alpha, \beta, \lambda\}$, $u = \alpha\beta$ é um prefixo de $w = \alpha\beta\lambda\alpha$, porque $\exists v \in \Sigma^*$ tal que $uv = w$ (neste caso $v = \lambda\alpha$).

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

Exe. 2.3 Dado um alfabeto $\Sigma = \{\alpha, \beta, \lambda\}$, se $s = \alpha\beta\lambda\alpha$, então $Pre(s) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta\lambda, \alpha\beta\lambda\alpha\}$.

Dada uma linguagem $L \subseteq \Sigma^*$, existe uma linguagem associada a ela, formada pelas palavras de L e todos seus prefixos.

Def. 2.11 O *Prefixo fechamento*, ou simplesmente *fechamento* de L, é dado por: $\overline{L} = \{u \mid \exists v \in \Sigma^* \wedge uv \in L\}$.

Uma conseqüência desta definição é:

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

Def. 2.12 Uma Linguagem L é *prefixo fechada* ou simplesmente *fechada* sse $L = \overline{L}$.

Se L é uma linguagem prefixo-fechada , então, para cada palavra r pertencente a L, pode-se afirmar que: $Pre(r) \subseteq L$.

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 se um determinado sistema produziu uma palavra qualquer r, então produziu anteriormente todos os seus prefixos. Considerando este fato, a seguinte proposição é válida:

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

2.1.2 Projeção natural

Def. 2.13 Sejam dois alfabetos Σ_l e Σ, com $\Sigma_l \subset \Sigma$. Define-se $P_l : \Sigma^* \rightarrow \Sigma_l^*$, a *projeção natural* de Σ em Σ_l de acordo com:.

1. $P_l(\varepsilon) = \varepsilon$
2. $P_l(\sigma) = \begin{cases} \varepsilon & se \quad \sigma \notin \Sigma_l \\ \sigma & se \quad \sigma \in \Sigma_l \end{cases}$.
3. $P_l(s\sigma) = P_l(s)P_l(\sigma)$ *para* $s \in \Sigma^*$, $\sigma \in \Sigma$

A ação de P_l sobre uma cadeia s é apenas a de apagar de s todas as ocorrências de σ tal que $\sigma \notin \Sigma_l$. Assim P_l é a projeção natural de Σ^* em Σ_l^*. Pode-se estender a noção acima a linguagens como segue:

$$P_l L = \{s_l \in \Sigma_l^* \mid (\exists s \in L | P_l s = s_l)\ \}$$

Def. 2.14 Dada uma projeção $P : \Sigma^* \rightarrow \Sigma_l^*$, a *projeção inversa*, denotada por P^{-1}, é definida como:
$P_l^{-1} L_l = \{s \mid s \in \Sigma^* \wedge P_l s \in L_l\}$

2.1.3 Composição de linguagens

Def. 2.15 Sejam duas linguagens $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$, onde é possível que $\Sigma_1 \cap \Sigma_2 \neq \phi$. Seja $\Sigma = \Sigma_1 \cup \Sigma_2$. Define-se o *Produto Síncrono* $L_1 ||_s L_2 \subset \Sigma^*$, como:
$L_1 ||_s L_2 = P_1^{-1} L_1 \cap P_2^{-1} L_2$

Casos extremos da composição de linguagens

- Caso quando $\Sigma_1 \cap \Sigma_2 = \phi$. Diz-se que $L_1 ||_s L_2$ é um *produto assíncrono*
- Caso quando $\Sigma_1 = \Sigma_2 = \Sigma$. $L_1 ||_s L_2$ corresponde à *interseção das linguagens* L_1 e L_2 ($L_1 \cap L_2$).

Exe. 2.4 Sejam: $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$

$\Sigma_1 = \{\alpha, \beta\}$; $L_1 = \epsilon + \alpha + \beta + \alpha\beta$
$\Sigma_2 = \{\beta, \gamma\}$; $L_2 = \epsilon + \gamma + \gamma\beta$
$\Sigma = \Sigma_1 \cup \Sigma_2 = \{\alpha, \beta, \gamma\}$

Então:

$P_1^{-1}L_1 = \gamma^* + \gamma^*\alpha\gamma^* + \gamma^*\beta\gamma^* + \gamma^*\alpha\gamma^*\beta\gamma^*$
$P_2^{-1}L_2 = \alpha^* + \alpha^*\gamma\alpha^* + \alpha^*\gamma\alpha^*\beta\alpha^*$
$L_1||_sL_2 = P_1^{-1}L_1 \cap P_2^{-1}L_2 = \epsilon + \alpha + \gamma + \alpha\gamma + \gamma\alpha + \alpha\gamma\beta + \gamma\alpha\beta + \gamma\beta$

2.1.4 Modelos de linguagens

Num sistema a eventos discretos pode-se apresentar os seguintes tipos de linguagens:

Def. 2.16 A linguagem prefixo-fechada que representa o comportamento lógico de um sistema a eventos discretos é denominada *linguagem gerada* do sistema.

Após partir do estado inicial e percorrer uma determinada trajetória em seu espaço de estados, um SED acabará via de regra, completando uma ou mais tarefas. As seqüências de eventos que levam a tarefas completas, formam também uma linguagem. Note que linguagens deste tipo não são necessariamente prefixo-fechadas.

Def. 2.17 A linguagem que representa o conjunto de tarefas que um sistema a eventos discretos é capaz de completar é denominada *linguagem marcada* do sistema.

Se L é a linguagem gerada de um sistema e L_m sua linguagem marcada, observe que, para gerar qualquer palavra da *linguagem marcada*, o SED em questão tem que gerar todos os seus prefixos. Em outras palavras, um SED que produza as palavras contidas numa linguagem L_m, também produzirá as palavras contidas em $\overline{L_m}$. Em conclusão:

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

2.1.5 Expressões regulares

Existe uma classe importante de linguagens, que podem ser expressas pelas assim chamadas *expressões regulares* [HU79] [CL89]. As expressões regulares são seqüências de símbolos obtidas pela aplicação repetitiva de um conjunto de regras de formação.

Def. 2.18 Dado um alfabeto $\Sigma = \{\alpha_1, \alpha_2, ...\alpha_m\}$. Uma *expressão regular* sobre Σ é qualquer expressão que representa uma linguagem e que pode ser formado por uma seqüência finita de aplicações da seguintes regras:

1. $\alpha_1, \alpha_2, ..., \alpha_m$ são expressões regulares.
2. ϕ é uma expressão regular.
3. ε é uma expressão regular
4. Se R_1 e R_2 são expressões regulares, então $R_1 R_2$ é uma expressão regular
5. Se R_1 e R_2 são expressões regulares, então $R_1 + R_2$ é uma expressão regular.
6. Se R_1 é uma expressão regular, então R_1^* é uma expressão regular.

Exe. 2.5 Dado um alfabeto $\Sigma = \{\alpha, \beta, \gamma\}$. A expressão regular $(\alpha\beta)^*\gamma$ representa a linguagem $L = \{\gamma, \alpha\beta\gamma, \alpha\beta\alpha\beta\gamma, \alpha\beta\alpha\beta\alpha\beta\gamma, ...\}$. Também; a expressão regular $(\alpha + \beta)\gamma^*$ representa a linguagem $L = \{\alpha, \beta, \alpha\gamma, \beta\gamma, \alpha\gamma\gamma, \beta\gamma\gamma, ...\}$.

Outra forma de representar linguagens regulares equivalente às expressões regulares, são os autômatos finitos, que são tratados na próxima seção. Expressões regulares e autômatos finitos têm o mesmo poder de representação, e modelam as linguagens regulares com a mesma expressividade [HU79].

2.2 Autômatos

Um *autômato finito determinístico*, às vezes simplesmente chamado de *autômato*, é um modelo matemático de uma máquina de estados, que aceita um conjunto particular de palavras sobre um alfabeto Σ [CL89].

Def. 2.19 Um *autômato finito determinístico* é uma quíntupla $A =< \Sigma, Q, \delta, q_0, Q_m >$, onde:

- Σ é um alfabeto;
- Q é um conjunto finito não-vazio de estados;
- $\delta : \Sigma \times Q \rightarrow Q$ é uma função de transição de estados;
- $q_0 \in Q$ é o estado inicial;
- $Q_m \subseteq Q$ é o conjunto de estados marcados.

Entende-se aqui que a função de transição é *completa*, ou seja definida para todo par $(\sigma, q) \in \Sigma \times Q$. Isto é a origem da palavra *determinístico* na frase *autômato finito determinístico.*

O alfabeto de entrada Σ para qualquer autômato finito determinístico A, é o conjunto de todos os eventos que podem aparecer nessa máquina de estados. Cada símbolo sucessivo numa palavra, causará a transição do estado atual para outro estado na máquina.

O conjunto de estados representa a memória da máquina. Desde que o número de estados na máquina é finito, o número das situações possíveis que podem ser recordadas pela máquina é também finito. Isto é a origem da palavra *finito* na frase *autômato finito determinístico.*

Um autômato pode ser representado a partir de um *diagrama de transição de estados* ou ainda por uma *tabela de transição de estados.* Um diagrama de transição de estados é um diagrama onde os estados da máquina podem ser representados pêlos vértices do diagrama, e as transições pêlos arcos. Formalmente:

Def. 2.20 Dado um autômato $A =< \Sigma, Q, \delta, q_0, Q_m >$, o *diagrama de transição de estados* associado a A é o Grafo $G = \{V, E\}$, onde:
$V = Q$, *e*
$E = \{< q, r, \sigma > \mid q, r \in Q, \sigma \in \Sigma, \wedge \delta(\sigma, q) = r\}$

V representa o conjunto de vértices do diagrama, e E o conjunto de arcos conectando esses vértices.

A *tabela de transição de estados* é uma matriz cujas linhas são enumeradas pelo conjun-

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

Exe. 2.6 No autômato mostrado na Figura 2.1.

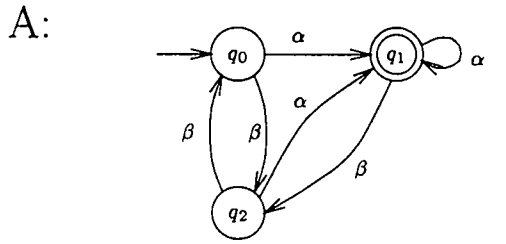


Figura 2.1: Diagrama de transição representando um autômato

$\Sigma = \{\alpha, \beta\}$
$Q = \{q_0, q_1, q_2\}$
$\delta(\alpha, q_0) = q_1, \qquad \delta(\beta, q_0) = q_2,$
$\delta(\alpha, q_1) = q_1, \qquad \delta(\beta, q_1) = q_2,$
$\delta(\alpha, q_2) = q_1, \qquad \delta(\beta, q_2) = q_0,$
$q_o = q_0$
$Q_m = \{q_1\}$

Segundo a definição 2.20, o diagrama correspondente é $G = \{V, E\}$, com:

$V = \{q_0, q_1, q_2\}$

$E = \{< q_0, q_1, \alpha >, < q_0, q_2, \beta >, < q_1, q_1, \alpha >, < q_1, q_2, \beta >, < q_2, q_1, \alpha >, < q_2, q_0, \beta >\}$

e sua tabela de transição de estados é a tab. 2.1.

$q\backslash\sigma$	α	β
q_0	q_1	q_2
q_1	q_1	q_2
q_2	q_1	q_0

Tabela 2.1: Tabela de transição de estados

Com δ, podemos descrever o estado em que nos encontraremos, após ter processado uma única letra. Queremos também poder descrever o estado em que nós chegaremos

após ter processado uma cadeia inteira. Estenderemos a função δ para cobrir cadeias inteiras; $\overline{\delta}(x, q)$ será o estado que é obtido, após começar em q e processar, em ordem, todas as letras da cadeia x.

Def. 2.21 Dado um autômato $A =< \Sigma, Q, \delta, q_0, Q_m >$, a *função de transição estendida* para A, denotado por $\overline{\delta}$, é uma função $\overline{\delta} : \Sigma^* \times Q \to Q$ definido recursivamente como segue:

- $(\forall q \in Q)(\forall \sigma \in \Sigma)$ $\quad \overline{\delta}(\sigma, q) = \delta(\sigma, q)$
- $(\forall q \in Q)$ $\quad \overline{\delta}(\varepsilon, q) = q$
- $(\forall q \in Q)(\forall x \in \Sigma^*)(\forall \sigma \in \Sigma)$ $\quad \overline{\delta}(x\sigma, q) = \delta(\sigma, \overline{\delta}(x, q))$

A função $\overline{\delta}$ estende a função δ de simples letras para palavras.

O estado em que uma cadeia termina é significativo; é importante determinar se o estado final para uma cadeia acontece ser um dos estados que foi designado para ser um estado marcado.

Def. 2.22 Dado um autômato $A =< \Sigma, Q, \delta, q_0, Q_m >$, A reconhece uma palavra $w \in \Sigma^*$ **sse** $\overline{\delta}(w, q_0) \in Q_m$..

Exe. 2.7 O autômato do exemplo 2.6, reconhece todas as palavras de Σ^* que terminam com a letra α, expressando em termos de linguagens regulares temos:

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

Conclui-se que é possível representar um SED de linguagem marcada L_m por um autômato A tal que: $L_m(A) = L_m$.

Note que a linguagem gerada por um autômato é o universo de todas as palavras de Σ^*. No entanto é desejável ter uma representação por máquinas de estados finitos para a linguagem gerada do sistema. A solução para este problema está em se permitir que a função de transição δ seja definida apenas para alguns pares (evento, estado) do conjunto $\Sigma \times Q$. Diz-se então que δ é uma função parcial.

Utiliza-se o termo *Gerador* quando se fala de *autômatos com função de transição parcial.*

2.3 Geradores

Formalizando a definição de um gerador:

Def. 2.23 Um *Gerador* é uma quíntupla $G =< \Sigma, Q, \delta, q_0, Q_m >$, onde:

- Σ é um alfabeto;
- Q é um conjunto finito não-vazio de estados;
- $\delta : \Sigma \times Q \rightarrow Q$ é uma função de transição parcial de estados;
- $q_0 \in Q$ é o estado inicial;
- $Q_m \subseteq Q$ é o conjunto de estados marcados.

A diferença entre autômatos e geradores é a função de transição. Nos geradores, a função de transição apenas está definida para um subconjunto de eventos em cada estado do gerador. Assim como os autômatos, os geradores também podem ser representados por diagramas de transição, com a diferença de que neste caso estão presentes apenas os arcos para os quais a função de transição está definida.

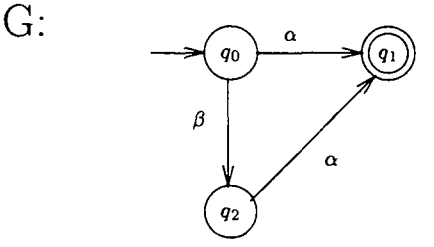


Figura 2.2: Exemplo de gerador

Def. 2.24 Para cada estado $q \in Q$ de um gerador $G =< \Sigma, Q, \delta, q_0, Q_m >$ é definido um conjunto de eventos $\Sigma(q)$, tal que:
$\Sigma(q) = \{\sigma \mid \sigma \in \Sigma \wedge \delta(\sigma, q)!\}$

onde a notação $\delta(\sigma, q)!$ indica que δ está definida para (σ, q).

Também para o caso de um gerador pode se estender a função de transição sob as seguintes condições.

Def. 2.25 Dado um gerador $G =< \Sigma, Q, \delta, q_0, Q_m >$, a função de transição estendida denotada por $\overline{\delta}$, é uma função $\overline{\delta} : \Sigma^* \times Q \rightarrow Q$ tal que :

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

A limitação da função de transição estendida limitará também o conjunto de palavras que o gerador pode processar.

Def. 2.26 Dado um gerador $G =< \Sigma, Q, \delta, q_0, Q_m >$, a *linguagem gerada* de G, dado por L(G) é:
$L(G) = \{s \in \Sigma^* \mid \overline{\delta}(s, q_0)!\}$

Enquanto que os autômatos, com sua função de transição completa, podem processar qualquer palavra sobre seu alfabeto, o caráter parcial da função de transição dos geradores faz com que a linguagem gerada seja, um subconjunto próprio de Σ^*. Além disso, esta linguagem é prefixo-fechada [Zil93]

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

De forma análoga ao caso dos autômatos, também há um conjunto de estados marcados para os geradores.

Def. 2.27 A *Linguagem marcada* $L_m(G) \subseteq \Sigma^*$ de um gerador $G =< \Sigma, Q, \delta, q_0, Q_m >$ é:
$L_m(G) = \{s \mid s \in \Sigma^* \wedge \delta(s, q_0) \in Q_m\}$

Portanto, dado um SED de linguagem gerada L e de linguagem marcada L_m, pode-se representa-lo por um gerador G tal que $L(G) = L$ e $L_m(G) = L_m$. Assim, os geradores permitem representar ambas linguagens associadas a um SED.

Pelas definições 2.26 e 2.27:

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

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

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

e, pela equação (2.3),

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

Considerando-se as equações (2.1) a (2.6), é possível ordenar as linguagens apresentadas da seguinte forma:

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

2.3.1 Acessibilidade e co-acessibilidade de um gerador

De forma geral, um gerador pode ter estados inacessíveis, isto é, estados que jamais podem ser alcançados a partir do estado inicial.

Def. 2.28 Um estado $q \in Q$ é um *estado acessível*, se: $\exists s \in \Sigma^* | \delta(s, q_0) = q$

Def. 2.29 Um gerador G é *acessível* se q é acessível para todo $q \in Q$.

A componente acessível de um gerador G é obtido pela eliminação dos estados não acessíveis e das transições associadas a eles.

Def. 2.30 Um gerador G é dito *co-acessível* se cada palavra $s \in L(G)$ for um prefixo de uma palavra em $L_m(G)$, ou seja sse $L(G) \subseteq \overline{L_m(G)}$.

Esta definição diz que um gerador é co-acessível se, a partir de qualquer um de seus estados, existir ao menos um caminho que leve a um estado marcado. Considerando-se que a equação (2.6) é valida para todo gerador G, é possível substituir a inclusão na definição 2.30 pela igualdade. Portanto, um gerador é co-acessível se e somente se.

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

Def. 2.31 Um gerador G é *Trim* se é acessível e co-acessível.

2.3.2 Bloqueio num SED

A equação (2.8), permite definir a idéia de ausência de bloqueio num sistema a eventos dicretos.

Def. 2.32 Um sistema a eventos discretos com comportamento $L(G)$ e $L_m(G)$ é dito *não bloqueante*, **sse** $\overline{L_m(G)} = L(G)$.

Na definição 2.32, se, a partir de uma seqüência definida pela palavra $u \in L_m$ não pode acontecer mais nada no sistema, este ainda é considerado *SED não bloqueante.* Então pode-se afirmar que um SED será bloqueante quando a partir de uma palavra $u \in (L - L_m)$, não se pode completar alguma tarefa no sistema.

Exe. 2.1 O gerador da Figura 2.3 modela um SED não bloqueante.

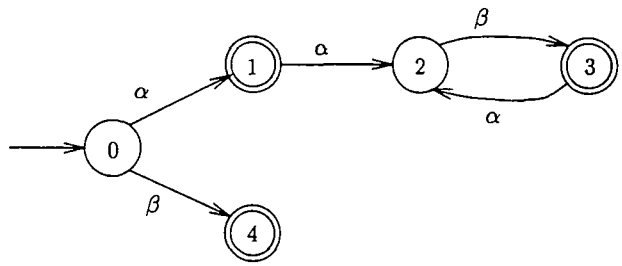


Figura 2.3: SED não Bloqueante.

Conforme segue:

$\Sigma = \{\alpha, \beta\}$
$Q = \{0, 1, 2, 3, 4\}$
$q_0 = 0$
$Q_m = \{1, 3, 4\}$
$L(G) = \varepsilon + \alpha + \beta + \alpha\alpha(\beta\alpha)^*(\varepsilon + \beta)$

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

Observe que este exemplo satisfaz a condição de *SED não bloqueante* ; $L(G) = \overline{L_m(G)}$, ainda que após o estado 4 não possa acontecer nenhum evento.

Exe. 2.2 O gerador da Figura 2.4 modela um SED bloqueante.

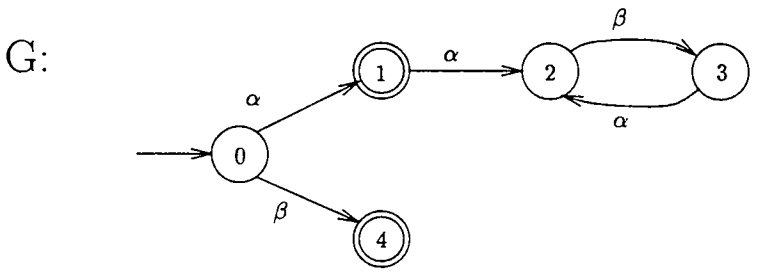


Figura 2.4: SED Bloqueante.

Conforme segue:

$\Sigma = \{\alpha, \beta\}$
$Q = \{0, 1, 2, 3, 4\}$
$q_0 = 0$
$Q_m = \{1, 4\}$
$L(G) = \varepsilon + \alpha + \beta + \alpha\alpha(\beta\alpha)^*(\varepsilon + \beta)$
$L_m(G) = \alpha + \beta$
$\overline{L_m(G)} = \varepsilon + \alpha + \beta$

Observe que $L(G) \neq \overline{L_m(G)}$, uma vez que o modelo apresenta dois estados de bloqueio (estados 2 e 3), a partir dos quais no se pode completar nenhuma tarefa no sistema. Observe ainda que, a partir destes estados existem eventos possíveis de ocorrerem.

2.4 Conclusão

Neste capítulo foi apresentada uma revisão dos conceitos básicos da teoria de linguagens e autômatos, e a relação destes com os sistemas a eventos discretos. As principais conclusões são:

- o comportamento de um SED pode ser modelado por uma linguagem ou por um autômato correspondente;
- a partir dos autômatos é obtida uma classe derivada chamada de Gerador;
- o comportamento de um SED também pode ser modelado por um gerador, no qual se representa conjuntamente os comportamentos gerado e marcado do sistema.

Capítulo 3

Supervisão de Sistemas a Eventos Discretos

Este capítulo apresenta a teoria clássica de controle supervisório para modelar Sistemas a Eventos Discretos, formulada inicialmente por Ramadge-Wonham [RW87], que está expressa em termos de observação e inibição de eventos, realizados por um supervisor minimamente restritivo. Apresenta-se ainda uma extensão desta teoria, formulada por Golaszewski-Ramadge [GR87], introduzindo o conceito de eventos forçáveis.

Sobre esta base se encara o problema de controle supervisório em relação à síntese da máxima sub-linguagem controlável.

O texto apresentado neste capítulo está baseado em: [Won98], [RW87], [KG95], [Zil93], [GR87].

3.1 Abordagem clássica Ramadge-Wonham

No modelo Ramadge-Wonham, um sistema a eventos discretos, de linguagem gerada L (seqüências parciais) e linguagem marcada L_m (tarefas completadas) pode ser representado por um gerador G tal que $L(G) = L$ e $L_m(G) = L_m$.

Neste modelo se assume que a evolução de um sistema é seqüencial. Conseqüentemente toda seqüência gerada de eventos é "prefixo fechada" e define um conjunto que pode ser descrito por uma linguagem prefixo-fechada. A linguagem marcada representa as tarefas

completas pelo sistema e também é parte da linguagem gerada.

3.1.1 Sistemas a eventos discretos com estruturas de controle

Até aqui, um SED foi visto simplesmente como um gerador espontâneo de eventos, sem nenhuma ação restritiva imposta por algum agente externo. Para controlar um SED, considera-se que alguns eventos do sistema podem ser inibidos quando desejado. Sob aspecto diz-se que a forma de controle é a de um *controle permissivo.* [1]. Para isto particiona-se o alfabeto do SED como segue:

$$\Sigma = \Sigma_c \cup \Sigma_u \tag{3.1}$$

onde:

Σ_c é o conjunto de eventos controláveis que podem ser inibidos.

Σ_u é o conjunto de eventos não controláveis, sobre os quais o agente de controle não tem influência.

3.1.1.1 Entradas de controle

Considera-se como entradas de controle, o conjunto de eventos que se deseja habilitar num determinado estado do sistema. Naturalmente, esta entrada de controle não deve tentar inibir eventos não controláveis. Por este motivo, uma entrada de controle é considerada válida somente se contiver o conjunto de eventos não controláveis.

Def. 3.1 Seja um Gerador $G =< \Sigma, Q, \delta, q_0, Q_m >$ e uma partição $\Sigma = \Sigma_u \cup \Sigma_c$. O conjunto de *entradas de controle válidas* associado a G é dado por $\Gamma = \{\gamma | \Sigma_u \subseteq \gamma \subseteq \Sigma\}$

A condição $\Sigma_u \subseteq \gamma$, indica simplesmente que os eventos não controláveis são necessariamente habilitados.

[1] Um tal controle é dito *permissivo* no sentido de que os eventos inibidos não podem ocorrer e os eventos autorizados não ocorrem obrigatoriamente

O conjunto de entradas de controle é representado por $\Gamma \subset 2^{\Sigma}$, onde 2^{Σ} [GR88] representa a combinação de todas as possíveis entradas de controle. $\Gamma = \{\gamma \in 2^{\Sigma} | \Sigma_u \subseteq \gamma \subseteq \Sigma\}$

Propriedades de Γ

- $\Gamma \subseteq 2^{\Sigma}$ é fechado sob a relação de união de conjuntos [Tis91] [GR87]. Por exemplo, se $\gamma, \gamma' \in \Gamma$ então $\gamma \cup \gamma' \in \Gamma$.
- $\Gamma \subseteq 2^{\Sigma}$ é fechado sob a relação de confinamento de conjuntos [TW94]. Por exemplo, se $\gamma \in \Gamma$ e $\gamma \subset \gamma' \subset 2^{\Sigma}$ então $\gamma' \in \Gamma$

Exe. 3.1 Seja uma máquina de 3 estados, como aquele mostrado na Figura 3.1, onde 2^{Σ} é dado por:

$$\begin{array}{llll} \gamma_1 = \{\varepsilon\} & \gamma_2 = \{\alpha\} & \gamma_3 = \{\beta\} & \gamma_4 = \{\lambda\} \\ \gamma_5 = \{\mu\} & \gamma_6 = \{\alpha, \beta\} & \gamma_7 = \{\alpha, \lambda\} & \gamma_8 = \{\alpha, \mu\} \\ \gamma_9 = \{\beta, \lambda\} & \gamma_{10} = \{\beta, \mu\} & \gamma_{11} = \{\lambda, \mu\} & \gamma_{12} = \{\alpha, \beta, \lambda\} \\ \gamma_{13} = \{\alpha, \beta, \mu\} & \gamma_{14} = \{\alpha, \lambda, \mu\} & \gamma_{15} = \{\beta, \lambda, \mu\} & \gamma_{16} = \{\alpha, \beta, \lambda, \mu\} \end{array}$$

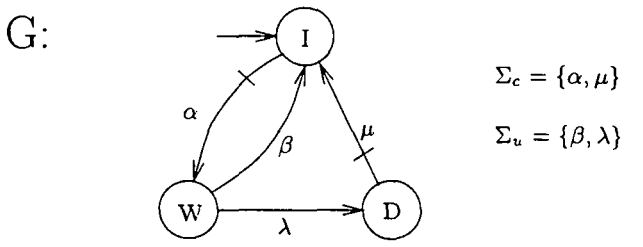


Figura 3.1: Maquina de 3 estados.

dos quais o conjunto de entradas de controle válidas são: $\Gamma = \{\gamma_9, \gamma_{12}, \gamma_{15}, \gamma_{16}\}$. Aplicar por exemplo a entrada de controle γ_{15} no estado inicial, significaria deter o sistema no estado de repouso.

3.1.2 Supervisores

O controle de um SED consiste no chaveamento das entradas de controle, em função do comportamento passado do sistema. Por analogia com a teoria de controle clássica, este comportamento controlado é denominado de *comportamento em malha fechada.*

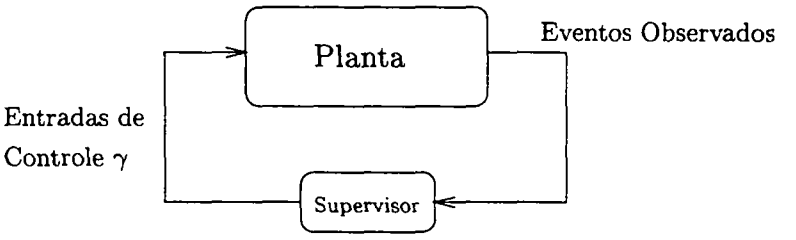


Figura 3.2: SED em malha fechada.

Este conceito leva à distinção do *sistema a controlar* (Planta) e do *agente de controle* (Supervisor), permitindo assim distinguir o funcionamento fisicamente possível do sistema e as restrições ligadas a funcionamentos não desejados. Formalmente pode-se definir supervisores como segue.

Def. 3.2 Um *Supervisor* f , é um mapeamento $f : L \rightarrow \Gamma$ que especifica, para cada cadeia possível de eventos gerados $w \in L$, uma entrada de controle $\gamma = f(w) \in \Gamma$.

O sistema a eventos discretos G controlado por f é denotado por f/G.

Def. 3.3 O comportamento do sistema sob a ação do supervisor, definido pela linguagem $L(f/G) \subseteq L(G)$ é descrito como segue:

1. $\varepsilon \in L(f/G)$.
2. $w\sigma \in L(f/G)$ sse $w \in L(f/G), w\sigma \in L(G)$ e $\sigma \in f(w)$

Def. 3.4 O comportamento marcado de f/G é:

$$L_m(f/G) = L(f/G) \cap L_m(G)$$

A linguagem $L_m(f/G)$ representa simplesmente a parte da linguagem marcada original que sobrevive sob a ação de controle. Se $L_m(G)$ representa as tarefas que podem ser completadas pela planta, então $L_m(S/G)$ representa as tarefas que podem ser completadas sob supervisão.

Propriedades

1. $\{\varepsilon\} \subseteq L(f/G) \subseteq L(G)$
2. $L(f/G)$ é não vazia e prefixo-fechada.
3. $\phi \subseteq L_m(f/G) \subseteq L_m(G)$

Def. 3.5 Diz-se que um supervisor f para a planta G é *não bloqueante* sse $\overline{L_m(f/G)} = L(f/G)$.

3.1.2.1 Realização de um supervisor por um SED

Pode-se realizar um supervisor simplesmente como um outro SED S. Nesse caso, a ação de controle de S sobre G é implícita na estrutura de transição de S.

Def. 3.6 Se $w \in L(f/G)$ então $w \in L(S)$, e $w\sigma \in L(S)$ somente se $\sigma \in f(w)$.

Esta definição garante que as transições não habilitadas pelo supervisor f não aparecem na estrutura de transição de S.

Def. 3.7 Se $w \in L(f/G)$, $w\sigma \in L(G)$ e $\sigma \in f(w)$, $\Rightarrow w\sigma \in L(S)$

Esta condição assegura que as transições habilitadas pelo supervisor f, e fisicamente possíveis, aparecem na estrutura de transição de S.

O gerador S que representa o supervisor pode ser interpretado como uma máquina de estado:

$$S = (\Sigma, X, \xi, x_0, X)$$

Observe que todos os estados do supervisor são considerados marcados, uma vez que a função deste é simplesmente habilitar ou desabilitar eventos sem se importar com a marcação. A ação de controle de S sobre G pode ser visualizada como segue: imediatamente após entrar no estado $x \in X$, e enquanto permanece lá, S desabilita em G aqueles eventos $\sigma \notin \Sigma(x)$.

O funcionamento em malha fechada f/G é representado pelo SED obtido da composição síncrona de S e G $(S||_sG)$. A composição síncrona $S||_sG$ significa que somente as transições permitidas tanto no sistema controlado G, como no supervisor S são permitidas em $S||_sG$.

3.1.3 Controlabilidade e existência de supervisores

De forma geral, para um comportamento fisicamente possível do sistema e um comportamento desejado sob supervisão, o objetivo é construir um supervisor para a planta, tal que o comportamento do sistema se limite ao comportamento desejado. A formulação do comportamento desejado pode ser feito em termos de linguagens marcadas ou linguagens geradas.

Formulação do comportamento desejado em termos de linguagem marcada.

Neste tipo de formulação, o sistema a ser controlado é representado por uma planta G e o comportamento desejado é especificado na forma de uma *linguagem alvo* $K \subseteq L_m(G)$, que representa as tarefas que se deseja sejam completáveis sob supervisão. O objetivo do problema é encontrar (se possível) um supervisor f para G, tal que o comportamento em malha fechada satisfaça a igualdade $L_m(f/G) = K$.

Formulação do comportamento desejado em termos de linguagem gerada.

Também pode-se especificar o comportamento desejado na forma de uma *linguagem alvo* prefixo-fechada $K \subseteq L(G)$, que representa o comportamento fisicamente possível desejado sob supervisão. O objetivo do problema é então encontrar (se possível) um supervisor f para G, tal que o comportamento em malha fechada satisfaça a igualdade $L(f/G) = K$.

3.1.3.1 Controlabilidade

Def. 3.8 Dados um alfabeto $\Sigma = \Sigma_u \cup \Sigma_c$ e linguagens K e L sobre Σ tais que $K \subseteq L$, a linguagem K é dita *controlável* em relação a L sse

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

Esta definição exige que qualquer prefixo w de uma palavra de K ($w \in \overline{K}$), quando seguido de um evento não controlável $\sigma \in \Sigma_u$, tal que $w\sigma \in L$, deve ser ainda prefixo de uma palavra de K ($w\sigma \in \overline{K}$).

3.1.3.2 L-fechamento

Este conceito de *fechamento em relação a uma linguagem* será, juntamente com o conceito de controlabilidade introduzido acima, fundamental para o estabelecimento das condições de existência de supervisores para problemas abstratos formulados em termos de linguagens marcadas.

Def. 3.9 Dadas duas linguagens quaisquer K, $L \subseteq \Sigma^*$, K é dita *fechada em relação a* L ou simplesmente *L-fechada* sse

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

Na abordagem de Ramadge-Wonhan [RW87] as linguagens K e L da definição 3.9 aparecem envolvendo a linguagem-alvo $K \subseteq L_m(G)$ e $L = L_m(G)$. Se K é *$L_m(G)$-fechada*, isto é, se $K = \overline{K} \cap L_m(G)$, então todo prefixo de qualquer palavra de K que pertença a $L_m(G)$ é também uma palavra de K.

Exe. 3.2 Seja o gerador da Figura 3.3:

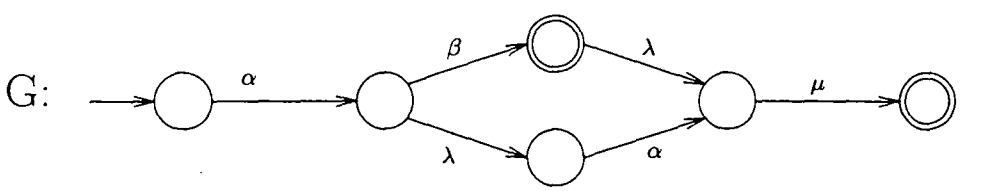


Figura 3.3:

A linguagem marcada do gerador é: $L_m(G) = \{\alpha\beta, \alpha\beta\lambda\mu, \alpha\lambda\alpha\mu\}$. Se a linguagem alvo desejada é $K \subseteq L_m(G)$, tal que $K = \{\alpha\beta\lambda\mu\}$, veja que K não é *$L_m(G)$-fechada*, pois, uma vez que a geração da palavra $\alpha\beta\lambda\mu$ exige a geração previa de seus prefixos, é inevitável que $\alpha\beta$ seja gerada cada vez que se gerar $\alpha\beta\lambda\mu$. O supervisor que permita a ocorrência de $\alpha\beta\lambda\mu$ é obrigado a permitir também a ocorrência de $\alpha\beta$, indicando que um supervisor para K só pode existir se K for *$L_m(G)$-fechada*.

3.1.3.3 Existência de supervisores

O objetivo desta seção é a de apresentar os teoremas de existência de supervisores para problemas formulados em termos de linguagens geradas e em termos de linguagens marcadas.

Previamente são apresentadas algumas definições acerca de supervisores que serão utilizadas no desenvolvimento a seguir.

Def. 3.10 Um supervisor f para uma planta G é dito *completo*, se a função de transição do supervisor é definida para todo evento habilitado fisicamente possível.

Def. 3.11 Um supervisor f para a planta G é dito "não bloqueante" sse

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

Esta definição implica que, de qualquer estado do comportamento em malha fechada da planta, uma tarefa pode ser completada no sistema.

As duas definições anteriores são articuladas na seguinte definição:

Def. 3.12 Um supervisor completo e não bloqueante é dito *supervisor próprio.*

A proposição a seguir estabelece as condições necessárias e suficientes de existência de supervisores para problemas formulados em termos de *linguagens marcadas* [Won98] [KG95].

Proposição. 3.1 " Dados um gerador G tal que $L_m(G)$ represente as tarefas que podem ser completadas na ausência de ação de controle, e uma linguagem alvo $K \subseteq L_m(G)$, $K \neq \phi$, existe um supervisor não bloqueante f tal que $L_m(f/G) = K$ se e somente se K é L_m-fechada e L-controlável ".

O resultado a seguir estabelece as condições necessárias e suficientes de existência de supervisores para problemas formulados em termos de *linguagens geradas* [Won98] [KG95]. Este problema pode ser visto como um caso particular do anterior, considerando todos os estados da planta como marcados.

Corolário 3.1 " Dados um gerador G tal que $L(G)$ represente o comportamento livre fisicamente possível do sistema, e uma linguagem alvo $K \subseteq L(G)$, $K \neq \phi$, existe um supervisor f tal que $L(f/G) = K$ se e somente se K é *prefixo-fechada* e *L-controlável* ".

3.1.4 Supervisor minimamente restritivo. Máxima linguagem controlável

Na seção precedente foram estabelecidas as condições de existência de um supervisor. Em particular, uma condição incontornável é a de controlabilidade da linguagem, que corresponde ao funcionamento em malha fechada desejado, em relação à linguagem gerada pelo SED a ser controlado.

No caso em que a linguagem K que especifica o comportamento desejado não é controlável, veremos nesta seção que existe uma *aproximação de K* que é a *máxima linguagem contida em K*, (denotado por $K\uparrow$) e que esta é única. Além disso, sob certas condições, esta linguagem satisfaz os requisitos de fechamento impostos pela proposição estabelecida, e por tanto as condições de um supervisor associada. A este supervisor chamaremos de *Supervisor minimamente restritivo* ou *ótimo.*

Pode-se então anunciar o seguinte problema abstrato de síntese de supervisores

Problema de Controle Supervisório (PCS): O problema de controle supervisório é a de encontrar um supervisor para um SED, de tal forma que as seqüências de eventos geradas sob controle permaneçam dentro de um subconjunto desejado.

Formalmente:
Dadas uma planta G com estrutura de controle Γ, uma linguagem-alvo marcada $E \subseteq L_m(G)$ e uma mínima linguagem admissível $A \subseteq E$, encontrar um supervisor próprio f tal que $A \subseteq L_m(f/G) \subseteq E$

3.1.4.1 Existência da máxima linguagem controlável contida numa dada linguagem K

Sejam $L(G)$ a linguagem gerada por um SED G e $K \subseteq \Sigma^*$ a linguagem do comportamento desejado. Seja:

$\mathcal{C}(K) = \{E \subseteq K | E$ é controlável em relação a G, ou seja , $\overline{E}\Sigma_u \cap L(G) \subseteq \overline{E}\}$

ou:

$\mathcal{C}(K) =$ Conjunto de linguagens controláveis contidas em K

As proposições a seguir detalham os características da máxima sub-linguagem controlável [Won98].

Proposição. 3.2 $\mathcal{C}(K)$ é não vazio e é fechado para operação de união de conjuntos. Em particular, $\mathcal{C}(K)$ contem um elemento supremo único denotado $sup\mathcal{C}(K)$ ou, $K\uparrow$.

Proposição. 3.3 Seja $K \subset L_m(G)$, uma linguagem L_m-*fechada* ($K = \overline{K} \cap L_m$). Então $sup\mathcal{C}(K)$ é L_m-*fechada.*

Proposição. 3.4 Seja $K \subset L(G)$ uma linguagem prefixo-fechada ($K = \overline{K}$). Então $sup\mathcal{C}(K)$ é fechada.

O corolário a seguir decorre diretamente das proposições 3.2, 3.3, 3.4.

Corolário 3.2 Dado um SED descrito por G, não bloqueante, com funcionamento $L(G)$ e funcionamento marcado $L_m(G)$ temos:

1. Se $K \subset L$, $K \neq \phi$ e $K = \overline{K}$, existe um supervisor f tal que $L(f/G) = sup\mathcal{C}(K)$.
2. Se $K \subset L_m$, $K \neq \phi$ e $K = \overline{K} \cap L_m$, existe um supervisor f tal que $L_m(f/G) = sup\mathcal{C}(K)$.

Os supervisores do corolário 3.2 são ótimos, no sentido que são os que geram compor-

tamento em malha fechada dentro das especificações descritas por K, da forma menos restritiva possível. Este corolário estabelece que sobre as condições citadas, a solução do problema de controle supervisório é dada pela máxima linguagem controlável contida em K.

A solução para o problema geral poderia ser obtida através de um algoritmo que calcule a máxima linguagem controlável e L_m- fechada contida numa especificação genérica K [Zil93].

Na seção seguinte será apresentado o algoritmo que compute a máxima sub-linguagem controlável contida em K ($supC(K)$).

3.1.4.2 Síntese da máxima linguagem controlável $supC(K)$

Sejam:

K: uma linguagem regular.
G: um SED com comportamento $L(G)$ e $L_m(G)$.
S: um gerador trim tal que $L_m(S) = K \subset L(G)$.
Σ: $\Sigma_u \cup \Sigma_c$.

O algoritmo abaixo permite calcular a máxima linguagem controlável contida na linguagem alvo K [KG95].

Algoritmo:

1. Obter G' , um gerador trim tal que $L(G') = L_m(G') = L(G)$ (Marcar todos os estados de G)

2. Construir $C_i = G'||_s S$ com $i = 0$;

   observe que:

   $L_m(C_0) = L_m(G') \cap L_m(S) = L(G) \cap L_m(S) = L_m(S) = K$

   note que $\Sigma_{(S)} = \Sigma_{(G)}$

3. Identificar os maus estados de C_i ;

   São os estados $(x, y) \in C_i$ tais que: transições não-controláveis no estado x da planta (G') não pertencem ao conjunto de transições de C_i no estado (x, y), (i.e. $\Sigma_u(G')(x) \not\subset \Sigma_u(C_i)(x, y)$);

4. Obter C_i' eliminando os maus estados de C_i e as transições a eles associadas e fazer $C_{i+1} = trim C_i'$.

5. Se $C_{i+1} = C_i'$, parar, pois $sup\mathcal{C}(K) = L_m(C_i)$. Se não, fazer $i = i + 1$ e voltar ao item 3.

3.2 Abordagem introduzindo o conceito de eventos "forçáveis "

Nesta seção estenderemos alguns conceitos da teoria clássica introduzidos em [RW89] para tratar uma nova classe de eventos, chamados *eventos forçáveis* [GR87].

A inclusão de eventos forçáveis na modelagem de SED's implica na revisão de tudo o que estiver relacionado com controlabilidade.

3.2.1 Controlabilidade de SED's com eventos forçáveis

Os conceito relacionados a controlabilidade são:

- Particionamento do conjunto de eventos Σ.
- Entradas de controle.
- Linguagens controláveis.
- Síntese da máxima linguagem controlável $sup\mathcal{C}(K)$.

3.2.1.1 Particionamento do conjunto de eventos Σ

No modelo original de [RW89], Σ está dividido em dois subconjuntos distintos Σ_u e Σ_c, onde Σ_u corresponde aos eventos não controláveis, e Σ_c aos eventos controláveis

Nesta nova abordagem , o conjunto de eventos Σ é particionado em três conjuntos distintos Σ_u, Σ_c e Σ_f. Eventos em Σ_u e Σ_c têm a mesma interpretação que no modelo original. Os eventos em Σ_f representam os eventos forçáveis, interpretados como eventos que ocorrem se e somente se eles são forçados por alguma entrada externa. Os eventos

forçáveis não são espontâneos, e assumimos que, ao forçar-se um evento, este ocorre imediatamente, antecipando-se à ocorrência de outros eventos. Na prática esta hipótese pode ser satisfeita com a implementação de um controlador de resposta mais rápida que o intervalo mínimo com que os eventos da planta possam ser gerados. (ex: Seja uma planta onde os eventos espontâneos são gerador por um CLP. Pela restrição tecnológica do CLP o intervalo mínimo entre dois eventos será de 20 *ms*. Se for implementado um controle feito pelo computador (486DX2 66Mhz), a resposta será 75 vezes mais rápida). Desta forma é garantida a antecipação de um evento forçável à ocorrência de qualquer evento não controlável.

3.2.1.2 Entradas de controle

Com esta nova abordagem, o conjunto de entradas de controle é modelado como:

$$\Gamma = \left\{ \begin{array}{c} \gamma | \gamma \in 2^{\Sigma} \wedge \quad \gamma = \{\sigma\} \quad Para\ algum \quad \sigma \in \Sigma_f \\ \vee \quad \gamma \in 2^{\Sigma_u \cup \Sigma_c} \ Com \ \Sigma_u \subseteq \gamma \end{array} \right\} \tag{3.2}$$

Note que Γ nem está fechado sob a união nem a interseção. Isto quer dizer que existem dois tipos de entrada de controle bem definidos, um que contem só eventos forçáveis e outro que contem só eventos controláveis/não-controláveis.

Ampliando a funcionalidade de um supervisor, é possível considerar um *supervisor não determinístico*, definido como o mapeamento $\phi(x) \rightarrow 2^{\Gamma}$. O supervisor pode selecionar qualquer entrada de controle dentro de um subconjunto $\phi(x) \subseteq \Gamma$ numa forma não deterministica. Pode-se ver que, se se considera o novo conjunto $cl(\Gamma)$ defindo pelo fechamento de Γ para a operação união de conjuntos, então a cada supervisor não determinístico ($\phi : X \rightarrow 2^{\Gamma}$) corresponde um supervisor deterministico equivalente ($\phi' : X \rightarrow cl(\Gamma)$).

3.2.1.3 Linguagens controláveis

Estendendo o conceito de linguagem controlável introduzido por [RW89] temos:

Def. 3.13 Para cada $K \subseteq \Sigma^*$ e cada $s \in \overline{K}$ define-se *o conjunto ativo de K após "s"* como:

$$\Sigma_K(s) = \{\sigma \in \Sigma | s\sigma \in \overline{K}\}$$

Por exemplo, para o caso de um gerador G , $\Sigma_{L(G)}(s)$ representa o conjunto de eventos possíveis de ocorrer após o estado $\delta(s, q_0)$ de G.

Uma linguagem controlável é definida como segue:

Def. 3.14 Uma linguagem $K \subseteq L(G)$ com estrutura de controle Γ é L(G)-controlável se para qualquer $s \in \overline{K}$ existe uma entrada de controle $\gamma \in \Gamma$ tal que :

1. $\Sigma_K(s) \in \gamma$; e
2. $s\gamma \cap L(G) \subseteq \overline{K}$.

O segundo item é reconhecido como a condição de controlabilidade de [RW89]. Esta condição assegura que exista uma entrada de controle para manter a evolução da cadeia em K.

O item 1 assegura que esta entrada de controle possa ser escolhida permitindo todas as alternativas possíveis em K. Estas duas condições conduzem a: $\forall s \in \overline{K} \exists \gamma \in \Gamma | \gamma \cap \Sigma_{L(G)}(s) = \Sigma_K(s)$.

Para o caso de um supervisor, onde o conjunto de entradas de controle é fechado sob a relação união de conjuntos, a condição de controlabilidade é interpretada como: $\forall s \in \overline{K} \exists \gamma \in cl(\Gamma) | \gamma \cap \Sigma_{L(G)}(s) = \Sigma_K(s)$

Por exemplo, para um estado da planta onde exista a possibilidade de ocorrerem eventos controláveis, não controláveis e forçáveis como é o caso do gerador G da Figura 3.4, tem-se três especificações possíveis não controláveis (E_1, E_2 e E_3). As especificações E_1 e E_2 são não controláveis uma vez que, ao permitir a ocorrência do evento controlável, o evento não controlável também poderá ocorrer. No caso da especificação E_3, não é possível desabilitar a ocorrência do evento não controlável.

No entanto, se houver uma especificação contendo apenas um evento forçável (E_6), ela será controlável, desde que este evento é considerado instantâneo e antecipa-se à ocorrência do evento não controlável.

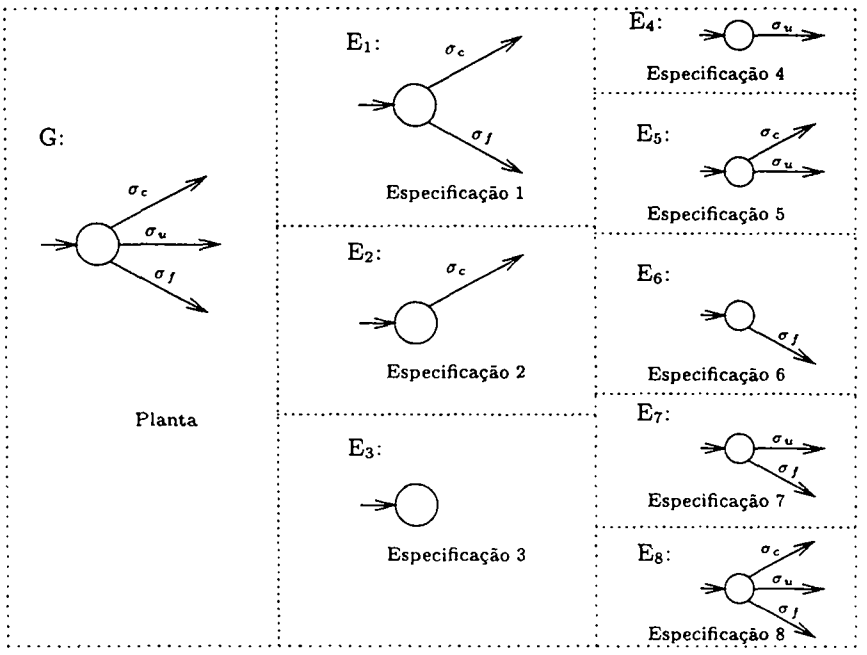


Figura 3.4: Especificações controláveis e não controláveis

3.2.1.4 Síntese da máxima linguagem controlável $sup\mathcal{C}(K)$

Sejam:

K: uma linguagem regular.
G: um SED com comportamento $L(G)$, $L_m(G)$, e com estruturas de controle $cl(\Gamma)$.
S: um gerador trim tal que $L_m(S) = K \subset L(G)$.
Σ: $\Sigma_f \cup \Sigma_u \cup \Sigma_c$.

O seguinte algoritmo permite o cálculo da máxima sub-linguagem controlável contida numa linguagem alvo K, quando no sistema são introduzidos eventos forçáveis na modelagem.

Este algoritmo é uma extensão daquele apresentado por Kumar [KG95] e uma das contribuições deste trabalho.

Algoritmo:

1. Obter G' , um gerador trim tal que $L(G') = L_m(G) = L(G)$ (Marcar todos os estados de G)

2. Construir $C_i = G'||_s S$ com $i = 0$;

   observe que:

   $L_m(C_0) = L_m(G') \cap L_m(S) = L(G) \cap L_m(S) = L_m(S) = K$

   note que $\Sigma_{(S)} = \Sigma_{(G)}$

3. Identificar os maus estados de C_i e/ou as más transições.

   - Identificar os *possíveis* maus estados de C_i;

     São os estados $(x, y) \in C_i$ tais que: transições não-controláveis no estado x da planta (G') não pertencem ao conjunto de transições de C_i no estado (x, y), (i.e. $\Sigma_u(G')(x) \not\subset \Sigma_u(C_i)(x, y)$);

   - Dos *possíveis* maus estados separar os bons; São os estados $(x, y) \in C_i$ que têm como saída eventos forçáveis (i.e. $(\exists \Sigma_f(C_i)(x, y) \in \Sigma(C_i)(x, y))$). Se não há, ir ao ítem 4.

   - Dos estados separados anteriormente identificar as más transições ; São aquelas transições controláveis e não controláveis que saem desses estados.

4. Obter C_i' eliminando os maus estados de C_i e as transições a eles associadas. Eliminar as más transições. fazer $C_{i+1} = trim C_i'$.

5. Se $C_{i+1} = C_i'$ parar, pois $sup\mathcal{C}(K) = L_m(C_i)$. Se não fazer $i = i + 1$ e voltar ao ítem 3.

Os Eventos forçáveis que não são saída dos *possíveis* maus estados comportam-se como eventos controláveis, no sentido que a ocorrência é considerado opcional. Neste caso a decisão será tomada por um agente externo.

Neste algoritmo note que, além de tratar maus estados, são consideradas más transições. O fato de apagar as más transições descritas no algoritmo, fisicamente significa aplicar um evento forçável à planta antecipando-se à ocorrência do algum evento (não controlável) que leve o sistema para fora do comportamento desejado.

3.3 Abordagem Modular

Nesta seção discutimos a abordagem modular para a síntese de supervisores em sistemas a eventos discretos [KG95] [Won98].

Na seção anterior projetava-se um supervisor para a especificação resultante do produto síncrono das diferentes especificações, abordagem conhecida como *abordagem monolítica*. Na *abordagem modular* projeta-se supervisores para cada especificação em separado, e aplica-se estes supervisores combinados, de modo a se atender a especificação global no sistema resultante em malha fechada.

O objetivo do controle modular é:

1. Diminuir a complexidade computacional em relação à explosão de estados quando é feito o produto síncrono entre a planta e a especificação.
2. Aumentar a flexibilidade em caso de mudanças. Por exemplo, se uma sub-tarefa é alterada, então deve somente ser necessário projetar novamente a componente do supervisor correspondente.

Infelizmente, estas vantagens não são sempre possíveis de serem exploradas. A abordagem modular pode ser usada, somente se as especificações são *modulares* [Won98].

3.3.1 Conjunção de supervisores

Sejam f_1 e f_2 supervisores próprios para G tais que cada um dos supervisores são controláveis em relação a G e (f_1/G), (f_2/G) são não bloqueantes, isto é:

$$\overline{L_m(f_1/G)} = L(f_1/G), \qquad \overline{L_m(f_1/G)} = L(f_1/G).$$

Então podemos definir.

Def. 3.1 Um evento σ é habilitado pela *conjunção de supervisores* $f_1 \wedge f_2$ **sse** σ é habilitado por f_1 e f_2 simultaneamente.

Def. 3.2 $K_1, K_2 \subseteq \Sigma^*$ são ditas serem *modulares* se $(\overline{K_1 \cap K_2}) = \overline{K}_1 \cap \overline{K}_2$

Para descrever a ação de f_1 e f_2 é formulada a seguinte proposição [Won98].

Proposição 3.1 Seja G uma planta e $K_1, K_2 \subset \Sigma^*$ controláveis e L_m-*fechadas*, e f_1, f_2 supervisores não-bloqueantes tais que $L_m(f_i/G) = K_i$.

Então:

1. $L_m((f_1 \wedge f_2)/G) = K_1 \cap K_2$; e
2. $f_1 \wedge f_2$ é não-bloqueante **sse** K_1, K_2 são modulares.

Obs: Se S_1, S_2 são geradores *trim* tais que $L_m(S_i) = K_i$, então $L_m(S_1||S_2) = K_1 \cap K_2$

3.4 Conclusão

As principais conclusões deste capítulo são:

- quando se introduzem eventos forçáveis na modelagem de SED's, a entrada de controle definida após uma seqüência gerada poderá ser tratada com um supervisor não determinístico, ou por seu equivalente determinístico obtido pelo fechamento do conjunto de entradas de controle($cl(\Gamma)$);
- o algoritmo para a síntese da máxima linguagem controlável dentro da abordagem clássica R-W, deve ser modificado para contemplar a consideração de eventos forçáveis. Esta modificação e sua implementação são contribuições desta dissertação.

Capítulo 4

Implementação do Controle Supervisório

Neste capítulo se apresenta uma arquitetura para implementação de controle supervisório de sistemas a eventos discretos realizáveis na prática, proposta por [MC97]. Esta metodologia abrange o estudo da natureza dos eventos utilizados ao modelar um sistema. Os eventos segundo a sua natureza se classificam em: controláveis, não-controláveis e forçáveis.

Com base nesta abordagem, descreve-se uma implementação em dois níveis: uma implementação *off-line*, onde se preparam os dados de entrada do sistema a ser controlado; outra implementação *on-line* que interage em tempo real com o sistema.

4.1 Arquitetura de suporte

A arquitetura de referência proposta por [MC97], a ser implementada, é mostrada na figura 4.1. Esta arquitetura consiste de 4 módulos

- Supervisor
- Módulo intermediário
- Planta ou nível físico
- Interface

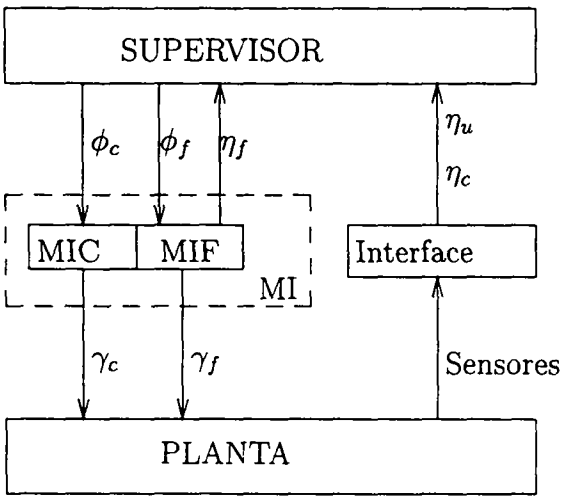


Figura 4.1: Arquitetura física para o controle supervisório de sistemas reais

Na continuação serão descritos em detalhe cada um destes módulos.

Quando o sistema começa a evoluir, é o supervisor quem inicia o processo. A função do supervisor é a de habilitar e desabilitar eventos correspondentes ao estado atual, e enviar esta informação para o elemento seguinte, chamado *Módulo intermediário* (*MI*). O *MI* é capaz de gerar eventos que são comandos enviados ao nível físico. De posse da informação vinda do supervisor, o *MI* decidirá sobre o envio de algum comando.

Este módulo pode ser subdividido em dois sub-módulos: um que trata dos eventos controláveis (*MIC*), e outro que trata dos eventos forçáveis (*MIF*).

Os eventos não-controláveis não necessitam ser tratados pelo *MI*. Estes eventos em geral são sempre mapeados em sinais dos sensores, e são gerados pela planta sem que o supervisor possa intervir em sua ocorrência.

A cada mudança de estado do sistema, o supervisor envia ao *MIC* a informação tanto dos eventos controláveis habilitados quanto dos desabilitados (ϕ_c). Ambas situações sempre representam comandos que devem ser enviados ao nível físico. Ao receber esta informação, o *MIC* deve imediatamente enviar os comandos associados ao nível físico (γ_c). O *MIC* não precisa realizar nenhum tipo de tomada de decisão sobre qual comando enviar.

No caso dos eventos *forçáveis*, o supervisor envia ao *MIF* somente os eventos habilitados (ϕ_f), uma vez que a desabilitação destes eventos significa apenas a não aplicação do comando associado. O *MIF* deve, em seguida, decidir se aplica ou não o comando associa-

do ao evento (γ_f). Além disso, no caso de aplicação de um comando, o *MIF* deve decidir, entre vários eventos forçáveis, qual efetivamente será aplicado. Neste caso exige-se uma tomada de decisão, que pode ser feita utilizando-se regras de prioridades, ou mesmo a aplicação de métodos probabilísticos.

Quando a informação enviada pelo supervisor contém a habilitação de eventos forçáveis e também a habilitação e desabilitação de eventos controláveis, a aplicação dos comandos associados aos eventos controláveis pelo *MIC* deve ser imediata, enquanto que a aplicação dos comandos associados aos eventos forçáveis depende de uma decisão a ser tomada pelo *MIF*. Neste caso, se não houver a aplicação de nenhum evento forçável, o próximo evento, (controlável ou não-controlável), será gerado pela planta. Porém, em casos onde a informação enviada pelo supervisor apenas contém a habilitação de eventos forçáveis, a aplicação destes eventos pelo *MIF* é obrigatório . Caso contrário, ou um evento não-controlável indesejado será gerado pela planta, ou nada mais acontece e o sistema pára de evoluir.

Se um evento forçável for aplicado (γ_f), o *MIF* deve enviar esta informação de volta ao Supervisor (η_f) de modo a alertá-lo sobre a ocorrência do evento, porque a planta não pode fornecer este tipo de informação, pois a mesma é gerada exclusivamente pelo MIF.

A *planta* é um gerador espontâneo de eventos que são enviados ao supervisor por meio de um outro módulo que realiza uma interface entre o nível físico e o nível supervisório. Este módulo, denominado *Interface*, recebe os sinais de sensores do nível físico e os mapea em eventos, não-controláveis ou controláveis, correspondentes no nível de supervisão (η_c, η_u).

4.2 Implementação

A implementação do sistema de controle foi desenvolvida em duas etapas:

- Etapa de preparação (off-line), que descreve a preconfiguração do sistema que servirá de entrada ao controlador.

- Etapa de execução (on-line), que descreve a maneira como o controlador interage com o nível físico em tempo real.

4.2.1 Etapa de preparação

Nesta etapa de implementação foram desenvolvidas duas ferramentas de suporte, que ajudam a preparar os dados de entrada do sistema a ser controlado.

Uma das ferramentas é o CONDES 3.0 *"com eventos forçáveis"* (Apêndice A), que permite modelar sistemas a eventos discretos, incluindo a noção de eventos forçáveis. Com o CONDES 3.0 o usuário poderá gerar a lei de controle de um SED segundo o comportamento desejado para o sistema, obtendo uma linguagem controlável minimamente restritiva. O CONDES 3.0 *"com eventos forçáveis"* é uma renovação da versão anterior CONDES 2.0 adicionando a funcionalidade dos eventos forçáveis na modelagem, além de facilitar a manipulação dos dados com uma interface gráfica mais intuitiva. Os algoritmos computacionais do CONDES 3.0 para o cálculo das funções implementadas são detalhados no apêndice D.

A outra ferramenta de suporte é o CONCEL. Esta ferramenta facilita preparar a base de dados que servirá de entrada de dados para o controlador a ser implementado (Gerenciador). O CONCEL permite fazer o mapeamento de um modelo de sistemas a eventos discretos, com os eventos físicos da planta associados a sensores e atuadores.

Os dados de entrada para o CONCEL são:

- A estrutura do supervisor e a Planta, que podem ser obtidas como resultado da modelagem no CONDES 3.0, ou serem calculadas e introduzidas manualmente
- As etiquetas correspondentes a eventos associados a sensores e atuadores do nível físico.

A ocorrência de um evento controlável e não controlável, é mapeado na ocorrência de um sensor do nível físico, e a habilitação ou desabilitação dos eventos controláveis são mapeados com os comandos dos atuadores. Os eventos forçáveis também são mapeados com os comandos dos atuadores.

Para maior detalhe acerca do mapeamento, referir-se ao manual de referência que está descrito no apêndice B.

4.2.2 Etapa de execução

Foi desenvolvida uma terceira ferramenta, chamada GERENCIADOR, encarregada de interagir em tempo real diretamente com o nível físico. O GERENCIADOR é o sistema de controle implementado segundo a arquitetura descrita anteriormente.

Para iniciar o funcionamento do GERENCIADOR, é preciso carregar primeiramente, os dados do sistema, que foram preparados anteriormente no CONCEL e executar. Para maior detalhe a cerca do uso e detalhes técnicos do GERENCIADOR, consultar o manual de referencia descrito no apêndice C.

Na seqüência descrevemos o comportamento interno do GERENCIADOR. Em termos gerais, o fluxograma da figura 4.2 descreve o comportamento de cada parte do GERENCIADOR.

O supervisor foi implementado como o jogador de um gerador $S = \{\sum, Q, \delta, q_0, Q\}$ e um conjunto de entradas de controle Γ. Para o jogador, $x = \delta(\sigma, x_i)$, é usado como uma estrutura de dados em forma de matriz, com estados como *linhas* e eventos como *colunas*:

$$State = \delta[State][Event]$$

O motivo de usar essa forma de estrutura de dados reside em que o acesso a um determinado dado é rápido, e para este caso satisfaz a problemática de resposta em tempo real requerida. A desvantagem em se usar esta estrutura está em que a memória requerida para armazenar os dados pode ser grande, mesmo que nem todos os campos estejam ocupados.

A função do supervisor é habilitar e desabilitar eventos, para depois enviá-los ao módulo intermediário. É implementado de tal forma que o supervisor conheça o comportamento livre da planta, e em conseqüência o supervisor envie ao modulo intermediário unicamente a habilitação e desabilitação dos eventos fisicamente possíveis. Evita-se assim o envio de comandos desnecessários.

São consideradas as seguintes hipóteses:

- Os números ímpares de 1 a 999 são reservados para os eventos controláveis
- Os números pares de 0 a 998 são reservados para os eventos não-controláveis

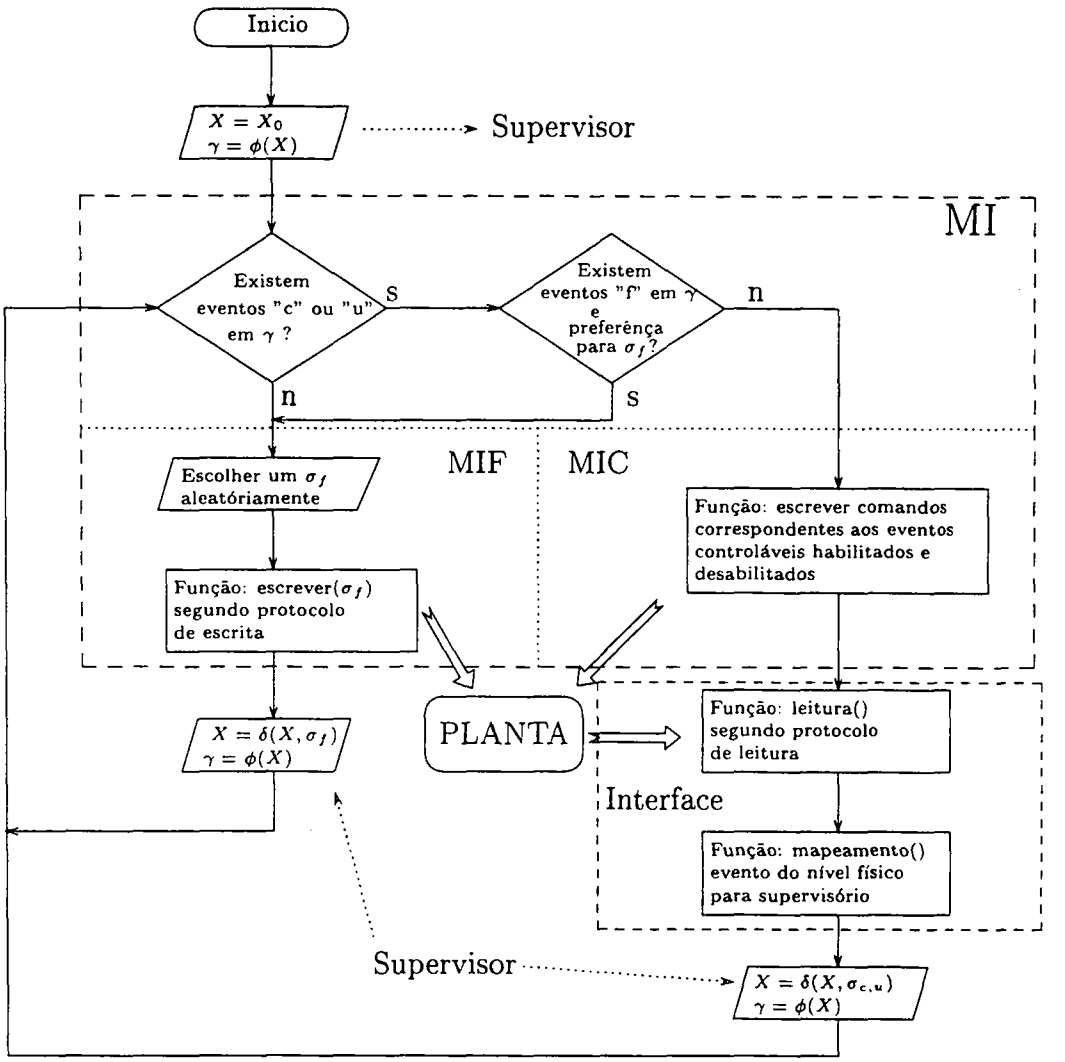


Figura 4.2: Fluxograma da implementação do sistema de controle

- Os números maiores ou iguais a 1000 são reservados para os eventos forçáveis

4.2.2.1 Módulo intermediário

Para cada estado do sistema, a entrada de dados para o módulo intermediário é fornecida pelo supervisor em forma de entrada de controle. Também é enviada ao MI a lista de eventos controláveis fisicamente possíveis que são desabilitados.

Segundo o conteúdo da entrada de controle, o MI reage aplicando comandos associados à habilitação ou desabilitação de eventos controláveis através do MIC, ou aplicando um comando associado a um evento forçável através do MIF.

No caso em que a entrada de controle contenha eventos controláveis/não-controláveis em conflito com eventos forçáveis , o MI terá que decidir qual dos sub-módulos entra em operação (MIC ou MIF) segundo critério aqui denominado *critério de prioridade.*

Uma vez feita a escolha do sub-módulo (MIC ou MIF), ou caso a entrada de controle não contenha eventos forçáveis em conflito com controláveis/não controláveis, a aplicação dos comandos associados aos evento forçáveis ou à habilitação, desabilitação de eventos controláveis, foram implementados segundo um critério chamado *critério de necessidade.*

Para efeito acadêmico foram implementados alguns de prioridade e necessidade, porém o sistema esta aberto a qualquer outro tipo de procedimento específico. Dependendo da escolha de um critério, o sistema vai ser dirigido dentro de um determinado subconjunto do comportamento em malha fechada.

Critério de prioridade

Caso a entrada de controle contenha *eventos controláveis/não-controláveis e eventos forçáveis* , só um dos módulos, MIC, MIF, ou nenhum deles entrará em operação. A escolha de qual módulo entra em operação foi implementada da forma seguinte:

- por "default" é dada maior prioridade aos eventos controláveis/não-controláveis, em relação aos eventos forçáveis, isto é, se para um determinado estado do sistema, a entrada de controle contiver eventos controláveis/não-controláveis e forçáveis, os comandos a serem enviados à planta correspondentes aos eventos forçáveis serão inibidos, ficando como entrada de controle unicamente eventos controláveis e não-controláveis. A habilitação e desabilitação dos eventos controláveis são tratadas pelo

MIC, sendo traduzidas em comandos a serem enviados ao nível físico. Em seguida, o supervisor fica aguardando a ocorrência de um evento na planta. Pode ocorrer que, para um determinado estado do sistema, somente existam eventos forçáveis e não-controláveis fisicamente possíveis. Neste caso, nem o MIF nem o MIC operam, uma vez que a prioridade é dos eventos não-controláveis. Além disso, não existem eventos controláveis habilitados nem desabilitados a serem tratados pelo MIC;

- o usuário tem a possibilidade de dar prioridade aos eventos forçáveis. Neste caso, para um determinado estado do sistema, quando a entrada de controle contenha eventos controláveis/não-controláveis e forçáveis, o MIF entra em operação. O MIF escolhe aleatoriamente dentre os eventos forçáveis habilitados e envia o comando correspondente à planta, antecipando-se à ocorrência de algum evento controlável/não-controlável. Finalmente, informa ao supervisor a mudança de estado;

- existe uma opção de escolha aleatória entre os dois casos anteriores.

Critério de necessidade

- para um estado do sistema, se a entrada de controle contiver *somente eventos forçáveis*, o MIF necessariamente entra em operação, escolhe um evento dentre os eventos forçáveis habilitados e envia o comando associado a esse evento à planta. No caso que não for aplicado, um evento não desejado poderia acontecer, ou o sistema poderá ficar em bloqueio (a escolha do evento forçável foi implementada como aleatória). Para finalizar o ciclo, informa ao supervisor a mudança de estado;

- se a entrada de controle contiver *somente eventos controláveis/não-controláveis*, a habilitação e desabilitação dos eventos controláveis, necessariamente é tratada pelo MIC. O MIC envia os comandos à planta referentes à habilitação e desabilitação dos eventos controláveis. O supervisor fica aguardando então a resposta da planta, que pode ser um evento controlável ou não-controlável.

4.2.2.2 Interface

A interface é um adaptador de eventos do nível físico para o nível supervisório. A interface fica monitorando a porta paralela do computador, aguardando a ocorrência de algum evento na planta. Se um evento acontece, a interface verifica se ele é um evento

relevante [1], caso em que mapeia o evento do nível físico com o evento correspondente do nível supervisório e envia o dado ao supervisor.

Os detalhes técnicos da interface são explicados no manual do apêndice C

4.3 Conclusão

Neste capítulo foi apresentada a arquitetura de suporte para implementação de controle supervisório proposta na referência [MC97]. Na seqüência a implementação foi desenvolvida em duas etapas: uma etapa de preparação, onde foram desenvolvidas ferramentas para modelar e preconfigurar SED's off-line (CONDES e CONCEL); na segunda etapa foi desenvolvido o controlador(GERENCIADOR) de acordo com a arquitetura proposta. O Gerenciador é uma ferramenta encarregada de interagir diretamente com o nível físico, em tempo real.

[1] Eventos relevantes num determinado estado são aqueles fisicamente possíveis de ocorrer naquele estado

Capítulo 5

Aplicação à Célula de Manufatura do Laboratório de Automação Industrial (LAI)

O sistema estudado a seguir é uma célula de manufatura localizada no Laboratório de Automação Industrial(LAI) do departamento de Automação e Sistemas da Universidade Federal de Santa Catarina. O sistema está composto por estações de espera, estações de inspeção, robô manipulador, estação de rejeição, entre outros, tal como pode ser visto na Figura 5.1.

O objetivo deste capítulo é fazer uma comparação da implementação do sistema de controle descrito anteriormente na célula de manufatura, utilizando se possível modelos alternativos, com ou sem eventos forçáveis. Para isto serão abordados problemas de controle associados a sub-sistemas do sistema global da célula .

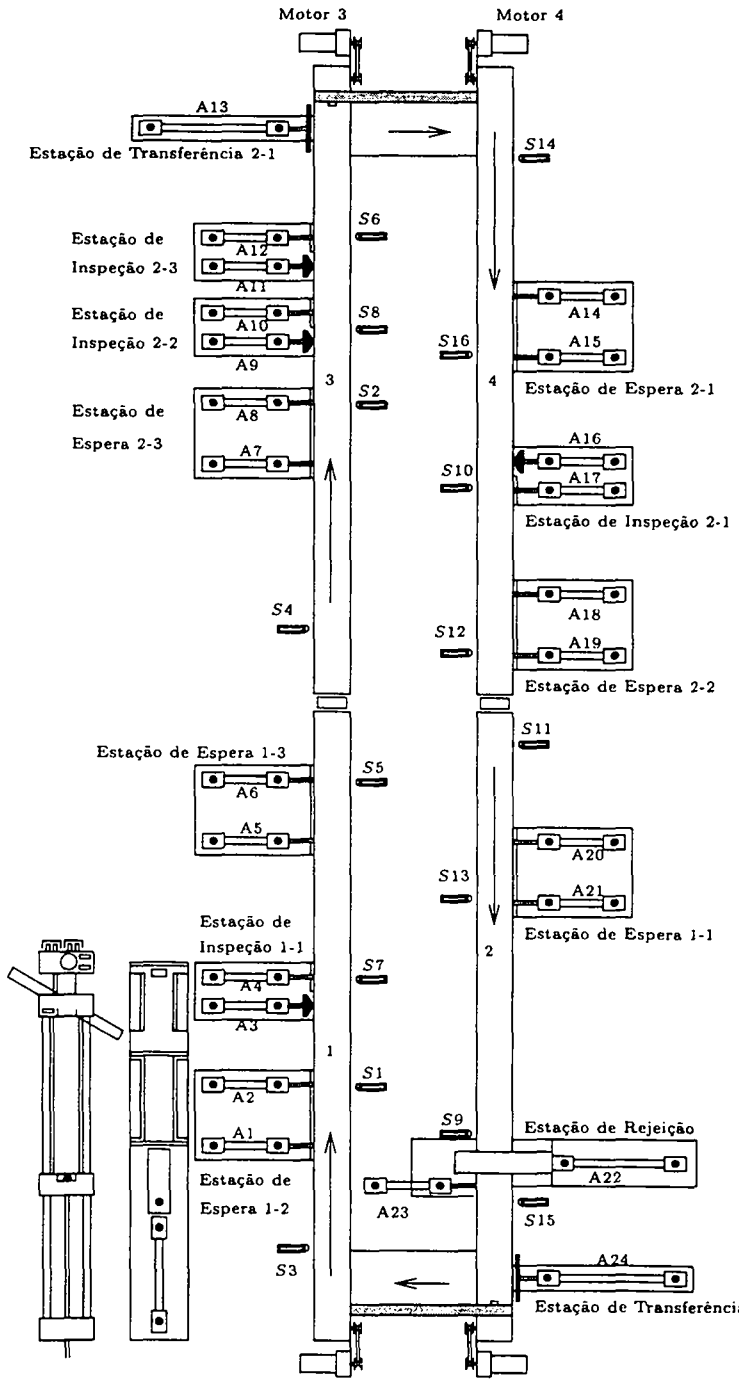


Figura 5.1: Célula Flexível de Manufatura 1 do L.A.I.

5.1 Estações de espera + buffer

A função da estação de espera é deter peças (se for necessário) enquanto alguma outra condição for satisfeita, por exemplo, o buffer estiver cheio.

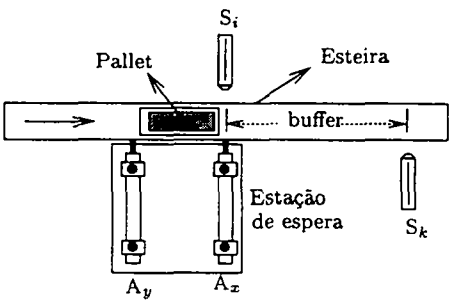


Figura 5.2: Sistema "Estação de Espera + Buffer"

A estação de espera, como mostrada na Figura 5.2 é dotada de dois atuadores pneumáticos A_x e A_y. S_i e S_k são sensores ópticos que detectam a passagem de *pallets*.

A seguir é descrito o comportamento livre da estação de espera.

O atuador A_x pode ser estendido ou recolhido livremente. Caso o atuador A_x esteja estendido, com a chegada de um *pallet* na estação de espera, evento identificado pela sensibilização do sensor S_i, o atuador A_y é acionado automaticamente pelo CLP [1] e só é recolhido quando o *pallet* sai da estação, evento identificado pela dessensibilização do sensor S_i. Esse procedimento automático garante que não existam dois *pallets* contíguos detectados como um só pelo sensor S_i. Caso dois *pallets* cheguem juntos à estação, se o primeiro parar na estação, o segundo será separado "à força" pelo atuador A_y.

O espaço existente entre os sensores S_i e S_k é modelado como um buffer.

Nesta seção a terminologia adotada para os comandos do nível físico é a seguinte:

$Trancar$ = estender o atuador A_x
$Liberar$ = recolher o atuador A_x
S_{ion} = sensibilização do sensor S_i quando um pallet obstrui a frente deste sensor.
S_{ioff} = dessensibilização do sensor S_i quando um pallet sai da frente deste sensor.
S_{koff} = dessensibilização do sensor S_k quando um pallet sai da frente deste sensor.

[1] O CLP é responsável pelo controle do atuador A_y

5.1.1 A modelagem utilizando "eventos forçáveis"

5.1.1.1 Comportamento livre do sistema

O gerador G_1 que representa o modelo do comportamento livre da estação de espera é mostrado na Figura 5.3. Note que no estado 2 do gerador G_1 o evento *Trancar* pode acontecer, levando o sistema a um estado indesejável. Fisicamente, a ocorrência deste evento, representa que o atuador A_x estende o embolo quando o pallet esteja passando em frente bloqueando-o contra a parede da esteira.

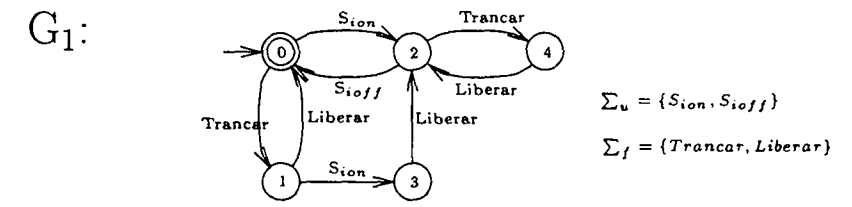


Figura 5.3: Modelo que representa o comportamento livre da "Estação de Espera"

O modelo do comportamento livre do *buffer* é descrito pelo gerador G_2 da Figura 5.4. A entrada de um pallet no buffer é identificada pela ocorrência do evento S_{ioff}, e a saída pela ocorrência do evento S_{koff}. No buffer, fisicamente só cabem dois pallets ao mesmo tempo.

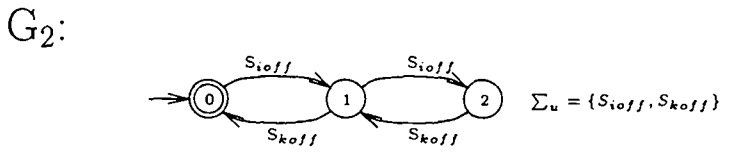


Figura 5.4: Modelo do comportamento do "Buffer"

O comportamento conjunto *buffer + estação de espera* (Planta) é dado pela composição síncrona dos dois modelos G_1 e G_2, e está representado pelo gerador G da Figura 5.5.

5.1.1.2 Especificação de segurança

Este tipo de especificação, como o nome diz, tem como objetivo evitar qualquer transição que danifique o sistema. No caso do exemplo, uma especificação deste tipo é:

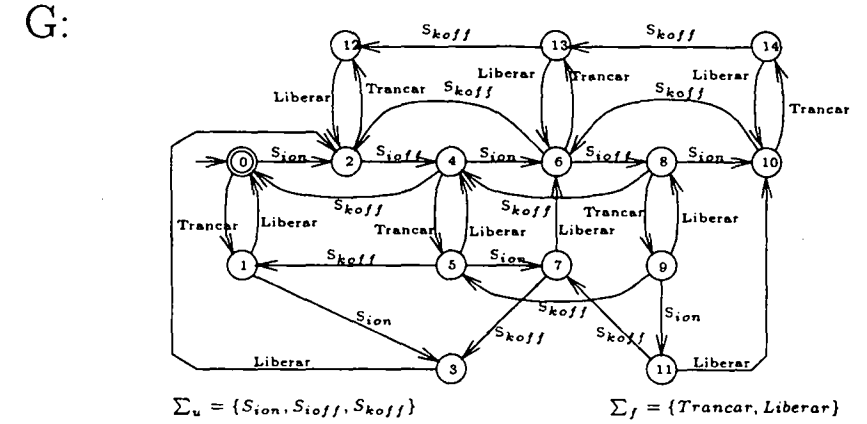


Figura 5.5: Comportamento do sistema "Estação de Espera + Buffer" (Planta)

- quando um pallet passa pela frente do sensor S_i, evitar que o atuador A_x possa ser estendido

O modelo que atende esta especificação é mostrado no gerador E_1 da Figura 5.6, que corresponde ao gerador G_1 com a supressão do estado "indesejável" 4.

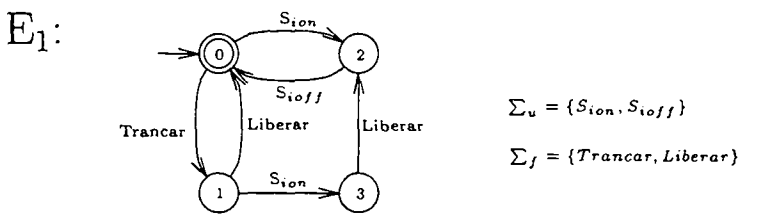


Figura 5.6: Especificação de segurança para a "Estação de Espera"

5.1.1.3 Especificação operacional

Este tipo de especificação corresponde ao que o usuário gostaria que fosse o funcionamento do sistema. Para o caso do exemplo assume-se que o buffer deva conter no máximo uma peça, ou seja, entre os sensores S_i e S_k deve existir no máximo uma peça. O modelo desta especificação é dado pelo gerador E_2 da Figura 5.7.

Então é obtido a especificação global como sendo, $E = E_1||_s E_2$

Com a planta G e a especificação E, é aplicado o algoritmo da máxima linguagem

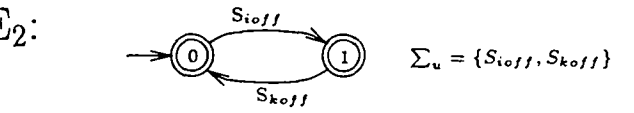


Figura 5.7: Especificação

controlável apresentado na seção 3.2.1.4 e obtém-se como solução o Gerador C descrito na Figura 5.8.

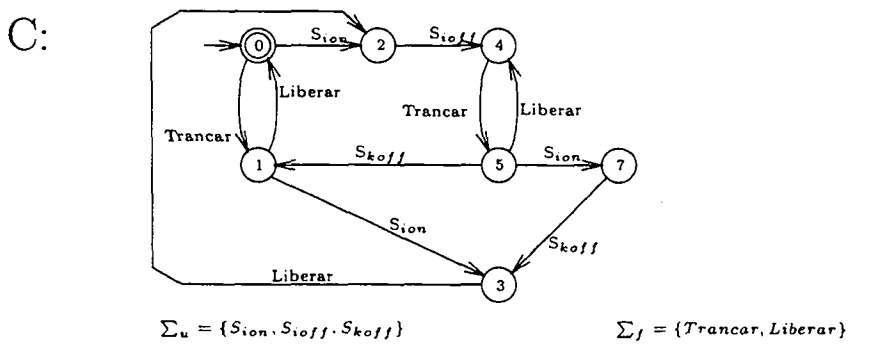


Figura 5.8: Estrutura do supervisor para o sistema "Estação de Espera + Buffer"

Observações:

- na estrutura do supervisor observe que para um estado que só tenha como saída eventos forçáveis, um comando para selecionar a ocorrência de um deles será obrigatório. Por exemplo, no *estado 4*, se o evento forçável *Trancar* não for executado, um evento não controlável S_{ion} poderá acontecer, levando o sistema para fora do comportamento desejado. No entanto para o estado que tenha como saída eventos forçáveis em conflito com eventos *controláveis/não controláveis*, a aplicação de um comando forçando a ocorrência do evento forçável será opcional (*ex: estados 0, 1, 5*), uma vez que o comportamento continua dentro da especificação desejada;

- observe que para os estados *0,1,4,5* o supervisor permite a ocorrência da seqüência *Trancar.Liberar.Trancar....* indefinidamente, a qual leva a um comportamento não desejado. Este pode ser evitado pela inclusão de mais uma especificação.

5.1.1.4 Especificação de coerência

Para obter um comportamento mais coerente, é necessário introduzir mais uma especificação que evite as seqüências descritas anteriormente.

A especificação está descrita pelo gerador $E3$ da Figura 5.9, e a nova máxima sublinguagem controlável obtida a partir desta nova situação é mostrada no gerador C' da Figura 5.10.

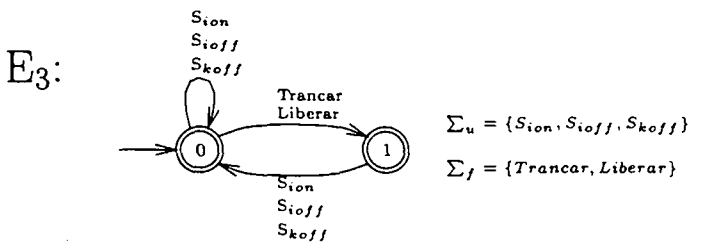


Figura 5.9: Especificação de Coerência

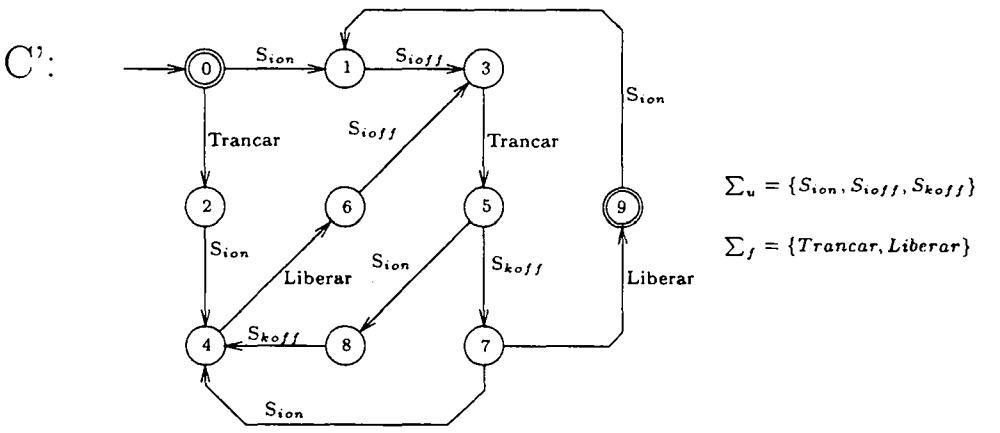


Figura 5.10: Nova estrutura do supervisor para o sistema "Estação de Espera + Buffer"

5.1.2 A modelagem sem usar eventos forçáveis

Relembrando a função do módulo intermediário MIC descrito na seção 4.1(pag. 39), para cada evento controlável vindo do supervisor, o MIC traduz a habilitação e desabilitação destes eventos em comandos físicos a serem enviados à planta.

A estação de espera é modelada por um estado e um evento controlável chamado "*passar*" como mostrado no gerador G_1 da Figura 5.11. Para a descrição do evento *passar* refira-se à tabela 5.1. Observe que a ocorrência deste evento está mapeada no evento S_{ioff}, enquanto que sua habilitação e desabilitação estão mapeadas nos comandos *Liberar* e *Trancar* respectivamente.

De fato, o evento *passar* interpreta-se como um pallet entrando no buffer e a condição que o atuador A_x esteja recolhido.

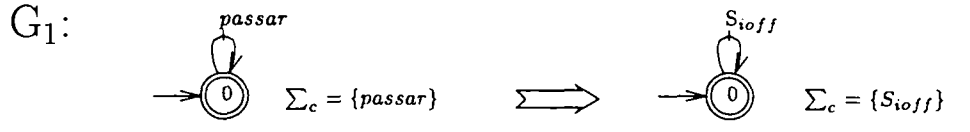


Figura 5.11: Modelo que representa o comportamento livre da "Estação de Espera"

O modelo do comportamento do *buffer* é descrito pelo gerador G_2 da Figura 5.12. Neste caso a ocorrência do evento não controlável *passou* é mapeado diretamente no comando da planta S_{koff}, como pode-se ver na tabela 5.1.

Neste caso o evento *passou*, significa que um pallet "termina" de sair do buffer, sem nenhuma condição adicional associada a este evento, devido a que este evento é não controlável.

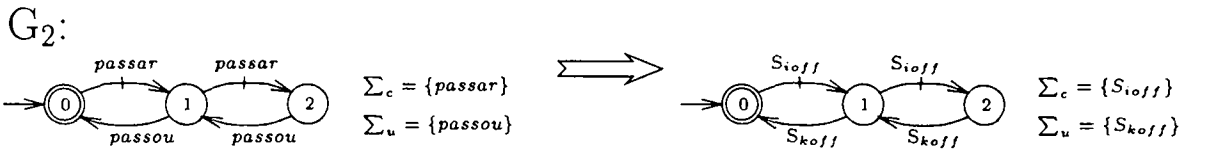


Figura 5.12: Modelo do comportamento do "Buffer" (Planta)

O mapeamento dos eventos usados na modelagem em comandos da planta é mostrado na tabela 5.1

A planta $G = G_1||_s G_2$, neste caso resulta ser a mesma do buffer ($G = G_2$).

Como no caso da seção 5.1.1 a especificação assumida para o buffer é que ele deva conter no máximo uma peça. Esta especificação está descrita no gerador E da Fig. 5.13.

evento \ característica	ocorrência	habilitação	desabilitação
passar	S_{ioff}	*Liberar*	*Trancar*
passou	S_{koff}		

Tabela 5.1: Mapeamento dos eventos controláveis

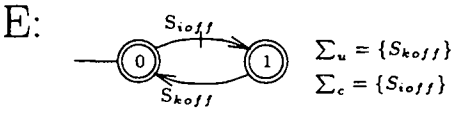


Figura 5.13: Especificação

Aplicando o algoritmo da máxima linguagem controlável, para o caso clássico (sem eventos forçáveis) o supervisor resulta ser aquele com a estrutura do gerador C da Figura 5.14.

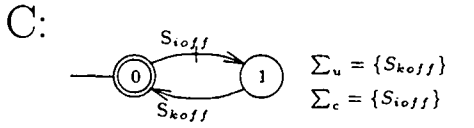


Figura 5.14: Estrutura do supervisor para a "Estação de Espera + Buffer"

Observações:

- note que para cada estado do supervisor existirá um conjunto de eventos relevantes habilitados e desabilitados que serão traduzidos em comandos do nível físico;
- para conhecer os eventos relevantes desabilitados é preciso conhecer o comportamento livre da planta. O conjunto de eventos não habilitados pelo supervisor será todo o alfabeto exceto os habilitados, mas nem todos os eventos não habilitados estarão presentes no estado correspondente da planta. Neste caso, os comandos de desabilitação se restringem aos eventos relevantes.

A tabela 5.2 mostra os eventos relevantes em cada estado do supervisor

Estado \ Evento	$Relev_{Habilitado}$	$Relev_{Deshabilitado}$
0	S_{ioff}	
1	S_{koff}	S_{ioff}

Tabela 5.2: Conjunto de eventos relevantes para cada estado do Supervisor

5.2 Estações de inspeção

A estação de inspeção tem a função de inspecionar pallets em forma seletiva, feita de forma automática quando o pallet é parado na estação.

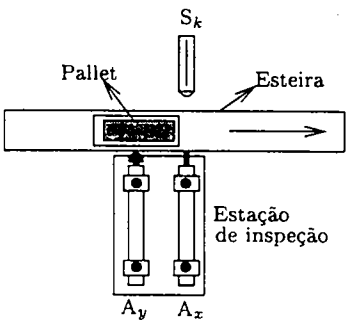


Figura 5.15: Sistema "Estação de Inspeção"

O Esquema de funcionamento da Estação de inspeção é semelhante ao da *Estação de Espera*. A estação de inspeção também está dotada de dois atuadores A_x e A_y, e um sensor S_k, que detecta a passagem de pallets.

Como no caso da estação de espera, o CLP comanda o acionamento do atuador A_y. Toda vez que A_x estiver estendido e chega uma peça à estação, o atuador A_y é estendido automaticamente, e é recolhido juntamente com o recolhimento do atuador A_x. Quando o atuador A_y pressiona o pallet contra a parede da esteira, imprime a identidade do pallet, o qual é captado por outro sensor (S_c) situado na parede da esteira.

Nesta seção a terminologia adotada para os comandos do nível físico é a seguinte:

$Trancar$ = estender o atuador A_x.
$Liberar$ = recolher o atuador A_x.
S_{kon} = sensibilização do sensor S_k quando um pallet obstrui a frente deste sensor .
S_{koff} = dessensibilização do sensor S_k quando um pallet sai da frente deste sensor.

5.2.1 A modelagem utilizando "eventos forçáveis"

5.2.1.1 Comportamento livre do sistema

O modelo que representa o comportamento livre da estação de inspeção está dado pelo gerador G da Figura 5.16. Quando um pallet está passando pela frente do sensor S_k (estado 1 de G), pode se dar o caso que o evento $Trancar$ aconteça, pressionando o pallet contra a parede da esteira, o que leva o sistema a um estado indesejável.

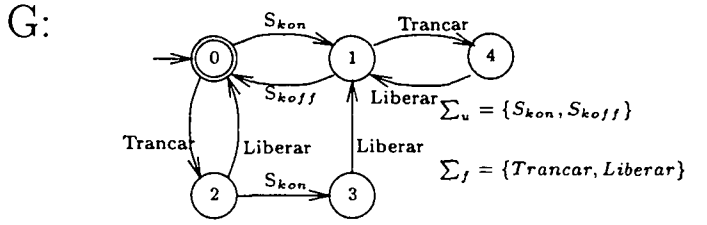


Figura 5.16: Modelo do comportamento livre da "Estação de Inspeção" (Planta)

5.2.1.2 Especificação de segurança

- evitar que o atuador A_x possa ser estendido, toda vez que um *pallet* passe pela frente do Sensor S_k

O modelo desta especificação de segurança está dado pelo gerador E_1 da Figura 5.17

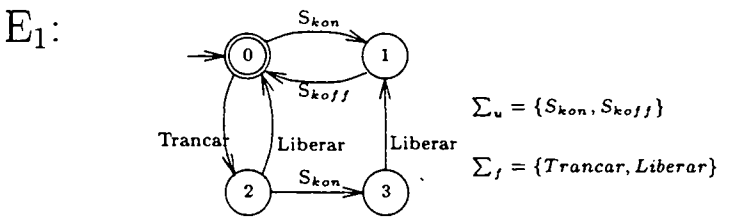


Figura 5.17: Especificação de segurança para a "Estação de Inspeção"

5.2.1.3 Especificação operacional

A especificação assumida para este caso é que a estação inspecione, toda vez que um *pallet* passe pela estação. Como a inspeção se dá após a seqüência de eventos $Trancar.S_{kon}$,

estado no qual o atuador A_y é estendido em forma automática, o modelo desta especificação está representado pelo gerador E_2 da Figura 5.18.

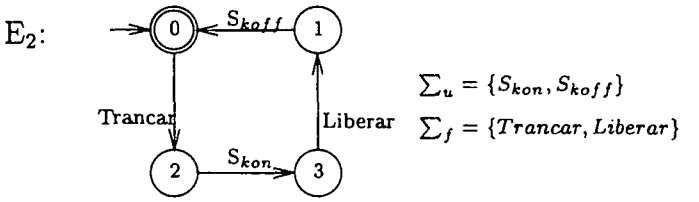


Figura 5.18: Especificação para a "Estação de Inspeção" (Supervisor)

Aplicando o algoritmo da máxima linguagem controlável para eventos forçáveis, o gerador que representa o supervisor, resulta ser igual ao gerador E_2 da especificação operacional (Fig 5.18).

A modelagem sem usar eventos forçáveis não foi realizada devido a que não foi possível encontrar, eventos que representem a espontaneidade da ocorrência e as condições de habilitação e desabilitação dos eventos controláveis.

5.3 Modelagem da estação de espera + estação de inspeção

Nesta seção é descrito o comportamento conjunto de uma estação de espera seguida de uma estação de inspeção com as mesmas funcionalidades descritas nas seções anteriores 5.1 e 5.2.

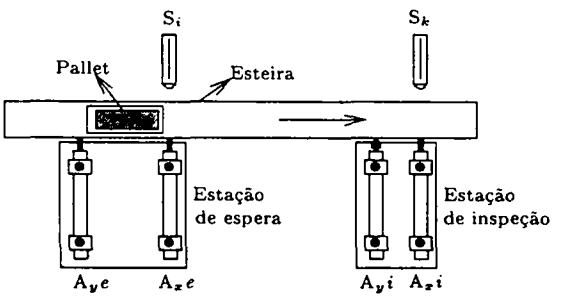


Figura 5.19: Sistema "Estação de Espera e Inspeção"

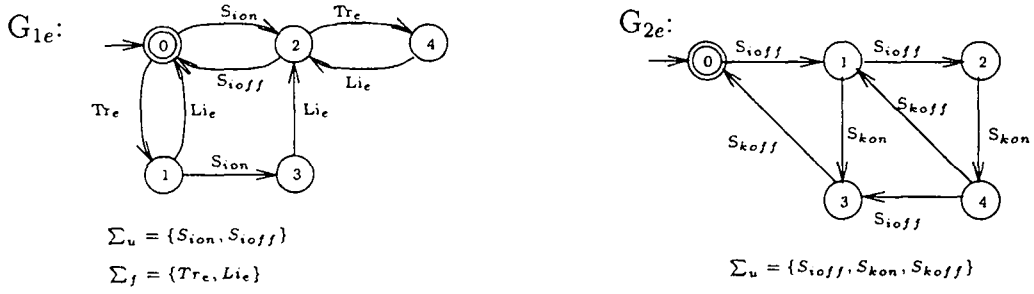
Observe que, na Figura 5.19, a estação de inspeção pode estar considerada dentro do buffer.

A notação adotada nesta seção é a seguinte:

$S_{ion}(S_{kon})$ = sensibilização do sensor $S_i(S_k)$ quando um pallet obstrui a frente deste sensor.
$S_{ioff}(S_{koff})$ = dessensibilização do sensor $S_i(S_k)$ quando um pallet sai da frente deste sensor.
Tr_e = estender o atuador A_x da estação de espera.
Li_e = recolher o atuador A_x da estação de espera.
Tr_i = estender o atuador A_x da estação de inspeção.
Li_i = recolher o atuador A_x da estação de inspeção.

5.3.1 Estação de espera *com* eventos forçáveis e estação de inspeção *com* eventos forçáveis

O modelo que representa o comportamento livre da estação de espera e o modelo do buffer estão dados pêlos geradores G_{1e} e G_{2e} da Figura 5.20 respectivamente.



Estação de espera Buffer

Figura 5.20: Modelo da "Estação de Espera e Buffer"

E o conjunto de especificações para este caso estão dados pêlos geradores E_{e1} e E_{2e} da Figura 5.21

O modelo que representa o comportamento livre da estação de inspeção está dado pelo gerador G_{1i} da Figura 5.22.

E o conjunto de especificações para este caso estão dados pêlos geradores E_{1i} e E_{2i} da Figura 5.23

Figura 5.21: Conjunto de especificações para o sistema "Estação de Espera + Buffer"

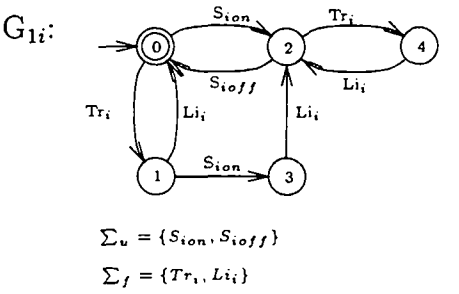


Figura 5.22: Modelo do comportamento livre da "Estação de Inspeção"

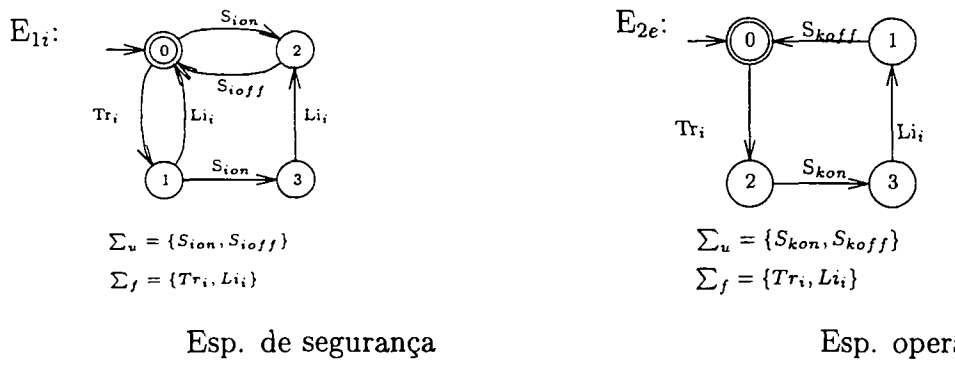


Figura 5.23: Conjunto de especificações para o sistema "Estação de Inspeção"

O comportamento livre do sistema em forma global, estará dada pela composição síncrona dos geradores G_{1e}, G_{2e} da *estação de espera + buffer* (Fig 5.20) e o gerador G_{1i} da *estação de inspeção* (Fig 5.22), isto é:

$$G = (G_{1e}||_s G_{2e})||_s G_{1i}$$

Como resultado desta composição síncrona resulta um gerador G com *75 estados* e *199 transições.*

A especificação conjunta também será o produto síncrono das especificações E_{1e}, E_{2e} da *estação de espera e buffer* (Fig 5.21) e das especificações E_{1i}, E_{2i} da *estação de inspeção* (Fig 5.23), isto e:

$$E = (E_{1e}||_s E_{2e})||_s (E_{1i}||_s E_{2i})$$

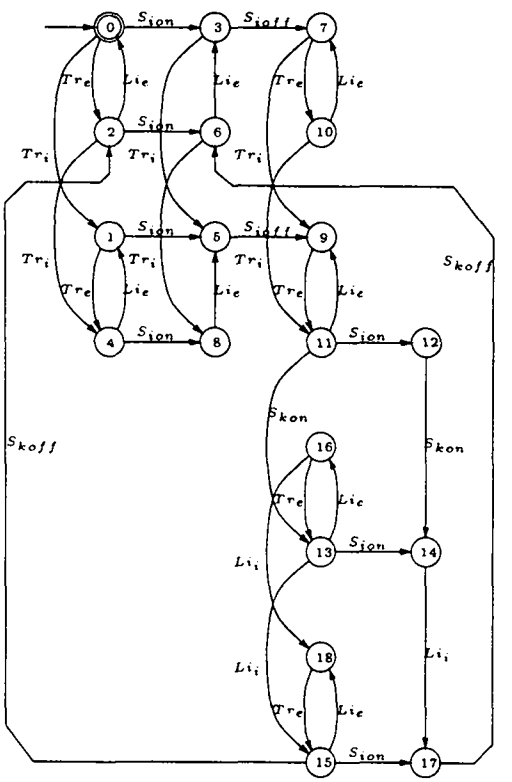
Esta especificação global resulta um gerador E com *24 estados* e *52 transições.*

Aplicando o algoritmo da síntese da máxima linguagem controlável entre a planta G e a especificação E é obtido o gerador C da Figura 5.24 que representa a estrutura do supervisor.

Como poderá ser visto também para este supervisor pode ocorrer a seqüência $Tr_e.Li_e.Tr_e\ldots$ que leva a um comportamento incoerente, sendo preciso introduzir uma especificação de coerência para evitá-lo.

Introduzindo a especificação se obtém a nova estrutura para o supervisor que está mostrado no gerador C' da Figura 5.25

C:

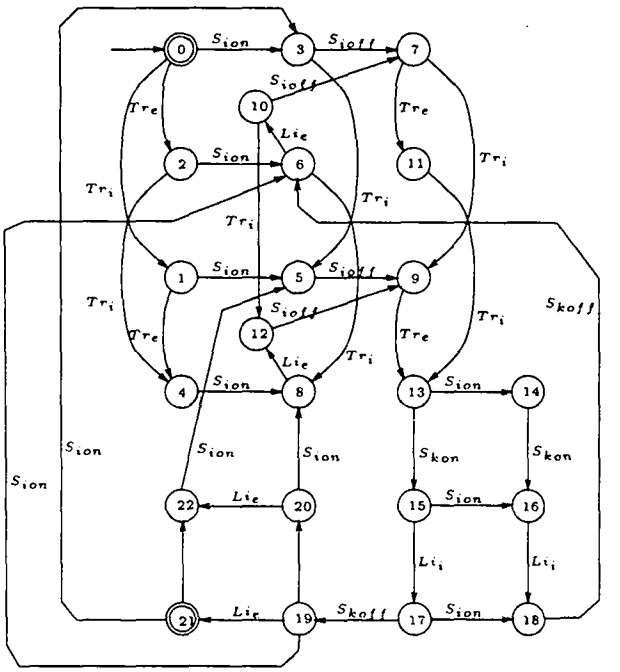


$$\sum_u = \{S_{ion}, S_{ioff}, S_{kon}, S_{koff}\}$$

$$\sum_f = \{Li_e, Tr_e, Li_i, Tr_i\}$$

Figura 5.24: Estrutura do supervisor para o sistema "Estação de Espera + Estação de Inspeção" usando eventos forçáveis

C':



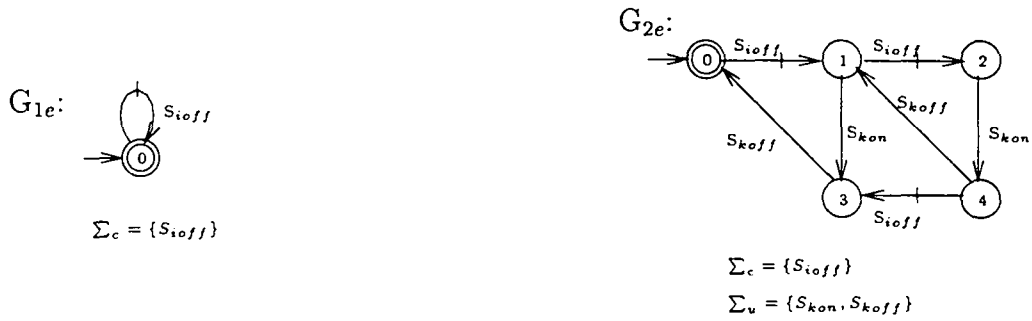
$\sum_u = \{S_{ion}, S_{ioff}, S_{kon}, S_{koff}\}$

$\sum_f = \{Li_e, Tr_e, Li_i, Tr_i\}$

Figura 5.25: Estrutura do supervisor para o sistema "Estação de Espera + Estação de Inspeção" usando eventos forçáveis e evitando a sequencia $Tr_e.Li_e.Tr_e \ldots$

5.3.2 Estação de espera *sem* eventos forçáveis e estação de inspeção *com* eventos forçáveis

O modelo que representa o comportamento livre da estação de espera e o modelo do buffer sem usar eventos forçáveis estão dados pêlos geradores G_{1e} e G_{2e} da Figura 5.26 respectivamente.



Estação de espera Buffer

Figura 5.26: Modelo da "Estação de Espera e Buffer"

E a especificação para este caso está dado pelo gerador E_e da Figura 5.27

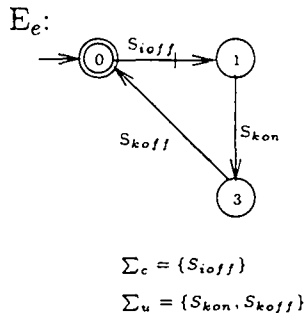


Figura 5.27: Especificação para o sistema "Estação de Espera + Buffer"

O modelo que representa o comportamento livre da estação de inspeção está dado pelo gerador G_{1i} da Figura 5.22. E o conjunto de especificações para este caso estão dados pêlos geradores E_{1i} e E_{2i} da Figura 5.23

O comportamento livre do sistema em forma global, estará dada pela composição síncrona dos geradores G_{1e}, G_{2e} da *estação de espera + buffer* (Fig 5.26) e o gerador G_{1i}

da *estação de inspeção* (Fig 5.22), isto é:

$$G = (G_{1e}||_s G_{2e})||_s G_{1i}$$

Como resultado desta composição síncrona resulta um gerador G com *15 estados* e *19 transições.*

A especificação conjunta também será o produto síncrono da especificação E_e da *estação de espera e buffer* (Fig 5.27) e das especificações E_{1i}, E_{2i} da *estação de inspeção* (Fig 5.23), isto e:

$$E = E_e||_s(E_{1i}||_s E_{2i})$$

Esta especificação global resulta um gerador E com *6 estados* e *7 transições.*

Aplicando o algoritmo da síntese da máxima linguagem controlável entre a planta G e a especificação E é obtido o gerador C da Figura 5.28 que representa a estrutura do supervisor.

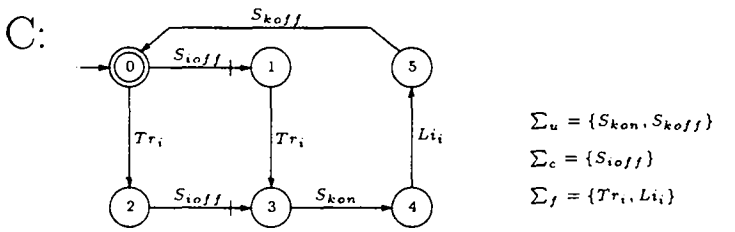


Figura 5.28: Estrutura do supervisor "Estação de Espera + Buffer" e " Estação de Inspeção"

Toda vez que na modelagem são usados eventos controláveis é preciso fazer o mapeamento correspondente da habilitação e desabilitação desses eventos em comandos do nível físico. O mapeamento dos eventos controláveis é mostrado na tabela 5.3

	S_{ioff}
Habilitado	Li_e
Desabilitado	Tr_e

Tabela 5.3: Mapeamento dos eventos controláveis

5.4 Estação de rejeição

A estação de rejeição tem a função de rejeitar pallets em forma seletiva, toda vez que um pallet é parado na estação.

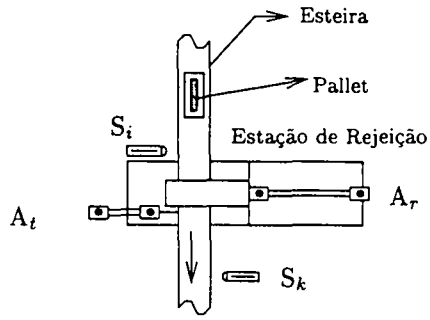


Figura 5.29: Sistema "Estação de Rejeição"

A Estação de Rejeição é dotada de dois atuadores A_t e A_r, e dois sensores que detectam a passagem de *pallets*, como pode ser visto na Figura 5.29. O atuador A_t se encarrega da detenção de *pallets* na estação e a tarefa do atuador A_r é a de rejeitá-los.

Enquanto não chega nenhum *pallet*, o atuador A_t pode estender e recolher livremente. Também um *pallet* pode passar livremente pela estação, sem que seja bloqueado ou rejeitado.

5.4.1 Modelagem utilizando "eventos forçáveis "

O gerador G da Figura 5.30 representa o comportamento livre da estação de rejeição.

5.4.1.1 Especificação operacional

A especificação operacional assumida para este caso é que a estação de rejeição rejeite *pallet's* alternadamente. Em termos de gerador, a especificação é mostrada no gerador E da Figura 5.31.

G:

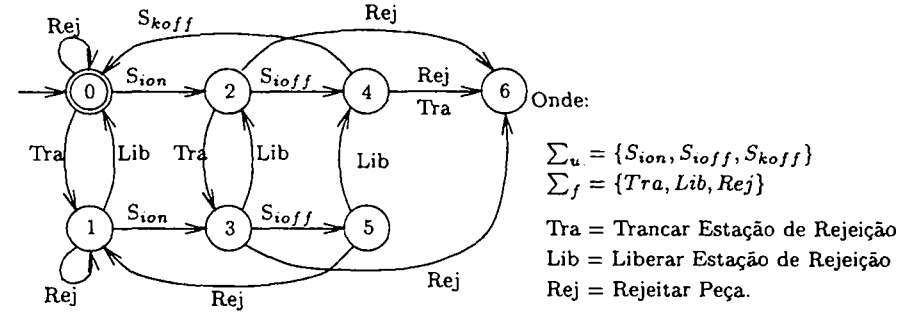


Figura 5.30: Comportamento livre da "Estação de Rejeição" (Planta)

E:

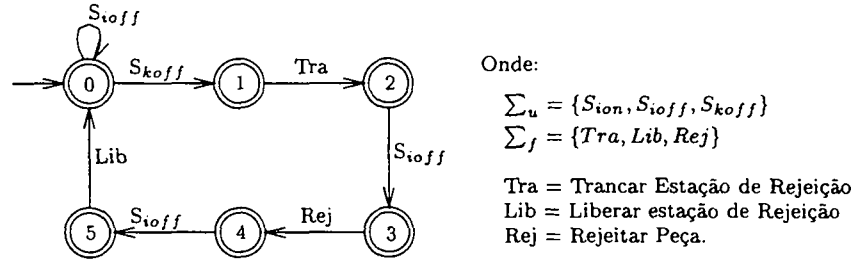


Figura 5.31: Especificação para a "Estação de Rejeição"

5.4.1.2 Especificação de segurança

- Um *pallet* só poderá ser rejeitado se ele antes tiver sido bloqueado pelo atuador A_t e tiver passado totalmente pelo sensor S_i.
- Se um *pallet* estiver passando pela frente do sensor S_i, o atuador A_r não pode ser acionado.
- Se um *pallet* estiver passando pela frente do sensor S_k, o atuadores A_r e A_t não podem ser acionados.

De fato pode se observar que esta especificação está contemplada na especificação operacional, uma vez que a mesma está expressa em temos de seqüência desejada, deixando fora os estados proibidos.

Aplicando o algoritmo da máxima linguagem controlável se obteve o gerador C da Figura 5.32, cuja estrutura representa o supervisor.

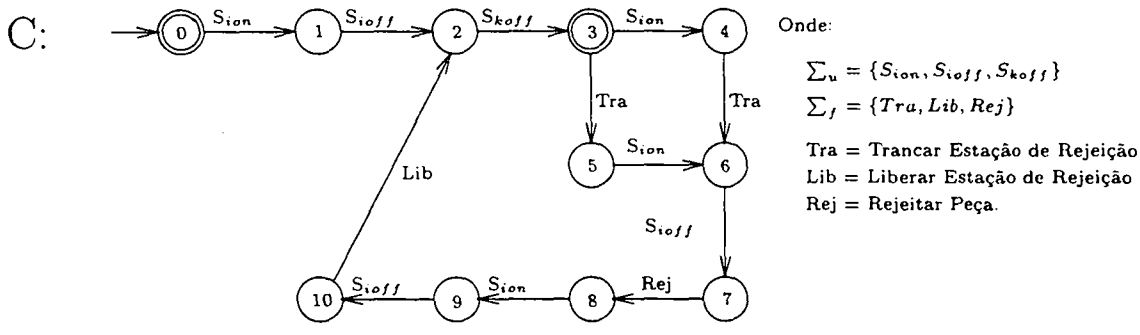


Figura 5.32: Estrutura do supervisor para a "Estação de Rejeição"

5.5 Estação de transferência

O comportamento da Estação de Transferência está a cargo do CLP. Ao chegar um *pallet* na Estação de transferência, a chave S_i é acionada e o embolo é estendido automaticamente, para garantir que o *pallet* chegue à outra esteira. No momento que se estende o êmbolo, o mesmo só será recolhido, também de forma automática, quando o *pallet* passe na frente do sensor S_k

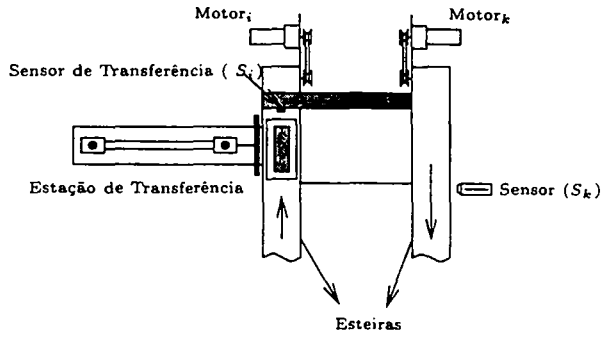


Figura 5.33: Sistema "Estação de Transferência"

5.5.1 Modelagem da estação de transferência

O comportamento livre para a Estação de Transferência é modelado pelo Gerador G da Figura 5.34

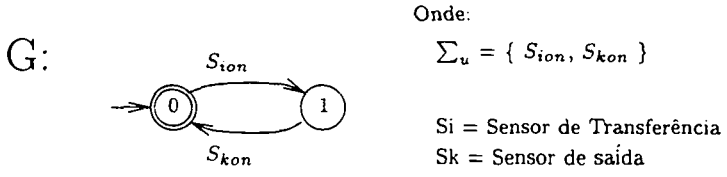


Figura 5.34: Comportamento livre da "Estação de Transferência" (Planta)

Note que quando o embolo é estendido, pode se dar o caso em que um novo pallet chegue à estação, o que provocaria um bloqueio total do sistema. Este problema não pode ser evitado pela consideração isolada da estação de transferência. Considera-se então o sistema que inclue a estação de Transferência e a Estação de Espera que a precede, como mostrado na Figura 5.35. Ou seja, modela-se somente a estação de espera e buffer, especificando que é permitido no máximo uma peça entre os sensores S_i e S_k (buffer).

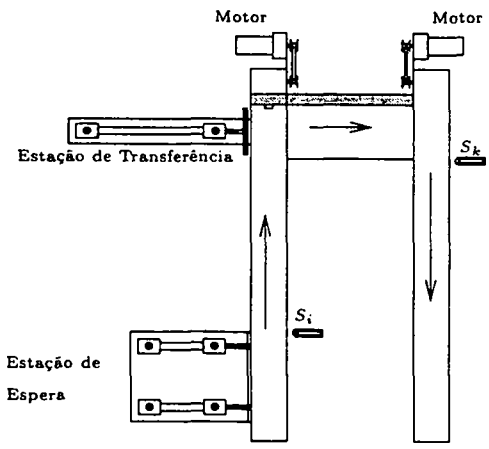


Figura 5.35: "Estação de Transferência" incluído dentro do buffer

5.6 Estação de espera *com* eventos forçáveis e estação de inspeção *com* eventos forçáveis (Abordagem Modular)

O objetivo desta subseção é calcular os supervisores parciais minimamente restritivos para a estação de espera e estação de inspeção e verificar se eles são modulares.

Dado um comportamento geral para ambas estações (produto síncrono entre os geradores da Figura 5.20 e o gerador da Figura 5.22) e calculando um supervisor para cada especificação, obteve-se que o produto síncrono dos supervisores independentes é o supervisor total descrito na seção 5.3.1

Neste caso, as linguagens em malha fechada das ações do supervisor da Estação de Espera sobre a Planta e do supervisor da Estação de Inspeção, verificou-se que são não bloqueantes, ou seja satisfazem as condições para síntese modular.

Sejam:
K_1 = Linguagem em malha fechada da ação do supervisor 1 sobre a Planta
K_2 = Linguagem em malha fechada da ação do Supervisor 2 sobre a Planta

Verificou-se que:

$$\overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2} \tag{5.1}$$

Então o supervisor resulta ser o produto síncrono dos supervisores parciais.

Observações:

- Não confundir como supervisor global resultado da composição modular, o produto síncrono dos supevisores sobre plantas localizadas, porque cada um deles atua sobre uma planta diferente.

5.7 Estação de espera sem eventos forçáveis e estação de inspeção com eventos forçáveis (Abordagem Modular)

Também para este caso foi possível aplicar técnicas de controle modular com sucesso, uma vez que os supervisores satisfazem as condições de modularidade.

5.8 Conclusão

As principais conclusões deste capítulo são as seguintes:

- a utilização de eventos forçáveis na modelagem de Sistemas a Eventos Discretos facilita a modelagem e a interpretação, porque o nível de abstração resultante é menor do que quando se usam somente eventos *controláveis/não-controláveis*;
- nem sempre é possível modelar um sistema sem usar eventos forçáveis, uma vez que a natureza de alguns eventos não o permite. Por exemplo, no caso da Estação de Rejeição, não é possível associar ao evento *Rejeitar* a noção de espontaneidade, que é a característica própria dos eventos *controláveis/não-controláveis*;
- se for possível fazer a modelagem sem usar eventos forçáveis, observa-se uma vantagem: este modelo em geral representa o mesmo comportamento com um menor

número de estados , uma vez que alguns comandos do nível físico estão já considerados na habilitação e desabilitação de eventos controláveis, os quais seriam eventos forçáveis na modelagem com eventos forçáveis. Portanto, a síntese da máxima linguagem controlável será menos complexa;

- é necessário destacar a diferença entre os tipos de especificação, desde que:

  especificações de segurança são obrigatórias;

  especificações de coerência são desejáveis;

  especificações operacionais representam uma estratégia particular requerida.

  Pode haver casos em que as especificações de segurança e coerência estejam incluídas implicitamente na especificação operacional.

Capítulo 6

Conclusão e Perspectivas

O presente trabalho teve como objetivo geral o desenvolvimento de ferramentas para modelagem e controle de sistemas a eventos discretos fundamentados no estudo desenvolvido em [RW87], [GR87] e [MC97].

Para tal partiu-se do estudo da teoria clássica de sistemas a eventos discretos proposto inicialmente por Ramadge-Wonhan [RW87] e de temas relacionados, como a teoria de autômatos e linguagens. Continuando, estudou-se uma extensão desta teoria apresentada por Golazewsky-Ramadge [GR87], referente à inclusão dos eventos forçáveis na modelagem. Com esta extensão se obtém maior facilidade e expressividade na modelagem de SED's. Finalmente e como complemento estudou-se a referência [MC97], que descreve a natureza dos eventos num sistema a eventos discretos e apresenta uma estrutura para implementação de controle supervisório em sistemas reais.

Durante a elaboração do trabalho, inicialmente foi necessário resolver o problema de síntese de leis de controle supervisório quando no modelo são introduzidos eventos forçáveis. Para isto foram desenvolvidos algoritmos para a síntese da máxima linguagem controlável. A etapa de implementação foi dividida em três partes, uma para modelagem (CONDES), outra para controle off-line(CONCEL), e a terceira para controle em tempo real (GERENCIADOR). O CONDES 3.0 *Com eventos forçáveis* é uma renovação do CONDES 2.0, o qual permite modelar sistemas a eventos discretos e sintetizar a lei de controle para o sistema em questão; o CONCEL permite a organização e o mapeamento dos eventos do nível físico com os eventos do nível supervisório; o GERENCIADOR ou controlador interage diretamente com o nível físico.

A implementação das ferramentas desenvolvidas neste trabalho foi feita na linguagem de programação C++, utilizando o ambiente C++ Builder versão 3.0 para windows 95/NT, da Borland.

As principais contribuições deste trabalho em ordem cronológica são:

- o desenvolvimento de um algoritmo para o cálculo da máxima sub-linguagem controlável quando eventos forçáveis são introduzidos na modelagem de SED's;
- a renovação do CONDES. Em relação ao CONDES 2.0, o CONDES 3.0 *com eventos forçáveis* permite a modelagem de SED's incluindo eventos forçáveis, além de apresentar algumas outras vantagens como por exemplo, a interface com o usuário mais intuitiva e de uso mais fácil;
- o desenvolvimento do CONCEL. Esta ferramenta está encarregada de pré-configurar o sistema de controle pelo mapeamento dos eventos do nível físico com os eventos do nível supervisório. Tal como a referência [MC97] apresenta, os eventos não controláveis do nível supervisório são mapeados em sinais de sensores do nível físico; os eventos forçáveis são mapeados em comandos a serem enviados ao nível físico; os eventos controláveis também são mapeados em sinais de sensores, mas a habilitação ou desabilitação são mapeadas em comandos a serem enviados ao nível físico. Esta base de dados servirá de entrada à próxima ferramenta implementada (GERENCIADOR);
- o desenvolvimento do GERENCIADOR. Esta ferramenta está baseada na arquitetura proposta por Martins e Cury [MC97]. O Gerenciador, ou controlador, interage diretamente com o nível físico em tempo real, e por meio da porta paralela do computador, envia e recebe eventos do nível físico.

Como aplicação estas ferramentas foram testadas na célula de manufatura do Laboratório de Automação Industrial (LAI) da Universidade Federal de Santa Catarina (UFSC). Em função as dificuldades observadas em relação ao esforço computacional, recomenda-se, sempre que for possível, fazer a modelagem sem usar eventos forçáveis, uma vez que esta modelagem possibilita um modelo com um número menor de estados.

Quanto a perspectivas para a continuidade deste trabalho, podemos indicar:

- explorar técnicas que permitam diminuir a explosão de estados na síntese do supervisor minimamente restritivo, ou seja:

* técnicas de modularidade que permitam o cálculo de supervisores parciais, levando em conta que os supervisores atuam sobre plantas localizadas;
* agregação de estados, que consiste em tirar proveito da estrutura de um sistema procedendo a uma redução no número de estados do sistema pela agregação destes;
* representação de SED's através de representações simetricamente reduzidas.

- trabalhar na parte de hardware e software para implementação de supervisores localizados (programação distribuída).

Referências Bibliográficas

[BW94] B.A. Brandin,W.M. Wonham. Supervisory control of timed discrete-event system. *IEEE Trans. on Automatic Control*, 39(2):329–342, February 1994.

[CDFV88] R. Cieslak, C. Desclaux, A.S. Fawaz,P. Varaiya. Supervisory control of discrete event systems with partial observations. *IEEE Transactions on Automatic Control*, 33(3):249–260, 1988.

[CL89] J. Carroll,D. Long. *Theory of Finite Automata.* Prentice-Hall International Editions, 1989.

[CW96] P.E. Caines,Y-J Wei. Hierarchical hybrid control system. *Proceeding of the Block Island Workshop*, pp 39–48, 1996.

[FE93] A. L. V. Forbellone,H. F. Eberspächer. *Lógica de Programação - A Construção de Algoritmos e Estruturas de Dados.* Makron Books, 1993.

[GR87] C.H. Golaszewski,P.J. Ramadge*. Control of discrete event processes with forced events. *IEEE Proceedings of the 26th Conference on Decision and Control*, pp 247–252, August 1987.

[GR88] C.H. Golaszewski,P.J. Ramadge. *Supervisory Control of Discrete Event Systems with Arbitrary Controls.* M.J. Laub, Eds. Springer, 1988.

[HU79] J.E. Hopcroft,J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley USA, 1979.

[KG95] R. Kumar,V. Garg. *Modeling and Control of Logical Discrete Event Systems.* Kluwer Academic Publishers, 1995.

[LW88] F. Lin,W.M. Wonham. On observability of discrete-event systems". *Information Sciences*, 44(3):173–198, 1988.

[MC97] E.D. Martins,J.R. Cury. Uma arquitetura para implementação de controle supervisório de sistemas a eventos discretos. *Terceiro Simpósio Brasileiro de Automação Inteligente*, pp 184–189, Setembro 1997.

[O'Y92] S.D. O'Young. On the synthesis of supervisors for timed discrete event processes. *IEEE Trans. on Automatic Control*, July 1992.

[RW87] P.J. Ramadge,W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25(1):206–230, 1987.

[RW89] P.J. Ramadge,W.M. Wonham. The control of discrete event systems. *Proceeding of the IEEE*, 77(1):81–98, January 1989.

[Tis91] J.G. Thistle. *Control of Infinite Behavior of Discrete Event Systems.* PhD thesis, University of Toronto, Toronto - Canada, January 1991.

[TW94] J.G. Thistle,W.M. Wonham. Supervision of infinite behavior of discrete event systems. *SIAM J. Control and Opt.*, 32(4):1098–1113, July 1994.

[Won98] W.M. Wonham. *Notes on Control of Discrete-Event Systems. Course Notes for ECE 1636F/1637S.* Revision 98.09.01, 1998.

[WR87] W.M. Wonham,P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25(3), May 1987.

[Zil93] R.M. Ziller. A abordagem ramadge-wonham no controle de sistemas a eventos discretos: Contribuições à teoria. Dissertação de Mestrado, Pós-Graduação em Engenharia Elétrica - Universidade Federal de Santa Catarina, Fpolis - SC, Outubro 1993.

Apêndice A

Manual do CONDES 3.0 "com eventos forçáveis"

CONDES (CONtrol of Discret Event Systems)

Versão: 3.0 whith Forced Events
Manual do Usuário
Novembro 1998
Por César R. Claure Torrico

Resumo:

Este manual oferece uma breve introdução à funcionalidade do CONDES. Supõe-se que o leitor está familiarizado com os conceitos e a teoria dos Sistemas a Eventos Discretos (DES) no contexto de Ramadge-Wonham [Won98]. Se o leitor quiser usar as funções programadas do software, deve adicionalmente ler o capítulos 2 e 3 desta dissertação.

A.1 Introdução geral

CONDES "com eventos forçados", é uma ferramenta que está estendendo a funcionalidade do CONDES versão 2.0.

Inicialmente o CONDES 2.0 foi desenvolvido no Laboratório de Automação Industrial do Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina,

sob a orientação do Prof. Roberto M. Ziller, com o objetivo de modelar sistemas a eventos discretos. A principal contribuição do CONDES 3.0 é a introdução da noção de eventos forçáveis na modelagem, além de uma interface gráfica mais amigável.

Esta extensão tem como base a teoria padrão de Ramadge-Wonham . Para compreender o fundamento teórico das funções fornecidas do CONDES "com eventos forçados", você deve ter lido [Won98]e [GR87].

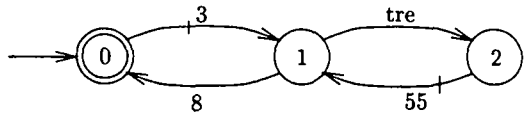
A.1.1 Características gerais do CONDES "com eventos forçados":

- Permite criação e edição de arquivos do tipo DES
- Disponível para Win95/NT (32 bit)
- Foi desenvolvido em Borland C++Builder 3 versão 3.0

A.2 Conceitos

Todo Sistema a Eventos Discretos (DES) é modelado como um grafo de estados e as transições etiquetadas com eventos apropriados. Todos os estados são codificados como um número inteiro positivo e os eventos são codificados, seja como um inteiro positivo ou um string de letras com 3 caracteres (veja o exemplo em Fig. A.1). A etiqueta máxima dos estados é atualmente "999". As etiquetas ímpares do evento indicam um evento controlável, etiquetas pares indicam eventos não controláveis e etiquetas com letras indicam eventos forçáveis.

Cada procedimento no CONDES faz um exame de alguma entrada do usuário a partir de um arquivo de dados ou uma janela gráfica e produz uma janela gráfica na resposta.



$\sum_c = \{3, 55\}$
$\sum_u = \{8\}$
$\sum_f = \{tre\}$

Figura A.1: Um exemplo de Grafo .DES

A.3 Menu principal do CONDES

O CONDES apresenta um menu principal dividida em *sub-menus*, com a finalidade de ajustar algumas variáveis globais para os procedimentos do CONDES.

A.3.1 *Menu* File

Ativa um *sub-menu* que permite a manipulação de arquivos .DES, com funções de criar, abrir, salvar, entre outras. Para imprimir um arquivo .DES primeiramente é necessário salvar como arquivo texto (File → Save As .txt) para depois ser recuperado por qualquer editor de texto.

Leve em conta que os itens do menu *File* atuam sobre a janela de trabalho ativa.

A.3.2 *Menu* Operations

Neste menu são implementadas as funções DES e que em geral requerem como argumentos dois parâmetros de entrada, como por exemplo: Interseção, Produto Síncrono, Máxima Linguagem Controlável.

A.3.2.1 *Item* Synchronous Product

Descrição formal:

As linguagem de $G = SynchronousProduct(G_1, G_2)$ em termos da projeção natural inversa é:

$$L(G) = P_1^{-1}L(G_1) \bigcap P_2^{-1}L(G_2)$$

$$L_m(G) = P_1^{-1}L_m(G_1) \bigcap P_2^{-1}L_m(G_2)$$

Uso

A sincronização pede dois arquivos .DES como argumentos do procedimento. A janela resultante incorpora o comportamento independente de ambos sistemas originais exceto

onde sincroniza eventos em comuns. O resultado do produto de dois DES com alfabetos disjuntos é justo o comportamento do produto assíncrono de ambos.

A.3.2.2 *Item* Intersection

Descrição formal:

As linguagem de $G = Intersection(G_1, G_2)$ é um caso particular de produto síncrono quando os alfabetos são iguais ($\sum = \sum_1 = \sum_2$). A descrição se reduz a:.

$$L(G) = L(G_1) \bigcap L(G_2)$$

$$L_m(G) = L_m(G_1) \bigcap L_m(G_2)$$

Uso

A interseção tem igual forma de uso que o produto síncrono.

A.3.2.3 *Item* SupCon

Permite calcular a máxima sub-linguagem controlável contida numa especificação dada, em relação a uma planta.

Descrição formal:

- G = Representa a Planta .DES
- E = Representa a Especificação .DES

A linguagem de $KDES = SupCon(G, E)$ é:

$$Lm(KDES) = K = Sup\mathcal{C}(L_m(G) \bigcap L_m(E)$$

$$L(KDES) = \overline{K}$$

Uso

SUPCON pede o .DES da planta e o .DES da especificação como argumentos. Admitindo que a especificação é $L_m - fechada$, o resultado é uma representação *Trim* de uma solução mínima restritiva que esteja obedecendo os requerimentos da especificação e seja não bloqueante. O resultado é ótimo no sentido que este restringe o comportamento original da planta em forma mínima sem violar a especificação.

A.3.3 *Menu* Window

Este item, como o de qualquer outro software para windows, permite a organização das janelas de trabalho em Cascata, Mosaico, etc.

A.4 Descrição da janela de trabalho .DES

Quando você cria uma nova janela de trabalho (File → New), será aberta na tela uma janela como a da Figura A.2. Esta janela permite a criação de um modelo de Sistemas a Eventos Discretos, expresso na forma de um gerador..

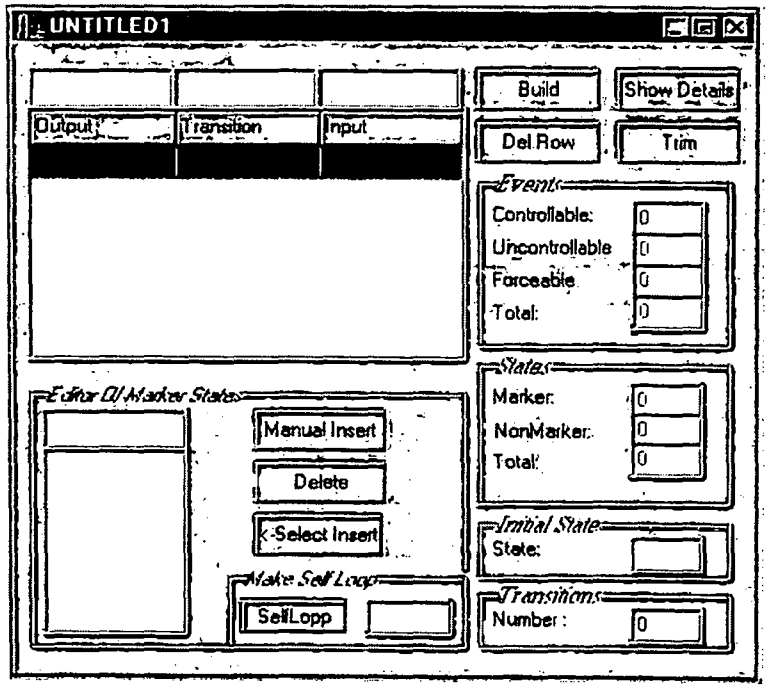


Figura A.2: janela de trabalho

A.4.1 Construção da estrutura

Uma vez preenchidas as três caixas de edição no painel principal, correspondente ao estado de saída, transição, e estado de entrada, clicar no botão *Build* ou pulsar a tecla *Enter* até completar com a estrutura. Para apagar uma transição basta clicar nela e seguidamente clicar no botão *Del Row* ou pulsar a tecla *Delete.*

A.4.2 Estado inicial

O Estado Inicial do autômato, por "Default", será o primeiro estado a ser construído. No entanto o usuário tem a possibilidade de modificar esta opção fazendo um click na caixa *Initial State* e então introduzir manualmente o novo estado inicial desejado.

A.4.3 Verificação

Se vôce deseja conferir a lista de estados e transições, pulsar a tecla *Show Details*,(ver Fig A.3) que lhe mostrará todos os detalhes referentes a esse sistema num painel *States* referindo-se à lista de estados e. Se esta janela é o resultado de algum produto síncrono ou interseção também é mostrada a correspondência de estados dos autômatos primitivos com cada estado do autômato atual [1]. Outro painel, *events* , mostra a lista de eventos forçáveis, controláveis e não controláveis usados no sistema .

A.4.4 Marcação de estados

A marcação dos estados pode se fazer de duas formas.

Manual.- Para marcar estados na forma manual, basta introduzir o número do estado na caixa de edição do painel *Editor of Marker States* e clicar no botão *Manual Insert* ou pulsar a tecla *Enter*

Selecionando.- Para marcar os estados desta forma, primeiro selecionar a lista de estados a marcar no painel *States* e em seguida clicar no botão ← *Insert Selected*

[1] Qualquer alteração na janela resultante do produto síncrono ou interseção, apaga a correspondência de estados com os estados primitivos

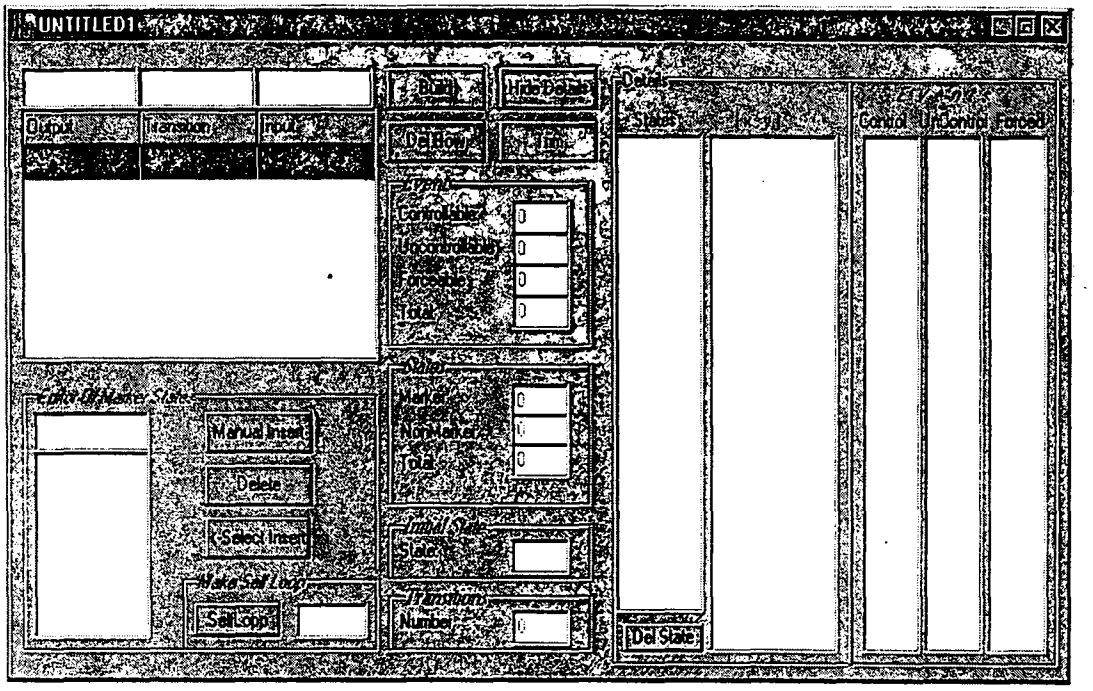


Figura A.3: janela de trabalho com detalhes

Para desmarcar um estado tem-se que selecionar os estados na lista de estados Marcados e clicar no botão *Delete* e/ou Pulsar a tecla *Delete*

A.4.5 Apagar estados

No Painel *States* Selecionar os estados a apagar e clicar no botão *Del State*, que esta situado na parte inferior do painel.

Este botão apaga o estado e as transições associadas a ele.

A.4.6 Self Loop

Na caixa de edição do painel *Make Self Loop* colocar o nome da Transição e pulsar o botão *Self Lopp*. Este comando constroe o "self Loop" com a transição selecionada para cada estado do autômato. Se a transição já estiver presente, uma mensagem de erro é mostrada.

A.4.7 Trim

O Botão *Trim* devolve um autômato correspondendo à componente Trim (Acessível e Co-acessível) do autômato original.

Apêndice B

Manual do CONCEL (CONDES-CELULA)

Versão: 1.0
Manual do Usuário
Novembro 1998
Por César R. Claure Torrico

B.1 Introdução geral

CONCEL, é uma ferramenta de suporte que ajuda a preparar um arquivo de dados para o controle da Célula de Manufatura 1, do Laboratório de Automação Industrial (LAI) da Universidade Federal de Santa Catarina (UFSC). Esta ferramenta usa os arquivos .DES gerados pelo *CONDES Whit Forced Events* e, a lista de comandos de atuadores e sinal de sensores da Célula.

Nada impede em usar esta ferramenta para outros sistemas de manufatura que não seja a célula, desde que a numeração dos comandos de entrada ou saída sejam adaptados para o sistema em questão.

B.2 Menu principal do CONCEL

É constituído de um menu principal cuja a finalidade ajustar algumas variáveis globais para os procedimentos do CONCEL.

B.2.1 *Menu* file.

Este *sub-menu*, permite a manipulação de três tipos de arquivos, por meio de janelas individuais

- Arquivos de comando .CDM.- Contem uma lista de comandos da Célula referente a atuadores e outra referente aos sinais dos sensores. Um arquivo deste tipo será mostrado numa janela como da Figura B.1.

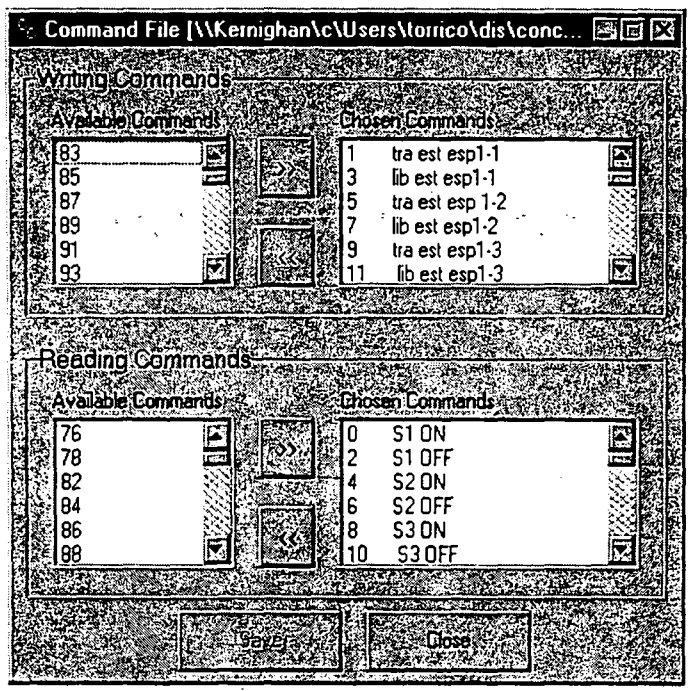


Figura B.1: Lista de Comandos da Célula

- Arquivos do Supervisor .SUP.- Este Arquivo contem a Estrutura do Supervisor.DES e a ligação com o comportamento da Planta .DES. Este tipo de arquivo é mostrado numa janela como da Figura B.2

- Arquivos de Ligação .CLN.- São arquivos de ligação, que ligam arquivos .CMD e .SUP. A interface é mostrada numa janela como da Figura B.3

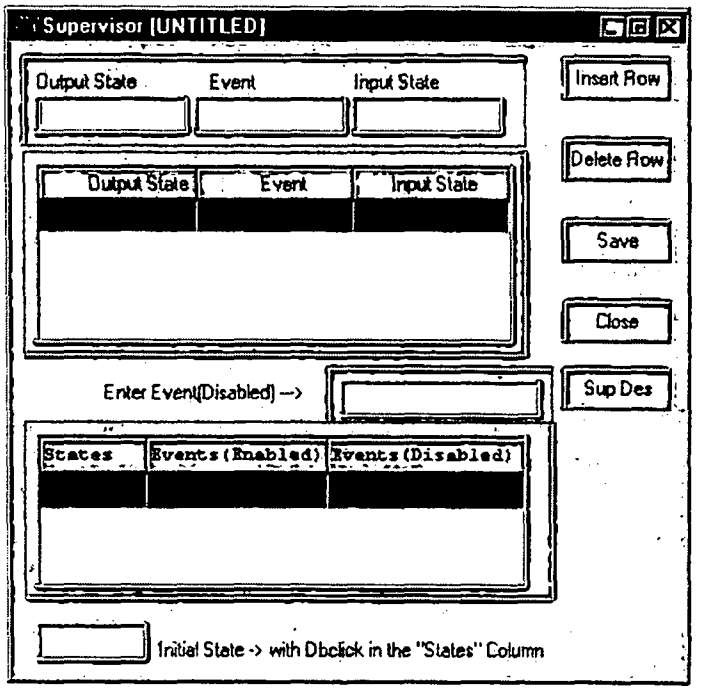


Figura B.2: Supervisor

B.2.2 Criação de arquivos de comando .CMD

Para começar, abre-se a janela de comandos (Fig B.1) com *New→File→Command File*. Observe que esta é dividida em 2 paneis. O primeiro (*Writing Commands*) que indica todos os comandos de escrita ou comandos relacionados com cada atuador da Célula. O Segundo painel *Reading Commands* são os valores dos sinais de leitura dos Sensores da Célula. Estão disponíveis 128 comandos para escrita 128 valores dos sinais de leitura.

Caso o usuário deseje controlar uma parte da Célula, ele tem a opção de escolher os comandos necessários para o controle. Para separar os comandos requeridos, basta selecionar com o Mouse na caixa de comandos disponíveis na esquerda do painel e clicar no botão >>. O usuário pode desistir de alguns comandos e voltar com o botão << para comandos disponíveis. Esse procedimento serve tanto para os comandos de escrita como leitura.

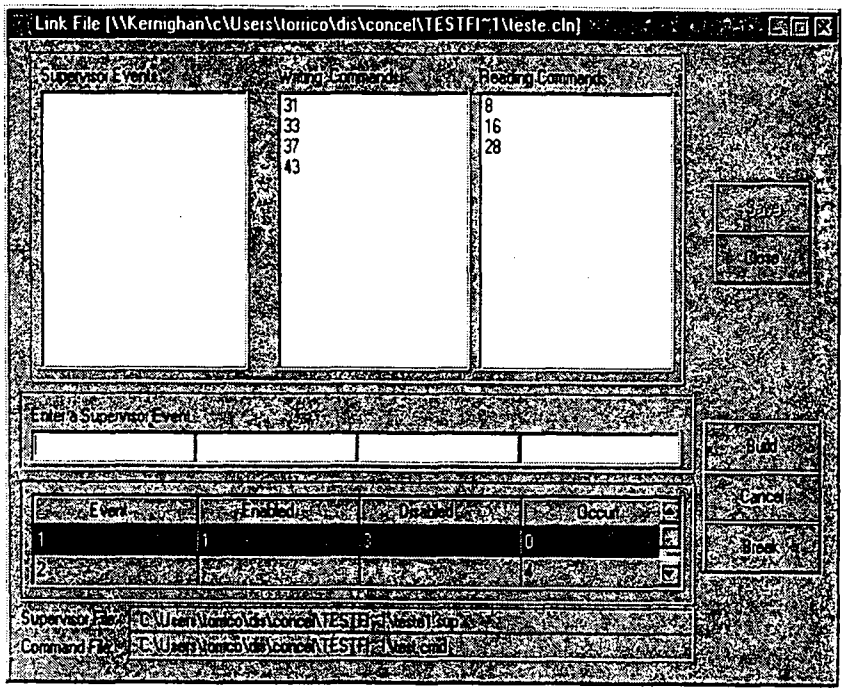


Figura B.3: Janela de ligação

B.2.2.1 Adicionando etiquetas a os comandos

Para facilitar a interpretação dos comandos é possível adicionar a eles uma etiqueta indicando o nome ou função desse comando.

Para adicionar, na caixa da direita (*Chosen Commands*) do painel *Writing Commands ou Reading Commands*, basta clicar duas vezes no comando desejado, que lhe mostrara um pequeno editor onde você introduz a etiqueta requerida.

B.2.3 Criação de arquivos do supervisor .SUP

Como primeiro passo, abre-se a janela do supervisor (*File→New→Supervisor File*). Ter-se-á uma janela como a da Figura B.2. Veja que ela esta formada por uma Grade na parte superior, referente à estrutura do supervisor e outra Grade na parte Inferior contendo a lista de estados com o respectivo conjunto de eventos relevantes habilitados e desabilitados[1].

[1]Por Default todos os eventos habilitados pelo supervisor são relevantes. Os eventos relevantes desabilitados num determinado estado são aqueles desabilitados pelo supervisor e que estejam no comportamento livre da Planta

Para criar Arquivos do Supervisor você tem duas opções:

1. **Manual.-** Uma vez preenchidas as três caixas de edição no painel principal, correspondente ao estado de saída, transição, e estado de entrada, clicar no botão *Insert Row* até completar com a estrutura. Para apagar uma transição basta clicar nela e seguidamente clicar no botão *Delete Row.*

   O Estado Inicial do autômato, por *Default* será o primeiro estado a ser construído, no entanto o usuário tem a possibilidade de modificar fazendo duplo "click" no estado, na lista de estados da Grade inferior.

   Para introduzir os eventos relevantes desabilitados num determinado estado da estrutura, clicar no estado requerido na Grade inferior e em seguida na caixa de edição etiquetada por *Enter Events (Disabled)* digitar a lista de eventos, separados por espaço, e finalmente pulsar a tecla , *Enter*

2. **Carregando arquivos .DES.-**

   Uma vez tendo os arquivos .DES do Supervisor e da Planta gerados pelo *CONDES With Forced Events* é muito simples: Clicar no botão *Sup Des* e introduzir o arquivo do Supervisor .DES e em seguida clicar no botão *Plant Des* e introduzir o arquivo da Planta .DES .

B.2.4 Criação de arquivos de ligação .CLN

Quando você executar a opção *New Link File* dentro do menu *File*, lhe será solicitado um arquivo de comando .CMD e um arquivo de Supervisor .SUP. O resultado é mostrado numa janela como a da Figura B.3.

Para construir o mapeamento a partir deste ponto, lhe será mostrada uma ajuda *On-Line* de Cor Azul que lhe indicará os passos que deve seguir. A escolha dos Eventos do Supervisor a construir é com duplo "click" no evento na lista *Supervisor Events.* De igual modo é feito para os comandos em *Writing Commands* e *Reading Commands*

Lembre que segundo a proposta de [MC97], a habilitação e desabilitação dos eventos controláveis, são traduzidos em comandos do nível físico (*Writing Commands*) e já a ocorrência do evento é mapeado na ocorrência de um sensor (*Reading Commands*). Os eventos não controláveis são só mapeados na ocorrência do evento com um sen-

sor(*Reading Commands*). Os eventos forçáveis são mapeados diretamente na ação de um atuador(*Writing Commands*).

Para destruir o mapeamento de um evento, seleciona-se o evento com o mouse na Grade inferior e clica-se no botão *Break*

Apêndice C

Manual do GERENCIADOR

GERENCIADOR

Versão: 1.0
Manual do Usuário
Novembro 1998
Por César R. Claure Torrico

C.1 Introdução

O "GERENCIADOR" é um aplicativo que interage diretamente em tempo real com sistemas de manufatura, através da porta paralela do computador. O objetivo deste aplicativo é controlar o comportamento de um processo de manufatura, usando a teoria de Sistemas a Eventos Discretos.

O fundamento teórico para o desenvolvimento deste aplicativo está no capítulo 4 desta dissertação.

C.2 Funcionamento

A janela da INTERFACE "CONDES-CÉLULA" é mostrada na figura C.1

Para começar com a interação, primeiro carrega-se o arquivo .CLN preparado no

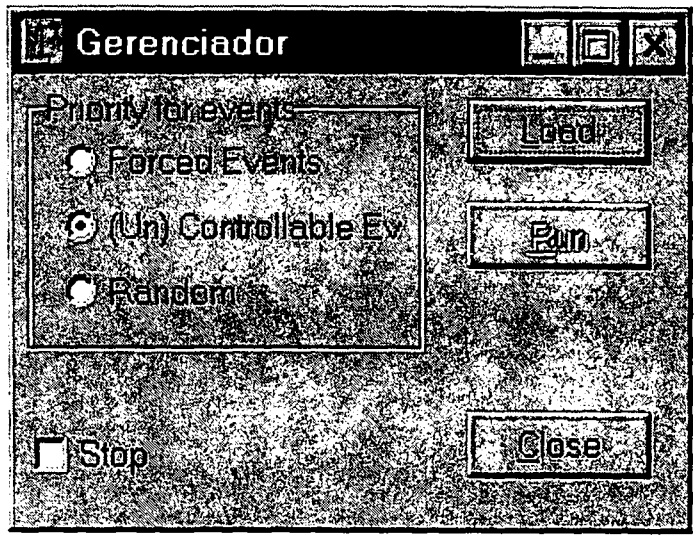


Figura C.1: "Gerenciador"

CONCEL , clicando no botão *Load*. Leve em conta que este arquivo tem como dado o caminho do arquivo .SUP do supervisor e .CMD da lista de comandos.

C.2.1 Descrição dos raio-botões

Num determinado estado, para um conjunto de eventos habilitados pelo supervisor (entrada de controle), poderia se dar a situação em que, um evento *forçável* esteja em conflito com eventos *controláveis/não controláveis* . Nesta situação o usuário pode fornecer um critério de prioridade, clicando na opção desejada. Para maior detalhe em relação ao critério de prioridade, refira-se à seção 4.2.2.1 da pag. 45

C.3 Informações técnicas

C.3.1 Descrição do funcionamento da INTERFACE EM GERAL

A troca de informações entre o computador (GERENCIADOR) e a Célula (CLP) é feita por intermédio da porta paralela do computador e por uma interface física colocada entre o computador e o CLP, conforme pode ser visto na figura C.2. A interface física, colocada entre o computador e o CLP, tem como principal objetivo compatibilizar os

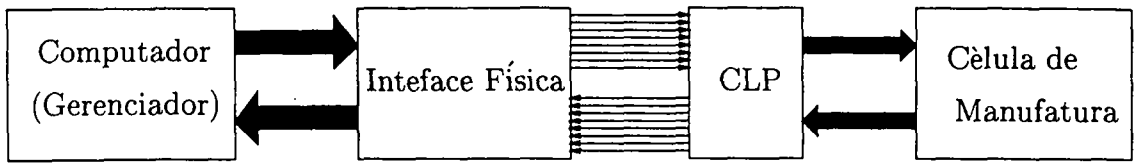


Figura C.2: Esquema de interligação do Computador com a célula de manufatura

níveis de tensão dos sinais elétricos usados no computador, entre 0 e 5[V], com os sinais elétricos do CLP, entre 0 e 24[V]

C.3.2 Protocolo de comunicação

A comunicação é do tipo *Full-Duplex.*

Os eventos são enviados e recebidos num byte, com 7 bits contendo a informação, e mais um bit de controle em forma de pulso (bit mais significativo) indicando a chegada de um novo evento. Para o envio de um comando do computador para a Célula, a duração do pulso do bit de controle é maior que 20[ms] para garantir que o dado seja lido pelo CLP, uma vez que um ciclo de varredura do CLP é 20[ms]. Na leitura de eventos que são enviados pelo CLP, os dados são capturados no instante em que o pulso cai para zero, garantindo a presença de todos os dados no momento da leitura. Na troca de informações entre o Computador e a Célula, os dados são mantidos na saída e correspondem ao último evento, até que um novo evento seja enviado

C.3.3 Interpretação dos eventos enviados do computador para a célula

Para interpretar os eventos enviados do computador para a célula utiliza-se as entradas do CLP com os endereços ímpares de **I:1/1** até **I:1/15**, as quais possuem o valor dos bits que forman o byte enviado pelo computador, sendo que o endereço **I:1/1** representa o bit menos significativo deste byte e o endereço **I:1/15** é usado para indicar a ocorrência de novo evento. Como o tempo de cada ciclo de scan do programa do CLP é muito pequeno, da ordem de alguns milisegundos, vários ciclos se passam durante o intervalo entre o envio de dois eventos. Para garantir então que um evento, após ser enviado para a célula, não seja lido e executado repetidamente pelo CLP utilizou-se um endereço auxiliar *(B3/4093)*

para indicar a ocorrência de um evento novo. Assim, o bit correspondente a este endereço é *setado* cada vez que um evento novo é enviado pelo computador e mantido em nível lógico alto durante apenas um *ciclo de scan*, assegurando que o evento seja identificado uma única vez. Para que esse endereço auxiliar permaneça com nível alto somente durante um *ciclo de scan* utiliza-se a instrução **[OSR]** (one short rising), que habilita a saída durante um ciclo quando o *rung* se torna verdadeiro e depois desabilita esta saída até que o *rung* se torne falso e volte a ser verdadeiro novamente (ocorra uma nova transição de zero para um), o que é feito a cada envio de um novo evento. Como a saída utilizada é do tipo OTE, que é *setada* quando habilitada e *resetada* quando desabilitada, consegue-se mantê-la em nível lógico alto por apenas um *ciclo de scan*

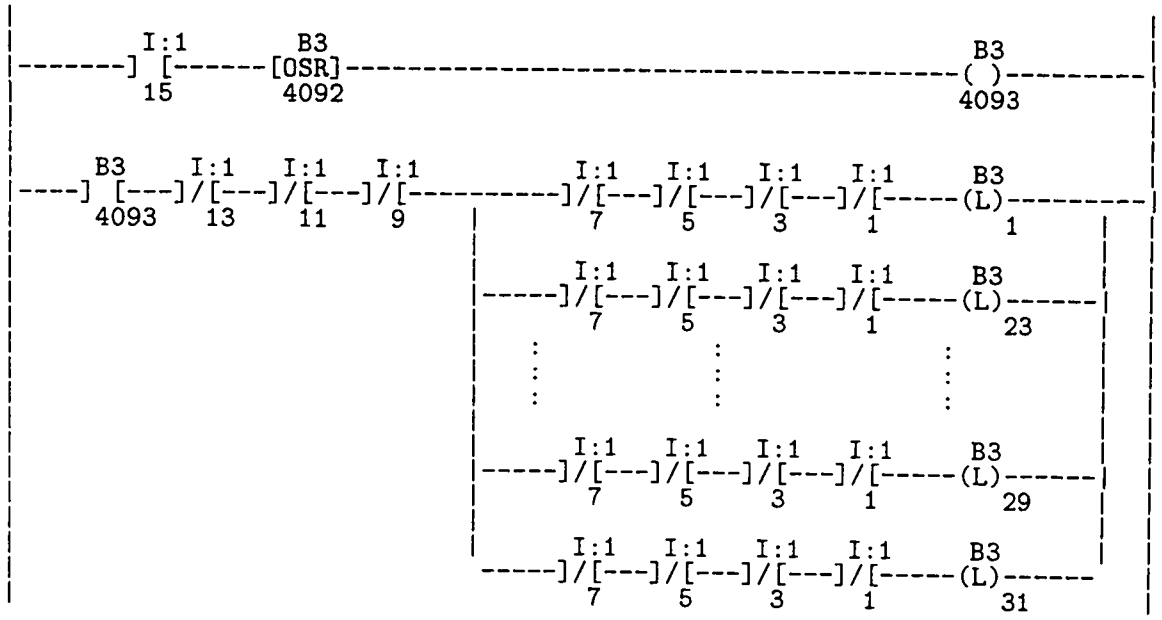


Figura C.3: Diagrama em ladder do envio de eventos do Computador para a célula

C.3.4 Interpretação dos eventos enviados da célula para o computador

Para interpretar os eventos enviados da célula para o computador, utiliza-se as saídas do CLP com os endereços pares de **O:9/0** até **O:9/14**, as quais possuem os valores dos bits que formam o byte enviado pelo CLP, sendo que o endereço **O:9/0** representa o bit menos significativo deste byte e o endereço **O:9/14** é usado para indicar a ocorrência de novo evento.

Num mesmo *ciclo de scan* podem ocorrer mais de um evento na célula, mas por uma questão de restrições tecnológicas do CLP, não é possível enviar mais de um evento a cada *ciclo de scan*. Como tem-se que informar ao computador quando o evento é novo, e essa informação é feita pela transição de zero para um do bit mais significativo do byte enviado, é necessário que entre dois eventos novos seja enviado essa informação de ocorrência de um novo evento. Assim os eventos propriamente ditos são enviados a cada dois ciclos de scan, e não um a cada ciclo como colocado anteriormente. A ordem de envio de eventos identificados num mesmo ciclo segue a ordem de sua identificação. Assim, utiliza-se um endereço auxiliar **(B3/4095)** que é *setado*, habilitando o envio de um novo evento, se no ciclo anterior nenhum foi enviado, é *resetado*, desabilitando o envio de um novo evento, se no ciclo anterior foi enviado um evento.

Apêndice D

Algoritmos computacionais no desenvolvimento do CONDES 3.0

Neste apêndice serão tratados os algoritmos utilizados para o desenvolvimento das funções implementadas no CONDES, ou seja, interseção de linguagens, produto síncrono, entre outros.

D.1 Armazenamento de dados

Um SED para ser representado em forma simbólica, precisa da especificação da estrutura, estado inicial e estados marcados.

Para os estados serem manipulados computacionalmente, são etiquetados por números inteiros e positivos.

As transições também são etiquetadas por números inteiros positivos, onde: números pares de 0 a 998 são reservados para os eventos não-controláveis ; números ímpares de 1 a 999 são reservados para os eventos controláveis; os eventos forçáveis são etiquetados por letras de 3 caracteres como máximo, onde a cada letra se associa um número inteiro maior ou igual a 1000

Exemplo

O SED descrito na Figura D.1.

pode ser representado pela estrutura:

$$\delta(1,0) = 2 \quad \delta(2,0) = 3$$
$$\delta(1,2) = 3 \quad \delta(2,3) = 0$$

com estado inicial e estados marcados:

$$q_0 = 0 \; ; \; q_m = \{3\}$$

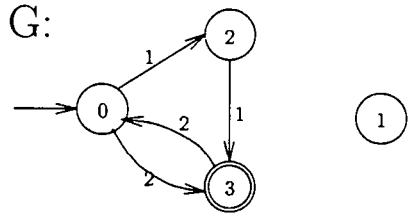


Figura D.1:

Para o tratamento computacional, a estrutura será representada pelos vetores:

$out\_state[k]$ = Vetor de estados de saída.
$eventk[k]$ = Vetor de eventos
$in\_state[k]$ = Vetor de estados de entrada.
como é mostrado na tabela D.1

O estado inicial e estados marcados são representados pelas variáveis;

$initial\_state = 0$
$marker\_states[*] = \{3\}$

k	$out\_state[k]$	$event[k]$	$in\_state[k]$
1	0	1	2
2	0	2	3
3	2	1	3
4	3	2	0

Tabela D.1:

D.1.1 Listas encadeadas

As listas encadeadas são estruturas de dados em forma de listas. Para o tratamento de Sistemas a Eventos Discretos, cada lista relaciona-se com um estado do SED. O algoritmo seguinte constrói uma lista encadeada por estados a partir dos vetores $out\_state[k]$ e $event[k]$

Algoritmo:

Este algoritmo constrói a lista encadeada por estados. A entrada de dados para este algoritmo consiste de:

- Uma variável m, que indica o máximo valor de etiqueta dentre os estados. Cabe notar que N^o *de estados* $\leq m$. Esta condição é necessária porque a enumeração dos estados pode-se fazer em forma desordenada e incompleta.
- $ntrans$ = número de transições.

O algoritmo constrói os vetores de lista encadeada $rfirst[*]$ e $rnext[*]$

```
for ( j=0 ;   j ≤ m  ;  j++ ) rfirst[j] = 0;
for ( j=1 ;   j ≤ ntrans ;  j++ ) rnext[j] = 0;
for ( k = 1 ; k ≤ ntrans ; k++)
{
      i = out_state[k];
      p = rfirst[i];
      rfirst[i] = k;
      rnext[k] = p;
}
```

Se ao exemplo anterior se aplica o algoritmo descrito anteriormente, obtém-se os vetores de lista encadeada mostrados na tabela D.2. Esta tabela D.2 é interpretada como segue:

Para o estado $i = 0$

A primeira transição encontra-se na posição $rfirst[i] = rfirst[0] = 2$ $(k = 2)$, quer dizer $out\_state[k] = out\_state[2] = 0$, $event[k] = event[2] = 2$ e $in\_state[k] = in\_state[2] =$

i	$rfirst[i]$	k	$rnext[k]$	$out\_state[k]$	$event[k]$	$in\_state[k]$
0	2	1	0	0	1	2
1	0	2	1	0	2	3
2	3	3	0	2	1	3
3	4	4	0	3	2	0

Tabela D.2:

3; e a próxima transição correspondente ao estado em questão, encontra-se na posição $rnext[k] = rnext[2] = 1$ $(k = 1)$, agora para $k = 1$ $rnext[k] = rnext[1] = 0$ significa que nesse estado $i = 0$ não tem mais transições de saída.

Para o estado $i = 1$,

$rfirst[i] = 0$, quer dizer que o estado 1 não tem transições de saída.

D.2 Gerador trim

Para análise considere o gerador da Figura D.1

Para que um gerador seja trim tem que satisfazer duas condições: ser acessível e co-acessível.

D.2.1 Gerador acessivel

Para obter o gerador acessível, basta fazer evoluir o gerador primitivo a partir do estado inicial. Logicamente serão visitados somente os estados acessíveis.

São declaradas duas variáveis que registram a evolução dos geradores resultante e primitivo.

$visit$ = Numero de estados a visitar + 1
$next[j] = i_1$

onde:

j = Estado do gerador resultante.

i_1 = Estado do gerador primitivo.
$next[*]$ = vetor que registra os estados visitados.

Às variáveis do gerador primitivo será acrescentado um subindice 1. Por exemplo as variáveis serão: k_1, $out\_state_1[k_1]$, $event_1[k_1]$, $in\_state_1[k_1]$,

A evolução começa com a primeira transição do estado inicial, isto é:

Para o estado inicial ($i = 0$)

$next[i] = initial\_state_1$,
$next[0] = 0$; o primeiro estado a ser registrado é o inicial.
$visit = 1$; é inicializado o contador de estados a visitar.

Obtem-se a posição da primeira transição do gerador primitivo ($rfirst_1[initial\_state1] = rfirst_1[0] = 2 = k_1$), para k_1 verifica-se se o estado alcançado ($in\_state_1[k_1]$) já está registrado. Se não tem-se um novo estado, como neste caso.

$next[visit] = in\_state_1[k_1]$
$next[1] = 3$; é registrado o primeiro estado visitado
$k = 1$; inicializamos o contador de transições do resultado.
$out\_state[k] = i$; $event[k] = event_1[k_1]$; $in\_state[k] = visit$;
$out\_state[1] = 0$; $event[1] = 2$; $in\_state[1] = 1$; (primeira transição do resultado)
$visit = 2$; os estados a visitar incrementa um

No gerador primitivo pega-se a próxima transição $rnext_1[k_1] = 1$, atribui-se a posição $k_1 = 1$ e faz-se a mesma análise

$next[visit] = in\_state_1[k_1]$
$next[2] = 2$; é registrado o segundo estado visitado.
$k = 2$; é incrementado o contador de transições.
$out\_state[k] = i$; $event[k] = event_1[k_1]$; $in\_state[k] = visit$;
$out\_state[2] = 0$; $event[2] = 1$; $in\_state[2] = 2$; (segunda transição)
$visit = 3$

Como não se tem mais transições no estado inicial do gerador primitivo($rnext_1[k_1] = 0$), passa-se para o próximo estado da lista.

Para o estado $i = 1$

$next[i] = next[1] = 3$; Pega-se o próximo estado da fila de estados registrados.

Obtem-se a primeira transição correspondente ao estado atual do gerador primitivo $rfirst_1[next[i]] = rfirst[3] = 4 = k_1$ e a partir de aqui continua-se da mesma forma que para o caso anterior.

A condição de parada, será quando $visit = i$, quer dizer que todos os estados a ser visitados foram visitados..

Até aqui os resultados obtidos estão mostrados na tabela D.3

k	$out\_state[k]$	$event[k]$	$in\_state[k]$	i	$(next[i])$
1	0	2	1	0	(0)
2	0	1	2	1	(3)
3	1	2	0	2	(2)
4	2	1	1		

Tabela D.3:

D.2.2 Gerador co-acessível

Para obter um gerador co-acessível, primeiro se identificam os estados não co-acessíveis.

Para cada estado do gerador primitivo, se faz o mesmo procedimento que para o cálculo do gerador acessível até alcançar um estado marcado. Se isto não for possível, elimina-se esse estado e os percorridos mais as transições associadas a esse estado.

D.3 Interseção de linguagens

Sejam os geradores da Figura D.1 e da Figura D.2.

No autômato da Figura D.2 as transições etiquetadas por letras, serão mapeados em números maiores ou iguais a 1000. Isto é:

$a \rightarrow 1000$
$b \rightarrow 1001$

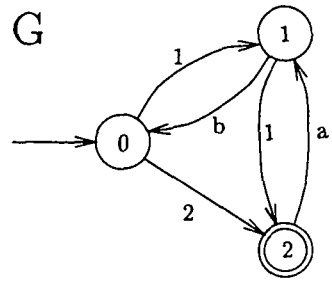


Figura D.2:

As características deste gerador são:

$$\delta(1,0) = 2 \qquad \delta(2,0) = 2$$
$$\delta(1,1) = 2 \qquad \delta(1001,1) = 0$$
$$\delta(1000,2) = 1$$

estado inicial e estado marcados.

$$q_0 = 0 \ ; \ q_m = \{2\}$$

Aplicando o algoritmo de listas encadeadas, obtém-se os vetores como descritos na tabela D.4

i	$rfirst[i]$	k	$rnext[k]$	$out\_state[k]$	$event[k]$	$in\_state[k]$
0	2	1	0	0	1	1
1	4	2	1	0	2	2
2	5	3	0	1	1	2
		4	3	1	1001	0
		5	0	2	1000	1

Tabela D.4:

D.3.0.1 Evolução da interseção

São declaradas 3 variáveis, que registram a evolução da interseção em cada gerador primitivo e resultante.

$visit$ = Numero de estados a visitar + 1

$next1[j] = i_1$
$next2[j] = i_2$

onde:

j = Estado do gerador resultante.
i_1 = Estado do gerador primitivo 1.
i_2 = Estado do gerador primitivo 2.
$next1[*]$ e $next2[*]$ = vetores que registram os estados visitados.

Às variáveis dos geradores primitivos $i = 1, 2$ serão associados um subíndice i. Por exemplo, para o gerador primitivo 1, as variáveis serão: k_1, $out\_state_1[k_1]$, $event_1[k_1]$, $in\_state_1[k_1]$, $initial\_state_1$, ...

Para o estado inicial da interseção ($i = 0$);

$next1[i] = initial\_state_1$ $\quad$ $next2[i] = initial\_state_2$
$next1[0] = 0$ $\quad$ $next2[0] = 0$
$visit = 1$

O sistema só evolui se para o estado atual dos geradores primitivos, existirem eventos de interseção como saída. A busca de eventos de interseção se faz por meio das listas encadeadas.

Neste caso, o primeiro evento (k = 1) de interseção encontrado está em $k_1 = 2$ e $k_2 = 2$ cujo valor é 2. O sistema evolui e verifica se este estado alcançado já está presente na lista de estados alcançados; se não está, como neste caso, então obtém-se um novo estado alcançado cujo valor é:

$next1[visit] = in\_state_1[k_1]$ $\quad$ $next2[visit] = in\_state_2[k_2]$
$next1[1] = 3$ $\quad$ $next2[1] = 2$,

$k = 1$; Inicia-se um contador de transições da interseção.

$out\_state[k] = i$; $event[k] = (event_1[k_1] \vee event_2[k_2])$; $in\_state[k] = in\_state_1[k_1]$;
$out\_state[1] = 0$; $event[1] = 2$; $in\_state[1] = 1$;
consequentemente os estados a visitar incrementa um $visit = 2$

O segundo evento de interseção (k = 2) encontrado está em $k_1 = 1$ e $k_2 = 1$ cujo valor é 1, o sistema evolui e verifica se este estado alcançado já está presente na lista de estados alcançados; se não está, como neste caso, então obtém-se um novo estado alcançado cujo

valor é:

$next1[2] = 2 \qquad next2[2] = 1;$
$visit = 3$

Para o estado $(i = 1)$

$next1[i] = next1[1] = 3;$
$next2[i] = next2[1] = 2;$

A partir deste ponto continua-se da mesma forma que para o estado inicial.

A condição de parada também será quando $i = visit$.

O resultado final obtido é aquele mostrado na tabela D.5

k	$out\_state[k]$	$event[k]$	$in\_state[k]$	i	$(next1[i]; next2[i])$
1	0	2	1	0	(0;0)
2	0	1	2	1	(3;2)
3	2	1	1	2	(2;1)

Tabela D.5:

D.4 Produto síncrono

Para ilustrar este problema também serão usados os geradores das figuras D.1 e D.2

D.4.1 Evolução do algoritmo do produto síncrono

A evolução do algoritmo do produto síncrono é muito semelhante ao da interseção. Para tratar a evolução do produto síncrono, primeiramente constróe-se um vetor de eventos de sincronismo.

$sync[*] = \{1, 2\}$

Como no caso da interseção, também se usa dois vetores que registram a evolução do produto síncrono.

Para o estado inicial do produto síncrono $(i = 0)$

$next1[i] = initial\_state_1$ $\quad$ $next2[i] = initial\_state_2$
$next1[0] = 0$ $\quad$ $next2[0] = 0$
$visit = 1$

O sistema para evoluir, pega a primeira transição do gerador primitivo1 e confere se este é um evento de sincronismo, se ele é, busca que este evento esteja na saída do gerador primitivo 2. Se o evento for encontrado então o sistema evolui nos dois geradores primitivos, de forma similar ao caso da interseção, se não o sistema pega a próxima transição no gerador 1 e continua com a análise. A busca dos eventos se faz por meio das listas encadeadas.

Se a transição que está sendo analisada não for evento de sincronismo, o sistema evolui só no gerador primitivo1. Terminadas as transições do gerador 1, a mesma analise é feita para os eventos de não sincronismo do gerador primitivo 2.

Continuando com o exemplo, o sistema pega a primeira transição no estado inicial do gerador 1 ($rfirst_1[initial\_state_1] = 2 = k_1$), confere que ele é um evento de sincronismo e está no gerador 2 na posição $k_2 = 2$ cujo valor é 2. Então o sistema evolui e confere se este estado alcançado já está presente na lista de estados registrados; se não está, como neste caso, obtém-se um novo estado alcançado cujo valor é:

$next1[visit] = in\_state_1[k_1]$ $\quad$ $next2[visit] = in\_state_2[k_2]$
$next1[1] = 3$ $\quad$ $next2[1] = 2,$

$k = 1$; Inicializa-se o contador do número da transição no produto síncrono.

$out\_state[k] = i$; $event[k] = (event_1[k_1] \vee event_2[k_2])$; $in\_state[k] = visit$;
$out\_state[1] = 0$; $event[1] = 2$; $in\_state[1] = 1$;

consequentemente os estados a visitar incrementa em um, isto é $visit = 2$

Com o segundo evento do gerador 1, casualmente ocorre o mesmo que para o primeiro (o evento a ser tratado está em $k_1 = 2$ e $k_2 = 2$ cujo valor é 2). Então seguindo o mesmo procedimento, obtém-se:

$k = 2$; segunda transição do produto síncrono.
$next1[2] = 3$ $\quad$ $next2[2] = 2$;
$visit = 3$; incrementa-se o número de estados registrados.

Para o estado ($i = 1$).

$next1[i] = next1[1] = 3$ $next2[i] = next2[1] = 2,$

No estado 3 do gerador 1 e no estado 2 do gerador 2, para o estado 3 do gerador 1, pega-se a primeira transição, que neste caso é a transição 2, confere-se que ela é uma transição de sincronismo, mas não é transição de saída do estado 2 do gerador 2, portanto passa-se para a próxima transição. Como o estado 3 do gerador 1 não tem mais transições, pega-se a primeira transição de não sincronismo do estado correspondente no gerador 2.

Neste caso a transição "a" evolui somente no gerador 2. Isto é: $k_2 = 5$. Verifica-se se o estado alcançado já está registrado. Se não, como neste caso, o novo estado é:

$next1[visit] = next1[i]$; no gerador 1 não evolui
$next2[visit] = in\_state_2[k_2]$
$next1[3] = 3$ $next2[3] = 1,$
$visit = 4$; incrementa-se o número de estados registrados.

Para o estado atual não existe mais transições de saída. Então passa-se para o próximo estado registrado na fila:

Para o estado ($i = 3$).

A partir deste ponto é o tratamento é o mesmo dos casos anteriores.

A condição de parada também é quando $visit = i$.

No final deste algoritmo obtém-se os valores da tabela D.6

D.4.2 Máxima linguagem controlável

Para o calculo do supervisor máximo controlável, baseamo-nos no algoritmo da seção 3.2.1.4. Dentro deste cálculo já foram considerados o produto síncrono, acessibilidade e co-acessibilidade de geradores.

k	$out\_state[k]$	$event[k]$	$in\_state[k]$	i	$(next1[i]; next2[i])$
1	0	2	1	0	(0;0)
2	0	1	2	1	(3;2)
3	1	1000	3	2	(2;1)
4	2	1	1	3	(3;1)
5	2	1001	4	4	(2;0)
6	3	1001	5	5	(3;0)
7	4	1	3	6	(0;2)
8	5	2	6	7	(0;1)
9	6	1000	7	8	(2;2)
10	7	1	8		
11	7	1001	0		
12	8	1000	2		

Tabela D.6: