PATRÍCIA NASCIMENTO PENA

VERIFICAÇÃO DE CONFLITO NA SUPERVISÃO DE SISTEMAS CONCORRENTES USANDO ABSTRAÇÕES

FLORIANÓPOLIS
2007

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

VERIFICAÇÃO DE CONFLITO NA SUPERVISÃO DE SISTEMAS CONCORRENTES USANDO ABSTRAÇÕES

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutor em Engenharia Elétrica.

PATRÍCIA NASCIMENTO PENA

Florianópolis, 18 de maio de 2007.

VERIFICAÇÃO DE CONFLITO NA SUPERVISÃO DE SISTEMAS CONCORRENTES USANDO ABSTRAÇÕES

Patrícia Nascimento Pena

Esta Tese foi julgada adequada para a obtenção do título de Doutor em Engenharia Elétrica, Área de Concentração em *Automação e Sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.

Prof. José Eduardo Ribeiro Cury, Dr.
Orientador

Prof. Nelson Sadowski, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. José Eduardo Ribeiro Cury, Dr.
Presidente

Prof. Antonio Eduardo Carrilho da Cunha, Dr.

Prof. Ricardo Lüders, Dr.

Prof. Guilherme Bittencourt, Dr.

Prof. Max Hering de Queiroz, Dr.

Prof. Victor Juliano De Negri, Dr.

"Mas agora, eu era muito menos que humana - e só realizaria o meu destino especificamente humano se me entregasse, como estava me entregando, ao que já não era eu, ao que já é inumano. E entregando-me com a confiança de pertencer ao desconhecido. Pois só posso rezar ao que não conheço. E só posso amar à evidência desconhecida das coisas, e só me posso agregar ao que desconheço. Só esta é que é uma entrega real."

Clarice Lispector - A paixão segundo G.H.

Ao meu pai.

Agradecimentos

Eu preciso começar esse agradecimento falando do Daniel. Sem o meu marido tudo teria sido muito mais difícil. Ele me acompanhou nessa empreitada desde o oitavo dia do meu doutorado até o último. Se mudou para Florianópolis, se mudou para Ann Arbor, Michigan. Parou sua vida por 4 anos, por mim. E fez muito mais.

Ao Cury tenho que agradecer pela orientação e por ter me recebido como aluna em um momento em que estava com sua quota de alunos excedida. Ele sabe que eu nunca aceitaria um não como resposta e acredito que nunca dei a ele a chance de me dizer não. Trabalhamos em várias frentes: SEDs, comissão do DAS, entre outros... Dividimos muitos doces na sobremesa e também na sexta-feira à tarde na FEESC. Criamos uma relação diferente, de amizade. Tirando alguns poucos momentos estressantes, só tenho a agradecer por ele ter aberto para mim as portas de uma nova área.

Falando do Cury, não posso deixar de falar de sua família. A família Cury cuidou de mim, me recebendo em sua casa como se eu fosse filha. Além de me hospedar na primeira semana, a Dóris me levou para a primeira compra de supermercado, me ajudou a encontrar minha primeira moradia, me emprestou panelas, pratos, talheres, travesseiros e cobertor. A família inteira foi como uma segunda família para mim: obrigada Pedro, Clara e João pela convivência e por me receberem em sua casa.

Muitos foram os amigos feitos durante a caminhada. Desde os primeiros dias, a Tatiana Garcia esteve por perto. A formação parecida fez com que o André Leal e eu nos identificássemos logo de início. Durante a fase de créditos conheci pessoas: Christianne Reiser, Fábio Baiano, Felipe Moura, Rodrigo Pinto. Sendo todos alunos de mestrado, se foram antes de mim. Com a mudança das baias para a BU coincidindo com o fim dos créditos, pude começar a conviver com os demais colegas do doutorado. Desde o princípio admirei a Michelle Wangham. Muitíssimo generosa, sempre disponível, sempre engajada. Longos papos com o Vallim, almoços e cafés com Rafael, Alysson Bessani, Cássia Tatibana, Emerson Mello, Fábio Pinga e Favarim, chocolates sempre bem vindos do Rodrigo Sumar. O Paulo Mafra tentou me converter para o software livre. Sem sucesso. Ganhei um amigo a quem recorro até hoje quando me enrolo com os bits.

A Tati, a Michelle, Karina Barbosa, a Cris Paim e a Chris Reiser são amigas que levo para toda a vida.

Pelo grupo de SEDs passaram várias pessoas: Gustavo Bouzon, Agnelo, Danilo, Daniel. Muitas e relevantes discussões ocorreram nos seminários de SEDs. Depois vieram dois conterrâneos, Frederico Mello e Rodrigo Braga que também fizeram seus mestrados no grupo de SEDs, e honraram o nome da UFMG, minha Universidade de origem.

A convivência na PGEEL foi sempre muito agradável. Wilson e Marcelo da secretaria

da Pós sempre prontos a ajudar. Tive o prazer de ter aulas com o Guilherme Bittencourt, Eduardo Camponogara, Rômulo Oliveira, Werner Kraus, Ricardo Rabelo e o próprio Cury. Além destes, cito o Jean-Marie Farines que foi o chefe do Departamento de Automação e Sistemas durante uma boa parte do tempo que estive na UFSC e com quem tive bastante contato enquanto "prefeita das baias" como foi batizado (pelo Vallim) o representante dos doutorandos associados ao LCMI, junto ao departamento. Além destes, cito o Joni Fraga. Quanto mais eu conheço, mais eu gosto. Entre os professores do DAS, menciono ainda o Max de Queiroz. Eu o conheci melhor quando ele voltou de seu estágio sanduíche de doutorado na Universidade de Toronto. O trabalho apresentado nesta tese baseia-se na abordagem de Controle Supervisório Modular Local, desenvolvida durante seu mestrado na UFSC. Indispensável, portanto, falar da importância do Max no meu trabalho. Ele foi também de minha banca de qualificação e defesa final do doutorado.

Em julho de 2001 conheci o Antônio Carrilho na Carnegie Mellon University. Ele estava começando seu estágio sanduíche nesta universidade e eu estava visitando o Prof. Bruce Krogh por um período de 15 dias. Àquele momento, meu plano era aplicar para fazer o doutorado naquela Universidade. O Antônio falou um pouco sobre a UFSC e seu orientador. Foi a primeira vez que ouvi falar do Prof. Cury. A tragédia do 11 de setembro de 2001 e as incertezas a respeito de uma potencial guerra me fizeram desistir de tentar o doutorado nos EUA. UFSC era então o meu futuro. Agradeço o Antônio por ter, involuntariamente, guiado minha escolha. Cheguei na UFSC a tempo de assistir sua defesa de doutorado. E o Antônio foi parte da minha banca de qualificação, relator e membro da banca de defesa final do doutorado.

Depois de dois anos e meio na UFSC fui para os Estados Unidos, trabalhar com o Prof. Stéphane Lafortune, na Universidade de Michigan em Ann Arbor. Essa foi uma experiência fantástica. Estava em uma universidade de renome, trabalhando com um dos professores mais importantes da área de SEDs. O Stéphane se integrou ao trabalho como orientador. Este período foi vital pois representou o início do sucesso do meu projeto. A maior parte dos resultados apresentados nesta tese foram alcançados durante o período do estágio sanduíche. A participação do Cury foi fundamental. Conseguimos trabalhar intensamente à distância. Formamos uma equipe, eu, Cury e Stéphane. A participação na organização do oitavo WODES (8th International Workshop on Discrete Event Systems) foi uma oportunidade muito bacana que me foi dada pelo Stéphane, a quem dedico um agradecimento especial.

Além dos membros da banca já mencionados, Guilherme Bittencourt (eu me livrei de quase todos os "implica que" da tese, antes de passar para o GB.. quase todos), Ricardo Lüders e Victor de Negri fizeram parte da minha banca. Agradeço pela paciência em ler o texto e pelas inúmeras contribuições dadas.

Por fim, mas não menos importante, quero agradecer à minha família. Sair de casa não foi fácil, mas eu tive apoio incondicional. Apoio psicológico, apoio financeiro.

O Prof. Bruce Krogh perguntou em 2001 em um almoço *brown bag* a razão que levou cada um dos presentes a estudar engenharia. A minha resposta causou estranheza. Até aquele

momento eu nunca havia refletido a respeito. Eu disse que foi por causa do meu pai. Foi a resposta que saiu, diante de uma pergunta à qual eu não tinha resposta. Passados 6 anos, eu continuo com a mesma resposta.

Minha mãe me apoiou desde o início, inclusive quando ninguém acreditava que algumas decisões que eu havia tomado eram acertadas. Esse apoio foi fundamental. Meus irmãos, Daniel e Renato, mesmo de longe me deixavam por dentro do que estava acontecendo em casa. Através de fotos, ou longas conversas no telefone e também na internet. E o Pedro... As visitas foram menos freqüentes que eu gostaria, mas a cada visita eu me sentia de novo amparada, no colo da mãe/pai, mesmo que a visita fosse do meu irmão. Valeu.

Por fim, agradeço o CNPq pela bolsa de doutorado, taxa de bancada e bolsa sanduíche que viabilizaram esse trabalho.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Doutor em Engenharia Elétrica.

VERIFICAÇÃO DE CONFLITO NA SUPERVISÃO DE SISTEMAS CONCORRENTES USANDO ABSTRAÇÕES

Patrícia Nascimento Pena

Maio/2007

Orientador: Prof. José Eduardo Ribeiro Cury, Dr.
Área de Concentração: Automação e Sistemas
Palavras-chave: sistemas a eventos discretos, controle supervisório, conflito, controle modular local, abstrações, propriedade do observador
Número de Páginas: xvi + 157

A explosão do espaço de estados associada ao teste para detecção do conflito é um dos principais problemas que impedem a aplicação da Teoria de Controle Supervisório de Sistemas a Eventos Discretos a sistemas industriais reais. O conflito é uma propriedade global dos sistemas concorrentes sendo que, para sua detecção, deve-se verificar não-bloqueio da composição dos subsistemas que estão sendo verificados. Esta tese trata do problema de detecção de conflito de forma eficiente. Neste trabalho, propõe-se um novo teste de não-conflito baseado em abstrações dos supervisores, obtidas pela operação de projeção natural. Apresentam-se dois conjuntos de condições sobre as abstrações para os quais o teste de não-conflito pode ser aplicado, com resultado equivalente àquele do teste sobre os supervisores originais. No primeiro conjunto de condições os eventos compartilhados são mantidos nas abstrações e a projeção deve possuir a propriedade do observador. O segundo conjunto de condições sobre as abstrações leva em conta propriedades estruturais dos supervisores originais para derivar o conjunto de eventos a serem mantidos nas abstrações, além da propriedade do observador sobre a projeção obtida. As duas abordagens podem ser utilizadas em conjunto para obter abstrações possivelmente melhores, de forma a obter maior redução do espaço de estados na verificação de não-conflito. Apresenta-se ainda um algoritmo para verificação da propriedade do observador. Esta propriedade é utilizada exaustivamente nos resultados apresentados e sua verificação torna-se de grande interesse para a aplicação dos resultados obtidos.

Abstract of Thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

VERIFICATION OF NONCONFLICT FOR CONCURRENT SUPERVISORS USING ABSTRACTIONS

Patrícia Nascimento Pena

May/2007

Advisor: Prof. José Eduardo Ribeiro Cury, Dr.
Area of Concentration: Systems and Automation
Key words: discrete event systems, supervisory control, conflicts, local modular control, abstractions, observer property
Number of Pages: xvi + 157

The state explosion associated to this nonconflict test is one of the main problems that prevent the use of Supervisory Control Theory to control industrial plants. The conflict is a global property of concurrent systems that can be detected by checking if the system, obtained by the composition of all the subsystems that are to be checked, is blocking. This thesis deals with the problem of detecting conflicts efficiently. We propose a novel test based on abstractions of the supervisors, obtained by computing the natural projection. Two sets of conditions over the abstractions are presented such that the result of the test over the abstractions is equivalent to the test over the original supervisors. The first set of conditions relies on the set of shared events, that have to be kept by the projection, and the observer property, a property of natural projections. The second set of conditions uses structural properties of the supervisors to derive the set of events to be kept by the projection, besides the observer property. The two approaches can be used together in order to obtain better (smaller) abstractions, providing a bigger reduction in the verification. We present also an algorithm for verifying if a natural projection has the observer property. Since this property is being widely used in our results, the verifications of its existence is of interest.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

Lista de Símbolos

Capítulo 2

Σ	alfabeto
Σ^*	conjunto de todas as cadeias de comprimento finito sobre Σ
σ	símbolo
$\sigma \in \Sigma$	símbolo em Σ
ϵ	cadeia vazia
s, t, u, v	cadeias
$s \in \Sigma^*$	cadeia sobre Σ
st	concatenação das cadeias s e t
$s \leq t$	s é prefixo de t
$L \subseteq \Sigma^*$	linguagem sobre Σ
$L_1 \| L_2$	produto síncrono das linguagens L_1 e L_2
A, A_G	autômato (determinístico, de estados finitos)
$(Q, \Sigma, \delta, q_0, Q_m)$	representação de um autômato A
$\delta : Q \times \Sigma^* \to Q$	função de transição estendida de A
Q	conjunto de estados de A
q_0	estado inicial de A
Q_m	conjunto de estados marcados de A
(q, σ, q')	transição com estado de saída q, estado de entrada q' e etiqueta de evento σ
$\mathcal{L}(A)$	linguagem gerada pelo autômato A
$\mathcal{L}_m(A)$	linguagem marcada pelo autômato A
Σ_c	conjunto de eventos controláveis
Σ_{uc}	conjunto de eventos não-controláveis
$\overline{L}$	prefixo-fechamento da linguagem L
$i \in I = \{1 \dots n\}$	índice
$j \in J = \{1 \dots m\}$	índice
$P_i : \Sigma^* \to \Sigma_i^*$	projeção natural de cadeias em Σ^* em cadeias em Σ_i^*
$P_i^{-1} : \Sigma_i^* \to \Sigma^*$	projeção inversa de cadeias em Σ_1^* em cadeias em Σ^*

G	planta global
$G_i = (Q_i, \Sigma_i, \delta_i, q_{0i}, Q_{mi})$	subsistemas da planta global, para $i \in I = \{1, \cdots, n\}$
E	especificação global
E_j	especificações locais, para $j \in J = \{1, \cdots, m\}$
K	linguagem desejada, $K = E \|\| \mathcal{L}_m(G)$
K_j	linguagens desejadas, para $j \in J = \{1, \cdots, m\}$, $K_j = E_j \|\| \mathcal{L}_m(G_j)$
$\Gamma = \{\gamma \in 2^{\Sigma} \| \gamma \supseteq \Sigma_{uc}\}$	conjunto de todos os padrões de controle
$V : \mathcal{L}(G) \to \Gamma$	mapa que associa cadeias de $\mathcal{L}(G)$ a elementos de Γ
V/G	representa o sistema sob a ação do supervisor
$\mathcal{C}(E, \mathcal{L}(G))$	conjunto de linguagens controláveis contida na linguagem E
$Sup\mathcal{C}(E, \mathcal{L}(G))$	máxima linguagem controlável contida em E
S	supervisor, $S = Sup\mathcal{C}(K, \mathcal{L}(G))$
S_j	supervisores, para $j \in J = \{1, \cdots, m\}$, $S_j = Sup\mathcal{C}(K_j, \mathcal{L}(G_j))$
L é controlável e.r.a $\mathcal{L}(G)$	$\overline{L}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{L}$
L é $\mathcal{L}_m$ fechada	$L = \overline{L} \cap \mathcal{L}_m(G)$

Capítulo 4

$\varrho_A(q)$	conjunto de estados do autômato A alcançáveis a partir do estado q (IDES)

Capítulo 5

$S_j \subseteq \Sigma_j^*$	linguagens que implementam a ação dos supervisores
$\Sigma = \bigcup_{j=1}^{m} \Sigma_j$	alfabetos
Σ_r	conjunto de eventos relevantes
$\Sigma' \subseteq \Sigma$	conjunto de eventos relevantes
$\Sigma'_i = \Sigma_i \cap \Sigma_r$	conjunto de eventos relevantes locais
Σ_s	conjunto de eventos compartilhados por dois ou mais supervisores
Γ, Υ	alfabetos genéricos
$P_{\Gamma \to \Upsilon} : \Gamma^* \to \Upsilon^*$	projeção natural de cadeias em Γ^* em cadeias em Υ^*
$\theta : \Sigma^* \to \Sigma_r^*$	projeções naturais de cadeias em Σ^* em cadeias em Σ_r^*
$\theta_j : \Sigma_j^* \to (\Sigma_j \cap \Sigma_r)^*$	projeções naturais de cadeias em Σ_j^* em cadeias em $(\Sigma_j \cap \Sigma_r)^*$
$s, u \in \Sigma^*$	cadeias em Σ^*
$u_j \in \Sigma_j^*$	cadeias em Σ_j^*
$t_j \in \Sigma_j'^*$	cadeias em $\Sigma_j'^*$
Σ_r^I	conjunto inicial de eventos relevantes
$\Sigma_j'^I = \Sigma_j \cap \Sigma_r^I$	conjunto inicial de eventos relevantes locais

Capítulo 6

$i \in I = \{1 \dots n\}$	índice
Σ_{E_j}	conjunto de eventos da especificação E_j
$a \in \Sigma_r^*$	cadeia em Σ_r^*
$s'_1, \mu_i^1, a_1 \in (\Sigma_1 \cup \Sigma_r)^*$	cadeias em $(\Sigma_1 \cup \Sigma_r)^*$
$s'_2, \mu_i^2, a_2 \in (\Sigma_2 \cup \Sigma_r)^*$	cadeias em $(\Sigma_2 \cup \Sigma_r)^*$
$\mu' \in \Sigma_1^*$	cadeias em Σ_1^*
$\mu \in \Sigma_2^*$	cadeias em Σ_2^*
$\sigma_i \in \Sigma_r$	evento relevante
$G_1 = G_c \|\| G_{1p}$	planta G_1
G_c	planta compartilhada de G_1
G_{1p}	planta privada de G_1
$\Sigma_{sa} = \Sigma_s \cup \Sigma_r$	alfabeto compartilhado por a_j
$s'_j, \mu_i^j, a_j \in (\Sigma_j \cup \Sigma_r)^*$	cadeias em $(\Sigma_j \cup \Sigma_r)^*$
η_j	cadeias em Σ_j^*
$G_{c\{j,m\}}$	planta compartilhada entre S_j e S_m
G_{jp}	planta privada de G_j
M_j	conjunto de indices de supervisores que compartilham plantas com S_j
$\eta_{jm} \in M_j$	subsequência que deve estar em η_j
Ξ	conjunto de cadeias disjuntas que formam η_j
E'_j	especificação genérica fictícia
Σ'_{E_j}	conjunto de eventos da especificação fictícia E'_j
Σ_{rT}	eventos relevantes obtidos pelo procedimento T
Σ_{rW}	eventos relevantes obtidos pelo procedimento W
$\Sigma_{rW \circ T}$	eventos relevantes obtidos pelo procedimento W $\circ$ T
Σ_{sT}	eventos compartilhados por quaisquer duas T-abstrações

Capítulo 7

$S = (Q^S, \Sigma, \delta^\tau, q_0^S, F^S)$	autômato que representa o supervisor
$\mathcal{L}_m(S)$	linguagem marcada de S
$\mathcal{L}(S)$	prefixo-fechamento de $\mathcal{L}_m(S)$
$M = (Q^M, \Sigma^\tau, \delta^\tau, q_0^M)$	estrutura sem marcação, correspondente a S
Q^M	estados de M
τ	evento especial usado para representar um estado marcado de S na estrutura M
$\Sigma^\tau = \Sigma \cup \tau$	alfabeto de M
δ^τ	função de transição de M
q_0^M	estado inicial de M
$\mathcal{L}(M) \subseteq \Sigma^{\tau *}$	linguagem implementada por M
$N = \mathcal{L}(M) \cap \Sigma^*$	sublinguagem de $\mathcal{L}(M)$
$T(s) \subseteq \Sigma^*$	conjunto de sufixos de $s \in N$ em $\mathcal{L}(M)$

$T_S(s) \subseteq \Sigma^*$	conjunto de sufixos de $s \in \overline{S}$ em $\mathcal{L}_m(S)$
$V_G = (Q, \Sigma^\tau, \delta, q_0)$	OP-verificador
$\Sigma_u = \Sigma - \Sigma_r$	eventos não-relevantes
Q_{T+1}	estados enumerados de O_S
Q_T	estados tratados de O_S
$Dead$	estado de O_S cuja alcançabilidade implica a violação da propriedade do observador
$q_1, q_2 \in Q^M$	estados de M
$En^M(q)$	conjunto de eventos habilitados no estado q de M
$T'(s) = T(s)\tau^*$	conjunto de sufixos de s em $\mathcal{L}(M)$

Lista de Abreviaturas

BDD	Bynary Decision Diagram
CFP	Conjunto de estados Fracamente Proibidos
CLP	Controlador Lógico Programável
FSM	Finite State Machine
FSMwP	Finite State Machine with Parameters
GMC	Gerador de Marcação Colorida
IDES	Interacting Discrete Event Systems
IMS	Sistema de Manufatura Integrado
MCP	Modular Control with Priorities
PCS	Problema do Controle Supervisório
RSC	Representação por Sistema Composto
RSP	Representação por Sistema Produto
R&W	Ramadge e Wonham
SED	Sistema a Eventos Discretos
SEDMT	Sistema a Eventos Discretos Multitarefa
STS	State Tree Structure
TCS	Teoria de Controle Supervisório

Capítulo 1

Introdução

1.1 Contextualização

Tradicionalmente, a teoria de controle lidou com o comportamento dinâmico dos sistemas, cujas variáveis são numéricas, contínuas ou descontínuas no tempo, com evolução que pode ser modelada por equações diferenciais ou a diferenças, respectivamente. Nesta tese, estudam-se questões relativas a uma outra classe de sistemas complexos, de grande ocorrência em situações reais e cuja teoria encontra-se ainda em evolução. Estes sistemas, cujos estados deixam de ter apenas valores numéricos e passam a ter valores booleanos e lógicos, não podem ser descritos adequadamente pelos modelos tradicionais [Heymann, 1990]. Neles, as mudanças de estados ocorrem em resposta à ocorrência de eventos discretos, como, por exemplo, o fechamento de um contato. Estes eventos são discretos no tempo, e ocorrem de forma assíncrona. Sistemas deste tipo são classificados como Sistemas a Eventos Discretos (SEDs). Em geral, os SEDs não temporizados são adequados para modelar os aspectos lógicos de qualquer sistema dinâmico [Gohari e Wonham, 2000].

Os SEDs de maior complexidade são formados pela interação de múltiplos subsistemas com evolução simultânea no tempo, sendo chamados de sistemas concorrentes. Cada subsistema de um sistema concorrente possui um comportamento característico bem definido, sendo responsável pela execução de tarefas particulares. Como exemplos de sistemas concorrentes, podem-se citar os sistemas de manufatura. Eles são compostos de diversos subsistemas menores usualmente representando operações concorrentes, tais como montar, transportar e armazenar subprodutos. A operação conjunta destes subsistemas faz com que o sistema global atinja os resultados esperados. O controle é essencial para garantir eficiência, economia e impedir situações indesejáveis. A lógica de controle coordena os subsistemas componentes. O problema de controle para um SED genérico consiste em determinar seu comportamento, para que ele satisfaça a(s) especificação(ões) [Balemi, 1992].

Os controladores lógico programáveis (CLPs) são utilizados na indústria para realizar o controle lógico do sistema. Em geral, o engenheiro responsável pela programação do CLP,

tendo conhecimento da lógica requerida para aquele sistema, programa as seqüências que devem ser executadas de forma a garantir que o comportamento do sistema seja adequado. A lógica de controle, portanto, garante que o funcionamento deverá seguir a seqüência do programa implementado no CLP. Nesse tipo de programação, a construção da trajetória do sistema controlado não está baseada num mapa de todas as trajetórias possíveis que o sistema pode gerar. Ela é, simplesmente, uma das trajetórias que satisfaz a especificação. Esta solução pode estar impedindo outros caminhos que podem também ser satisfatórios, levando a um comportamento mais restritivo do sistema. Este comportamento mais restritivo implica, em muitos casos, um desempenho pior do sistema de controle.

Na literatura, encontram-se vários formalismos para modelagem e controle de SEDs, incluindo máquinas de estados finitos [Ramadge e Wonham, 1983], [Ramadge e Wonham, 1987], [Cassandras e Lafortune, 1999], álgebra de dióides [Baccelli et al., 1992], redes de Petri [Murata, 1989], teoria de filas [Kleinrock, 1975] e cadeias de Markov [Çinclair, 1975], entre outras. Esta variedade de abordagens reflete a diversidade de áreas em que os sistemas a eventos discretos têm papel importante [Ramadge e Wonham, 1983].

Ramadge e Wonham [1987] iniciaram o desenvolvimento da teoria de controle de sistemas a eventos discretos sob o formalismo de máquinas de estados finitos e linguagens formais. A introdução da teoria R&W[1] teve grande impacto na área de controle e gerou um crescente interesse por processos a eventos discretos, o que pode ser evidenciado pelas inúmeras contribuições a esta teoria relatadas na literatura. Na abordagem de Ramadge e Wonham, os sistemas são modelados como reconhecedores de linguagens, onde os alfabetos destas linguagens são formados de símbolos representando a ocorrência de eventos. A linguagem descreve o conjunto de possíveis trajetórias do sistema.

A Teoria de Controle Supervisório (TCS) clássica considera a planta como um bloco único, formado pela composição de subsistemas. O sistema é controlado por meio do supervisor, que observa os eventos ocorridos na planta e atua sobre ela, através da desabilitação de eventos (controláveis) da mesma. Alguns eventos não podem ser desabilitados pelo supervisor, sendo chamados de não-controláveis. O supervisor restringe o conjunto de cadeias possíveis da planta em malha fechada, de tal forma que as cadeias remanescentes satisfaçam as especificações dadas. Há algoritmos que, dada a especificação e o modelo da planta, geram um supervisor que implementa a especificação de forma minimamente restritiva, ou seja, restringindo o comportamento da planta o mínimo possível.

Diversas extensões para a Teoria do Controle Supervisório foram propostas de forma a tratar adequadamente as especificidades de cada tipo de sistema. No controle hierárquico, o sistema é dividido em níveis hierárquicos, onde a informação do sistema é agregada e um modelo abstraído do sistema é obtido para o nível superior. O supervisor é obtido para o nível mais alto e posteriormente traduzido para ser implementado nos níveis inferiores [Wong e Wonham, 1996].

[1] Ramadge e Wonham

O uso do controle modular [Wonham e Ramadge, 1988] e [de Queiroz e Cury, 2002b] se torna vantajoso quando é possível dividir as tarefas em subtarefas. Cada subtarefa é implementada por um supervisor modular. Segundo Rohloff e Lafortune [2003b] o uso de modelos modulares leva a uma grande redução no espaço de armazenamento, o que justifica o interesse no uso de módulos para modelar SEDs.

Para tratar casos em que há restrição de informação, ou seja, o supervisor não observa todos os eventos do sistema, utiliza-se a abordagem descentralizada que, com a introdução do conceito de co-observabilidade, garante que as especificações podem ser implementadas.

No caso de se utilizarem técnicas formais para modelagem e projeto da lógica de controle a ser programada em um CLP tem-se garantias de que o sistema vai sempre atuar dentro das especificações, desde que o modelo seja adequado. A obtenção do controlador baseada em modelos do comportamento livre dos subsistemas e especificações, faz com com que este controlador tenha ações de controle previstas para qualquer combinação de comportamentos dos subsistemas. O modelo do comportamento livre do subsistema, ou seja, todas as seqüências de eventos possíveis para cada subsistema sem controle torna-se, portanto vital, para o sucesso do sistema de controle. Desenvolver um modelo para o sistema, por mais complicado que possa ser do ponto de vista do engenheiro, é menos complicado que obter a seqüência controlada do sistema completo a ser implementada no CLP. Esta reflexão leva à conclusão de que, em geral, a qualidade do controle obtido por técnicas formais deverá ser superior à qualidade do controle baseado apenas na experiência do projetista. Por outro lado, o uso de técnicas formais fica inviabilizado para sistemas de grande porte, devido ao problema da explosão do espaço de estados.

O tamanho do sistema global cresce exponencialmente com o número de componentes. Um sistema formado de 50 subsistemas de 2 estados, por exemplo, terá o modelo global com até 2^{50} estados. Os algoritmos para síntese de supervisores ótimos têm complexidade polinomial no número de estados dos modelos da planta e especificação. Como o modelo da planta global é potencialmente exponencial ao número de componentes, a síntese por intermédio dos algoritmos originais fica inviabilizada para sistemas de grande porte.

Com o objetivo de contornar esse problema da explosão de estados, várias abordagens vêm sendo propostas. Algumas delas usam estruturas de dados eficientes, aumentando a capacidade computacional dos algoritmos [Balemi et al., 1993], [Gunnarsson et al., 1996], [Vahidi, 2004], [Lawesson et al., 2003], [Sztipanovits e Misra, 1996], [Ma e Wonham, 2003], Ma [2004] [Chen e Lin, 2000], [Chen e Lin, 2001b], etc. Outras exploram características estruturais, como simetria entre componentes do sistema [Eyzell e Cury, 2001], [Rohloff e Lafortune, 2003a].

Várias extensões da Teoria do Controle Supervisório vêm sendo desenvolvidas, mas o problema da complexidade computacional ainda persiste. Um tópico fundamental, portanto, é o desafio de se expandir a capacidade de utilização desta teoria, com o objetivo de torná-la viável para aplicação em sistemas reais de grande porte. O Controle Modular Local [de Queiroz e

Cury, 2000] foi desenvolvido com o objetivo de redução da complexidade de síntese dos supervisores. Nesta abordagem, a síntese de cada supervisor é realizada a partir de uma visão local da planta. Desta forma, a planta global não precisa ser obtida, reduzindo a complexidade do procedimento, em geral. No entanto, apesar de permitir reduzir a complexidade da síntese, a verificação de não-conflito permanece um problema de complexidade computacional alta, como será detalhado na próxima seção.

Esta tese trata da verificação eficiente de conflito em sistemas, modelados por máquinas de estados finitos.

1.1.1 Conflito

Nos casos em que há diversos supervisores atuando paralelamente sobre uma planta há a possibilidade dos supervisores implementarem comportamentos contraditórios. Como os supervisores não possuem informação a respeito das especificações implementadas pelos outros supervisores, sua ação conjunta pode gerar conflito. O conflito pode ser caracterizado pela incapacidade do sistema completar tarefas. Mesmo que todos os supervisores sejam não-bloqueantes em relação a (e.r.a) sua planta, a ação conjunta pode ser conflitante. Portanto, uma questão de grande interesse e importância é o problema de verificar se os supervisores são não-conflitantes.

O conflito é uma característica do sistema global e, portanto, sua verificação possui alta complexidade computacional, pois passa pela composição de todos os supervisores utilizados.

Apesar do problema do conflito ser inerente às abordagens que utilizam diversos supervisores executando em paralelo, muitos autores desconsideram sua existência, tratando de linguagens prefixo-fechadas (este conceito é apresentado no Capítulo 2). Esta consideração simplifica bastante o problema a ser resolvido, pois mascara um possível bloqueio do sistema e permite tratar os sistemas localmente, sem analisar o comportamento global. No entanto, com o objetivo de tornar o formalismo de Controle Supervisório de Sistemas a Eventos Discretos aplicável a sistemas reais, faz-se necessário enfrentar a questão.

A idéia deste trabalho de pesquisa surgiu de uma avaliação de que pretende-se tornar o formalismo de Controle Modular Supervisório de Sistemas a Eventos Discretos uma ferramenta prática para problemas de engenharia, faz-se necessário tratar a questão do conflito. Sua existência tem que ser detectada de forma eficiente. Recentemente observa-se um interesse crescente nesse assunto [Abdelwahed e Wonham, 2003a], [Malik, 2004] associado a [Brandin et al., 2000] e [Flordal e Malik, 2006].

Além de detectar o conflito, deve-se ser capaz de evitar sua ocorrência, seja através de abordagens que evitam-no conflito por construção dos supervisores [Krishnan, 2004], [Hill e Tilbury, 2006], seja usando outros métodos, como [Wong et al., 2000], [Leduc et al., 2005a], etc.

Nesta tese apresentam-se contribuições relacionadas à detecção eficiente de conflito entre supervisores. Os resultados apresentados utilizam extensivamente uma propriedade das projeções naturais chamada *propriedade do observador*.

1.1.2 Propriedade do Observador

A propriedade do observador foi introduzida no contexto do controle hierárquico como uma das condições para a obtenção do modelo do nível hierarquicamente superior [Wong e Wonham, 1996], [Wong et al., 2000]. Esta propriedade foi estendida para o contexto das projeções naturais de linguagens, tendo sido apresentada como a condição que garante que o autômato mínimo que representa uma dada projeção possui menos (ou o mesmo número de) estados que o autômato mínimo que reconhece a linguagem original [Wong, 1998]. Como os resultados desta tese baseiam-se em projeções naturais, a utilização deste conceito como uma das condições impostas às abstrações se mostrou natural. As abstrações com a propriedade do observador são chamadas, no contexto deste trabalho, de OP-abstrações.

Na próxima Seção, apresentam-se as contribuições da tese, relacionando-as com artigos publicados, alguns ainda em processo de revisão, durante o período do doutoramento.

1.2 Contribuições da Tese

Neste trabalho, apresentam-se métodos de verificação de não-conflito utilizando abstrações dos supervisores com o objetivo de reduzir a complexidade computacional desta verificação. São obtidas abstrações para o sistema, através da operação de projeção natural e a verificação de não-conflito é aplicada sobre estas abstrações. Dois conjuntos de condições sobre as abstrações são apresentados. Os resultados desta parte do trabalho foram publicados em duas conferências internacionais [Pena et al., 2006c] e [Pena et al., 2006b] e um congresso nacional [Pena et al., 2006a]. Além destes, um artigo [Pena et al., 2006d] está submetido para *IEEE-Transactions on Automatic Control*, já com a primeira revisão realizada, solicitando pequenos ajustes. Este artigo inclui, além dos resultados apresentados nas conferências, três métodos de verificação de não-conflito baseados nos dois tipos de abstrações apresentados.

Além disso, apresenta-se um algoritmo para verificação da propriedade do observador de uma projeção inspirado pelo verificador de diagnosticabilidade apresentado por Yoo e Lafortune [2002a]. Este algoritmo possui complexidade polinomial no número de estados do autômato que representa a linguagem original. Há na literatura trabalhos que apresentam algoritmos de verificação desta propriedade e obtenção de projeções com tal característica. No entanto, as ferramentas disponíveis não possuem tal rotina implementada. Como o grupo do Professor José Cury, orientador da tese, vem desenvolvendo ao longo dos anos uma ferramenta de controle supervisório (GRAIL para Controle Supervisório [Reiser et al., 2006]) sobre o kernel básico da versão 2.5 da distribuição do GRAIL, há o interesse de ter a verificação desta propriedade implementada nesta ferramenta.

Por fim, durante o período do estágio sanduíche, realizado na *University of Michigan* - Ann Arbor, sob supervisão do Professor Stéphane Lafortune, a autora recebeu um convite para visitar o grupo de pesquisa dos Professores Knut Åkesson e Martin Fabian, da *Chalmers University of Technology* - Gotemburgo, Suécia. Há interseção entre o trabalho desenvolvido pelo grupo e os resultados apresentados neste tese. Nesta visita, essa interseção foi explorada, relacionando os resultados apresentados pela autora [Pena et al., 2006c] e por um dos colaboradores do Professor Åkesson [Flordal e Malik, 2006] no WODES'06 (*VIII International Workshop on Discrete Event Systems*), realizado em Ann Arbor, EUA, em julho de 2006. Esta discussão gerou um artigo [Flordal et al., 2007] que foi submetido e aceito para o DCDES (*I IFAC Workshop on Dependable Control of Discrete Systems*), a ser realizado em Paris, em junho de 2007. O assunto deste artigo é apresentado na Seção 6.6.

1.3 Organização do Texto

Após este capítulo introdutório, no Capítulo 2 apresentam-se os conceitos básicos relacionados com a Teoria de Controle Supervisório, além da abordagem Monolítica e Modular Clássica.

Os Capítulos 3 e 4 trazem as principais extensões à Teoria do Controle Supervisório. A leitura do Capítulo 3 é recomendada para aqueles que desejam revisar as principais extensões do controle supervisório. As extensões apresentadas neste capítulo não possuem relação direta com os resultados apresentados nesta tese e, por conseguinte, pode ser omitido em uma primeira leitura. No Capítulo 4 faz-se uma revisão das extensões da Teoria do Controle Supervisório que lidam com os sistemas concorrentes, foco deste trabalho. Além disso, apresentam-se técnicas para detecção de conflito. Nos Capítulos 5, 6 e 7 localizam-se as contribuições da tese.

No Capítulo 5, apresenta-se um conjunto de resultados que estabelece que, utilizando-se um certo conjunto de abstrações das linguagens (cujas características são discutidas) pode-se verificar o não-conflito das linguagens originais. Um procedimento para realização da verificação utilizando tais abstrações é apresentado, além de um exemplo.

Um novo conjunto de abstrações (cujas características são discutidas no Capítulo 6) é apresentado. Estas abstrações utilizam características estruturais das linguagens (segundo a forma como foram obtidas), não podendo ser aplicadas a um conjunto arbitrário de linguagens. Ainda neste capítulo são apresentados dois procedimentos, um deles para aplicação do método apresentado no Capítulo 6 e o outro consiste em uma combinação dos procedimentos apresentados nos dois Capítulos 5 e 6. O mesmo exemplo utilizado para ilustrar o procedimento do Capítulo 5 é utilizado para ilustrar os dois métodos apresentados no Capítulo 6.

O algoritmo para verificação da propriedade do observador é apresentado no Capítulo 7.

Por fim, no Capítulo 8 apresenta-se uma reflexão a respeito dos assuntos abordados, bem como sugestões de continuidade do trabalho.

Capítulo 2

Controle Supervisório de Sistemas a Eventos Discretos

Neste capítulo são introduzidos os principais conceitos da Teoria de Controle Supervisório, desenvolvida por Ramadge e Wonham [1987], [Wonham e Ramadge, 1987], que aborda o problema do Controle de Sistemas a Eventos Discretos. Este texto não é exaustivo, mas apresenta os conceitos que serão utilizados na seqüência do documento. Para uma revisão detalhada do conteúdo que segue, consulte [Wonham, 2004] e Cassandras e Lafortune [1999].

Um sistema a eventos discretos (SED) é um sistema dinâmico que evolui de acordo com a ocorrência abrupta, em intervalos irregulares e possivelmente desconhecidos, de eventos físicos. Exemplos típicos deste tipo de sistemas são: sistemas de manufatura, redes de comunicação e de computadores, robótica, logística, etc. Estas aplicações necessitam de controle e coordenação para garantir o fluxo correto dos eventos [Ramadge e Wonham, 1989].

Para Cassandras e Lafortune [1999], um sistema a eventos discretos é um sistema que pode ser descrito por um conjunto de estados e suas transições de estados são observadas em pontos discretos no tempo. Às transições de estados associam-se "eventos". Um evento pode ser identificado como uma ação específica realizada (por exemplo, acionamento de um contato) ou uma ocorrência espontânea relacionada ao ambiente externo (por exemplo, desligamento espontâneo de um computador por alguma razão desconhecida), ou ainda o resultado de um conjunto de condições simultaneamente atendidas (o nível de um fluido em um tanque ultrapassa certo valor).

Da forma como o SED é modelado, não importa o tempo de ocorrência dos eventos, mas a ordem em que eles ocorrem. A ocorrência simultânea de dois ou mais eventos não é admitida.

Este capítulo é formado por três seções. Inicialmente, apresentam-se os principais conceitos da teoria de linguagens controláveis, introduzida por Ramadge e Wonham, para o controle de sistemas a eventos discretos, Seção 2.1. Na Seção 2.2 define-se o controle supervisório e apresenta-se o Controle Supervisório Monolítico e Modular Clássico. A Seção 2.4 apresenta uma breve discussão.

2.1 Teoria de Linguagens

Antes de falar sobre as estruturas de controle supervisório de sistemas a eventos discretos, apresenta-se um resumo de definições e conceitos necessários para representar os sistemas a eventos discretos.

2.1.1 Linguagens

Seja Σ um conjunto finito de símbolos distintos. Σ é chamado de alfabeto e Σ^* é o conjunto de todas as cadeias finitas da forma $\sigma_1\sigma_2\cdots\sigma_k$, com $\sigma_i \in \Sigma$ e $k \geq 1$, que podem ser geradas a partir de Σ, incluindo a cadeia vazia ϵ. Um elemento de Σ^* é uma cadeia ou palavra sobre o alfabeto Σ. A cadeia $u \in \Sigma^*$ é prefixo de $v \in \Sigma^*$ se existe pelo menos uma cadeia $s \in \Sigma^*$ tal que $v = us$.

Uma linguagem sobre Σ é um subconjunto de Σ^*.

2.1.2 Autômatos e Geradores

Toda linguagem regular, que pode ser representada por uma expressão regular, pode ser gerada por um autômato. Se um sistema a eventos discretos pode ser descrito por uma linguagem regular, esta, por sua vez, pode ser gerada por um dispositivo chamado autômato. Neste contexto, a definição de autômatos é feita a seguir.

O autômato A sobre o alfabeto Σ é definido por uma quíntupla $A = (Q, \Sigma, \hat{\delta}, q_0, Q_m)$ tal que:

- $Q \mapsto$ conjunto não-vazio de estados,
- $\Sigma \mapsto$ alfabeto de eventos,
- $q_0 \in Q \mapsto$ estado inicial do autômato,
- $Q_m \subseteq Q \mapsto$ conjunto de estados finais ou marcados,
- $\hat{\delta} : Q \times \Sigma \to Q \mapsto$ função de transição definida em todo estado para todo o conjunto de eventos de Σ.

A função de transição $\hat{\delta}$ pode ser estendida para cadeias de eventos como a função $\delta : Q \times \Sigma^* \to Q$ tal que $q \in Q$, $s \in \Sigma^*$ e $\sigma \in \Sigma$,

$$\begin{aligned} \delta(q, \epsilon) &= q; \\ \delta(q, s\sigma) &= \delta(\hat{\delta}(q, s), \sigma). \end{aligned}$$

Esta função mapeia para qual estado de Q o sistema transita a partir de um estado de Q com a ocorrência de uma seqüência $s \in \Sigma^*$.

A representação dos autômatos pode ser feita através de grafos dirigidos ou diagramas de transição de estados onde os nós representam os estados e os arcos representam os eventos. Estados representados com dois círculos co-cêntricos são os estados marcados e o estado inicial é identificado com uma seta.

O autômato $A = (Q, \Sigma, \delta, q_0, Q_m)$ reconhece a linguagem $\mathcal{L}_m(A) = \{s \in \Sigma^* | \, \delta(q_0, s) \in Q_m\}$.

Na abordagem R&W de linguagens controláveis, a modelagem do comportamento do SED é feita através de um gerador. O gerador é um autômato $A_G = (Q, \Sigma, \delta, q_0, Q_m)$ cuja função de transição δ é parcial ou seja, está definida em cada estado de Q apenas para um subconjunto de Σ.

No caso do gerador, a função de transição δ pode ser estendida para cadeias de eventos como a função $\delta : Q \times \Sigma^* \to Q$ tal que $q \in Q$, $s \in \Sigma^*$ e $\sigma \in \Sigma$,

$$\begin{aligned} \delta(q, \epsilon) &= q; \\ \delta(q, s\delta) &= \delta(\delta(q, s), \sigma), \end{aligned}$$

sempre que $q' = \delta(q, s)$ e $\delta(q', \sigma)$ estiverem definidas.

O gerador A_G é interpretado como um dispositivo que começa no estado inicial q_0 e executa transições de estado, gerando uma seqüência de eventos, seguindo seu grafo. As transições ocorrem espontaneamente, de forma assíncrona e instantânea e sua ocorrência é sinalizada pelo evento correspondente, $\alpha \in \Sigma$. Sob controle, a ocorrência de alguns eventos pode ser evitada. Assim, o conjunto de eventos é particionado em eventos controláveis e não-controláveis ($\Sigma = \Sigma_c \cup \Sigma_{uc}$). Os eventos em Σ_c podem ser desabilitados enquanto os eventos de Σ_{uc} não sofrem influência do controle ou seja, são considerados permanentemente habilitados. Sinais de comando para a planta são modelados como eventos controláveis e sinais de resposta da planta são modelados como eventos não-controláveis. Em alguns exemplos deste documento, os eventos são rotulados com números inteiros positivos sendo que os números ímpares representam eventos controláveis e os pares representam eventos não-controláveis.

O gerador A_G é associado a duas linguagens, a linguagem gerada $\mathcal{L}(A_G) := \{s \in \Sigma^* | \, \delta(q_0, s)$ é definida$\}$ e a linguagem marcada $\mathcal{L}_m(A_G) = \{s \in \Sigma^* | \, \delta(q_0, s) \in Q_m\}$. A linguagem marcada é um subconjunto da linguagem gerada composto das cadeias que terminam em estados marcados. O autômato é um caso particular do gerador, onde a função de transição é completa.

A linguagem gerada por uma planta G é o conjunto de cadeias que podem ser geradas pela planta. A linguagem marcada é interpretada como o conjunto de cadeias que representam tarefas completas para a planta. A linguagem da planta, $\mathcal{L}(G)$, é prefixo-fechada (propriedade

de linguagens definida na subseção 2.1.3), pois, para que o sistema produza uma cadeia, pode-se afirmar que ele produziu todos os seus prefixos anteriormente.

Neste trabalho, apenas geradores são usados. No entanto, não se faz essa distinção e refere-se a eles como autômatos.

2.1.3 Operações sobre linguagens e geradores

O prefixo fechamento de uma linguagem L é a linguagem

$$\overline{L} = \{u \,|\, \exists v \in \Sigma^* \text{ tal que } uv \in L\},$$

que consiste no conjunto de prefixos das cadeias de L. Uma linguagem é dita fechada (ou prefixo-fechada) se $L = \overline{L}$.

A projeção natural $P_i(L)$ é uma operação realizada sobre linguagens definida a partir de dois conjuntos de eventos, Σ e Σ_i, com $\Sigma_i \subseteq \Sigma$ que apaga os eventos de $(\Sigma - \Sigma_i)$ das cadeias de L. A projeção inversa $P_i^{-1}(L)$ completa o alfabeto de L com os eventos de $(\Sigma - \Sigma_i)$, colocando-os em auto-laços.

O produto síncrono (ou composição síncrona) de linguagens $L_i \subseteq \Sigma_i^*$, com $i = \{1, \cdots, n\}$, é definido como:

$$\mathop{\Big\|}_{i=1}^{n} L_i = \bigcap_{i=1}^{n} P_i^{-1}(L_i) \subseteq \Sigma^*,$$

onde $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$

O produto síncrono é uma operação comutativa e associativa. Assim, o produto de múltiplos autômatos pode ser obtido pelo produto síncrono de autômatos dois a dois.

No contexto da TCS, a composição síncrona de geradores $G_i = (Q_i, \Sigma_i, \delta_i, q_{0i}, Q_{mi})$, com $i = \{1, \cdots, n\}$ é obtida fazendo-se a evolução em paralelo dos n geradores. Um evento compartilhado por mais de um gerador só é executado quando estiver habilitado em todos os geradores que o contêm.

A composição de plantas modeladas por G_i gera uma planta global G definida sobre o alfabeto $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$. Os comportamentos gerado e marcado por G são dados, respectivamente, por:

$$\mathcal{L}(G) = \mathop{\Big\|}_{i=1}^{n} \mathcal{L}(G_i)$$

$$\mathcal{L}_m(G) = \mathop{\Big\|}_{i=1}^{n} \mathcal{L}_m(G_i).$$

No pior caso, alfabetos Σ_i assíncronos, o número de estados de G cresce exponencialmente com o valor de n.

O SED é dito acessível se todos os seus estados forem acessíveis. Um estado $q \in Q$ é dito acessível se existe $s \in \Sigma^*$ tal que $\delta(q_0, s) = q$, ou seja, se existir pelo menos uma cadeia que, a partir do estado inicial, leve ao estado q.

O bloqueio de um SED está associado à impossibilidade de completar uma tarefa (atingir uma cadeia marcada) a partir de um dado estado. Quando isto acontece, nem todos os estados acessíveis do SED são co-acessíveis e o sistema é dito bloqueante. Um estado $q \in Q$ de um gerador é dito co-acessível se existe $s \in \Sigma^*$ tal que $\delta(q, s) \in Q_m$ ou seja, se existe uma cadeia a partir de $q \in Q$ que leve o gerador a um estado marcado $q \in Q_m$. Um SED é co-acessível se todos os seus estados são co-acessíveis.

Um SED é dito aparado se for acessível e co-acessível.

2.2 Controle Supervisório de SEDs

Seja $G = (Q, \Sigma, \delta, q_o, Q_m)$ um SED (não-vazio) controlado, onde $\Sigma = \Sigma_c \cup \Sigma_{uc}$. Um subconjunto particular de eventos a serem habilitados pode ser selecionado pela especificação de um conjunto de eventos controláveis. É conveniente adicionar a esse conjunto todos os eventos não-controláveis, já que esses estão automaticamente habilitados na planta. Cada um desses subconjuntos de eventos é chamado de padrão de controle. Em seguida, introduz-se o conjunto de todos os padrões de controle:

$$\Gamma = \{\gamma \in 2^{\Sigma} | \gamma \supseteq \Sigma_{uc}\}.$$

Um controle supervisório para G é qualquer mapa $V : \mathcal{L}(G) \rightarrow \Gamma$, ou seja, qualquer mapa que associa cadeias da planta à elementos de Γ. O par (G, V) pode ser escrito como V/G para denotar: *G sob supervisão de V*. O comportamento em malha fechada de V/G é definido como a linguagem $\mathcal{L}(V/G) \subseteq \mathcal{L}(G)$ tal que:

i. $\epsilon \in \mathcal{L}(G)$;

ii. Se $s \in \mathcal{L}(V/G), \sigma \in V(s)$ e $s\sigma \in \mathcal{L}(G)$ então $s\sigma \in \mathcal{L}(V/G)$;

iii. Nenhuma outra cadeia pertence a $\mathcal{L}(V/G)$.

O conjunto $\{\epsilon\} \subseteq \mathcal{L}(V/G) \subseteq \mathcal{L}(G)$ e, portanto, $\mathcal{L}(V/G)$ é não-vazia. O comportamento marcado de V/G é $\mathcal{L}_m(V/G) = \mathcal{L}(V/G) \cap \mathcal{L}_m(G)$. Portanto, o comportamento marcado de V/G consiste das cadeias marcadas (de $\mathcal{L}_m(G)$) que sobrevivem à supervisão de V. Neste trabalho, a entidade "supervisor"é representada por S.

Pode-se dizer que uma dada linguagem $K \subseteq \Sigma^*$ é controlável em relação a (e.r.a) $\mathcal{L}(G)$ se:

$$\overline{K}\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}$$

Esta propriedade garante que o supervisor que implementa a linguagem K não vai tentar desabilitar eventos não-controláveis na planta. A classe de linguagens controláveis contidas numa linguagem E é denotada por $\mathcal{C}(E, \mathcal{L}(G)) = \{K | K \subseteq E$ e K é controlável e.r.a $\mathcal{L}(G)\}$. Este conjunto é fechado para união, o que implica que ele tem um elemento supremo, chamado de $Sup\mathcal{C}(E, \mathcal{L}(G))$, que é a máxima sublinguagem controlável contida em K.

Uma linguagem é $\mathcal{L}_m$-fechada se $K = \overline{K} \cap \mathcal{L}_m(G)$. Isto significa que todos os prefixos de K que forem cadeias de $\mathcal{L}_m(G)$ também são cadeias de K.

2.2.1 Controle Monolítico

O controle supervisório visa controlar a planta (geradora de uma linguagem). Nesta abordagem isto significa que ao adicionar uma estrutura de controle é possível variar a linguagem gerada pelo sistema controlado dentro de certos limites [Wonham, 2004]. O objetivo de controle deve ser representado através de uma linguagem chamada especificação. A partir desta especificação é obtida a linguagem desejada para o sistema em malha fechada. Dada a linguagem desejada para o sistema, pode ser obtido um supervisor cuja função é desabilitar eventos controláveis da planta.

Seja um sistema composto de subplantas G_i, com $i = \{1, \cdots, n\}$. Pela abordagem monolítica, a planta G a ser controlada deve ser única, e pode ser obtida pela composição das subplantas G_i. Essa planta deve ser restringida pelas especificações E_j, com $j = \{1, \cdots, m\}$. De forma análoga, a especificação E deve ser única, e pode ser obtida pela composição das especificações E_j dadas.

O procedimento para projetar um supervisor (controlador) através da abordagem monolítica é resumido a seguir.

1. Obtenção da planta global G pela composição de G_i, para $i = \{1, \cdots, n\}$.
2. Obtenção da especificação monolítica E pela composição das especificações E_j, com $j = \{1, \cdots, m\}$.
3. Obtenção da linguagem desejada K para o sistema em malha fechada pela composição de E e $\mathcal{L}_m(G)$, $(K = E||\mathcal{L}_m(G))$ e obtenção da componente aparada.
4. Obtenção do supervisor S, cuja linguagem $\mathcal{L}(S)$ é a máxima sublinguagem controlável contida em K.

Uma fase do procedimento que não foi enumerada acima é a modelagem das subplantas e especificações. Esta fase é imprescindível para o sucesso do projeto. Entretanto, considera-se o procedimento de modelagem dos subsistemas componentes e especificações como já resolvido.

Dada uma linguagem $K \subseteq \mathcal{L}_m(G)$, não vazia, representando a linguagem desejada para o sistema G, existe um supervisor próprio S que implementa K se e somente se:

- K é $\mathcal{L}_m$-fechada;

- K é controlável.

No caso de K não satisfazer as condições necessárias para a síntese monolítica, é possível projetar um supervisor próprio S que implemente a máxima sublinguagem controlável contida em K. Neste caso, $S = Sup\mathcal{C}(K, \mathcal{L}(G))$.

No contexto desse trabalho, S poderá também aparecer representando o autômato que implementa a linguagem $Sup\mathcal{C}(K, G)$. Quando se fizer necessário, a distinção entre o autômato e a linguagem implementada pelo autômato será feita. O mesmo se aplica às linguagens K e E que poderão aparecer nomeando os autômatos que implementaiiim suas linguagens.

O exemplo abaixo ilustra a metodologia da síntese monolítica.

Exemplo 2.2.1. *Sejam os subsistemas G_1, G_2, G_3 e G_4 cujos eventos são todos controláveis e os autômatos E_1, E_2 e E_3 representando as especificações, apresentados na Figura 2.1.*

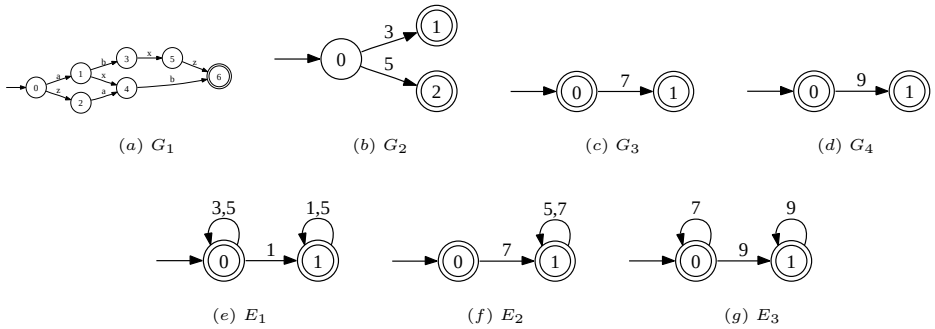


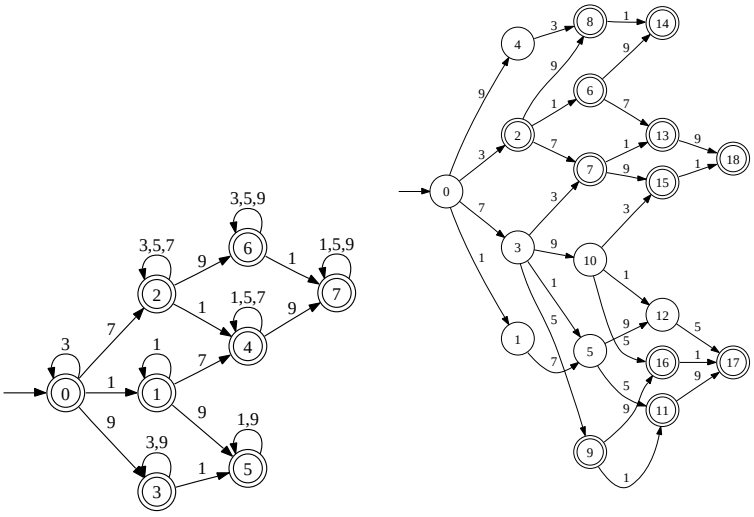
Figura 2.1: Exemplo 2.2.1: (a)-(d) Subplantas (e)-(g) Especificações

A planta global é obtida pela composição síncrona das subplantas G_i, $G = G_1||G_2||G_3||\ G_4$. A especificação global é obtida pela composição síncrona das especificações $E = E_1||E_2||E_3$ e é apresentada na Figura 2.2(a). A linguagem desejada para o sistema é obtida fazendo $K = E||\mathcal{L}_m(G)$. Como todos os eventos são controláveis, tem-se que K é controlável. Portanto, a linguagem da planta global sobre ação do supervisor monolítico é apresentada na Figura 2.2(b).

Caso K não fosse controlável, um supervisor seria calculado como sendo a máxima sublinguagem controlável contida em K, $S = Sup\mathcal{C}(K, \mathcal{L}(G))$. O algoritmo para cálculo da máxima linguagem controlável é polinomial no produto do número de estados da planta e especificação [Wonham e Ramadge, 1987]. Este procedimento não é discutido nesta tese, mas detalhes podem ser encontrados em [Cury, 2001].

2.2.2 Controle Modular Clássico

O controle modular, proposto por Wonham e Ramadge [1988] é uma extensão do controle monolítico que visa facilitar a síntese de supervisores e aumentar a flexibilidade. Enquanto



(a) E (b) S

Figura 2.2: Exemplo 2.2.1: (a) Especificação global $E = E_1||E_2||E_3$; (b) Supervisor monolítico S e sua linguagem $S = K$

no Controle Monolítico projeta-se apenas um supervisor que deve restringir o comportamento da planta de forma que o sistema em malha fechada obedeça a um conjunto de restrições, no Controle Modular Clássico a tarefa de fazer com que o sistema em malha fechada obedeça ao conjunto de restrições é dividida em subtarefas que são realizadas por diferentes supervisores.

O grande interesse da comunidade de sistemas a eventos discretos na modelagem de sistemas de forma modular se justifica na constatação de que algumas plantas são muito complicadas para serem modeladas como um sistema a eventos discretos monolítico, mas que podem ser facilmente modeladas como conjuntos de subsistemas que interagem entre si. A manipulação de plantas como agentes separados evita o problema de explosão de estados na modelagem. Quando sistemas de estados finitos são compostos, o tamanho do espaço de estados é exponencial no número de componentes a serem compostos, no pior caso. Da mesma forma, pode ser que haja diversas especificações e pode ser vantajoso tratar as especificações de forma modular.

A combinação da ação dos supervisores soluciona o problema original. A ação conjunta dos supervisores desabilita eventos que forem desabilitados por pelo menos um supervisor. Esta construção é chamada de síntese modular. A síntese modular propicia uma maior flexibilidade sendo mais fácil atualizar, modificar e corrigir supervisores que na síntese monolítica, já que é necessário apenas modificar a especificação correspondente e recalcular o supervisor relativo àquela especificação. Por outro lado, a ação dos diferentes supervisores, que têm apenas uma visão local do sistema, pode ser conflitante.

Muitas vezes a solução modular é degradada em relação à solução monolítica. Isto ocorre porque cada supervisor tem sua ação de controle baseada no funcionamento parcial da planta (um supervisor não tem informação a respeito da ação de controle aplicada pelos outros

supervisores). Esta falta de informação pode causar um conflito entre os supervisores, quando os supervisores em conjunto desabilitam todos os eventos que tornam possível alcançar estados marcados a partir de um determinado estado do sistema. Neste caso o sistema global bloqueia e os supervisores são ditos conflitantes. Se não há conflito entre os supervisores, tem-se que a ação dos supervisores modulares é igual à ação de um supervisor monolítico projetado para o sistema. Para garantir que a ação conjunta de todos supervisores seja igual à ação de um supervisor monolítico deve-se aplicar o teste de não-conflito ou teste da modularidade.

O teste da modularidade consiste em testar se todo prefixo de uma cadeia marcada nos supervisores é também prefixo de pelo menos uma cadeia da interseção das linguagens dos supervisores, ou seja, testar se:

$$\bigcap_{j=1}^{m} \overline{S_j} = \overline{\bigcap_{j=1}^{m} S_j}$$

Portanto, seja um sistema modelado por n subplantas G_i, com $i = \{1, \cdots, n\}$ que deve ser restringido pelas especificações E_j, com $j = \{1, \cdots, m\}$. O procedimento para projetar os supervisores através da abordagem modular é resumido a seguir.

1. Obtenção da planta global G pela composição de G_i, $\forall i$.
2. Obtenção das linguagens desejadas K_j, $K_j = E_j||\mathcal{L}(G)$ e obtenção das componentes aparadas $\forall j$.
3. Obtenção dos supervisores que implementam as máximas linguagens controláveis contidas em K_j, $S_j = Sup\mathcal{C}(K_j, \mathcal{L}(G))$, $\forall j$.
4. Teste da modularidade entre os supervisores obtidos.

Os supervisores são obtidos como no controle monolítico, ou seja, para cada linguagem-alvo K_j, obtém-se $S_j = Sup\mathcal{C}(K_j, \mathcal{L}(G))$, $j = \{1, \cdots, m\}$.

Um exemplo em que os supervisores obtidos pela síntese modular são conflitantes é apresentado a seguir.

Exemplo 2.2.2. *O mesmo exemplo apresentado na seção anterior é agora resolvido usando a abordagem modular. Sejam os subsistemas G_1, G_2, G_3 e G_4, cujos eventos são todos controláveis, e os autômatos E_1, E_2 e E_3 que representam as especificações, apresentados na Figura 2.1.*

Nesta abordagem um supervisor é obtido para cada especificação. As linguagens desejadas para a planta são obtidas fazendo $K_1 = E_1||\mathcal{L}_m(G)$, $K_2 = E_2||\mathcal{L}_m(G)$ e $K_3 = E_3||\mathcal{L}_m(G)$. K_1, K_2 e K_3 são controláveis e.r.a G e os supervisores S_1, S_2 e S_3 implementam as linguagens desejadas para G. Para saber se a síntese modular é não-conflitante e ótima, ou seja, tem o mesmo desempenho do supervisor monolítico, deve-se aplicar o teste da modularidade.

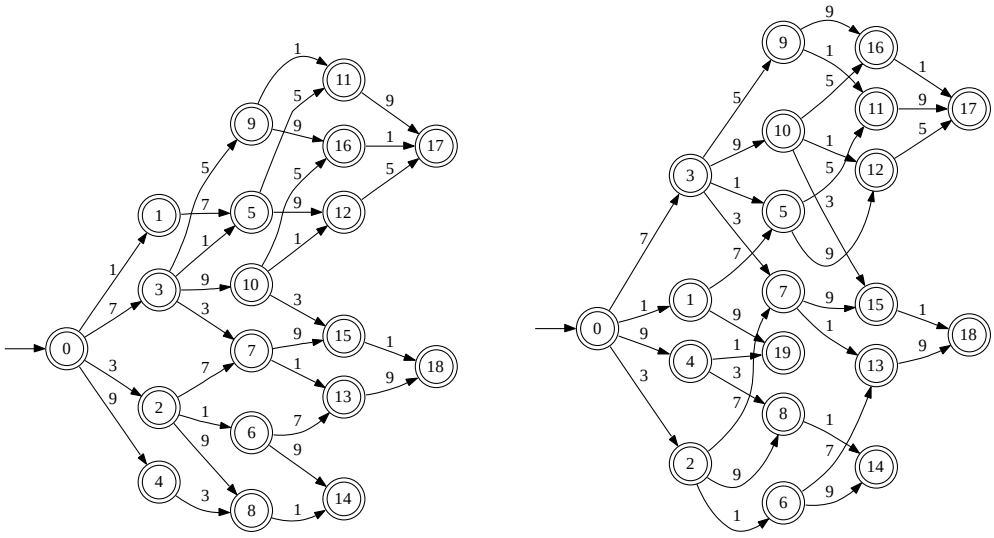


Figura 2.3: Teste da modularidade: (a) $\overline{S_1 \cap S_2 \cap S_3}$ (b) $\overline{S_1} \cap \overline{S_2} \cap \overline{S_3}$;

O teste da modularidade consiste em obter as linguagens $\overline{S_1 \cap S_2 \cap S_3}$ *(Figura 2.3(a)) e* $\overline{S_1} \cap \overline{S_2} \cap \overline{S_3}$ *(Figura 2.3(b)) e verificar sua igualdade. Se essa igualdade não for verificada, conclui-se que a ação conjunta dos supervisores é conflitante.*

Pode-se observar que os geradores representados nas Figuras 2.3(a) e 2.3(b) não são isomorfos. O estado 19 do gerador da Figura 2.3(b), alcançado pela execução das cadeias 1 9 *e/ou* 9 1*, não aparece no gerador da Figura 2.3(a)), o que implica que os supervisores não são modulares.*

Segundo Gohari e Wonham [2000], a complexidade computacional envolvida na solução do problema do controle supervisório pela abordagem monolítica não é reduzida pela utilização da abordagem modular. Ou seja, o problema da explosão de estados é inevitável, em geral. Esforços para reduzir a complexidade computacional do PCS[1] devem ser feitos no sentido de identificar casos especiais. Para esses casos, pode ser possível obter algoritmos eficientes.

2.3 Extensões do Controle Supervisório de Sistemas a Eventos Discretos

Diversas extensões do Controle Supervisório de Sistemas a Eventos vêm sendo propostas na literatura. Pode-se considerar que o PCS se divide em dois grandes grupos: (*i*) controle supervisório com observação total e (*ii*) controle supervisório com observação parcial. No controle supervisório com observação parcial considera-se o problema em que nem todos os

[1]Problema do Controle Supervisório

eventos gerados pela planta podem ser observados pelo supervisor, ou seja, o supervisor deve tomar suas decisões de controle baseado em uma visão incompleta de sua planta.

Uma outra característica que pode ser considerada para classificar os problemas é quanto à arquitetura, ou seja, quanto à forma em que o problema de controle é dividido no procedimento de obtenção da solução. Duas arquiteturas principais são reconhecidas: a arquitetura vertical e a arquitetura horizontal. Na arquitetura vertical, o problema de controle é dividido em níveis, onde apenas um determinado nível é envolvido na síntese de um supervisor para aquele nível. Na arquitetura horizontal, o problema de controle é dividido em subproblemas, onde cada supervisor é responsável pela solução de um subproblema.

A abordagem hierárquica pode ser classificada como sendo de arquitetura vertical. Esta abordagem trata de dividir o problema em dois níveis hierárquicos distintos e foi desenvolvida para lidar com sistemas em que há observação total dos eventos, no nível mais baixo da hierarquia. Várias extensões do controle hierárquico vêm sendo desenvolvidas e são citadas na Seção 3.1.

No caso da arquitetura horizontal, há abordagens que lidam com o problema de observação parcial e outras que lidam com observação total de eventos. A planta pode ser tratada como sendo única e com observação total, como no controle modular clássico (apresentado na Seção 2.2.2), composta de subplantas e com observação total, como no controle modular local (apresentado na Seção 4.2), ou composta de subplantas e com observação parcial como no controle descentralizado (apresentado na Seção 3.2). Nesta arquitetura, a propriedade do sistema global controlado ser não-bloqueante deve ser garantida. Mais recentemente, a nomenclatura "descentralizado" vem sendo utilizada para denotar tanto problemas em que os sistemas são concorrentes ou modulares (com observação completa) quanto problemas em que se tem observação parcial dos eventos [Wonham, 2006], [Wonham, 2005].

No presente trabalho mantém-se a nomenclatura anteriormente utilizada. Nela, distingue-se controle descentralizado do controle modular da forma descrita a seguir:

- no controle descentralizado cada supervisor pode ter acesso a apenas parte da informação e desabilitar apenas parte dos eventos controláveis, e no controle modular não há restrição de informação;
- no controle descentralizado, a descentralização é uma "exigência" do sistema e no controle modular a estrutura descentralizada é uma consequência do procedimento de modelagem.

2.4 Síntese

Neste capítulo foi apresentada uma breve revisão da teoria de linguagens controláveis, necessária para o entendimento do restante do documento. Apresentou-se também o Controle

Supervisório Monolítico e Modular Clássico. Foram apresentados pequenos exemplos para ilustrar a metodologia de síntese de supervisores pelas diversas abordagens. Além destas, uma breve discussão a respeito das principais extensões do controle supervisório de sistemas a eventos discretos foi apresentada.

O controle hierárquico e descentralizado, assim como algumas outras extensões são apresentadas no Capítulo 3. Considerando que a classe de sistemas que pode se beneficiar diretamente dos resultados apresentados nesta tese é a classe dos sistemas compostos de mais de uma subplanta, com observação total de eventos, os trabalhos que lidam com esta classe de sistemas são reunidos no Capítulo 4.

Capítulo 3

Controle Hierárquico, Controle Descentralizado e Outras Extensões

O presente capítulo apresenta uma revisão das principais extensões, excetuando-se as extensões para controle de sistemas concorrentes, apresentadas no Capítulo 4. Os principais resultados do controle hierárquico e controle descentralizado são apresentados. Na Seção 3.3 apresenta-se um conjunto de trabalhos que faz uso de estruturas eficientes de representação e/ou arquiteturas que explorem aspectos estruturais do sistema visando reduzir a complexidade associada à solução do problema de controle. Além destas, algumas outras extensões do Controle Supervisório são apresentadas na Seção 3.4. Na Seção 3.5 é realizada uma breve discussão.

3.1 Controle Hierárquico

Por controle hierárquico entende-se a subdivisão de um problema de controle complexo em problemas associados a níveis de abstração diferentes (gerência e operacional), onde o nível de abstração cresce, quanto mais se sobe na hierarquia. Apesar desta arquitetura ser usualmente justificada pela redução da complexidade, isto nem sempre se verifica pois a obtenção da abstração do modelo operacional para um modelo do gerente pode ser complexa.

Zhong e Wonham [1990] introduzem o controle hierárquico na abordagem R&W. Trata-se de uma hierarquia de dois níveis, consistindo de uma planta e um controlador no baixo nível e uma planta e um controlador no alto nível, como apresentado na Figura 3.1.

A estrutura pode ser apresentada como uma metáfora de uma fábrica onde o alto nível estaria associado ao gerente e o baixo nível ao operador. Os comportamentos dos sistemas são descritos por linguagens prefixo-fechadas e existe uma estrutura de controle, segundo a abordagem R&W, nos dois níveis. G_{op} representa a planta real a ser controlada por C_{op} e G_{ge} é um modelo simplificado de G_{op}. C_{ge} é o supervisor projetado para G_{ge}, utilizando

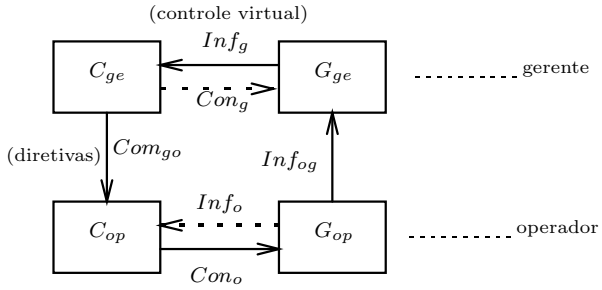


Figura 3.1: Arquitetura Hierárquica

informação do canal Inf_g. A malha fechada para o gerente é representada por C_{ge}/G_{ge}. O controle que o supervisor no nível gerencial exerce é virtual isto é, não existe o canal de comando Con_g, representado pela linha tracejada. O controle é exercido pelo supervisor do operador C_{op} utilizando os canais de controle Con_o, de informação Inf_o e comando Com_{go}. A função do canal de comando Com_{go} é traduzir as diretivas do gerente em vigor no momento, em sequências de controle para o operador. Existe ainda o canal de informação Inf_{og} que envia informações da evolução da planta de baixo nível para a planta de alto nível e é chamado de mapa-repórter.

A principal propriedade de uma estrutura hierárquica é que o modelo de controle disponível em qualquer nível da hierarquia possa ser utilizado com a garantia de que o nível imediatamente inferior vai responder como desejado ou esperado. Esta propriedade é denominada consistência hierárquica.

O problema do controle hierárquico consiste na determinação do modelo do gerente com refinamento suficiente para garantir a consistência hierárquica, ou seja, que o controle exercido pelo operador faça com que a malha fechada do operador C_{op}/G_{op}, ao ser reportada ao gerente, $Inf_{og}(C_{op}/G_{op})$, seja equivalente à malha fechada virtual C_{ge}/G_{ge}. Isto é feito pelo cálculo do canal de informação Inf_{og}.

O canal de informação (ou mapa-repórter) tem a função de, com base em uma sequência de eventos gerada pela planta de baixo nível, gerar uma sequência de eventos de alto nível. A cada novo evento gerado pelo observador, o mapa repórter pode silenciar ou informar ao gerente a ocorrência de um novo evento do alfabeto do gerente. O canal de informação Inf_{og} pode ser modelado como $\theta : \mathcal{L}(G_{op}) \rightarrow T^*$, onde T é o alfabeto do gerente não necessariamente contido e possivelmente diferente do alfabeto do operador, com as seguintes propriedades:

$$\theta(\epsilon) = \epsilon$$
$$\theta(s\sigma) = \begin{cases} \theta(s) & \text{ou} \\ \theta(s)\tau & \text{para algum } \tau \in T \end{cases}$$

O comportamento do operador em conjunto com o canal de informação é modelado por um autômato de Moore. O modelo do gerente é obtido como imagem da linguagem do operador

através do canal de informação e a estrutura de controle de G_{ge} deve ter consistência de controle.

A consistência de controle garante que a estrutura de controle no nível gerencial tenha uma estrutura natural de controle. Ou seja, um par (G_{op}, θ) tem consistência de controle se o alfabeto T do gerente puder ser particionado em um subconjunto de eventos controláveis e outro de eventos não-controláveis.

Caso a consistência de controle não seja verificada, Zhong e Wonham [1990] propõem um refinamento de G_{op} para obter uma estrutura com consistência de controle no nível gerencial. Este refinamento é feito pela modificação da estrutura de transição de G_{op} e do alfabeto do gerente (T), estendendo-o de forma a ser possível particioná-lo em um subconjunto de eventos controláveis e outro de eventos não-controláveis. Para isto, usa-se a árvore de alcançabilidade de $\mathcal{L}(G_{op})$ [Zhong e Wonham, 1990]. O refinamento é feito em G_{op} de forma que a consistência de controle no nível gerencial seja verificada. Ou seja, se um evento de T for não-controlável, todos os seus trechos silenciosos (que não geram eventos no nível do gerente) serão também não-controláveis. Equivalentemente, se um evento de T for controlável, todos os seus trechos silenciosos também serão controláveis. A partir da árvore de alcançabilidade e de um mapa repórter estendido, o novo G_{ge}, com consistência de controle, é obtido.

O modelo, da forma como foi proposto, considera apenas o comportamento gerado dos sistemas. Wong e Wonham [1996] apresentam uma generalização do trabalho de Zhong e Wonham [1990] que considera a marcação do sistema. Este trabalho introduz o conceito de consistência hierárquica forte. A consistência hierárquica forte garante que todas as linguagens controláveis para o operador são mapeadas em linguagens controláveis no nível do gerente e todas as linguagens controláveis no nível do gerente são imagens de linguagens controláveis para o operador. As condições para que se possa garantir consistência hierárquica forte são:

- consistência de controle estrita;
- consistência de marcação;
- mapa repórter observador.

Schmidt [2005] apresenta uma abordagem hierárquica onde o nível hierarquicamente superior é obtido pela projeção natural do autômato do baixo nível. Em seguida, verifica-se a propriedade chamada pelo autor de "aceitação de cadeia marcada"[1] e a condição de ser localmente não-bloqueante. Essas duas propriedades correspondem à propriedade do observador [Wong e Wonham, 1996], posteriormente discutida e utilizada nesta tese. Pode-se perceber que o autor utiliza projeção natural ao invés da idéia do mapa-repórter na construção do autômato do alto nível. Schmidt [2005] apresenta as condições sob as quais o supervisor correspondente do baixo nível é não-bloqueante. Esta abordagem é então estendida para o caso de sistemas compostos em Schmidt e Moor [2006],

[1] do inglês *marked string acceptance*

Outras extensões foram propostas para o modelo hierárquico. Cunha e Cury [2007] propõem um método construtivo para controle hierárquico de SEDs, baseado nos SEDs com marcação flexível, uma nova classe de SEDs dotados de controle, onde garante-se a consistência hierárquica (propriamente dita e a forte) por construção. O SED com marcação flexível foi originalmente apresentado em [Cury et al., 2001]. Cury et al. [2004] apresentam uma abordagem baseada em agregação de estados em que o modelo para o alto nível é baseado nos SEDs com marcação flexível, de forma que o modelo hierárquico precise de refinamento mínimo para alcançar a propriedade de consistência hierárquica forte.

3.2 Controle Descentralizado

O controle descentralizado de SEDs [Lin e Wonham, 1991] surgiu pela necessidade de tratar de sistemas de forma descentralizada. Esta necessidade pode ser causada tanto pela natureza física descentralizada do sistema e dos equipamentos como também pela necessidade de se tratar o problema em que há restrição de informação. No caso da restrição de informação, podem ocorrer dois tipos de problemas: aquele em que realmente existem eventos que não podem ser observados pelo sistema de controle e aquele em que todos os eventos são observáveis, mas existe restrição de informação para os supervisores (por exemplo, supervisores que só podem observar e atuar em subconjuntos de eventos do sistema).

Em muitas aplicações, o controle descentralizado é mais adequado que o controle centralizado. Por exemplo, no caso em que a tarefa de controle pode ser facilmente dividida em sub-tarefas para as quais é fácil obter supervisores locais [Lin e Wonham, 1991]. A ação de controle é exercida por mais de um supervisor, sendo que cada um observa um determinado subconjunto de eventos (Σ_o). Ou seja, o controle é feito sob a condição de observação parcial de eventos. É possível estabelecer que um conjunto de eventos seja observável por um supervisor e não-observável pelo outro, bem como um subconjunto de eventos seja controlável para um supervisor e não seja controlável para o outro.

Com a impossibilidade do supervisor observar todos os eventos, introduz-se, na modelagem, uma máscara de observação que filtra os eventos para o supervisor. A condição de co-observabilidade, definida adiante, é necessária para garantir a existência dos supervisores.

Uma cadeia é chamada de ambígua se pode ser obtida através da projeção de cadeias diferentes. Sejam os alfabetos $\Sigma = \{\alpha, \beta, \gamma\}$ e $\Sigma_a = \{\alpha, \beta\}$ e as cadeias $\alpha\beta\gamma\alpha\beta$ e $\gamma\alpha\beta\alpha\beta$. As projeções $P_{\Sigma_a}(\alpha\beta\gamma\alpha\beta)$ e $P_{\Sigma_a}(\gamma\alpha\beta\alpha\beta)$ são iguais a $\alpha\beta\alpha\beta$. A cadeia $\alpha\beta\alpha\beta$ é uma cadeia ambígua.

Sejam s e s' pertencentes a uma linguagem $\overline{K}$ e o evento $\sigma \in \Sigma$. De acordo com Rudie e Wonham [1992], uma linguagem K é co-observável se:

- depois da ocorrência de uma cadeia ambígua s em K, a decisão de habilitar ou desabilitar σ é forçada pela ação que um supervisor, que possa controlar σ, possa ter sobre outras cadeias que se pareçam com s (tenham mesma projeção que s);

- a decisão de marcar ou não uma cadeia s (potencialmente ambígua) é determinada por pelo menos um dos supervisores.

Em palavras, a propriedade de co-observabilidade garante que, em qualquer tempo, pelo menos um supervisor tem informação suficiente do comportamento da planta para tomar a decisão de controle correta.

Rudie e Wonham [1992] dividem o problema do controle descentralizado em duas classes, aqueles em que se tem especificações locais (sobre um subconjunto dos eventos da planta) [Lin e Wonham, 1988], [Lin e Wonham, 1990], [Rudie e Wonham, 1992] e aqueles em que se tem uma especificação global (sobre o conjunto completo de eventos da planta) [Jiang et al., 2001], [Rudie e Wonham, 1992].

Inicialmente, a teoria do controle descentralizado tratou da classe de problemas em que se tem especificações locais.

O trabalho de Lin e Wonham [1988] apresenta a descentralização como sendo mais natural quando o objetivo de controle pode ser dividido em subtarefas. Para as subtarefas, pode-se obter supervisores locais que atuam sobre um subconjunto de eventos controláveis. A coordenação entre os supervisores locais é feita pelo uso de um supervisor coordenador que atua em um nível hierárquico superior.

Lin e Wonham [1988] apresentam condições, em termos da propriedade de normalidade, para existência de solução para o problema de controle supervisório descentralizado com observação parcial, condições suficientes para existência do supervisor coordenador e sob que condições não haverá perda de desempenho em relação à solução centralizada. Este trabalho deixa um problema importante em aberto: como dividir as tarefas de controle entre os dois níveis de forma que (i) o desempenho do sistema seja ótimo; (ii) intervenção do alto nível seja mínima [Lin e Wonham, 1990].

Lin e Wonham [1991] apresentam condições em que o não-bloqueio local dos supervisores implica não-bloqueio do sistema global.

No caso do problema do controle supervisório descentralizado com especificação global, a especificação não pode ser quebrada em subtarefas a serem implementadas por cada supervisor local. O trabalho de Rudie e Wonham [1992] apresenta condições necessárias e suficientes, em termos da propriedade de co-observabilidade, para a existência de solução através de controle local para este problema.

A satisfação da especificação global, mesmo quando dada pela conjunção de especificações locais, não implica a satisfação de cada especificação local. Jiang et al. [2001] estudam o problema em que as especificações locais devam ser satisfeitas e apresentam condições necessárias e suficientes para existência de supervisores de forma a garantir que cada especificação local seja atendida.

Na abordagem descentralizada convencional, os eventos habilitados globalmente são os eventos que estão habilitados por todos os supervisores locais ou seja, que não foram desabilitados por nenhum dos supervisores locais. Yoo e Lafortune [2002b] denominam esta arquitetura de conjuntiva. Eles apresentam, em [Yoo e Lafortune, 2002b], [Yoo e Lafortune, 2000b] e [Yoo e Lafortune, 2000a], uma forma generalizada da arquitetura descentralizada, onde as ações de controle de um conjunto de supervisores podem ser obtidas pela união (arquitetura conjuntiva) e/ou interseção (arquitetura disjuntiva) de eventos habilitados. Na nova arquitetura, os supervisores locais "decidem" de antemão que alguns eventos controláveis serão processados através da união e outros serão processados pela interseção.

O fato desta arquitetura requerer particionamento do conjunto de eventos controláveis quanto à forma de processamento reflete-se no enunciado do problema de controle. Além de buscar um supervisor não-bloqueante que implemente o comportamento desejado, deve-se derivar condições necessárias e suficientes para a existência da partição de Σ_c. Tanto o método de verificação da existência de supervisores locais como a técnica para encontrar a combinação apropriada das regras (conjuntivas e disjuntivas) podem ser realizados em tempo polinomial. Algumas variações sobre esta arquitetura são apresentadas em [Yoo e Lafortune, 2002c], [Yoo e Lafortune, 2004], [Yoo e Lafortune, 2005].

Barrett e Lafortune [1998] apresentam um formalismo para lidar com sistemas descentralizados cujos supervisores se comunicam. Ou seja, os controladores observam eventos gerados pelo sistema e podem trocar mensagens.

Mais recentemente, Huang et al. [2006] estenderam o paradigma de SEDs decentralizados para incluir o caso em que a observação é baseada no estado. A condição necessária e suficiente para a solução do problema é análoga à propriedade de co-observabilidade, chamada de co-observabilidade baseada em estados[2]. No caso da propriedade de co-observabilidade ser violada, a troca de informações entre os supervisores (comunicação) pode ajudar a garantir que, para cada evento a ser desabilitado, pelo menos um supervisor tem informação suficiente sobre a planta para desabilitar o evento. Desta forma, apenas algumas ocorrências dos eventos são comunicadas.

Kumar e Takai [2005] apresentam uma abordagem para tomada de decisão em que se classifica cada cadeia ambígua com uma gradação. O nível de ambigüidade de uma decisão de controle local é computada a partir de sua auto-ambigüidade e também das ambigüidades das outras decisões locais. Esta gradação é utilizada no processo de tomada de decisão ou seja, baseado no nível de ambigüidade decide-se qual a decisão a ser tomada (inferência). A decisão com um nível de ambigüidade mais baixo é a decisão escolhida nesta abordagem. Segundo os autores, um indicador de que esta é a forma mais natural de abordar o problema da tomada de decisão é que não há a necessidade de impor partições dos eventos controláveis em conjuntos permissivo e não-permissivo e a abordagem também não requer a computação global baseada em regras conjuntivas ou disjuntivas como outros trabalhos na área [Kumar e

[2]do inglês *state-based coobservability*

Takai, 2005]. Como desvantagem dessa abordagem, cita-se que o uso de inferência no contexto do controle descentralizado introduz um custo computacional na solução do problema.

Su e Thistle [2006] apresentam uma abordagem para síntese de supervisores distribuídos usando abstrações locais obtidas usando o conceito de bissimulação fraca. A bissimulação fraca, diferentemente da projeção natural, preserva a co-alcançabilidade do autômato, o que torna essa relação de equivalência mais interessante, segundo Su e Thistle [2006]. Abstrações do sistema local são obtidas, relacionadas à cada subsistema local. Os supervisores são então obtidos baseados nas composições dos componentes locais com cada abstração global (chamada de modelo centralizado abstraído). A comunicação da ocorrência de alguns eventos é permitida nessa abordagem. Os autores apresentam três procedimentos, para: (*i*) obtenção das abstrações eficientemente; (*ii*) definição do subconjunto de eventos comunicáveis (eventos cuja execução deve ser comunicada aos outros supervisores); (*iii*) obtenção do supervisor distribuído.

3.3 Extensões que visam Redução de Complexidade

Apesar dos grandes avanços teóricos alcançados nos últimos anos, as técnicas formais não vêm sendo utilizadas no ambiente industrial [Tilbury e Khargonekar, 2000]. Este fato justifica-se, em grande parte, pela complexidade computacional envolvida na solução do problema de controle. Alguma melhora na eficiência computacional é obtida fazendo uso de estruturas eficientes de representação e/ou arquiteturas que explorem aspectos estruturais do sistema.

Nas Seções 3.3.1 e 3.3.2 são apresentadas a representação de SEDs por BDDs [3] [Hoffman e Wong-Toi, 1992] e [Asarin et al., 1995], e por árvores de estados (STS[4]) [Ma e Wonham, 2006], respectivamente. Segundo Åkesson et al. [2002], as implementações baseadas em BDDs não tiram vantagem da estrutura modular da planta e especificação. Eles acreditam que a utilização de BDDs em conjunto com a abordagem modular deverá expandir a capacidade de resolver problemas. A mesma possibilidade se estende ao uso combinado de STS com a abordagem modular. Além destas, a modelagem de SEDs através de FSMwP[5] ou máquinas de estados finitos parametrizadas visa reduzir a complexidade do modelo pela associação de um conjunto de parâmetros ao modelo [Chen e Lin, 2000], [Oliveira et al., 2004]. Essa abordagem é apresentada na Seção 3.3.3.

De acordo com a classe de sistemas tratada, pode-se escolher uma arquitetura que explore determinadas características estruturais do sistema, de forma a obter algoritmos mais eficientes.

Eyzell e Cury [2001] exploram a simetria dos subsistemas para síntese de supervisores. Esta abordagem, apresentada na Seção 3.3.4, não foi desenvolvida com o objetivo de reduzir

[3] do inglês *Binary Decision Diagrams*
[4] do inglês *State Tree Structure*
[5] do inglês *Finite State Machine with Parameters*

complexidade mas, para o caso em que os subsistemas têm uma determinada característica topológica, a redução é obtida.

3.3.1 Diagramas de Decisão Binários (BDDs)

A complexidade de um modelo é, muitas vezes, medida pelo seu número de estados. Em sistemas compostos este número pode crescer substancialmente inviabilizando o seu armazenamento na memória do computador. BDD é um método de representação simbólica compacto que evita a enumeração explícita de todo o espaço de estados do sistema [Hoffman e Wong-Toi, 1992]. Ao usar BDDs para representar SEDs, obtém-se uma compressão significativa do espaço de estados.

Os sistemas compostos são formados de subplantas e o espaço de estados global é obtido pela composição das subplantas. Ao invés de gerar este autômato composto e aplicar os algoritmos de síntese e verificação, usam-se métodos simbólicos para manter a representação em uma forma mais concisa e aplicam-se as operações sobre esta representação [Asarin et al., 1995].

A representação do sistema através de autômatos é transformada em uma representação através de fórmulas booleanas sobre variáveis de estado [Asarin et al., 1995]. Estas fórmulas devem ser armazenadas e manipuladas de forma eficiente, o que pode ser feito usando BDDs [Bryant, 1986].

BDDs são árvores de decisão binárias com a restrição de que a ordem das decisões respeitam uma ordenação fixa. A árvore de decisão é representada por um grafo acíclico direcionado. A Figura 3.2 apresenta um BDD para a função $f = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$.

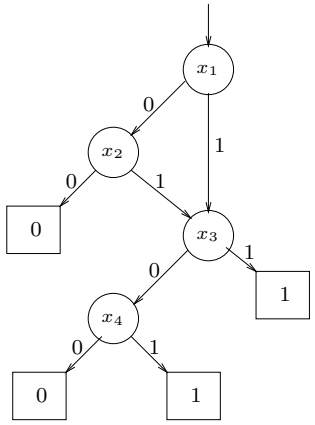


Figura 3.2: BDD para a função booleana $f = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$ [Balemi et al., 1993]

O valor de uma função para uma atribuição de variáveis particular pode ser lido seguindo o caminho entre a raiz e a folha. Cada nó interno é rotulado com uma variável booleana. Em cada nó, o caminho segue o ramo que corresponde ao valor (na atribuição de variáveis)

daquela variável. No exemplo da Figura 3.2, a atribuição de variáveis ($x_1 = 0$, $x_2 = 1$, $x_3 = 1$ e $x_4 = 1$) leva a uma folha marcada com 1, indicando que f é verdade para esta atribuição de variáveis. As operações lógicas E, OU, negação e quantificação universal podem ser realizadas em tempo polinomial [Balemi et al., 1993].

Com o uso de BDDs, a complexidade computacional da síntese deixa de ser polinomial no número de estados do modelo e passa a ser polinomial no número de nós do BDD. No pior caso, o número de nós é comparável ao número de estados do sistema que ele representa, mas em grande parte dos problemas práticos, tem-se um número de nós muito menor que o número de estados do modelo [Ma, 2004]. A razão para essa redução no número de nós é que com BDDs é possível remover nós redundantes usando regras simples e alcançando compressão substancial da função booleana que eles representam [Vahidi et al., 2005].

Segundo Vahidi et al. [2005], o gargalo para melhorar a eficiência da TCS em termos de memória e tempo de execução está em obter métodos mais eficientes de busca de alcançabilidade. Os autores argumentam que, mesmo o uso de BDDs pode não ser suficientemente eficiente, para os casos de funções mais complexas como, por exemplo, as funções de transição de um autômato (δ). Eles dizem ainda que, apesar da representação final por BDDs ser compacta, os BDDs intermediários costumam ser muito grandes, gerando muitas vezes explosão do número de nós do BDD. Como possíveis soluções para os problemas descritos, os autores apresentam técnicas de particionamento [Vahidi et al., 2005], [Byröd et al., 2006] usadas para auxiliar na representação das funções de transição δ e algoritmos de alcançabilidade especiais utilizados para reduzir o tamanho dos BDDs intermediários, tornando possível sintetizar e analisar supervisores para SEDs de larga escala.

BDDs e suas extensões vêm sendo utilizados, no contexto de SEDs, para viabilizar a verificação de propriedades [Vahidi, 2004], [Ma e Wonham, 2003], [Gunnarsson et al., 1996], na síntese de supervisores [Vahidi, 2004], [Ma e Wonham, 2003], [Balemi et al., 1993], [Hoffman e Wong-Toi, 1992], na detecção e isolamento de falhas [Lawesson et al., 2003], [Sztipanovits e Misra, 1996], entre outras.

3.3.2 Árvores de Estados

O trabalho de Ma e Wonham [2006] apresenta um modelo baseado em árvores de estado (STS), bem como um procedimento de síntese de supervisores não-bloqueantes ótimos. A idéia é gerenciar a complexidade através da organização do sistema como uma STS [Ma e Wonham, 2003].

A estrutura em árvore de estado (STS) é um espaço de estados que implementa tanto a estrutura horizontal (de concorrência) quanto a estrutura vertical (hierárquica) de forma natural e compacta. STS, quando utilizado para modelar SEDs, confere estrutura ao espaço de estados. Ao invés de implementar todo o grafo de transição através de um único predicado, a implementação é feita através de um conjunto de predicados simples. A estrutura

hierárquica multinível permite colocar informações locais (transições rotuladas com eventos não compartilhados) em níveis inferiores e informações de acoplamento (transições com eventos compartilhados) no nível superior.

BDDs são usados para representar e manipular os predicados. Segundo Ma e Wonham [2006], a lógica simbólica é uma das melhores formas para representar sistemas de transição e portanto a STS é representada simbolicamente. A síntese é também feita simbolicamente e de forma recursiva. Nesta abordagem, o controlador é representado por dois agentes. Um deles, o Agente 1, é responsável por acompanhar as mudanças de estado na planta. O outro, Agente 2, é responsável pelas decisões de controle. O Agente 1 comunica para o Agente 2 em qual estado a planta está e o Agente 2 realiza a desabilitação baseado nesta informação. A justificativa para esta divisão é o fato do controlador, na abordagem R&W, ter duas funções: acompanhamento da evolução dos estados da planta e ação de controle sobre a planta. O preço da tarefa de acompanhar a evolução da planta para sistemas complexos é alto, podendo significar controladores de milhões de estados, justificando assim a divisão do controlador em dois agentes.

A eficiência da síntese nesta abordagem se dá por vários fatores. O principal deles é o fato do algoritmo recursivo de síntese simbólica se beneficiar da estrutura hierárquica do STS e com isto efetivamente "controlar" o tamanho do BDD dos predicados intermediários através de uma boa ordenação das variáveis.

Dessa forma, a explosão de estados que pode ocorrer usando BDDs é evitada, expandindo-se a capacidade de representação, síntese e verificação para sistemas com número de estados maior que 10^{24} [Ma e Wonham, 2005].

Máquinas de estados finitos (FSM[6]) são um tipo especial de STS. Portanto, qualquer problema de controle descrito através de FSM pode ser reescrito usando STS. Esta abordagem resolve o PCS de forma centralizada, o que sugere que resultados ainda melhores podem ser obtidos pela união desta abordagem com a síntese modular ou modular local.

As principais vantagens do uso de STS foram resumidas por Ma e Wonham [2006]:

- STS são capazes de modelar sistemas complexos com estrutura concorrente e hierárquica;
- A síntese simbólica pode computar o supervisor não-bloqueante ótimo para sistemas em larga escala com uso de memória e tempo razoáveis;
- O controle pode ser facilmente implementado por funções de controle.

3.3.3 Máquinas de Estados Finitos Parametrizadas

Chen e Lin [2000] e Oliveira et al. [2004] modelam o SED por uma máquina de estados finitos ao qual é associado um conjunto finito de parâmetros (FSMwP). Muitas aplicações podem ser representadas eficientemente usando esta abordagem evitando a explosão de estados.

[6]do inglês *Finite State Machine*

Segundo Chen e Lin [2000], o grande desenvolvimento alcançado no estudo da teoria de controle supervisório em tão pouco tempo pode ser atribuído à simplicidade do modelo de máquinas de estados finitos. Depois de se ter os principais problemas e conceitos como controlabilidade, observabilidade, co-observabilidade, normalidade, entre outros, bem entendidos e resolvidos, surge o interesse por modelos que sejam mais úteis em aplicações reais.

O problema de explosão de estados inviabiliza a aplicação desta teoria em muitos sistemas reais. Esta explosão de estados pode ser evitada pelo uso de um parâmetro p inteiro para descrever alguma característica do sistema e, desta forma, o autômato que represente o sistema pode ter uma redução drástica do número de estados.

Por exemplo, para modelar um armazém de capacidade n usando um autômato seria necessário pelo menos n estados. Ao usar a abordagem com parâmetros, o autômato do armazém poderá ter 2 estados (armazém vazio e armazém com peças) e ao segundo estado estaria associado um parâmetro i que indicaria o número de peças no armazém. Caso fosse necessário mudar a capacidade do armazém, bastaria mudar o valor de n. As transições entre os estados são realizadas através de guardas, como será apresentado a seguir.

Existem dois tipo de transições, as transições dinâmicas e as transições de evento. As transições dinâmicas são representadas por guardas, que são predicados sobre os parâmetros, e as transições de evento são definidas como no caso clássico. As transições na FSM podem ser compostas de transições dinâmicas e de eventos. Neste caso, a transição entre os estados ocorre quando a guarda é satisfeita e o evento ocorre. Qualquer transição composta de guardas e eventos pode ser decomposta em transições de evento e transições dinâmicas pela introdução de estados extras no modelo. Devido a essa estrutura de transição, a FSMwP pode ser considerada um caso especial de Máquina Híbrida, cuja dinâmica contínua é inexistente ($\dot{p} = 0$ em todos os estados).

Chen e Lin [2001a] apresentam um mecanismo de abstração do modelo que agrega parâmetros e sequências de eventos para um modelo com granularidade maior, gerando uma representação, em geral, mais compacta da FSMwP. Esta abordagem é adequada para modelar sistemas com características hieráquicas.

Para realizar o controle nesta abordagem, utiliza-se a idéia de eventos forçáveis, restringindo o comportamento do sistema, fazendo-o atingir estados marcados.

Os mesmos autores [Chen e Lin, 2001b] apresentam um procedimento de síntese de supervisores para o problema de evitar que o sistema evolua para algum estado de um conjunto de estados proibidos. Dessa forma, uma especificação do tipo "o parâmetro p deve ser sempre menor que n" pode ser transformada em uma especificação de estado proibido através da introdução de um estado extra cuja guarda é $p > n$. Quando p for maior que n, o sistema transita para este estado extra que será então considerado proibido.

O objetivo de controle passa a ser impedir que o sistema visite os estados proibidos. Os vizinhos de um estado proibido são legais se as transições para o estado proibido puderem

ser evitadas (através de desabilitação ou pelo forçamento de eventos). Se a transição não puder ser evitada, o estado vizinho também é considerado proibido. Se a transição pode ser evitada sob algumas condições, o estado é particionado em estados legais e proibidos. Este procedimento é aplicado iterativamente até que o conjunto de estados proibidos convirja e o conjunto de desabilitações e eventos a serem forçados em cada estado seja conhecido.

Recentemente, Bherer et al. [2005] apresentam um método de síntese que garante maximalidade e requisitos de segurança em problemas em que os processos são simétricos. Os resultados obtidos são válidos para sistemas parcialmente observados, em que dois ou mais processos podem, simultaneamente, utilizar o recurso. Bherer et al. [2006] apresentam algoritmos para determinar se propriedades como alcançabilidade e não-bloqueio são violadas em sistemas parametrizados replicados com observação total de eventos.

Oliveira e Cury [2006] utilizam SEDs parametrizados para modelar sistemas. Um método para síntese de supervisores minimamente restritivos é apresentado. Segundo os autores, a característica principal desta abordagem é a possibilidade de capturar comportamentos não-regulares e a construção de modelos parametrizados que levam a soluções genéricas para uma dada classe de sistemas.

3.3.4 Simetria entre os Componentes

Eyzell e Cury [2001] formalizam a noção de simetria entre componentes de um sistema a partir de suas linguagens. A teoria de grupos é utilizada para definir as classes de estados e transições em um sistema, que são equivalentes sob as operações de permutação definidas. O grupo simétrico de uma linguagem é um conjunto composto de linguagens geradas através de um grupo de permutações especiais sobre seu alfabeto.

A partir deste grupo simétrico é construído um autômato quociente que representa a linguagem que tem, em geral, espaço de estados muito menor, que a representação original (quanto maior o grau de simetria, maior é a redução no espaço de estados). Sob certas condições, o problema do controle supervisório pode ser resolvido com um algoritmo similar ao algoritmo clássico, mas definido sobre o autômato quociente para a planta e especificação. O supevisor resultante pode também ser implementado usando uma estrutura de transição reduzida.

O cálculo do grupo simétrico é, em geral, complexo. Apesar disto, existem situações em que o grupo simétrico emerge naturalmente como em sistemas cujas subplantas tenham modelos com estrutura replicada, sendo formado por rotações sobre componentes replicados. Neste caso, esta abordagem pode ser aplicada com vantagem [Eyzell e Cury, 2001], [Eyzell, 2000].

Mais recentemente, Rohloff e Lafortune [2003a] apresentaram um modelo para agentes (ou módulos) similares que evita os problemas computacionais inerentes à verificação. Os

eventos do sistema são divididos em globais e privados sendo que os eventos globais representam a coordenação entre os módulos e os eventos privados afetam apenas os módulos em que ocorrem. Os módulos são cópias exatas a menos dos rótulos de seus eventos privados. Assume-se a existência de um controlador por módulo e os controladores dos vários módulos são também similares a não ser pelos rótulos dos eventos. O modelo é bem parecido com aquele apresentado em [Eyzell e Cury, 2001], mas o trabalho trata da verificação de algumas propriedades do sistema global como não-bloqueio e ausência de *deadlock*.

Os mesmos autores expandem os resultados apresentados em [Eyzell e Cury, 2001] e [Rohloff e Lafortune, 2003a], fazendo uma discussão em torno de classes de sistemas em que se pode assumir a presença de simetria por permutação [Rohloff e Lafortune, 2004a] e [Rohloff e Lafortune, 2004b].

3.4 Abordagens Combinadas

O uso combinado de algumas das abordagens apresentadas pode levar a uma solução mais adequada para alguns sistemas. Esta seção apresenta trabalhos que combinam abordagens tratadas neste capítulo.

Alguns autores já citados neste trabalho tratam de composições de alguns destes métodos. Destacam-se os trabalhos de Ma [2004] e Torrico [2003]. Ma [2004] usa BDDs associado a uma estrutura em árvore de estados, gerando uma representação ainda mais eficiente para os modelos. Torrico [2003] faz a combinação da arquitetura de controle hierárquico desenvolvida (decomposição vertical) com a arquitetura modular (decomposição horizontal). Esta arquitetura permite um comportamento consistente entre os níveis de hierarquia.

Leduc et al. [2005b] apresentam uma arquitetura baseada em interface (HISC[7]), com características hierárquicas e modulares. Este método decompõe o sistema em um subsistema de alto nível que comunica com $n \geq 1$ subsistemas paralelos de baixo nível através de interfaces, de forma a restringir a interação entre os subsistemas. Estas interfaces são projetadas para cada subsistema e estabelecem a comunicação entre cada subsistema do baixo nível e o subsistema de alto nível. São derivadas propriedades de consistência local que podem ser usadas para verificar se o SED é globalmente não-bloqueante e controlável, sem computar o modelo completo do sistema. Assim, a verificação pode ser feita apenas através da verificação destas propriedades do conjunto constituído pelo subsistema de baixo nível e interface daquele subsistema. Esta abordagem é vantajosa em termos computacionais, quando as interfaces obtidas possuem espaço de estados menor que o espaço de estados de seus subsistemas [Leduc et al., 2005a], [Leduc et al., 2005b]. Em [Leduc et al., 2006], os autores aplicam a metodologia à célula de manufatura flexível AIP[8], ilustrando assim a teoria.

Hill e Tilbury [2006] apresentam uma abordagem em que os supervisores modulares são não-conflitantes por construção. Uma hierarquia em níveis é utilizada, onde supervisores de

[7] do inglês *Hierarchical Interface-Based Supervisory Control*
[8] do francês *Atelier Interétablissement de Productique*

um mesmo nível são assíncronos entre si. Os supervisores do nível inferior são projetados primeiro. Então, os subsistemas juntamente com seu supervisor são abstraídos para servir como subsistema da hierarquia imediatamente superior e assim por diante. As abstrações, obtidas através de projeções naturais, devem possuir a propriedade do observador [Wong e Wonham, 1996] e consistência de marcação. Depois de calculados os supervisores do nível superior, sobre as abstrações do nível mais baixo, os eventos que foram apagados pelas projeções são adicionados aos supervisores em auto-laços. Esse procedimento é repetido até que não haja mais especificações a serem tratadas. Segundo Hill e Tilbury [2006], não há a garantia de que a solução modular proposta será ótima.

Krishnan [2004] apresenta uma técnica para decomposição do supervisor monolítico em supervisores distribuídos não-conflitantes através de técnicas clássicas de partição. A solução proposta é dividida em duas partes, a primeira em que o supervisor monolítico é considerado uma entidade independente ou seja, um sistema em malha aberta. Nessa fase, dois supervisores não-conflitantes são obtidos. Na segunda parte, a planta é considerada e os supervisores distribuídos são simplificados utilizando o comportamento da planta. Essa abordagem, apesar de levar a supervisores não-conflitantes, passa pela construção do supervisor monolítico e portanto está suscetível à explosão de estados.

de Queiroz et al. [2005] apresentam uma extensão da TCS que introduz um novo modelo para SEDs, que permite a diferenciação das classes de tarefas envolvidas no problema de controle. O uso de uma única marcação para representar todas as tarefas provoca uma perda de informação que pode comprometer a qualidade da solução de controle. Um modelo obtido pela composição de subsistemas é marcado se o estado correspondente em cada subsistema for marcado. Assim, uma tarefa global reflete as situações em que todos os subsistemas estão simultaneamente em estados marcados. Tal supervisor pode ser muito conservador para alguns problemas. A marcação do modelo global é então definida para representar situações em que pelo menos um objetivo de controle seja atingido [de Queiroz et al., 2005]. O supervisor garante que sempre seja possível alcançar estados em que pelo menos uma tarefa é completa. Este supervisor pode bloquear tarefas e ainda assim ser considerado fracamente não-bloqueante, desde que atinja algum de seus objetivos de controle.

3.5 Síntese

Neste capítulo foram apresentadas algumas extensões propostas para a Teoria do Controle Supervisório, bem como alguns trabalhos que visam reduzir a complexidade computacional através de uma representação eficiente e também pelo uso de arquiteturas que exploram características estruturais do sistema.

No Capítulo 4, apresentam-se extensões da TCS desenvolvidas para sistemas concorrentes, ou seja, sistemas compostos de subsistemas que executam em paralelo, onde não há restrição na observação de eventos. A abordagem modular local e alguns trabalhos que abordam técnicas eficientes para a detecção e resolução do conflito são apresentadas.

Capítulo 4

Controle de Sistemas Concorrentes

Os sistemas concorrentes (ou compostos), muito comuns no ambiente industrial, são constituídos de diversos subsistemas que operam concorrentemente. No contexto do Controle Supervisório de SEDs, a modelagem das partes dos sistemas é feita separadamente, o que costuma ser mais compreensível para o projetista. O comportamento global do sistema é obtido pela composição síncrona de suas partes ou subplantas. Os conjuntos de eventos podem ser parcialmente superpostos, gerando uma certo grau de sincronismo.

Este capítulo apresenta algumas extensões para a TCS desenvolvidas para tratar do controle de sistemas concorrentes. Nas Seções 4.1 e 4.2 apresentam-se tais abordagens. Quando se têm diversos supervisores atuando sobre a planta, há a possibilidade de ocorrer conflito entre os supervisores. A Seção 4.3 trata especificamente do problema do conflito e está subdividida em quatro partes. Nas três primeiras partes (4.3.1, 4.3.2 e 4.3.3), apresentam-se quatro abordagens para detecção do conflito. Na quarta parte (4.3.4) apresentam-se as abordagens de resolução de conflito, em que o conflito é evitado por construção. Por fim, a Seção 4.4 apresenta uma breve discussão.

4.1 Controle de Sistemas Concorrentes

Alguns autores investigaram o problema do controle de sistemas concorrentes em SEDs para os quais são sintetizados supervisores locais para a planta. A forma como estes supervisores locais são sintetizados e/ou coordenados para garantir a operação conjunta adequada dos diversos subsistemas controlados é assunto de alguns artigos.

Willner e Heymann [1991] estudaram o problema do Controle Concorrente em subsistemas que compartilham apenas eventos controláveis, o que significa que não há sincronização entre eventos não-controláveis pois eles representam comportamento espontâneo e imprevisível dos subsistemas. Este trabalho não considera marcação. Segundo Willner e Heymann [1991], sob certas condições, supervisores locais podem ter desempenho ótimo. Neste trabalho, o conceito de separabilidade é introduzido. Uma linguagem L é separável se existe um conjunto

de linguagens $L_i \subseteq \Sigma_i^*$, com $1 \leq i \leq n$, tal que $L = \mathop{||}_{i=1}^{n} L_i$. O principal resultado de [Willner e Heymann, 1991] estabelece que a ação de supervisores locais que implementam uma linguagem K tem o mesmo efeito sobre a planta que a ação de um supervisor centralizado que também implemente K, se K for separável. O trabalho apresenta ainda um algoritmo para testar a separabilidade de K em linguagens sobre alfabetos disjuntos.

Abdelwahed e Wonham [2002] apresentam uma extensão do esquema de controle supervisório descentralizado para uma classe de problemas com interação entre os componentes bem definida. Estes sistemas são chamados de Sistemas a Eventos Discretos Interativos ou IDES[1]. Um IDES consiste em um conjunto de componentes operando concorrentemente e uma especificação de interação que restringe o comportamento concorrente dos componentes. Neste modelo, a interação entre os componentes é considerada uma variável do modelo e é incorporada à estrutura do mesmo. Mais detalhes sobre este modelo são apresentados na Seção 4.3.1. Este modelo é estendido para sistemas hierárquicos multiníveis em [Abdelwahed e Wonham, 2003b].

Gaudin e Marchand [2003c] apresentam uma metodologia de síntese de supervisores para sistemas concorrentes, representados por um modelo estruturado, baseada na decomposição do conjunto de estados proibidos. O modelo do sistema é estruturado de forma a ter dois conjuntos de estados, os conjuntos de estados fracamente proibidos e o conjunto de borda. O conjunto de estados fracamente proibidos (CFP) é composto dos estados a partir dos quais é possível atingir estados proibidos por um caminho de eventos não-controláveis. O conjunto de borda é o conjunto de estados a partir dos quais ainda é possível aplicar controle sobre a planta antes dela entrar em estado de CFP. O problema de controle que é abordado neste trabalho é o problema de evitar que o sistema atinja estados proibidos.

Divididos os estados proibidos em subconjuntos de estados proibidos, supervisores locais que resolvem os problemas de controle locais são obtidos. Além destes supervisores locais, um supervisor global é obtido. Sua função é ativar e desativar os supervisores locais, de forma a coordenar suas ações para evitar que as configurações ilegais globais sejam alcançadas. As condições necessárias para que o supervisor global implemente o objetivo global de forma ótima são apresentadas em [Gaudin e Marchand, 2003c]. Extensões para esta abordagem são apresentadas em [Gaudin e Marchand, 2003a], [Gaudin e Marchand, 2003b] e [Gaudin e Marchand, 2005]

Em [Gaudin, 2006], o autor utiliza a arquitetura disjuntiva de Yoo e Lafortune [2002b] de forma a melhorar a eficiência computacional na síntese e ainda a possibilitar a representação do supervisor por FSM. Os cálculos são realizados sobre as aproximações $P_i^{-1}\mathcal{L}(G_i)$ ao invés de $\mathcal{L}(G)$. O conjunto de estados a serem evitados (E) é representado pela união de subconjuntos de estados dos subsistemas. Então, a linguagem da especificação é composta das cadeias que levam àqueles estados representados. A linguagem K é representada pelo conjunto de cadeias que não levam aos estados de E e modela o objetivo de controle de evitar que o sistema evolua para estados proibidos. O objetivo de controle pode ser decomposto em diversas

[1] do inglês *Interactive Discrete Event System*

linguagens K_j que vão gerar supervisores S_j. O autor mostra que o cálculo dos supervisores locais da forma apresentada no artigo é suficiente para garantir a solução de controle, desde que os eventos compartilhados sejam todos controláveis. O supervisor global é obtido como a combinação dos vários supervisores, compostos em uma arquitetura conjuntiva e disjuntiva. Apesar da hipótese de que os eventos compartilhados por dois ou mais subsistemas sejam controláveis, Gaudin [2006] argumenta que qualquer problema em que eventos não-controláveis são compartilhados pode ser transformado de forma que esses eventos não sejam utilizados no cálculo dos supervisores. A complexidade desta transformação deverá, no entanto, ser considerada.

Lee e Wong [1997] apresentam resultados para o problema em que se tem um sistema concorrente e um conjunto de especificações e são obtidos supervisores locais que implementem as especificações. Os sistemas podem compartilhar eventos controláveis e/ou não-controláveis desde que eles concordem em relação à classificação dos eventos compartilhados (se são controláveis ou não-controláveis). As especificações podem não ser prefixo-fechadas. As condições suficientes para garantir que a ação dos supervisores locais seja ótima são estabelecidas sobre a estrutura dos subsistemas locais e sua sincronização.

Komenda et al. [2005] tratam o problema em que, apesar dos subsistemas serem concorrentes, as especificações são indecomponíveis e não contidas na linguagem da planta. Por especificações indecomponíveis, entende-se que a especificação é global, e não pode ser decomposta em especificações locais. Os autores apresentam condições suficientes para que supervisores modulares tenham desempenho ótimo. Componentes locais da especificação global são utilizados para obtenção dos supervisores modulares e condições estruturais são impostas de tal forma que a interseção dos supervisores modulares implemente a mesma linguagem que um supervisor global obtido através do cálculo da planta global. Em trabalho anterior, Gaudin e Marchand [2004] apresentaram condições suficientes para solução do problema descrito acima, para o caso em que todos os eventos compartilhados pelos subsistemas são controláveis. Já em [Komenda et al., 2005], os autores consideram que os subsistemas concordam na controlabilidade dos eventos compartilhados. Os resultados foram estendidos para o caso de observação parcial dos eventos, em [Komenda e van Schuppen, 2006] e [Komenda e van Schuppen, 2005].

Schmidt et al. [2006] aplicam o método de síntese de supervisores para sistemas concorrentes, apresentado anteriormente em [Komenda et al., 2005], para sintetizar supervisores sobre o modelo reduzido dos subsistemas com objetivo de redução de complexidade. Os autores apresentam uma implementação destes supervisores, mostrando que os supervisores obtidos sobre modelos reduzidos podem ser usados sobre os modelos originais sob certas condições.

O controle modular local é apresentado em detalhe na próxima seção.

4.2 Controle Modular Local

A aplicabilidade da teoria de Controle Supervisório é bastante reduzida quando se está lidando com sistemas complexos como sistemas automatizados de manufatura. Como no controle supervisório, tanto a planta como as especificações são modeladas através de uma única linguagem (cada), apenas a modelagem do problema pode induzir a uma explosão de estados pela composição dos diversos subsistemas.

Os sistemas de manufatura são compostos de diversos subsistemas menores, usualmente representando operações concorrentes como montar, transportar e armazenar subprodutos. O projeto de controladores para esses sistemas é caracterizado pela existência de várias especificações atuando sobre apenas partes do sistema global. O principal objetivo é a coordenação dos subsistemas de forma que atendam a uma série de tarefas individuais e conjuntas, garantindo o bom funcionamento global do sistema [de Queiroz e Cury, 2002b].

A abordagem Controle Modular Local [de Queiroz, 2000] surgiu da tentativa de reduzir a complexidade de síntese de supervisores para sistemas de grande porte. Enquanto o controle monolítico e o controle modular usam a linguagem da planta completa para gerar as linguagens desejadas, o controle modular local usa apenas as subplantas que são restringidas pelas especificações. Dessa forma, menos autômatos são compostos e os autômatos que representam as linguagens desejadas são menores que nas abordagens monolítica e modular clássica. Conseqüentemente, a complexidade da síntese e os supervisores obtidos são menores prevenindo-se assim a explosão de estados.

Assim como no controle modular, a tarefa de fazer com que o sistema em malha fechada obedeça ao conjunto de restrições é dividida em subtarefas que são realizadas por diferentes supervisores. A diferença básica entre o controle modular e o modular local é que as especificações são tratadas localmente e os supervisores são projetados com uma visão parcial da planta.

Para utilizar a abordagem modular local, deve-se modelar o sistema através da Representação por Sistema Produto, apresentada na Seção 4.2.1. Em seguida, discute-se o procedimento de síntese de supervisores utilizando esta abordagem, na Seção 4.2.2.

4.2.1 Modelagem por Sistema Produto

Como os sistemas de manufatura são, em geral, compostos por um grande número de componentes interagindo, a modelagem do sistema deve ser feita de forma eficiente para evitar a explosão de estados. Nesta abordagem, o sistema é representado como um Sistema Produto, que consiste em um conjunto de subsistemas assíncronos entre si.

Qualquer sistema pode ser modelado por uma representação por sistema produto, RSP. Obtido o modelo para cada subsistema (representado de forma conceitual na Figura 4.1(a), onde os círculos superpostos indicam que há eventos compartilhados), a RSP pode ser obtida

pela composição de seus subsistemas de forma a obter um conjunto de subsistemas assíncronos (Figura 4.1(b)). A RSP mais refinada é obtida pela composição apenas dos subsistemas síncronos, criando o maior número possível de subsistemas assíncronos (nesse caso, quatro subsistemas assíncronos).

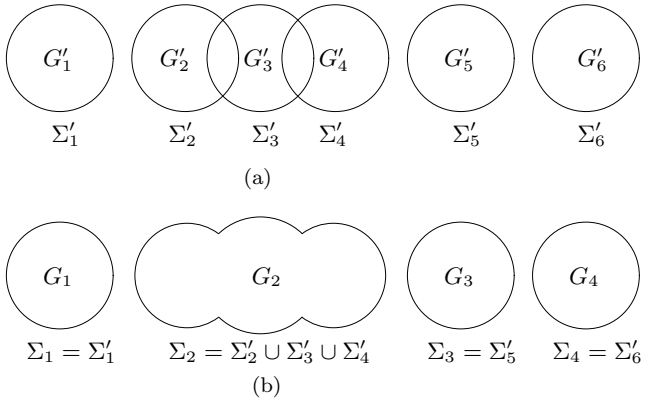


Figura 4.1: Representação por sistema produto: (a) modelagem dos diversos componentes; (b) obtenção da representação por sistema produto mais refinada.

No caso em que todos os subsistemas compartilham eventos entre si, a RSP mais refinada é a planta global. Os resultados obtidos pela abordagem modular local são tanto melhores quanto mais refinada for a RSP.

4.2.2 Obtenção dos Supervisores

Seja a RSP de um sistema G formada por subsistemas $G_i = (\Sigma_i, Y_i, \delta_i, y_{0i}, Y_{mi})$, $i \in I = \{1, \cdots, n\}$. O funcionamento da planta em malha aberta inclui cadeias de eventos indesejáveis. Para realizar o controle e com isto restringir o sistema de forma a ele não mais executar estas cadeias indesejáveis, precisa-se definir o comportamento desejado do sistema, o que é feito através das especificações. Seja a especificação modelada considerando apenas os eventos fundamentais para a descrição do comportamento desejado. Esta especificação é chamada de especificação genérica local, denotada por E_{xj} e definida sobre $\Sigma_{xj} \subseteq \Sigma$, para $j \in J = \{1, \cdots, m\}$. A planta local é composta pelos subsistemas da modelagem original que estão restringidos pela especificação genérica local. A planta local $G_{Lj} = (\Sigma_{Lj}, Q_{Lj}, \delta_{Lj}, q_{0Lj}, Q_{mLj})$, $\forall j \in J$, associada à especificação E_{xj}, é definida por $G_{Lj} = \Big\| \limits_{i \in N} G_i$, com $N = \{k \in I | \Sigma_k \cap \Sigma_{xj} \neq \emptyset\}$. Pode-se definir ainda a planta enxuta G_e como a composição de todas os subsistemas restringidos por pelo menos uma especificação e a planta complementar G_e^c como a composição dos subsistemas que não fazem parte de nenhuma planta local.

As especificações locais E_{Lj} expressam as linguagens desejadas para cada parte da planta global. Essa linguagem é obtida pela composição síncrona de cada especificação genérica com sua planta local correspondente $E_{Lj} = E_{xj} || G_{Lj}$. A especificação genérica pode ser expressa

em termos da planta enxuta (especificação enxuta) e da planta global (especificação global). A especificação enxuta é obtida fazendo $K_{Lje} = E_{xj}||G_e$ e a especificação global é obtida fazendo $K_{Lj} = E_{xj}||G$. Cada supervisor é então obtido pela operação $Sup\mathcal{C}(E_{L_j}, \mathcal{L}(G_{L_j}))$, buscando a máxima linguagem controlável contida na especificação local. Para ilustrar esta idéia, apresenta-se um exemplo a seguir.

Exemplo 4.2.1. *Seja o sistema G representado na Figura 4.1. Considere duas especificações genéricas $E_a \subseteq \Sigma_a^*$ e $E_b \subseteq \Sigma_b^*$ sendo que seus alfabetos intersecionam com os alfabetos das subplantas segundo a Figura 4.2 (E_a: $\Sigma_a \subseteq (\Sigma_1 \cup \Sigma_2)$ e E_b: $\Sigma_b \subseteq (\Sigma_2 \cup \Sigma_3)$. Como são duas*

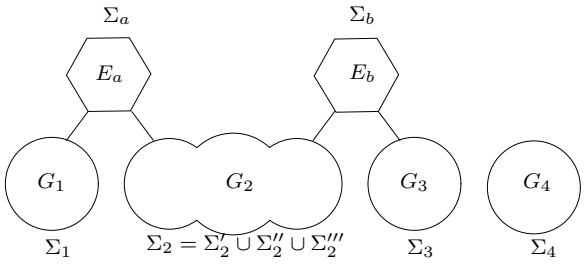


Figura 4.2: Exemplo 4.2.1: $\Sigma_a \subseteq \Sigma_1 \cup \Sigma_2$ e $\Sigma_b \subseteq \Sigma_2 \cup \Sigma_3$

especificações, têm-se duas plantas locais. A planta local relacionada a E_a é $G_A = G_1||G_2$, a planta local relacionada a E_b é $G_B = G_2||G_3$ e a planta enxuta $G_e = G_1||G_2||G_3 \subseteq \Sigma_e^*$. As especificações locais E_A e E_B podem então ser obtidas fazendo $E_A = E_a||\mathcal{L}_m(G_A)$ e $E_B = E_b||\mathcal{L}_m(G_B)$. As especificações podem ser expressas ainda em termos da planta global ($K_A = E_a||\mathcal{L}_m(G)$ e $K_B = E_b||\mathcal{L}_m(G)$) e da planta enxuta ($K_{Ae} = E_a||\mathcal{L}_m(G_e)$ e $K_{Be} = E_b||\mathcal{L}_m(G_e)$).

O subsistema G_4 pode ser desconsiderado na modelagem e controle de G para as especificações E_a e E_b. Por G_4 ser assíncrono, a composição dele com o resto da planta não afeta a controlabilidade e o não-bloqueio das especificações.

Esta abordagem explora a característica modular da planta considerando a existência de diversos subsistemas executando concorrentemente. Na arquitetura de controle proposta, cada módulo atua somente sobre os subsistemas atingidos. A Figura 4.3 ilustra esta idéia.

Dado um conjunto de especificações genéricas para um sistema composto, deve-se expressá-las em termos das plantas locais e computar as máximas linguagens controláveis $Sup\mathcal{C}(E_{Lj}, \mathcal{L}(G_{Lj})), \forall j$, que levam a supervisores locais, um para cada especificação.

Na síntese dos supervisores é utilizado um modelo parcial da planta global. Cada supervisor é então projetado com uma visão parcial da planta e também com a especificação local, sem ter informação das outras especificações a serem implementadas pelos outros supervisores. Pode ser o caso das especificações implementarem ações conflitantes. O conflito se caracteriza pela impossibilidade de completar tarefas, ou seja, a ação conjunta dos supervisores faz com que, para pelo menos uma cadeia da linguagem da planta em malha fechada, não seja possível atingir um estado marcado. No contexto do controle modular local, a forma

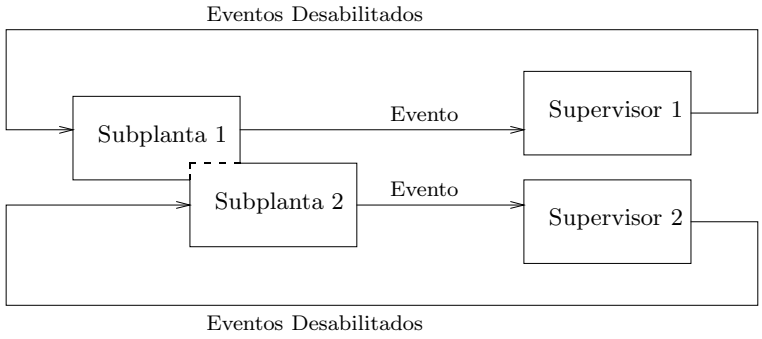


Figura 4.3: Diagrama do funcionamento dos supervisores na abordagem modular local, para dois supervisores

de detectar a presença do conflito é aplicando um teste após o projeto dos supervisores. Esta propriedade, também chamada de modularidade local, garante que a malha fechada do sistema sob ação dos supervisores modulares é não-bloqueante. Nesse caso, sabe-se que o desempenho dos supervisores locais é igual ao desempenho de um supervisor obtido pela abordagem monolítica [Wonham e Ramadge, 1988].

Na abordagem modular clássica, modularidade das linguagens dos supervisores é um requisito para garantir não-conflito dos supervisores. As linguagens L_i, $i \in \{1, \cdots, n\}$, são não-conflitantes (ou modulares) se:

$$\overline{\bigcap_{i \in I} L_i} = \bigcap_{i \in I} \overline{L_i}.$$

Em palavras, significa que duas linguagens são modulares se, sempre que compartilharem um prefixo, elas compartilharem também uma palavra contendo o prefixo.

Na abordagem modular local, o conceito de modularidade é expandido para o conceito de modularidade local. Duas linguagens $E_A, E_B \subseteq \Sigma_e^*$ são ditas localmente modulares se:

$$\overline{E_A || E_B} = \overline{E_A} || \overline{E_B}.$$

Portanto, a verificação da modularidade local das especificações pode ser feita diretamente sobre as especificações locais, e não sobre as especificações obtidas pela composição com a planta enxuta.

Pode-se garantir, através do Teorema 4.2.1 apresentado a seguir, que duas especificações são localmente modulares se e somente se elas são modulares em termos da planta global, ou seja, que os conceitos são equivalentes.

Teorema 4.2.1 (de Queiroz [2000])**.** *Sejam $E_A, E_B \subseteq \Sigma_e^*$ e $K_A, K_B \subseteq \Sigma^*$, $K_A = E_a || \mathcal{L}_m(G)$ e $K_B = E_b || \mathcal{L}_m(G)$ são modulares se se somente se $E_A = E_a || \mathcal{L}_m(G_A)$ e $E_B = E_b || \mathcal{L}_m(G_B)$ são localmente modulares, ou seja,*

$$\overline{K_A || K_B} = \overline{K_A} || \overline{K_B} \iff \overline{E_A || E_B} = \overline{E_A} || \overline{E_B}.$$

Em termos da controlabilidade das especificações, pode-se dizer que ela não varia pela adição ou subtração de subsistemas assíncronos ao subsistema original. Este resultado é apresentado no Teorema 4.2.2.

Teorema 4.2.2 (de Queiroz [2000]). *Sejam* $K_1, \mathcal{L}(G_1) \subseteq \Sigma_1^*$ *e* $K_2, \mathcal{L}(G_2) \subseteq \Sigma_2^*$, *com* $\Sigma_1 \cap \Sigma_2 = \emptyset$. *Então,*

$$Sup\mathcal{C}(K_1, \mathcal{L}(G_1)) \|_{as} Sup\mathcal{C}(K_2, \mathcal{L}(G_2)) = Sup\mathcal{C}(K_1 \|_{as} K_2, \mathcal{L}(G_1) \|_{as} \mathcal{L}(G_2)).$$

Esse resultado indica uma forma alternativa de sintetizar controladores para sistemas compostos sem passar pela composição de todos os subsistemas com cada especificação. Subsistemas que não são restringidos por nenhuma especificação não afetam a controlabilidade do sistema. Esta idéia é apresentada no Corolário 4.2.2.1.

Corolário 4.2.2.1 (de Queiroz [2000]). *Dado um sistema composto* G_i, *com* $i = \{1, \cdots, n\}$ *e as especificações* E_a *e* E_b. *Sejam* K_{Ae}, K_{Be}, K_A, K_B, G_e, G_e^c *e* G *como definidos anteriormente. Então,*

$$Sup\mathcal{C}(K_{Ae} \cap K_{Be}, \mathcal{L}(G_e)) \|_{as} \mathcal{L}_m(G_e^c) = Sup\mathcal{C}(K_A \cap K_B, \mathcal{L}(G)).$$

Ao desconsiderar subsistemas que são assíncronos e que não são restringidos por nenhuma especificação na síntese já há uma possível redução de complexidade. Alguma redução adicional é obtida na verificação da modularidade de especificações expressas em termos de suas plantas locais.

Para dar mais clareza, os teoremas foram apresentados para duas especificações. Todos eles podem ser estendidos diretamente para o caso de múltiplas especificações. Estas extensões podem ser encontradas em [de Queiroz, 2000].

Em seguida, o principal teorema desta abordagem, que estabelece as condições para que supervisores locais operando em conjunto tenham o mesmo desempenho que um supervisor monolítico projetado para o mesmo sistema e especificações, é apresentado.

Teorema 4.2.3 (de Queiroz [2000]). *Sejam* I *e* J *conjuntos de índices. Sejam* G *uma representação por sistema produto dada por subplantas* G_i, $i \in I$ *e* E_j *especificações genéricas, com* $j \in J$, *e para cada* j, *uma planta local* G_{Lj}, *uma especificação local* E_{Lj} *e uma especificação enxuta* K_{je}. *Seja* G_e *a planta enxuta. Se, para todo* $j \in J$, $Sup\mathcal{C}(E_{Lj}, \mathcal{L}(G_{Lj}))$, *são localmente modulares, então,*

$$Sup\mathcal{C}(\bigcap_{j \in J} K_{je}, \mathcal{L}(G_e)) = \|_{j \in J} Sup\mathcal{C}(E_{Lj}, \mathcal{L}(G_{Lj})).$$

Pelo Teorema 4.2.3 a modularidade local das linguagens garante que a ação conjunta dos supervisores locais seja ótima, ou seja, não apresenta perda de desempenho quando comparada à ação do supervisor centralizado, projetado pela abordagem monolítica.

Dado um conjunto de subsistemas G_i, $i \in I$, e um conjunto de especificações genéricas locais E_{xj}, $j \in J$, o procedimento para projetar supervisores através da abordagem modular local é resumido a seguir.

1. Obtenção das plantas locais G_{Lj} para cada especificação pela composição das subplantas que compartilham eventos com a especificação, ou seja, $G_{Lj} = \|_{i \in N} G_i$ com $N = \{k \in I | \Sigma_k \cap \Sigma_{xj} \neq \emptyset\}$, $\forall j \in J$.
2. Obtenção das especificações locais E_{Lj} pela composição das especificações genéricas locais E_{xj} com as plantas locais G_{Lj}, ou seja, $E_{Lj} = E_{xj} || \mathcal{L}_m(G_{Lj})$, $\forall j \in J$.
3. Obtenção das componentes aparadas das especificações locais.
4. Obtenção das linguagens implementadas pelos supervisores S_j, chamadas também de S_j, pela operação $Sup\mathcal{C}(E_{Lj}, \mathcal{L}(G_{Lj}))$, que resulta na máxima linguagem controlável e.r.a G_{Lj} contida em E_{Lj}, , $\forall j \in J$.
5. Teste da modularidade local entre os supervisores locais obtidos.

Um exemplo é apresentado a seguir.

Exemplo 4.2.2. *O mesmo exemplo apresentado na Seção 2.2.2 para ilustrar o problema do conflito entre supervisores modulares é agora resolvido usando a abordagem modular local. Sejam os subsistemas G_1, G_2, G_3 e G_4 sobre os alfabetos Σ_1, Σ_2, Σ_3 e Σ_4, respectivamente, cujos eventos são todos controláveis, e as especificações E_{x1}, E_{x2} e E_{x3} sobre os alfabetos Σ_{x1}, Σ_{x2} e Σ_{x3}, respectivamente, apresentados na Figura 4.4.*

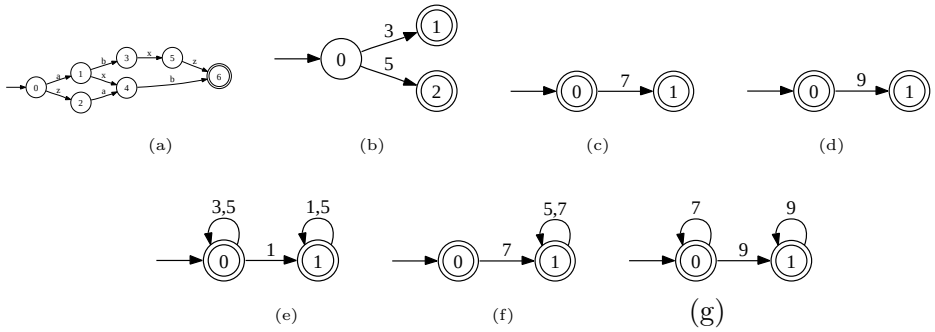


Figura 4.4: Exemplo 4.2.2: (a) G_1; (b) G_2; (c) G_3; (d) G_4; (e) E_{x1}; (f) E_{x2}; (g) E_{x3}.

O primeiro passo é obter as plantas locais:

- $G_{L1} = G_1 || G_2$ *pois* $\Sigma_{x1} \subseteq \Sigma_1 \cup \Sigma_2$;
- $G_{L2} = G_2 || G_3$ *pois* $\Sigma_{x2} \subseteq \Sigma_2 \cup \Sigma_3$;
- $G_{L3} = G_3 || G_4$ *pois* $\Sigma_{x3} \subseteq \Sigma_3 \cup \Sigma_4$.

Em seguida, obtém-se as especificações locais:

- $E_1 = E_{x1}||G_{L1}$;

- $E_2 = E_{x2}||G_{L2}$;

- $E_3 = E_{x3}||G_{L3}$.

As especificações locais são apresentadas na Figura 4.5

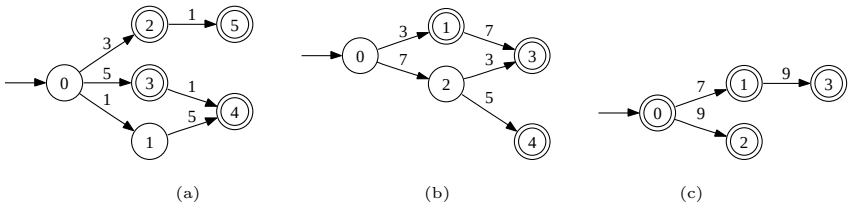


Figura 4.5: Exemplo 4.2.2: (a) E_1; (b) E_2; (c) E_3.

Como todos os eventos são controláveis, estas especificações são controláveis. Portanto, os supervisores obtidos implementam as linguagens geradas por E_1, E_2 e E_3 e portanto, $S_1 = E_1$, $S_2 = E_2$ e $S_3 = E_3$. Caso as especificações não fossem controláveis, a máxima sublinguagem controlável contida em E_1, E_2 e E_3 ($S_j = Sup\mathcal{C}(E_j, \mathcal{L}(G_{Lj}))$) seria obtida e os supervisores implementariam estas sublinguagens máximas.

Finalmente, deve-se proceder o teste da modularidade local. Sejam S_1, S_2 e S_3 as linguagens implementadas pelos supervisores obtidos. O teste da modularidade local consiste em obter as linguagens $\overline{S_1||S_2||S_3}$ (Figura 4.6(a)) e $\overline{S_1}||\overline{S_2}||\overline{S_3}$ (Figura 4.6(b)) e verificar sua igualdade. Se essa igualdade não for verificada, conclui-se que a ação conjunta dos supervisores é conflitante.

Pode-se observar que os geradores representados nas Figuras 4.6(a) e 4.6(b) não são isomorfos. O estado 19 do gerador da Figura 2.3(b), alcançado pela execução das cadeias 1 9 *e/ou* 9 1*, não aparece no gerador da Figura 4.6(a)), o que implica que os supervisores são conflitantes, ou seja, não possuem a propriedade de modularidade local.*

É importante observar que a modularidade local de todos os subconjuntos de linguagens não garante a modularidade local do conjunto completo, o que implica a verificação da modularidade requerer o cálculo da composição síncrona do conjunto completo de linguagens [de Queiroz e Cury, 2002a]. O exemplo que se segue ilustra esse fato.

Exemplo 4.2.3. *Considerando o exemplo anterior, o teste da modularidade local realizado entre subconjuntos das linguagens implementadas pelos supervisores apresenta resultado positivo, ou seja,*

- $\overline{S_1}||\overline{S_2} = \overline{S_1||S_2}$

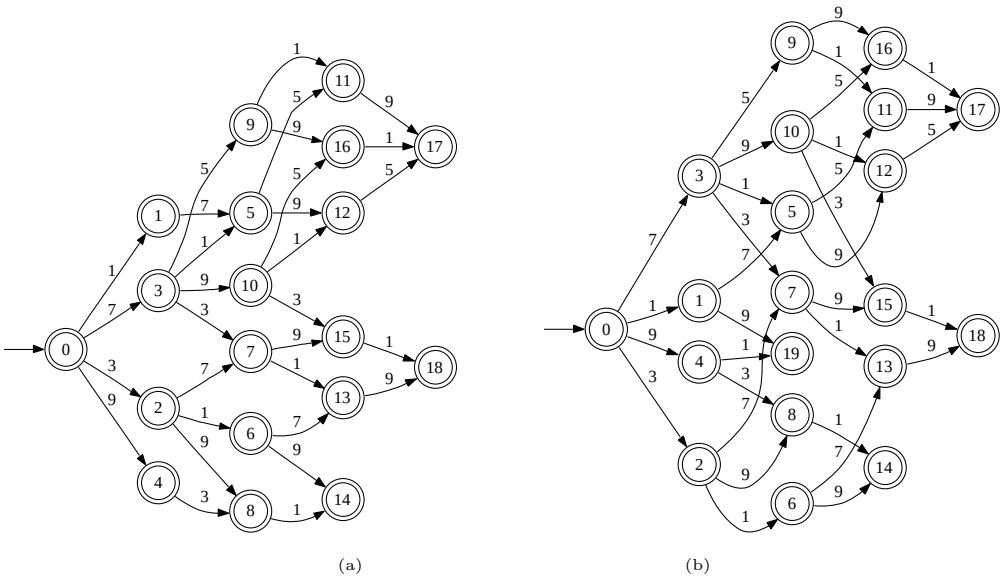


Figura 4.6: Exemplo 4.2.2 - Teste da modularidade local: Autômatos que geram: (a) $\overline{S_1||S_2||S_3}$; (b) $\overline{S_1}||\ \overline{S_2}||\overline{S_3}$

- $\overline{S_2}||\overline{S_3} = \overline{S_2||S_3}$.

Na Figura 4.7 apresentam-se os autômatos que representam o resultado do teste da modularidade local entre os dois pares de supervisores. Como S_1 e S_3 são assíncronos, tem-se que

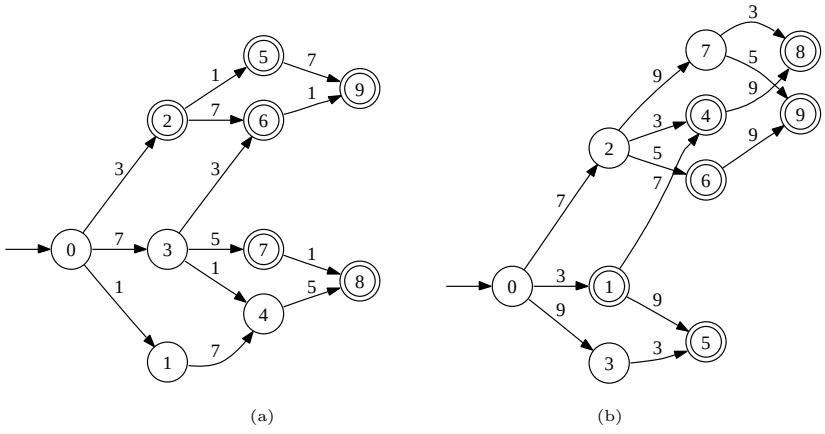


Figura 4.7: Exemplo 4.2.3 - Teste da modularidade local sobre subconjuntos dos supervisores. Autômatos que implementam: (a) $\overline{S_1}||\overline{S_2} = \overline{S_1||S_2}$; (b) $\overline{S_2}||\overline{S_3} = \overline{S_2||S_3}$

$\overline{S_1}||\overline{S_3} = \overline{S_1||S_3}$, *por definição.*

No entanto, $\overline{S_1}||\overline{S_2}||\overline{S_3} \neq \overline{S_1||S_2||S_3}$, *como mostrado na Figura 4.6.*

Essa é a grande limitação da abordagem modular local. Apesar da síntese dos supervisores ter a complexidade reduzida em relação às outras abordagens, o procedimento de verificação

exige que se faça a composição de todos os supervisores, ficando sujeito ao problema da explosão de estados.

Recentemente, tem havido um crescente interesse no assunto. Um estudo de alguns trabalhos que tratam de formas eficientes para detecção do conflito é apresentado na seqüência do documento.

4.3 Conflito

Seja o PCS em que G é uma planta composta por subsistemas G_i e E_{xj} é um conjunto de restrições a serem aplicadas à planta, com $i \in \{1, \cdots, n\}$ e $j \in \{1, \cdots, m\}$. Soluções para esse problema de controle podem ser obtidas por diversas abordagens. Ao usar as abordagens que geram diversos supervisores, deve-se levar em conta a possibilidade de ocorrer conflito.

O problema do conflito é inerente às abordagens em que são gerados mais de um supervisor para implementar as especificações. Apesar dos supervisores serem entidades independentes, sua ação é aplicada sobre a mesma planta e, portanto, os efeitos dos diversos supervisores sobre a planta são interdependentes.

Mesmo que todos os supervisores sejam não-bloqueantes em relação à planta, sua ação conjunta pode causar o bloqueio do sistema global. Ainda na fase de projeto de supervisores modulares e localmente modulares, deve-se proceder uma verificação de modularidade que garanta a ausência de conflito. Se os supervisores forem modulares, há a garantia de que o sistema global é não-bloqueante e que não há perda de desempenho quando comparada ao supervisor monolítico projetado para as mesmas especificações.

Uma forma de abordar esse problema é realizar a síntese de supervisores que implementam a máxima sublinguagem controlável e não-conflitante [Chen e Lafortune, 1991]. Nesse trabalho, a condição de não-conflito é em relação a uma segunda linguagem dada, e a condição de controlabilidade é em relação a uma terceira linguagem e um conjunto fixo de eventos não-controláveis. Esta solução pode ser muito restritiva.

Abdelwahed e Wonham [2003a] propõem um método de detecção de conflito entre sistemas, sem passar pela composição de todos eles (Seção 4.3.1). Malik [2004] e Flordal e Malik [2006] propõem um método para detecção de conflito a partir da caracterização do conflito e, em certos casos, é possível detectar o conflito sem compor todas os sistemas, usando o método incremental de Brandin et al. [2000] (Seções 4.3.2 e 4.3.3). Na Seção 4.3.4 apresentam-se as abordagens de resolução de conflito em que o conflito é evitado por construção ([Wong et al., 2000], [Chen e Lafortune, 2000], [Wong e Wonham, 1998] e [Wong et al., 1995]).

As subseções que se seguem abordam esses trabalhos.

4.3.1 Abordagem de detecção do conflito: detecta-primeiro

Abdelwahed [2002] apresenta uma nova estrutura de modelo chamada IDES. Um IDES é composto de um conjunto de componentes operando concorrentemente e uma linguagem que sincroniza os componentes, denotada por K. O conjunto de componentes é representado por um vetor de linguagens, $\mathbf{L} = [L_1, L_2, \cdots, L_n]$, onde $L_i \subseteq \Sigma_i^*$ e I é um conjunto de índices. $\mathbf{\Sigma}$ representa o vetor de alfabetos $\mathbf{\Sigma} = \{\Sigma_i | i \in I\}$. A união de todos os alfabetos é denotada por Σ. O conjunto de todos os vetores de linguagens sobre Σ é denotado por $\mathfrak{L}(\Sigma)$. Sejam ainda $\mathbf{A}$ um conjunto de autômatos $\{A_i = (\Sigma_i, Q_i, \delta_i, q0i, Q_{mi}), i \in I\}$ sobre o espaço $\mathbf{\Sigma}$ e $A = ||A_i = (\Sigma, Q, \delta, q0, Q_m)$ seu produto síncrono.

Um IDES sobre Σ é um par $\mathcal{L} = (\mathbf{L}, K)$, onde $\mathbf{L}$ é um vetor de linguagens em $\mathfrak{L}(\Sigma)$ e K é uma linguagem em Σ^*. L_i é usado para denotar o i-ésimo componente de $\mathbf{L}$. A linguagem gerada por $\mathcal{L}$ é dada por:

$$B_\Sigma = ||\mathbf{L} \cap K.$$

Daqui em diante, considere $K = \Sigma^*$ e, portanto,

$$B_\Sigma = ||\mathbf{L}.$$

O vetor de projeção $\mathbf{P_\Sigma} : \mathcal{L}(\Sigma) \rightarrow \mathfrak{L}(\Sigma)$ associa cada linguagem $L \in \mathcal{L}(\Sigma)$ a um vetor de linguagens $\{P_i L | \, i \in I\}$, onde $P_i : \Sigma^* \rightarrow \Sigma_i^*$ é a projeção natural.

Sejam $\overline{B_\Sigma(\mathbf{L})} = \overline{\underset{i \in I}{||} L_i}$ e $B_\Sigma(\overline{\mathbf{L}}) = \underset{i \in I}{||} \overline{L_i}$. Um vetor de linguagens é dito não bloqueante se:

$$\overline{B_\Sigma(\mathbf{L})} = B_\Sigma(\overline{\mathbf{L}}),$$

ou seja, se o conjunto $\{P_i^{-1} L_i | i \in I\}$ é não-conflitante, ou ainda se

$$\overline{\underset{i \in I}{\cap} P_i^{-1} L_i} = \underset{i \in I}{\cap} \overline{P_i^{-1} L_i}.$$

Denota-se por $\varrho_A(q)$ o conjunto de estados alcançáveis a partir de q no autômato A e por Σ_s o conjunto de eventos compartilhados, ou seja, $\Sigma_s = \underset{i \neq j}{\cup} (\Sigma_i \cap \Sigma_j)$.

Para mais detalhes sobre esta estrutura, consulte [Abdelwahed, 2002].

O vetor de linguagens $\mathbf{L}$ é equivalente ao IDES $\mathcal{L} = (\mathbf{L}, \Sigma^*)$, o que sugere que o IDES pode ser usado para representar um conjunto de componentes, sem considerar a restrição.

A verificação do bloqueio é feita através da busca exaustiva em todo espaço de estados do sistema e portanto está suscetível ao problema da explosão de estados. Para sistemas complexos, esse processo se torna pouco eficiente. Além de introduzir um novo modelo, Abdelwahed [2002] apresenta um método para detecção de bloqueio em IDES. Ele apresenta um conjunto de procedimentos indiretos para verificação de bloqueio no sistema, em que não

é necessário explorar todo os espaço de estados do sistema, mas somente regiões relevantes. A abordagem proposta é particularmente eficiente para o caso de sistemas fracamente acoplados, ou seja, com pouca interação entre os componentes.

O bloqueio corresponde à situação em que o sistema atinge uma região no espaço de estados a partir da qual ele não consegue atingir nenhum estado marcado. De acordo com a possibilidade de movimento dentro desta região, o estado pode ser descrito como de *deadlock* ou *livelock*. O *deadlock* ocorre quando um estado alcançável não marcado não consegue atingir nenhum outro estado. Esse estado é chamado de estado de *deadlock*. Quando o estado alcançável não marcado pode atingir apenas estados não marcados, ele é chamado de estado de *livelock*. Um estado bloqueante pode ser de *deadlock* ou *livelock*, mas não ambos [Abdelwahed e Wonham, 2003a].

A abordagem a ser apresentada tem o objetivo de detectar e identificar estados bloqueantes de ambas as formas. Esta abordagem é chamada de *detecta-primeiro*. O método para detecção de *deadlock* é apresentado a seguir e em seguida é apresentado o método para detecção de *livelock*.

4.3.1.1 Detecção de *deadlock*

Sejam $\mathbf{L}$ um vetor de linguagens em Σ e $K = \Sigma^*$, portanto $B_\Sigma = ||\mathbf{L}$. Uma cadeia $s \in B_\Sigma(\overline{\mathbf{L}})$ é chamada não-terminal se ela não está em $B_\Sigma(\mathbf{L})$. Um vetor de linguagens é livre de *deadlock* se o conjunto de eventos ativos após a ocorrência de qualquer cadeia não-terminal em $B_\Sigma(\overline{\mathbf{L}})$ é não-vazia. Formalmente,

$$(\forall s \in (B_\Sigma(\overline{\mathbf{L}}) - B_\Sigma(\mathbf{L})))(\exists \sigma \in \Sigma) \quad s\sigma \in B_\Sigma(\overline{\mathbf{L}}).$$

Portanto, *deadlock* pode ser detectado através da exploração do espaço de estados completo do sistema e identificação dos estados não-marcados que não tenham eventos ativos. Usa-se $Ativ(s)$ para denotar os eventos ativos após a ocorrência de uma cadeia s. A abordagem detecta-primeiro é uma alternativa a essa abordagem, em que identifica-se os estados potencialmente bloqueantes primeiro e depois testa-se a alcançabilidade desses estados.

Para um dado subconjunto J de I, seja $\Sigma_{(J)}$ o conjunto de eventos compartilhados exclusivamente pelo conjunto J de componentes, ou seja, $\Sigma_{(J)} = \bigcap_{j \in J} \Sigma_j - \bigcup_{i \in I-J} \Sigma_i$.

Exemplo 4.3.1. *Sejam os alfabetos* $\Sigma_1 = \{\sigma, \alpha, \delta, \lambda\}$, $\Sigma_2 = \{\gamma, \alpha, \delta, \kappa\}$ *e* $\Sigma_3 = \{\beta, \lambda, \delta, \kappa\}$, *tal que* σ, γ *e* β *são eventos locais de* Σ_1, Σ_2 *e* Σ_3, *respectivamente. Seja* $J = \{1, 2\}$, *a Figura 4.8 ilustra o compartilhamento de eventos entre os alfabetos. Apesar de a interseção entre* Σ_1 *e* Σ_2 *ser o conjunto* $\{\alpha, \delta\}$, *o único evento que é compartilhado apenas por* Σ_1 *e* Σ_2 *é* α *e, portanto,* $\Sigma_{(J)} = \{\alpha\}$.

A questão principal desta abordagem está no Teorema 4.3.1 em que é feita uma caracterização do estado livre de *deadlock*.

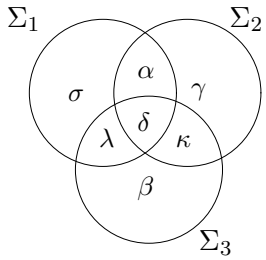


Figura 4.8: Exemplo 4.3.1 - Alfabeto $\Sigma_{(J)}$

Teorema 4.3.1. *O vetor de linguagens* $\mathbf{L}$ *é livre de deadlock se e somente se*

$$(\forall s \in B_\Sigma(\overline{\mathbf{L}}) - B_\Sigma(\mathbf{L}))(\exists J \subseteq I) \bigcap_{j \in J} Ativ_{L_j}(P_j s) \cap \Sigma_{(J)} \neq \emptyset.$$

Esse teorema mostra que o *deadlock* pode ser identificado examinando o conjunto de eventos ativos após a ocorrência de cadeias locais, geradas pelos componentes do sistema. Portanto, um vetor de linguagens $\mathbf{L}$ está livre de deadlock se e somente se existe um evento local (não compartilhado) habilitado em qualquer dos componentes do vetor de projeção da cadeia. Conseqüentemente, para detectar os estados de *deadlock* globais, basta considerar as combinações de estados locais em que apenas eventos compartilhados estejam ativos. Esses são os estados globais potencialmente bloqueantes.

A seguir, apresenta-se um exemplo. O vetor de linguagens do exemplo tem apenas dois componentes, para facilitar o entendimento. As linguagens componentes do vetor são resultantes da ação de um supervisor obtido pela abordagem modular local sobre sua planta local. Estas linguagens são conflitantes e o objetivo deste exemplo é mostrar isto através da abordagem detecta-primeiro.

Exemplo 4.3.2. *Sejam o vetor de linguagens* $\mathbf{L} = [L_1, L_2]$*, sendo que estas linguagens são geradas pelos autômatos* S_1 *(*$L_1 = \mathcal{L}(S_1) \subseteq \Sigma_1^*$*) e* S_2 *(*$L_2 = \mathcal{L}(S_2) \subseteq \Sigma_2^*$*) representados na Figura 4.9. Sejam os conjuntos de eventos:*

- $\Sigma_1 = \{53, 54, 65, 66, 71, 72, 73, 74\}$
- $\Sigma_2 = \{71, 72, 73, 74, 81, 82\}$.

Os eventos controláveis são representados por números ímpares e os não-controláveis por números pares. Nesse exemplo, $J = \{1, 2\}$ *e* $\Sigma_{(J)} = \Sigma_s$.

As linguagens L_1 *e* L_2 *compartilham os eventos* $\Sigma_s = \{71, 72, 73, 74\}$. *Na Tabela 4.1 apresentam-se os eventos ativos em cada estado de* S_1 *e* S_2.

A detecção de deadlock é feita através da identificação de estados potencialmente bloqueantes e posterior teste de alcançabilidade desses estados. Para obter o conjunto de estados potencialmente bloqueantes são consideradas apenas as combinações de estados locais tais que

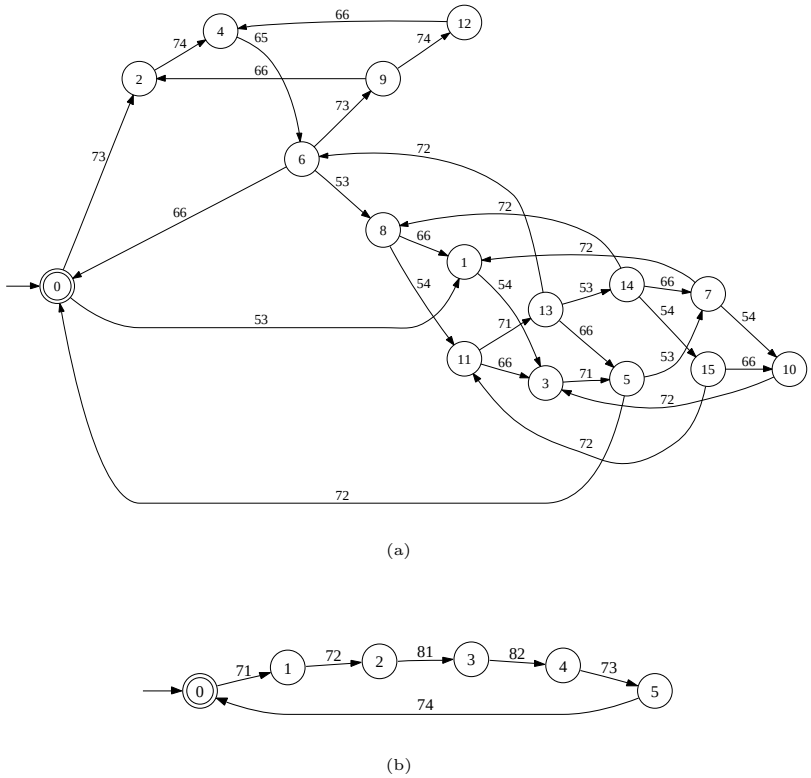


Figura 4.9: Exemplo 4.3.2: (a) S_1; (b) S_2.

S_1				S_2	
Estados	Eventos	Estados	Eventos	Estados	Eventos
0	73,53	8	66,54	0	71
1	54	9	74,66	1	72
2	74	10	72	2	81
3	71	11	71,66	3	82
4	65	12	66	4	73
5	72,53	13	72,53,66	5	74
6	73,53,66	14	72,54,66		
7	72,54	15	72,66		

Tabela 4.1: Exemplo 4.3.2 - Eventos ativos nos estados de S_1 e de S_2

apenas eventos assíncronos estejam ativos. Por exemplo, o estado 0 *de* S_1 *tem os eventos* 73 *e* 53 *ativos. Portanto, nenhum estado global composto do estado* 0 *de* S_1 *é potencialmente bloqueante. O mesmo ocorre para todos os estados compostos dos estados* 1, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15 *de* S_1 *e estados* 2 *e* 3 *de* S_2*. Então, restam apenas algumas combinações. São elas:*

$$\begin{array}{cccc} (2,0) & (2,1) & (2,4) & (2,5) \\ (3,0) & (3,1) & (3,4) & (3,5) \\ (10,0) & (10,1) & (10,4) & (10,5). \end{array}$$

Em conseqüência do Teorema 4.3.1, se a interseção dos eventos ativos nos estados locais de um par de estados com o conjunto de eventos compartilhados (Σ_s) *for não-vazia, esse estado global está livre de deadlock. Portanto, o par* $(2,5)$ *pode ser tirado da lista de pares potencialmente bloqueantes, pois seus estados locais compartilham o mesmo evento (*74*). O mesmo ocorre com os pares* $(3,0)$ *e* $(10,1)$*, que compartilham os eventos* 71 *e* 72*, respectivamente.*

Depois de identificados os estados potencialmente bloqueantes, aplica-se o teste de alcançabilidade sobre esse conjunto de pares.

$S_1 \| S_2$							
Estados	Par	Estados	Par	Estados	Par	Estados	Par
0	(0,0)	9	(3,2)	18	(8,0)	27	(8,3)
1	(1,0)	10	(1,3)	19	(11,0)	28	(6,4)
2	(3,0)	11	(0,4)	20	(13,1)	29	(11,3)
3	(5,1)	12	(3,3)	21	(14,1)	30	(8,4)
4	(7,1)	13	(1,4)	22	(6,2)	31	(9,5)
5	(0,2)	14	(2,5)	23	(15,1)	32	(11,4)
6	(10,1)	15	(3,4)	24	(8,2)	33	(12,0)
7	(1,2)	16	(4,0)	25	(6,3)		
8	(0,3)	17	(6,0)	26	(11,2)		

Tabela 4.2: Exemplo 4.3.2 - Correspondência entre o estado global e o par de estados (a, b) sendo que a corresponde a um estado de S_1 e b corresponde a um estado de S_2.

A Tabela 4.2 mostra a correspondência entre os estados globais (de $S_1||S_2$*) e os pares locais. Uma inspeção na Tabela 4.2 mostra que entre os pares listados, somente o par* $(3,4)$ *é alcançável, que corresponde ao estado* 15 *do autômato apresentado na Figura 4.10.*

O estado de *deadlock* foi identificado através do método detecta-primeiro. Nesse caso, para verificar a alcançabilidade do conjunto de pares, o produto síncrono dos autômatos foi obtido. Existem vários métodos para se fazer a verificação de alcançabilidade, cujos detalhes estão fora do escopo deste trabalho.

A Proposição 4.3.1 mostra que a busca de alcançabilidade pode ser realizada considerando apenas o comportamento compartilhado do sistema.

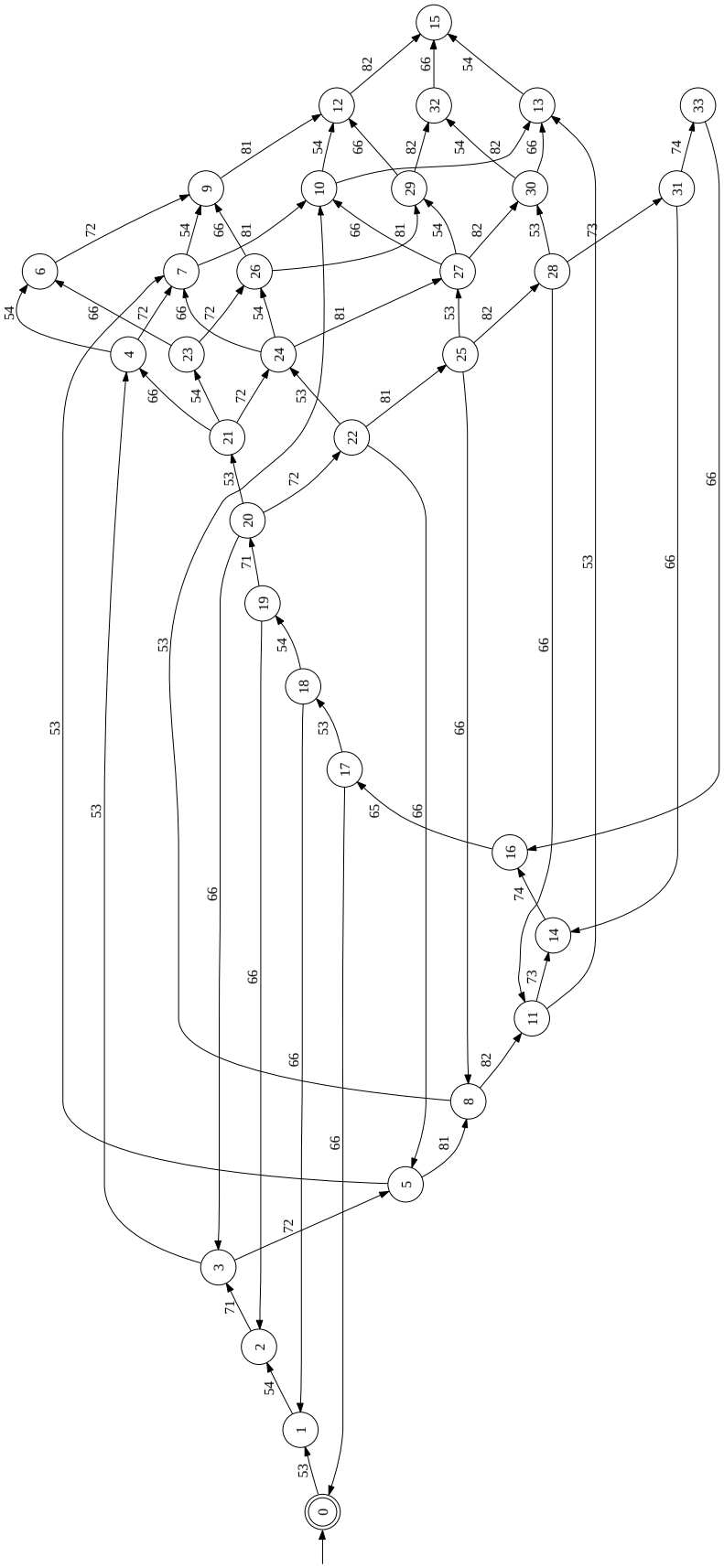


Figura 4.10: Exemplo 4.3.2 - Autômato resultante do produto síncrono $S_1||S_2$

Proposição 4.3.1. *Seja* $\mathbf{L}$ *um vetor de linguagens em* Σ*. Então*

$$B_\Sigma(\mathbf{L}) = \emptyset \Longleftrightarrow B_\Sigma(P_s(\mathbf{L})) = \emptyset$$

onde $P_s : \Sigma \rightarrow \Sigma_s$*.*

Segundo os autores, um estado de *deadlock* potencial é alcançável se e somente se o estado correspondente na projeção do sistema composto é alcançável [Abdelwahed e Wonham, 2003a], [Abdelwahed, 2002]. O teste de alcançabilidade sobre o sistema projetado, $P_s(B_\Sigma(\mathbf{L}))$, pode aumentar bastante a eficiência, para o caso de sistemas fracamente acoplados, dado que $P_s(B_\Sigma(\mathbf{L})) = B_\Sigma(P_s(\mathbf{L}))$.

Como existe um conjunto de eventos compartilhados, a evolução nas linguagens de $\mathbf{L}$ deve ser sincronizada no que diz respeito a esses eventos compartilhados. Por exemplo, para atingir o estado $(2, 0)$ é necessária a ocorrência de uma cadeia em S_1 que leve do estado inicial até o estado 2, e que S_2 atinja o estado 0. Nos autômatos do exemplo anterior (Figura 4.9), para que S_2 atinja o estado 0 alguma cadeia do conjunto $\{(71\ 72\ 81\ 82\ 73\ 74)^*\}$ deverá ocorrer. O teste que tem que ser feito para verificar se $(2, 0)$ é alcançável, é avaliar se é possível atingir o estado 2 de S_1 pela ocorrência de alguma cadeia cuja projeção no conjunto de eventos compartilhados seja $\{(71\ 72\ 73\ 74)^*\}$. No caso do par $(2, 0)$, pode-se verificar que ele não é alcançável.

A ocorrência de eventos locais em S_1 e S_2 é livre ou seja, esses eventos podem estar intercalados com os eventos compartilhados, de forma diversa. O importante é que os eventos compartilhados apareçam na mesma ordem e mesma quantidade em alguma cadeia que leve ao estado 2 de S_2 que em alguma cadeia que leve ao estado 0 de S_1. Dessa forma, o teste de alcançabilidade pode ser feito para todos os pares potencialmente bloqueantes. Aqueles pares que sobreviverem ao teste podem ser considerados alcançáveis e portanto, estados de *deadlock*.

4.3.1.2 Detecção de *livelock*

Um estado de *livelock* apresenta um conjunto não-vazio de eventos ativos e pode portanto, atingir outros estados (possivelmente ele mesmo). A dificuldade de detectar *livelock* está na sua caracterização. Particularmente, um estado de *livelock* pode ser definido recursivamente como um estado que tem um domínio alcançável consistindo apenas de estados de *deadlock* ou *livelock*. A existência de estados de *livelock*, então, depende da existência de estados de *deadlock*. Portanto, a detecção de *deadlock* deve ser feita primeiro.

A detecção de *livelock* pode ser simplificada considerando apenas os estados que podem atingir estados de *livelock*. Estados que podem atingir somente estados de *deadlock* podem ser considerados bloqueados. Tais estados bloqueados serão referenciados como estados de *semi-livelock*. A existência de estados de *semi-livelock* está relacionada à existência de estados

de *deadlock*. Assumindo que o sistema seja livre de *deadlock*, só é necessário considerar os estados de *livelock*.

Assumindo-se espaço de estados finito, a caracterização recursiva do *livelock* leva à conclusão de que qualquer estado de *livelock* deve existir em um laço, ou mais precisamente um *clique*, no sistema global.

Seja $A = (\Sigma, Q, \delta, q_0, Q_m)$ um autômato livre de *deadlock*. Um estado $q \in Q$ é um estado de *livelock* se e somente se,

$$\varrho_A(q) \neq \emptyset \qquad \wedge \qquad \varrho_A(q) \cap Q_m = \emptyset \qquad \wedge \qquad (\forall q' \in \varrho_A(q)) \quad \varrho_A(q') \neq \emptyset.$$

As duas primeiras condições caracterizam também estados de *semi-livelock*. A terceira condição distingue os estados de *livelock* como sendo aqueles que levam a outros estados de *livelock*. Portanto, o domínio alcançável desses estados é um conjunto não-vazio de estados de *livelock*. Um autômato é dito livre de *livelock* se não contiver nenhum estado de *livelock*.

Seja X um subconjunto não-vazio de Q em A. Seja δ^X a restrição de δ aos estados de X e seja $\varrho_A^X : X \to 2^X$ o mapa de alcançabilidade que associa cada estado $x \in X$ aos estados de X que são alcançáveis a partir de x através de transições em δ^X. O conjunto X é chamado clique em A se todo estado em X pode alcançar qualquer outro estado em X através de transições em δ^X, ou seja,

$$(\forall x, x' \in X) \quad x' \in \varrho_A^X(x).$$

Um estado $x_0 \in X$ é chamado estado de entrada para o clique X se x_0 for estado inicial ou

$$(\exists q \in Q - X)(\exists \sigma \in \Sigma) \quad \delta(q, \sigma) = x_0.$$

Nesse caso, a transição (q, σ, x_0) é dita ser uma transição de entrada para o clique X. Um estado $x_m \in X$ é chamado estado de saída em X se x_m for estado marcado ou se existe uma transição (q_m, σ, y) com $q \in Q - X$. Esta transição é chamada de transição de saída do clique. X_0 é o conjunto de estados de entrada e X_m é o conjunto de estados de saída. Um sistema só pode entrar no clique através de um estado de entrada e então pode ficar indefinidamente no clique ou sair dele através de um estado de saída. A tupla (X, X_0, X_m) é um estrutura clique em A.

Um clique é dito acessível se qualquer (e portanto todos) de seus estados for acessível ou seja, o clique tem pelo menos um estado de entrada. É dito co-acessível se qualquer (e portanto todos) seus estados forem co-acessíveis ou seja, todo clique tem pelo menos um estado de saída. Um clique é máximo se não há outro clique X' em A tal que $X \subseteq X'$, $X_0 \subseteq X_0'$ e $X_m \subseteq X_m'$. Um clique é terminal se todo estado acessível a partir do clique está no clique ou seja $\varrho_A(X) = X$.

Exemplo 4.3.3. *Na Figura 4.11, existem quatro cliques,* $\{0, 1, 2\}$, $\{1, 2\}$, $\{2\}$ *e* $\{3, 4\}$. *Todos os cliques são acessíveis e apenas o clique* $\{3, 4\}$ *não é co-acessível.*

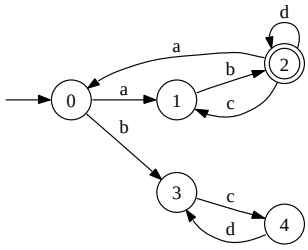


Figura 4.11: Exemplo 4.3.3 - Clique

Os cliques $\{0, 1, 2\}$ *e* $\{3, 4\}$ *são máximos mas apenas o clique* $\{3, 4\}$ *é terminal.*

Proposição 4.3.2. *Assumindo que A é livre de deadlock, então A é livre de livelock se e somente se todo clique terminal em A contiver um estado marcado.*

Portanto, estados de *livelock* podem ser detectados testando-se apenas o conjunto de cliques terminais do sistema. O resultado apresentado na Proposição 4.3.2 pode ser usado para o desenvolvimento de um procedimento detecta-primeiro para detecção de *livelock* em sistemas compostos. Mais detalhes sobre o procedimento podem ser encontrados em [Abdelwahed, 2002].

4.3.2 Conjunto de Conflitos Certos e Abordagem Incremental

Malik [2004] propõe um método para facilitar a detecção de conflito em sistemas modulares, a partir de uma caracterização do conflito. O conjunto de conflitos certos[2] de uma linguagem é um conjunto de comportamentos que pode ser tratado de forma especial para verificação, melhorando consideravelmente a eficiência da verificação.

Depois de caracterizado o conjunto de conflitos certos, a abordagem incremental é utilizada para testar se o comportamento do sistema é conflitante sem construir o produto síncrono completo. São propostas abstrações, que são modelos simplificados que preservam o conflito, de forma que o teste do conflito se torna mais simples.

O grande problema desta abordagem é que para o caso de sistema não-conflitante ainda é necessário compor todos os subsistemas para concluir que ele é não-conflitante. Além disso, é necessário conhecer de início algum subsistema (ou composição de subsistemas) que seja bloqueante.

4.3.2.1 Conjunto de Conflitos Certos

Em muitos casos é suficiente analisar subsistemas do sistema composto para estabelecer que ele satisfaz uma propriedade de interesse [Åkesson et al., 2002], [Brandin et al., 2004].

[2] do inglês *set of certain conflicts*

Entretanto, a propriedade do conflito é mais sofisticada. Mesmo que cada subsistema individual esteja livre de bloqueio, o sistema composto pelos subsistemas pode ser bloqueante. O exemplo apresentado na Figura 4.12 ilustra um caso em que os subsistemas são não bloqueantes. Entretanto, sua composição é conflitante. Este exemplo será utilizado no decorrer dessa subseção para ilustrar alguns conceitos.

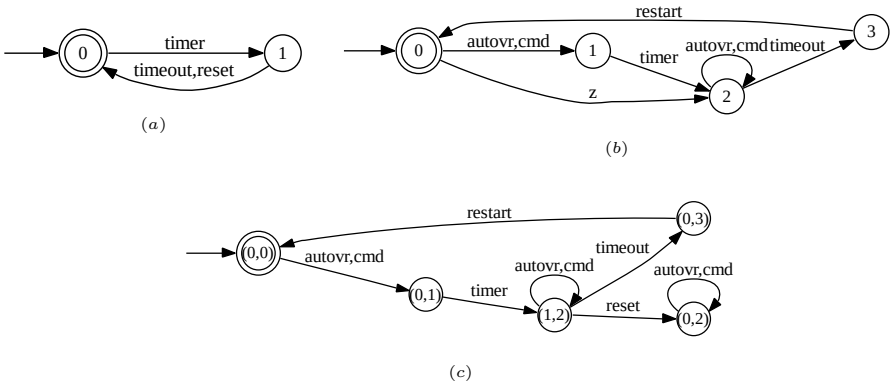


Figura 4.12: Exemplo de subsistemas não-bloqueantes cuja composição síncrona é bloqueante: (a) G_1; (b) G_2; (c) $G = G_1||G_2$.

Em sistemas complexos, onde diversos subsistemas executam em paralelo, o comportamento do sistema global (G sobre Σ) é obtido pelo produto síncrono dos subsistemas componentes (G_i sobre Σ_i, para $i = \{1, \cdots, n\}$). Para tanto, os alfabetos Σ_i de G_i são estendidos para o alfabeto comum Σ usando a projeção inversa. Considerando o SED estendido, é interessante diferenciar quais eventos caracterizam o comportamento e quais eventos foram introduzidos pela projeção inversa. Portanto, um conjunto de eventos $\Omega \subseteq \Sigma$ é *adequado* para uma linguagem $L \subseteq \Sigma^*$ se $P_\Omega^{-1}(P_\Omega(L)) = L$, isto é, se a linguagem pode ser reconstruída a partir de sua projeção. Ω é dito adequado para um SED G se Ω é adequado para $\mathcal{L}(G)$ e $\mathcal{L}_m(G)$. Apenas os eventos que caracterizam o comportamento do sistema são mostrados nas figuras. Subentende-se que os eventos que não aparecem no SED estão em auto-laços em todos os estados do SED.

Considere que G (Figura 4.12(c)) faz parte de um sistemas composto. Se for possível mostrar que todos os componentes do sistema aceitam a seqüência *cmd timer reset* que leva ao estado não co-acessível $(0, 2)$, pode-se concluir que o sistema composto é bloqueante. Ainda neste exemplo, assume-se que os eventos *timer* e *reset* nunca são desabilitados pelos outros componentes. Nesse caso, o sistema é bloqueante se for possível para G entrar no estado $(0, 1)$. Se G está nesse estado, o sistema composto pode executar a seqüência *timer reset* e levar G para o estado $(0, 2)$. Portanto, os estados $(0, 1)$, $(0, 2)$ e $(1, 2)$ são considerados estados de conflitos certos. Qualquer sistema não-conflitante com G deve impedi-lo de entrar nesses estados. Em seguida define-se o conjunto de conflitos certos, usando linguagens.

Seja um SED G que interage com outros subsistemas. O conjunto de eventos Σ de G deve ser dividido em um conjunto Υ de eventos que são utilizados somente por G e são irrelevantes para os outros sistemas e um conjunto Ω de eventos que são compartilhados entre G e outros

componentes, tal que $\Sigma = \Omega \cup \Upsilon$.

Exemplo 4.3.4. *Sejam o SED G_1 (Figura 4.12(a)), e os conjuntos de eventos $\Upsilon = \{autovr, cmd, restart\}$ e $\Omega = \{timeout, reset, timer\}$. Ω é adequado para G_1.*

Se G é bloqueante, então existe um conjunto de cadeias aceitas por G tal que, quando aceitas por outro SED H executando em paralelo com G, torna G e H conflitantes. Estas cadeias constituem o conjunto de conflitos certos.

Definição 4.3.1. *Sejam G um SED em Σ e $\Omega \subseteq \Sigma$. Uma cadeia $s \in \Omega^*$ é dita Ω-bloqueante e.r.a G se, para todo SED H em Σ tal que Ω seja adequado para H e $s \in \mathcal{L}(H)$, os SEDs G e H sejam conflitantes. Caso contrário, s é dito Ω-não-bloqueante e.r.a G*

Ω-bloqueio é uma propriedade de cadeias. Para s ser Ω-bloqueante, todos os SEDs H com as propriedades mencionadas devem ser conflitantes com G.

Definição 4.3.2. *Sejam G um SED sobre Σ e $\Omega \subseteq \Sigma$,*

$$\textsc{Conf}(G, \Omega) = \{s \in \Omega^* | s \text{ é } \Omega\text{-bloqueante e.r.a } G\};$$

$$\text{NConf}(G, \Omega) = \{s \in \Omega^* | s \text{ é } \Omega\text{-não-bloqueante e.r.a } G\}.$$

$\textsc{Conf}(G, \Omega)$ é o conjunto de conflitos certos de G e.r.a Ω. Seu complemento, $\text{NConf}(G, \Omega)$ é o conjunto de todas as cadeias s de eventos compartilhados para os quais existe outro sistema que aceita s e é não-conflitante com G.

Exemplo 4.3.5. *Sejam G_1 o SED representado na Figura 4.12(c) e $\Upsilon = \{reset, timer\}$. Então, todo SED H que tiver os eventos de Υ em auto-laços em todos os estados e aceitar os eventos autovr ou cmd deverá ser conflitante com G. Portanto,*

$$\textsc{Conf}(G_1, \Omega) = \{autovr, cmd\}\Omega^*.$$

Pode-se dizer que um SED H é não-conflitante com G se seu comportamento está contido em $\text{NConf}(G, \Omega)$.

Proposição 4.3.3. *Sejam G e H dois SEDs sobre Σ e $\Omega \subseteq \Sigma$ adequado e.r.a H. Se G e H são não-conflitantes então:*

$$P_\Omega(\mathcal{L}(H)) \subseteq P_\Omega^{-1}\ \text{NConf}(G, \Omega).$$

Portanto, seja $G = G_1||G_2||\cdots||G_n$. Se o comportamento de $G_2||\cdots||G_n$ não estiver contido na linguagem $\text{NConf}(G_1, \Omega)$ isto é,

$$\mathcal{L}(G_2||\cdots||G_n) \nsubseteq (P_\Omega^{-1}\text{NConf}(G_1, \Omega)),$$

então o sistema composto é conflitante.

Se $\mathcal{L}(G_2||\cdots||G_n) \subseteq P_\Omega^{-1}(\text{NConf}(G_1,\Omega))$, nada se pode concluir a respeito da existência ou não de conflito em G. Entretanto, a informação do conjunto de conflitos certos pode ser usada para simplificar a análise do conflito. O componente G_1 pode ser substituído por um modelo mais simples, G_1' e a análise do conflito pode ser feita pelo cálculo de $G_1'||G_2||\cdots||G_n$. G_1' deve preservar todas as possibilidades de G_1 de estar em conflito com os outros componentes do sistema.

Proposição 4.3.4. *Dois SEDs G_1 e G_2 sobre Σ são chamados conflito-equivalentes e.r.a $\Omega \subseteq \Sigma$ se, para todo SED H tal que Ω seja adequado para H, G_1 e H são não-conflitantes se e somente se G_2 e H são não-conflitantes*

Exemplo 4.3.6. *Seja G_1 o SED representado na Figura 4.13(a). O SED G_1' (Figura 4.13(b)) é conflito-equivalente a G_1.*

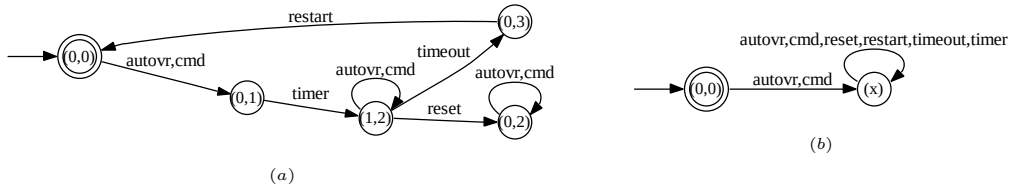


Figura 4.13: Exemplo 4.3.6: (a) G_1 (b) G_1', conflito-equivalente a G_1

Qualquer SED H, que não use os eventos timer e cmd, é conflitante tanto com G_1 como com G_1' se e somente eles aceitam algum dos eventos autovr e cmd. Portanto, G_1 pode ser substituído por G_1' antes de testar se os subsistemas são conflitantes.

Para obter a simplificação, agrupam-se os estados que representam conflitos certos em um único estado, onde todos os eventos são possíveis. As duas Proposições que se seguem fundamentam esta idéia.

Proposição 4.3.5. *Sejam G um SED sobre Σ e $\Omega \subseteq \Sigma$. G é conflito-equivalente a G' e.r.a Ω se*

$$\mathcal{L}(G') = \mathcal{L}(G) \cup P_\Omega^{-1}(\text{Conf}(G,\Omega))$$

$$\mathcal{L}_m(G') = \mathcal{L}_m(G).$$

Portanto, cadeias do conjunto de conflitos certos podem ser adicionadas à linguagem $\mathcal{L}(G)$ de um SED sem afetar seus possíveis conflitos.

Proposição 4.3.6. *Sejam G um SED sobre Σ e $\Omega \subseteq \Sigma$. G é conflito-equivalente a G' e.r.a Ω se*

$$\mathcal{L}(G') = \mathcal{L}(G)$$

$$\mathcal{L}_m(G') = \mathcal{L}_m(G) \cap P_\Omega^{-1}(\text{NConf}(G,\Omega)).$$

Portanto, como os conflitos certos não representam tarefas completas, eles podem ser removidos da linguagem $\mathcal{L}_m(G)$. O subsistema simplificado G_1' pode ser obtido pela aplicação da Proposição 4.3.6 seguida da aplicação da Proposição 4.3.5.

Em paralelo com [Malik, 2004], os autores formalizaram alguns conceitos como o conceito do conjunto de conflitos certos usando a teoria de álgebra de processos [Malik et al., 2004]. Flordal e Malik [2006] propõem procedimentos para redução dos autômatos utilizando conceitos da teoria de álgebra de processos. A Seção 4.3.3 apresenta esses resultados.

Em seguida, apresenta-se a abordagem incremental, introduzida por Brandin et al. [2000], para verificação de propriedades. Esse método é usado em conjunto com o conceito de conflitos certos para detectar conflito em sistemas compostos.

4.3.2.2 Abordagem Incremental

Em geral, o comportamento do sistema a ser verificado não é descrito completamente, mas apenas aspectos particulares são considerados. Sejam um sistema e uma propriedade a ser verificada. Quando se conclui que não é possível mostrar que o comportamento do sistema satisfaz a propriedade, duas interpretações podem ser dadas. São elas:

- o sistema considerado não satisfaz a propriedade;
- o sistema satisfaz a propriedade, mas a descrição de seu comportamento não é suficientemente completa para permitir que se mostre que a propriedade é satisfeita.

O questionamento que surge é, então: será que a descrição do sistema pode ser refinada de alguma forma para mostrar que a propriedade é satisfeita ou não? Se a resposta a esse primeiro questionamento for sim, surge um segundo questionamento: Como fazer?

Usando resultados da TCS é possível: (i) determinar quando comportamentos adicionais, acoplados ao comportamento original do sistema, garantem que a propriedade é sempre satisfeita; (ii) sintetizar comportamentos que, quando acoplados ao sistema original, garantem a satisfação da propriedade.

As propriedades podem ser descritas modularmente, por um conjunto de autômatos. Neste caso, esta abordagem pode ser considerada poderosa para verificação, que permite que o comportamento do sistema seja aumentado incrementalmente, com o objetivo de verificar se especificações comportamentais são satisfeitas, aumentando a eficiência computacional. O procedimento incremental trata da verificação de propriedades de segurança e verificação de controlabilidade como um conjunto de passos de verificação sucessivos. As linguagens consideradas são prefixo-fechadas. Depois de cada passo, ou os requisitos comportamentais são verificados ou contra-exemplos são obtidos. Esses conta-exemplos, juntamente com a heurística, são a base para o aumento do subsistema, pela composição de outros subsistemas. Só será apresentado o procedimento incremental para testar propriedades de segurança.

Antes de apresentar o procedimento de verificação incremental, algumas idéias e a notação utilizada são introduzidas. Sejam I e J conjuntos de índices. Sejam um sistema G composto de subsistemas G_i, $i \in I$, e um requisito comportamental R a ser satisfeito por G. R é composto de autômatos R_j, com $j \in J$. O comportamento de um sistema satisfaz requisitos comportamentais quando está contido ou é igual ao requisito ou seja, G satisfaz R se $\mathcal{L}(G) \subseteq \mathcal{L}(R)$, ou seja, se toda cadeia de eventos aceita pelo autômato G é também aceita pelo autômato R.

A Proposição 4.3.7 apresentada a seguir formaliza a idéia de que a composição de subsistemas sempre restringe o comportamento de um sistema.

Proposição 4.3.7. *Sejam* $K, L \subseteq \Sigma^*$ *e* $L = L_1 \cap L_2$ *se*

$$L_1 \subseteq K \Longrightarrow L \subseteq K.$$

Portanto, pode-se construir o produto síncrono de G e R e checar se em cada combinação de estados alcançáveis existe um evento possível em G que não seja possível em R. Se existir, G não satisfaz R.

Exemplo 4.3.7. *Sejam um alfabeto* Σ *e duas linguagens* $L_1, L_2 \subseteq \Sigma^*$. *A linguagem* $L = L_1 \cap L_2$ *está contida em* L_1 *e em* L_2, *como mostrado na Figura 4.14. Portanto, se a linguagem*

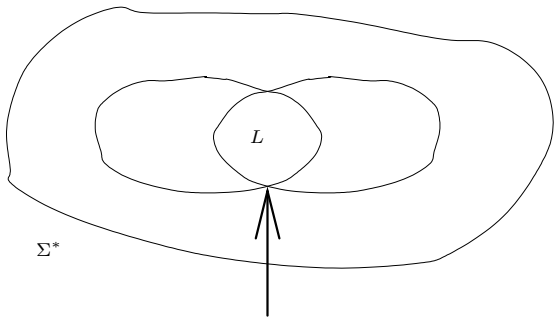


Figura 4.14: Exemplo 4.3.7: $L = L_1 \cap L_2$

L_1 *atende a um requisito de inclusão comportamental* ($L_1 \subseteq \mathcal{L}(R)$), L *também atende* ($L \subseteq \mathcal{L}(R)$).

Como conseqüência, se um subsistema E de G atende a um requisito de inclusão comportamental, G também atende.

Sejam G_1 e G_2 autômatos definidos sobre o alfabeto Σ_1 e Σ_2 respectivamente. Para que $\mathcal{L}(G_1||G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$, os alfabetos dos dois subsistemas devem ser estendidos através da inserção de autolaços em todos os estados, de forma que os alfabetos de G_1 e G_2 fiquem iguais a $\Sigma = \Sigma_1 \cup \Sigma_2$. Os eventos podem ser controláveis ou não-controláveis, $\Sigma = \Sigma_c \cup \Sigma_{uc}$. Além disto, podem ser classificados em relevantes e irrelevantes. Para uma linguagem $L \subseteq \Sigma$,

o evento $\sigma \in \Sigma$ é irrelevante para L se $\forall s, t \in \Sigma^*$

$$st \in L \quad \text{se e somente se} \quad s\sigma t \in L.$$

Caso contrário, σ é chamado relevante. O conjunto de eventos relevantes é definido como

$$\text{rel } L = \{\sigma \in \Sigma \,|\, \sigma \text{ é relevante para L}\}.$$

Considere G_{Σ^*} como um elemente neutro, tal que $\mathcal{L}(G_{\Sigma^*}) = \Sigma^*$ e $\mathcal{L}(G||G_{\Sigma^*}) = \mathcal{L}(G)$ para qualquer G. O procedimento incremental para verificação da inclusão comportamental é feito partindo do elemento neutro representando o modelo para o sistema. Esse modelo vai sendo refinado pela composição de subsistemas de G que não aceitem uma cadeia que seja contra-exemplo. O contra-exemplo é alguma cadeia de $\mathcal{L}(G')$ que não pertença a $\mathcal{L}(R)$. A obtenção de um contra-exemplo que seja aceito por todos os subsistemas de G implica G não atender o requisito comportamental R.

O procedimento incremental será ilustrado através de um exemplo.

Exemplo 4.3.8. *Sejam um sistema G, composto das subplantas M_1, M_2, $Armazem$ e $Reparo$, e uma restrição R, também chamada de $AltInicio$. Verificar se $G = M_1||M_2||Armazem||Reparo$ satisfaz restrições de comportamento R.*

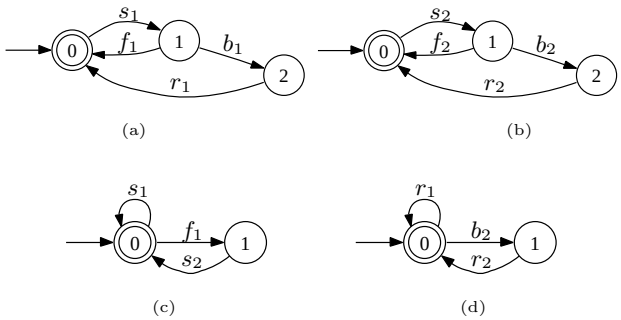


Figura 4.15: Exemplo 4.3.8: (a) Máquina M_1; (b) Máquina M_2; (c) *Armazem*; (d) *Reparo*.

A restrição $AltInicio$ (Figura 4.16) pode ser interpretada como a verificação de que as máquinas começam a operar alternadamente, ou seja, verificar se

$$\mathcal{L}(M_1||M_2||Armazem||Reparo) \subseteq \mathcal{L}(AltInicio).$$

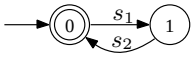


Figura 4.16: Exemplo 4.3.8: restrição $AltInicio$

Considere G' um subconjunto de G.

- *se G' satisfaz $R \Rightarrow$ G também satisfaz*

- *se G' não satisfaz $R \Rightarrow$ busca de contra-exemplo $s \in \mathcal{L}(G')$ tal que $s \notin \mathcal{L}(R)$*

Pela Proposição 4.3.7, deve-se encontrar G' de G que satisfaça $AltInicio$. Inicialmente, escolhe-se o elementro neutro e, portanto $G' = G_{\Sigma^*}$. Verifica-se que $\Sigma^* \nsubseteq \mathcal{L}(AltInicio)$ (Figura 4.17).

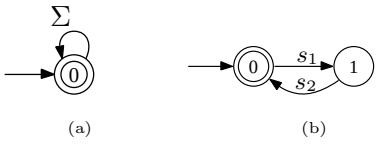


Figura 4.17: Exemplo 4.3.8 - Primeiro passo do procedimento incremental, busca de contra-exemplo: (a) $G' = G_{\Sigma^*}$; (b) $AltInicio$.

A cadeia s_2 é possível em $\mathcal{L}(G')$ e não é possível em $\mathcal{L}(AltInicio)$ e portanto, é considerada um contra-exemplo. Verifica-se se algum dos subsistemas do sistema G (M_1, M_2, Armazem ou Reparo) não aceitam s_2. Caso algum subsistema não aceite o contra-exemplo, esse subsistema é composto com G', refinando o modelo.

O subsistema Armazem não aceita a cadeia s_2. Portanto, no próximo passo do procedimento $G' = G_{\Sigma^*}||Armazem = Armazem$ e deve-se verificar se $\mathcal{L}(Armazem) \subseteq \mathcal{L}(AltInicio)$ (Figura 4.18).

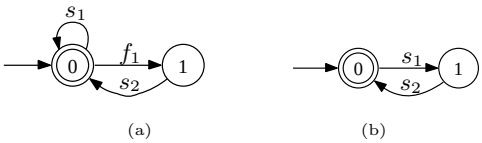


Figura 4.18: Exemplo 4.3.8 - Segundo passo do procedimento incremental, busca de contra-exemplo: (a) $G' = Armazem$; (b) $AltInicio$.

Um contra-exemplo para esse G' é a cadeia $f_1 s_2$, aceita por G', mas não por $AltInicio$. Por inspeção, pode-se perceber que o subsistema M_1 não aceita esta cadeia.

O modelo G', utilizado para a verificação, é então refinado, no próximo passo do procedimento, pela composição de M_1, $G' = G'||M_1 = Armazem||M_1$. Ao testar se $\mathcal{L}(G)' \subseteq \mathcal{L}(AltInicio)$ pode-se obter a cadeia $s_1 b_1 r_1 s_1$. Todos os subsistemas aceitam essa cadeia. Pode-se concluir que G', e portanto G, não satisfaz o requisito $AltInicio$.

Para chegar à conclusão de que o sistema G não atende o requisito $AltInicio$ foi necessário compor apenas dois subsistemas de um total de quatro subsistemas componentes de G.

Para o caso em que há vários requisitos K_i, pode-se derivar o seguinte resultado.

Proposição 4.3.8. *Dado $K, L \subseteq \Sigma^*$, $K = K_1 \cap K_2$, tem-se que*

$$L \subseteq K \iff L \subseteq K_1 \wedge L \subseteq K_2.$$

Ou seja, se G satisfaz $R_j, \forall j \in \{1, \ldots, m\}$, G satisfaz R. Basta provar que, para cada R_j de R, existe algum G' de G que satisfaz a propriedade. O procedimento incremental é aplicado para comprovar que para cada R_j existe algum G' tal que $\mathcal{L}(G') \subseteq \mathcal{L}(R)$ e portanto que $\mathcal{L}(G) \subseteq \mathcal{L}(R)$.

Caso mais de um subsistema (restrição) aceite o contra-exemplo, deve-se escolher entre eles qual ou quais devem ser compostos com G' (R'). Algumas heurísticas são propostas em [Brandin et al., 2004]. Segundo os autores, não há uma heurística que seja sempre melhor. Eles sugerem que se utilizem várias heurísticas, limitando o tempo de computação delas com um valor empírico.

Esta abordagem foi testada pelos autores em vários exemplos industriais. A partir dos exemplos foi possível perceber que é possível resolver alguns problemas em tempo polinomial, apesar desta metodologia ter complexidade exponencial, no pior caso.

Como mencionado anteriormente, a abordagem incremental pode ser usada em conjunto com a idéia do conjunto de conflitos certos para detectar o conflito com maior eficiência computacional.

Para concluir que há conflito entre um subsistema G_1 de G e o restante do sistema $G_2||\cdots||G_n$, deve-se verificar que

$$\mathcal{L}(G_2||\cdots||G_n) \nsubseteq \mathrm{NConf}(G_1, \Omega).$$

Este teste pode ser feito usando o teste de inclusão comportamental descrito nesta subseção.

4.3.3 Conflito Equivalência

Flordal e Malik [2006] utilizam resultados da álgebra de processos para obter abstrações que preservam conflito. Esse artigo apresenta um conjunto de procedimentos para redução dos autômatos de forma a garantir a propriedade de conflito-equivalência (Definição 4.3.4). A notação utilizada para representar uma relação de transição é $q \xrightarrow{a} q'$, onde q e q' representam estados e a é a cadeia de eventos que leva o sistema do estado q para o estado q'. Dois outros eventos são utilizados, o *evento silencioso* $\tau \notin \Sigma$ o *evento de término* $\omega \in \Sigma$. O evento τ não está no alfabeto Σ a menos que explicitamente dito, $\Sigma_\tau = \Sigma \cup \{\tau\}$. Seja $a = \sigma_1\sigma_2 \ldots \sigma_n$, a transição dupla, $q \stackrel{a}{\Rightarrow} q'$, denota a existência de uma cadeia $t \in \tau^*\sigma_1\tau^*\sigma_2\tau^* \ldots \tau^*\sigma_n\tau^*$ tal que $q \xrightarrow{t} q'$. A transição simples $\xrightarrow{t}$ denota, portanto, a existência de um caminho com exatamente os eventos de t enquanto a transição dupla $\stackrel{a}{\Rightarrow}$ denota um caminho com um número arbitrário de eventos τ intercalados.

A representação dos estados marcados do sistema é através do evento ω onde o estado que seria marcado possui uma transição rotulada com o evento ω para um estado que não possui nenhuma transição de saída, fazendo de ω o último evento de qualquer execução.

Um autômato $G = (Q, \Sigma, \rightarrow, Q^i, Q^m)$ aceita uma cadeia $s \in \Sigma^*$ se $Q^i \stackrel{s}{\Rightarrow}$, e o conjunto de eventos ativos no estado q é definido como $\Sigma(q) = \{\sigma \in \Sigma | q \stackrel{\sigma}{\Rightarrow}\}$. Na composição síncrona, os eventos compartilhados e também ω devem ser sincronizados, enquanto os outros eventos, incluindo τ, são executados independentemente.

Usando a notação apresentada acima, o conceito de não-bloqueio é apresentado a seguir:

Definição 4.3.3. *Um autômato $G = (Q, \Sigma, \rightarrow, Q^i, Q^m)$ é não-bloqueante se, para todo estado $q \in Q$ e toda cadeia $s \in \Sigma^*$ tal que $Q^i \stackrel{s}{\Rightarrow} q$, existe um cadeia $t \in \Sigma^*$ tal que $q \stackrel{t\omega}{\Rightarrow}$.*

Relembrando, dois autômatos são não-conflitantes se sua composição síncrona é não-bloqueante. O conceito de conflito-equivalência foi desenvolvido para caracterizar autômatos que possam ser usados indistintamente para teste de conflito.

Definição 4.3.4. *Dois autômatos G_1 e G_2 são conflito-equivalentes, $G_1 \simeq_{conf} G_2$, se, para qualquer teste T, $G_1||T$ é não-bloqueante se e somente se $G_2||T$ for não-bloqueante.*

Sejam G um autômato e s uma cadeia desse autômato. A linguagem $\mathcal{C} \subset \Sigma^*\omega$ é chamada de *sufixo não-conflitante* para s em G se para todo teste T tal que T e G sejam não-conflitantes e $Q^i \stackrel{s}{\Rightarrow} q$, existe um sufixo $t \in \mathcal{C}$ tal que $q \stackrel{t}{\Rightarrow}$. A semântica dos sufixos não-conflitantes é dada a seguir.

$$CC(G) = \{(s, \mathcal{C}) \in \Sigma^* \times \mathbb{P}(\Sigma^*\omega) |\ \mathcal{C} \text{ é sufixo não-conflitante e.r.a } s \in \mathcal{L}(G)\}.$$

O par $(s, \mathcal{C})$ representa, na definição de $CC(G)$, a noção de que um autômato (teste) que aceite s deve aceitar pelo menos um dos sufixos em $\mathcal{C}$ depois de s para ser não-conflitante com G. Dois autômatos são conflito-equivalentes se e somente se a semântica de seus sufixos não-conflitantes coincide.

Teorema 4.3.2. *Sejam dois autômatos G_1 e G_2. $G_1 \simeq_{conf} G_2$ se e somente se $CC(G_1) = CC(G_2)$.*

A relação de equivalência $\sim_{inc}$ é apresentada a seguir e utilizada no procedimento de obtenção das abstrações. A relação $\sim_{inc}$ relaciona estados que podem ser alcançados da mesma forma.

Definição 4.3.5. *Seja $G = (Q, \Sigma, \rightarrow, Q^i, Q^m)$ um autômato. A relação binária $\sim_{inc} \subseteq Q \times Q$ é definida tal que $q \sim_{inc} q'$ se:*

$$\begin{aligned} Q^i \stackrel{\epsilon}{\Rightarrow} q \quad &\iff \quad Q^i \stackrel{\epsilon}{\Rightarrow} q'; \\ \forall p \in Q, \forall \sigma \in \Sigma : p \stackrel{\sigma}{\Rightarrow} q \quad &\iff \quad p \stackrel{\sigma}{\Rightarrow} q'. \end{aligned}$$

O ato de *esconder* eventos consiste em transformar um evento σ no evento silencioso τ. O resultado de esconder eventos $\Upsilon \subseteq \Sigma$ de G é o autômato $G \backslash \Upsilon$ obtido pela substituição de cada transição $q \xrightarrow{v} q'$ por $q \xrightarrow{\tau} q'$, com $v \in \Upsilon$ e removendo de Σ os eventos de Υ. A questão que se coloca é: como obter Υ de forma a manter a propriedade de conflito-equivalência no autômato resultante? Flordal e Malik [2006] discutem dois métodos possíveis.

4.3.3.1 Equivalência de Observação ou Bissimulação Fraca

A *equivalência de observação* ou bissimulação fraca é uma equivalência forte de autômatos não-determinísticos que considera apenas estados com a mesma estrutura e o mesmo futuro como equivalentes.

Definição 4.3.6. *Uma relação de equivalência $\sim$ é dita uma bissimulação fraca se $q \sim p$ implicar, para qualquer $s \in \Sigma^*$,*

$$\begin{aligned} &\text{se } p \overset{s}{\Rightarrow} p' \text{ então } \exists q' \text{ tal que } q \overset{s}{\Rightarrow} q' \text{ e } p' \sim q'; \\ &\text{se } q \overset{s}{\Rightarrow} q' \text{ então } \exists p' \text{ tal que } p \overset{s}{\Rightarrow} p' \text{ e } q' \sim p'. \end{aligned}$$

Dois autômatos G_1 e G_2 são *equivalentes de observação* se existe uma bissimulação fraca tal que para cada estado inicial q_1 de G_1 existe um estado inicial q_2 de G_2 tal que $q_1 \sim q_2$ e vice-versa [Flordal e Malik, 2006]. Autômatos que sejam equivalentes de observação são conflito-equivalentes [Malik et al., 2004].

4.3.3.2 Reduções que preservam Conflito-Equivalência

Há casos em que dois autômatos são conflito-equivalentes, mesmo não sendo equivalentes de observação. Flordal e Malik [2006] apresentam um conjunto de regras de redução que preservam a propriedade de conflito-equivalência. As regras são aplicadas localmente sobre alguns estados do autômato tentando identificar estados que sejam conflito-equivalentes (estados que não possam ser distinguidos pela propriedade de conflito-equivalência). Esses estados são, então agrupados, reduzindo o espaço de estados do sistema. As regras são descritas a seguir.

1. Eventos ativos: dois estados que são equivalentes em relação a $\sim_{inc}$ e possuem os mesmos eventos ativos são conflito-equivalentes.

Figura 4.19: Exemplo da aplicação da regra de eventos ativos: (a) G; (b) G'.

Na Figura 4.19, os estados 1 e 2 de G possuem os mesmos eventos de entrada α vindo do estado 0 e o evento β vindo do estado 1, estabelecendo a relação $\sim_{inc}$. Além disso,

os dois estados têm o mesmo evento ativo β. Portanto, a regra dos eventos ativos pode ser aplicada, e os estados 1 e 2 podem ser agrupados em um estado 1,2 como mostrado em G'.

2. Continuação silenciosa: dois estados que são equivalentes em relação a $\sim_{inc}$ e a partir do qual estados estáveis (que não possuem eventos τ ativos) podem ser alcançados por uma seqüência não-vazia de transições τ são conflito-equivalentes.

Figura 4.20: Exemplo da aplicação da regra de continuação silenciosa: (a) G; (b) G'.

Na Figura 4.20, os estados 0 e 1 de G são ambos iniciais uma vez que podem ser alcançados por um cadeia silenciosa, a partir do estado inicial 0. Portanto os estados 0 e 1 satisfazem $\sim_{inc}$ já que nenhum dos estados é alcançável por qualquer outro evento que não τ. Além disso, os dois estados podem alcançar o estado 3 que não tem eventos τ habilitados, pela execução de pelo menos uma transição silenciosa τ. Os estados 0 e 1 são conflito-equivalentes e podem ser agrupados em um estado 0,1 como mostrado em G'.

3. Conflitos certos: em um autômato H, todos os estados q tais que $Q^i \xrightarrow{s} q$ e $(s, \emptyset) \in CC(H)$ podem ser substituídos por um único estado bloqueante que não possui transições ativas.

Figura 4.21: Exemplo da aplicação da regra de conflitos certos: (a) H; (b) H'.

Na Fig. 4.21, a cadeia $\beta\beta$ leva ao estado bloqueante 5. Portanto, qualquer teste, para impedir o bloqueio, não pode aceitar a cadeia $\beta\beta$. Nesse caso, após a ocorrência da cadeia β, já não é possível evitar a ocorrência de $\beta\beta$ e portanto o autômato atinge um estado de conflito certo.

4.3.3.3 Verificação da Modularidade

Flordal e Malik [2006] apresentam um método para verificar se um conjunto de autômatos G_i é conflitante. As abstrações G_i' obtidas através das reduções descritas anteriormente são usadas no teste, no lugar dos autômatos originais. Como G_i e G_i' são conflito-equivalentes por construção, segue-se que o sistema obtido pela composição das abstrações será não-bloqueante se e somente se a composição dos autômatos originais for não-bloqueante.

Depois de feitas as reduções de G_i, a composição síncrona é obtida passo-a-passo, aplicando-se reduções intermediárias. Um subconjunto de autômatos é substituído pela sua composição síncrona H, e aqueles eventos considerados locais para H (não compartilhados com outros subsistemas) são *escondidos* (substituídos por τ). H é então substituído por $H\backslash\Upsilon$.

A seguir, apresenta-se um exemplo para ilustrar a aplicação da metodologia.

Exemplo 4.3.9. *Sejam o autômato H da Figura 4.21 e um autômato genérico G_1. Para detectar a existência de conflito entre os autômatos H e G_1, deve-se obter a composição síncrona $H||G_1$ e verificar se é não-bloqueante. O primeiro passo do procedimento apresentado em [Flordal e Malik, 2006] consiste em aplicar as reduções possíveis. Neste caso, a regra 3 pode ser aplicada a H e esse autômato pode então ser substituído por H' (Figura 4.21), que é conflito-equivalente a H e [$H||G_1$ não-bloqueante $\Longleftrightarrow$ $H'||G_1$ não-bloqueante]. O próximo passo consiste em verificar quais transições são rotuladas com eventos locais e substituir por τ. Para ilustrar esse passo, considere que o autômato genérico G_1 possui transições rotuladas com eventos locais (isto é, não compartilhados com os outros autômatos), que podem ser substituídos por τ, tal que $G_1' = G$ (Figura 4.20). Em seguida, processa-se reduções sobre os novos autômatos obtidos. Nesse caso, observa-se que $G_1' = G$ pode ser reduzido usando a regra 2 e G_1' pode ser substituído por G' da (Figura 4.20). No caso de mais de dois autômatos, composições intermediárias podem ser feitas e reduções e substituições por τ podem ser aplicadas à estas composições. Os dois passos são repetidos até que não seja mais possível colapsar estados e nem substituir eventos ou então até que a redução obtida seja satisfatória.*

A ordem em que os autômatos são escolhidos para serem compostos pode mudar o resultado das reduções, para melhor ou para pior. Em alguns casos, pode ser mais vantajoso construir pequenos subsistemas e depois compor. Em outros, pode ser melhor compor os autômatos um a um em um único sistema. Em [Flordal e Malik, 2006], a ordem da composição é definida baseada num procedimento em dois passos: (*i*) alguns candidatos são compostos para serem usados no próximo passo; (*ii*) a identificação do melhor candidato é realizada. Heurísticas são utilizadas no procedimento.

4.3.4 Resolução de Conflito

Dado que o conflito é um problema inerente às arquiteturas descentralizadas, vários trabalhos tratam formas alternativas de modelar, refinar e até projetar sistemas livres de conflito

por construção. Entre eles, cita-se [Wong et al., 2000], [Chen e Lafortune, 2000] e [Wong e Wonham, 1998]. Apesar dessas abordagens serem apresentadas brevemente nesta seção, esse não é o foco desta tese.

Segundo Chen e Lafortune [2000], o uso de prioridades é muito comum em aplicações reais para resolver situações de conflito. Esse fato motiva o uso de prioridades para evitar o conflito dos supervisores modulares. Alguns trabalhos apresentam esquemas de resolução de conflito usando prioridades, entre eles [Wong et al., 1995], [Wong et al., 2000] e [Chen e Lafortune, 2000].

Em [Wong et al., 1995] é apresentado um esquema para resolução de conflito pela introdução de uma interface entre um dos supervisores conflitantes e a planta. Esta idéia foi aprimorada e, em [Wong et al., 2000], um esquema em que cada supervisor local é dotado de uma interface é apresentado. Cada supervisor tem uma prioridade, que pode ser fixa, ou seja, que não depende da dinâmica do sistema, ou flexível, em que a prioridade é função da evolução do sistema. Na situação de conflito entre dois supervisores, o supervisor de prioridade mais alta assume o controle "sozinho" e a ação do supervisor de menor prioridade é suspensa e reativada de forma a evitar o conflito. Esse esquema é formalizado pela introdução de interfaces adequadas entre os supervisores e a planta (Figura 4.22), que fazem a mediação do fluxo de informação e ação de controle entre planta e supervisores. A suspensão de um supervisor S_i é realizada pela filtragem e interpretação apropriadas das decisões de controle tomadas por S_i. Sejam G a planta e inf_i o canal de informação entre planta e o supervisor modular S_i. A informação observada por S_i pode ser representada pelo SED $G_{i,hi}$. As decisões de controle tomadas por S_i são comunicadas, através de com_i, ao agente $S_{i,lo}$, que interpreta e aplica o controle a G. O efeito do controle é filtrado e informado a S_i através de $G_{i,hi}$. Nesse sentido, (inf_i, com_i) representam a interface entre S_i e G, que media o fluxo de informação de G para S_i e a ação de controle de S_i para G.

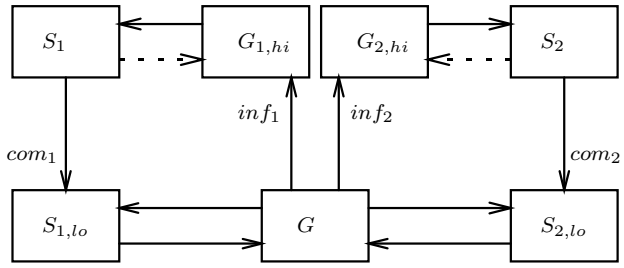


Figura 4.22: Esquema para resolução de conflito apresentada em [Wong et al., 2000]

Esta estrutura apresenta características hierárquicas em que o controle no baixo nível é induzido por um controle de alto nível. O estudo do esquema de prioridades passa a ser um estudo sobre as propriedades dos mapas repórteres, e condições suficientes sobre os mapas-repórteres, para que se garanta a resolução de conflito, são obtidas [Wong et al., 2000].

Wong e Wonham [1998] propõem um esquema de coordenação hierárquica em dois níveis (Figura 4.23).

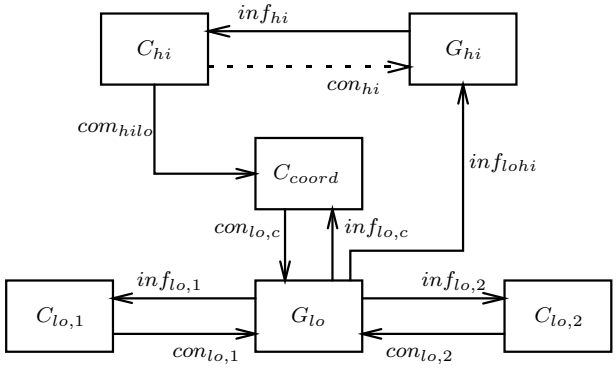


Figura 4.23: Esquema para resolução de conflito via coordenação hierárquica [Wong e Wonham, 1998]

O funcionamento do esquema é descrito a seguir. O comportamento da planta G_{lo} sob supervisão concorrente de $C_{lo,1}$ e $C_{lo,2}$ é filtrado através do canal de informação inf_{lohi}. O modelo de alto nível G_{hi} representa o comportamento de G_{lo}, depois de filtrado. O filtro remove a informação irrelevante para a coordenação e com isto simplifica o projeto do coordenador. O coordenador de alto nível C_{hi} vê o comportamento no baixo nível através da monitoração de G_{hi}. A situação crítica é detectada quando a execução de transições consideradas seguras por $C_{lo,1}$ e $C_{lo,2}$ vão causar o conflito. Essas transições são imediatamente desabilitadas por C_{hi} em G_{hi}. Esta desabilitação é virtual, assim como no controle hierárquico de Zhong e Wonham [1990]. O comando de desabilitação é enviado para o agente de coordenação C_{coord} que desabilita os eventos controláveis de baixo nível apropriados para prevenir que o conflito aconteça. O resultado da desabilitação é reportado a C_{hi} como a desabilitação (virtual) em G_{hi}. Condições necessárias e suficientes para que C_{hi} seja capaz de detectar e prevenir os potenciais conflitos em tempo hábil são apresentados em [Wong e Wonham, 1998].

Chen e Lafortune [2000] apresentam um esquema, denominado MCP[3] ou controle modular com prioridades (Figura 4.24). Nesse esquema, os supervisores individuais S_i geram ações de controle, denotadas por $\gamma_i(s)$, baseadas na cadeia de eventos s gerada pelo sistema G e nas especificações que o supervisor implementa. A função Σ_i^P associada a cada supervisor é chamada de função de prioridade. O coordenador, $\widehat{P}$, combina as ações de controle de acordo com os respectivos conjuntos de prioridades dos supervisores depois de ocorrida a cadeia s.

Se as funções de prioridades forem omitidas e $\widehat{P}$ for substituído pela conjunção, a arquitetura modular original é resgatada. Deve-se garantir que o efeito da suspensão (como resultado da ação do coordenador) de algum supervisor seja temporário e que este supervisor, ao voltar a atuar sobre a planta, funcionará corretamente. Algoritmos com os quais é possível sintetizar supervisores modulares não-conflitantes pela escolha adequada das funções de prioridade são apresentados. Detalhes sobre o coordenador e o projeto das funções de prioridade podem ser obtidos em [Chen e Lafortune, 2000].

[3]do inglês *modular control with priorities*

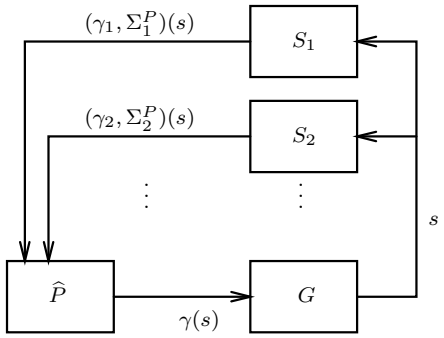


Figura 4.24: Esquema para resolução de conflito: MCP [Chen e Lafortune, 2000]

4.4 Síntese

Nesse capítulo, apresentaram-se abordagens que tratam do controle de sistemas concorrentes. Inicialmente, apresentou-se uma revisão da literatura sobre essas abordagens, na Seção 4.1. O controle modular local foi apresentado na Seção 4.2. Esta abordagem obtém redução da complexidade computacional do procedimento de síntese de supervisores, para o caso geral. Entretanto, o teste da modularidade, necessário para garantir desempenho equivalente ao desempenho da síntese monolítica, passa pela composição síncrona de todos os supervisores ficando, assim, sujeito à explosão de estados. Alguns trabalhos que tratam do problema do conflito foram também apresentados, na Seção 4.3.

Os resultados da tese são apresentados nos próximos três capítulos.

Capítulo 5

Verificação de Não-Conflito de Supervisores Modulares

Um problema importante a ser resolvido de forma a viabilizar a aplicação da teoria de controle supervisório a sistemas reais é a redução da complexidade do teste de não-conflito (ou teste da modularidade) de supervisores. Um grande esforço vem sendo feito pela comunidade de SEDs no sentido de tornar essa teoria realmente útil para tratar problemas reais.

Este capítulo apresenta um novo teste de não-conflito (ou teste da modularidade) realizado sobre abstrações dos supervisores. Condições suficientes para que as abstrações possam ser utilizadas no lugar dos supervisores para teste de não-conflito são apresentadas. O objetivo com este novo teste é reduzir a complexidade do teste original.

O capítulo é dividido em 5 seções. Na Seção 5.1 apresentam-se o teste de não-conflito clássico e o novo teste baseado em abstrações. Na Seção 5.2 discutem-se a operação de projeção natural e, especificamente, duas propriedades amplamente utilizadas neste trabalho. Na seção que segue apresentam-se os resultados obtidos para os casos de duas e m linguagens e, na Seção 5.4, apresentam-se os procedimentos para obtenção das abstrações com as propriedades desejadas, além de um exemplo ilustrativo. Por fim, na Seção 5.5, apresenta-se uma breve discussão dos resultados obtidos.

5.1 Teste de Não-Conflito

Uma forma de verificar se duas linguagens são conflitantes é aplicar o teste de não-conflito, ou teste da modularidade. Sejam S_j linguagens que implementam a ação dos supervisores que, por conveniência, são também chamados de S_j, para $j \in J = \{1, \ldots, m\}$. O teste consiste em verificar se a igualdade da equação (5.1) é satisfeita

$$\mathop{||}_{j=1}^{m} \overline{S_j} = \overline{\mathop{||}_{j=1}^{m} S_j}, \tag{5.1}$$

ou seja, verificar que, para todo prefixo "compartilhado"por duas ou mais linguagens, exista pelo menos uma cadeia "compartilhada"que o contenha. Este teste, como se pode observar na equação (5.1), requer a composição de todas as linguagens e, portanto, está suscetível à explosão do espaço de estados. O teste proposto neste capítulo, que substitui o teste clássico, é apresentado a seguir.

Sejam a linguagem S_j sobre o conjunto de eventos Σ_j ($S_j \subseteq \Sigma_j^*$), $\Sigma = \bigcup_{j=1}^{m} \Sigma_j$ o conjunto total de eventos, $\Sigma_r \subseteq \Sigma$ o conjunto de eventos relevantes e a projeção natural $\theta_j : \Sigma_j^* \rightarrow (\Sigma_j \cap \Sigma_r)^*$. Condições suficientes são apresentadas para que o teste sobre as abstrações $\theta_j(S_j)$ tenha o mesmo resultado que o teste sobre as linguagens originais, ou seja, para que:

$$\|_{j=1}^{m} \overline{\theta_j(S_j)} = \overline{\|_{j=1}^{m} \theta_j(S_j)} \Longleftrightarrow \|_{j=1}^{m} \overline{S_j} = \overline{\|_{j=1}^{m} S_j}.$$

5.2 Propriedades das Projeções Naturais

Nesta seção pretende-se discutir a projeção natural, incluindo seu processo de obtenção e duas propriedades que são amplamente utilizadas nesse trabalho, e sobre as quais se baseiam os resultados desta tese.

A projeção natural de Σ^* em Σ_i^*, foi previamente definida na Seção 2.1.3. Esta operação aplicada sobre uma linguagem L apaga os eventos de $(\Sigma - \Sigma_i)$ das cadeias de L. A projeção inversa $P_i^{-1}(L)$ completa o alfabeto de L com os eventos de $(\Sigma - \Sigma_i)$, o que corresponde a compor a linguagem original com $(\Sigma - \Sigma_i)^*$. Em termos do autômato que implementa a linguagem L, a operação de projeção inversa consiste em acrescentar auto-laços em todos os estados do autômato.

A operação projeção natural P_i pode ser aplicada diretamente sobre um autômato. O procedimento, em linhas gerais, é apresentado a seguir e consiste em dois passos:

1. os rótulos das transições que estão em $\Sigma - \Sigma_i$ são substituídos pelo rótulo ε, gerando um autômato que pode ser não-determinístico;
2. o autômato resultante, caso seja não-determinístico, deve ser transformado em determinístico.

Apresenta-se um exemplo em que o autômato resultante do primeiro passo do procedimento é não-determinístico.

Exemplo 5.2.1. *Sejam $\Sigma = \{\alpha, \beta, \gamma, \delta\}$, $\Sigma' = \{\alpha, \beta\}$ alfabetos e $P : \Sigma^* \rightarrow \Sigma'^*$. Sejam G um autômato sobre Σ e o autômato resultante do primeiro passo do algoritmo de obtenção da projeção, como apresentado nas Figuras 5.1(a) e (b), respectivamente. Os eventos γ e δ são substituídos por ε no autômato G tornando o autômato resultante não-determinístico (Figura 5.1(b)).*

Figura 5.1: Exemplo 5.2.1: (a) G; (b) Autômato resultante do primeiro passo do procedimento de projeção.

O segundo passo do procedimento de projeção natural consiste na determinização do autômato resultante no passo 1. Segundo Hopcroft et al. [2001], o procedimento de determinização e o autômato resultante têm complexidade exponencial no número de estados do autômato não-determinístico, no pior caso.

A propriedade do observador foi apresentada inicialmente no contexto do controle hierárquico [Wong e Wonham, 1996]. Em Wong e Wonham [1996], a propriedade do observador foi apresentada para linguagens prefixo-fechadas, tendo sido posteriormente estendida para o caso de linguagens não-prefixo-fechadas [Wong et al., 2000],[Wong et al., 1995].

Esta propriedade pode também ser utilizada para caracterizar projeções naturais. A propriedade é apresentada pela Definição 5.2.1.

Definição 5.2.1 ([Wong et al., 2000])**.** *Sejam $S \subseteq \Sigma^*$ uma linguagem, $\Sigma' \subseteq \Sigma$ um alfabeto e $\theta : \Sigma^* \to \Sigma'^*$ a projeção natural de cadeias de Σ^* em cadeias de Σ'^*. Se*

$$(\forall a \in \overline{S})(\forall b \in \Sigma'^*)\ \ \theta(a)b \in \theta(S) \Longrightarrow (\exists c \in \Sigma^*)\ \ \theta(ac) = \theta(a)b \quad e \quad ac \in S$$

então a projeção $\theta(S)$ possui a propriedade do observador.

Em palavras, a projeção aplicada a um autômato não pode implicar o agrupamento de estados com caminhos distintos para estados marcados.

No contexto deste trabalho, sempre que uma abstração, obtida através da operação de projeção natural, tiver a propriedade do observador, ela é caracterizada como uma *OP-abstração.*

Apresentam-se na seqüência alguns exemplos em que se destaca se a projeção é OP-abstração. Caso não seja, detalhes sobre a violação da propriedade do observador são apontados.

Exemplo 5.2.2. *Seja G um autômato que reconhece a linguagem $S = \mathcal{L}_m(G) \subset \Sigma^*$, onde $\Sigma = \{x, y, u\}$, na Figura 5.2(a). Seja a projeção natural $\theta : \Sigma^* \to \Sigma'^*$, onde $\Sigma' = \{x, y\}$, que apaga todas as ocorrências do evento u em G. A Figura 5.2(b) mostra um autômato reconhecedor de $\theta(S)$. Para mostrar que esse exemplo viola a propriedade do observador, escolhe-se $a = xu$, $b = \epsilon$ (Definição 5.2.1). Com essas escolhas $\theta(a)b = \theta(xu) \in \theta(\overline{S})$. No entanto, $\nexists c \in \Sigma^*$ tal que $\theta(xu) = \theta(xuc)$ e $ac \in S$.*

Exemplo 5.2.3. *Seja G um autômato que reconhece a linguagem $S = \mathcal{L}_m(G) \subseteq \Sigma^*$, onde $\Sigma = \{x, y, z, u\}$, na Figura 5.3(a). Seja a projeção natural $\theta : \Sigma^* \to \Sigma'^*$. Considere*

Figura 5.2: Exemplo 5.2.2: (a) G; (b) $\theta(G)$.

$\Sigma' = \{x, y, z\}$. *Nesse caso, a projeção* $\theta(S)$ *apaga todas as ocorrências do evento* u *em* G. *A Figura 5.3(b) mostra um autômato que representa* $\theta(S)$. *Pode-se observar que essa não é uma OP-abstração. Sejam* $a = u \in \overline{S}$ *e* $b = xyz$ *tal que* $\theta(a)b = \theta(u)xyz = xyz \in \theta(S)$. *Para essas escolhas,* $\nexists c \in \Sigma^*$ *tal que* $\theta(ac) = \theta(a)NEstasb$ *e* $ac \in S$. *Já na Figura 5.3(c),* $\Sigma' = \{y, z\}$, *ou seja, os eventos* x *e* u *são apagados de* G. *Neste caso, pode-se perceber que* $\forall a \in \overline{S}$ *e* $\forall b \in \Sigma'^*$ *tal que* $\theta(a)b \in \theta(S)$, *pode-se afirmar que* $\exists c \in \Sigma^*$ *tal que* $\theta(a)b = \theta(ac)$ *e* $ac \in S$ *portanto,* $\theta(S)$ *é uma OP-abstração.*

Ao apagar z *apenas não obteve-se uma OP-abstração. No entanto, apagando o par* $\{x, z\}$ *foi*

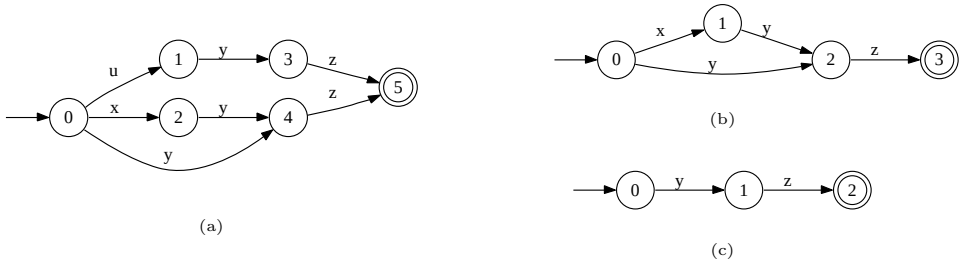


Figura 5.3: Exemplo 5.2.3: (a) G; (b) $\theta(G)$, com $\Sigma' = \{x, y, z\}$; (c) $\theta(G)$, com $\Sigma' = \{y, z\}$.

possível obter uma OP-abstração. Esse exemplo mostra que dado um subconjunto Σ' *de eventos para o qual obtém-se uma OP-abstração, nada pode ser afirmado em relação à obtenção de uma OP-abstração utilizando outro alfabeto (*Σ'_1*) que seja subconjunto do primeiro (*$\Sigma'_1 \subset \Sigma'$*). De forma semelhante, nada pode-se afirmar a respeito da obtenção de uma OP-abstração utilizando um conjunto* Σ'_2 *que contenha o primeiro (*$\Sigma'_2 \supset \Sigma'$*).*

Segundo Wong [1998], a complexidade de computar projeções é exponencial no tempo no pior caso e que o espaço de estados do autômato que representa a linguagem projetada pode crescer exponencialmente com o número de estados do sistema original.

Entretanto, se a projeção possui a propriedade do observador, há a garantia de que o autômato mínimo que representa a projeção sempre tenha número de estados menor ou igual ao número de estados do autômato mínimo para a linguagem original e ainda que pode ser obtida em tempo polinomial [Wong, 1998]. A obtenção de OP-abstrações é discutida no Capítulo 7.

A propriedade da distributividade de projeções naturais sobre a composição síncrona, apresentada como exercício em Wonham [2004], e extensivamente utilizada nas demonstrações das próximas seções, é apresentada a seguir como a Proposição 5.2.1.

Proposição 5.2.1 ([Wonham, 2004]). *Sejam* $L_j \subseteq \Sigma_j^*$, $j \in J = \{1, \ldots, m\}$, $\Sigma = \bigcup_{j=1}^{m} \Sigma_j$,

$\Sigma_r \subseteq \Sigma$, $P_{\Sigma\to\Sigma_r} : \Sigma^* \to \Sigma_r^*$ *e* $P_{\Sigma_j\to(\Sigma_j\cap\Sigma_r)} : \Sigma_j^* \to (\Sigma_j \cap \Sigma_r)^*$. *Então,*

$$P_{\Sigma\to\Sigma_r}\big(\mathop{||}_{j=1}^{m} L_j\big) = \mathop{||}_{j=1}^{m} P_{\Sigma_j\to(\Sigma_j\cap\Sigma_r)}(L_j)$$

se $\Sigma_s \subseteq \Sigma_r$, *onde* $\Sigma_s = \mathop{\cup}_{k,l\in J}^{k\neq l} (\Sigma_k \cap \Sigma_l)$.

5.3 Teste baseado em abstrações

Nesta seção apresentam-se os resultados obtidos para o teste baseado em abstrações. Para tornar a leitura mais compreensível apresentam-se, primeiramente, os resultados (lemas, teoremas e suas respectivas demonstrações) para o caso simples em que se tem apenas 2 supervisores. Em seguida, apresentam-se, em uma seção análoga à anterior, os resultados generalizados para o caso de m supervisores.

Os resultados apresentados nesta seção foram publicados em [Pena et al., 2006c] e [Pena et al., 2006a]. O caso simples, com 2 linguagens, foi simultaneamente publicado (sem demonstração) por Feng e Wonham [2006a].

No próximo capítulo, pretende-se substituir a condição sobre os eventos relevantes. Portanto, nas demonstrações que seguem, vai-se marcar com um $\triangleright$ *passagem* $\triangleleft$ todas as passagens que utilizam a condição sobre os eventos relevantes. Essas passagens da prova serão analisadas sob a nova condição a ser proposta no próximo capítulo.

5.3.1 Dois supervisores

Sejam as linguagens $S_1 \subseteq \Sigma_1^*$ e $S_2 \subseteq \Sigma_2^*$ que implementam as ações dos supervisores S_1 e S_2. Sejam ainda o conjunto $\Sigma = \Sigma_1 \cup \Sigma_2$, o conjunto de eventos relevantes $\Sigma_r \subseteq \Sigma$, as projeções naturais $\theta_1 : \Sigma_1^* \to (\Sigma_1 \cap \Sigma_r)^*$, $\theta_2 : \Sigma_2^* \to (\Sigma_2 \cap \Sigma_r)^*$ e $\theta : \Sigma^* \to \Sigma_r^*$. A condição imposta ao conjunto Σ_r é a de que todos os eventos compartilhados por S_1 e S_2 façam parte de Σ_r, ou seja, $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$. Para simplificar a notação, usa-se $\Sigma_1' = \Sigma_1 \cap \Sigma_r$ e $\Sigma_2' = \Sigma_2 \cap \Sigma_r$. Sejam ainda a projeção $P_{\Gamma\to\Upsilon} : \Gamma^* \to \Upsilon^*$ e sua inversa $P^{-1}_{\Gamma\to\Upsilon} : \Upsilon^* \to \Gamma^*$, onde Γ e Υ podem assumir valores diversos.

O Lema 5.3.1 será utilizado na prova do teorema principal desta seção e é, portanto, apresentado a seguir.

Lema 5.3.1. *Sejam* $S_1 \subseteq \Sigma_1^*$, $S_2 \subseteq \Sigma_2^*$, $s \in \Sigma^*$, $t \in \Sigma_r^*$, *as projeções naturais* θ_1, θ_2, θ, $P_{\Sigma\to\Sigma_1}$, $P_{\Sigma\to\Sigma_2}$, $P_{\Sigma_r\to\Sigma_1'}$, $P_{\Sigma_r\to\Sigma_2'}$, $P^{-1}_{\Sigma_r\to\Sigma_1'}$ *e* $P^{-1}_{\Sigma_r\to\Sigma_2'}$ *como definidos anteriormente. Se* $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$ *e* $\theta(s)t \in \theta_1(S_1)||\,\theta_2(S_2)$ *então* $\exists t_1 \in \Sigma_1'^*$ *e* $\exists t_2 \in \Sigma_2'^*$ *tal que*

i. $\theta(s)t \in \theta_1(P_{\Sigma\to\Sigma_1}s)t_1||\,\theta_2(P_{\Sigma\to\Sigma_2}s)t_2$;

ii. $\theta_1(P_{\Sigma\to\Sigma_1}s)t_1 \in \theta_1(S_1)$;

iii. $\theta_2(P_{\Sigma\to\Sigma_2}s)t_2 \in \theta_2(S_2)$;

iv. $\theta_1(P_{\Sigma\to\Sigma_1}s)t_1||\,\theta_2(P_{\Sigma\to\Sigma_2}s)t_2 \subseteq \theta_1(S_1)||\,\theta_2(S_2)$.

Demonstração. Para $s \in \overline{S_1||\,S_2} \subseteq \Sigma^*$, escolhe-se $t_1 = P_{\Sigma_r\to\Sigma_1'}t$ e $t_2 = P_{\Sigma_r\to\Sigma_2'}t$. Então, $t \in P^{-1}_{\Sigma_r\to\Sigma_1'}t_1 \cap P^{-1}_{\Sigma_r\to\Sigma_2'}t_2$. Para demonstrar a hipótese *i.*, deve-se mostrar que $[\theta_1(P_{\Sigma\to\Sigma_1}s)||\ \theta_2(P_{\Sigma\to\Sigma_2}s)]t \subseteq \theta_1(P_{\Sigma\to\Sigma_1}s)t_1||\,\theta_2(P_{\Sigma\to\Sigma_2}s)t_2$. Para uniformizar a notação, as projeções θ_1, θ_2 e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_1 \approx P_{\Sigma_1\to\Sigma_1'}$, $\theta_2 \approx P_{\Sigma_2\to\Sigma_2'}$ e $\theta \approx P_{\Sigma\to\Sigma_r}$. Portanto,

$$[P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)||\,P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)]t =$$

$$\begin{aligned}
&= [P^{-1}_{\Sigma_r\to\Sigma_1'}\big(P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)\big) \cap P^{-1}_{\Sigma_r\to\Sigma_2'}\big(P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)\big)]t\\
&= P^{-1}_{\Sigma_r\to\Sigma_1'}[P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)]t \cap P^{-1}_{\Sigma_r\to\Sigma_2'}[P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)]t\\
&\subseteq P^{-1}_{\Sigma_r\to\Sigma_1'}[P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)]P^{-1}_{\Sigma_r\to\Sigma_1'}t_1 \cap P^{-1}_{\Sigma_r\to\Sigma_2'}[P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)]P^{-1}_{\Sigma_r\to\Sigma_2'}t_2\\
&\quad = P^{-1}_{\Sigma_r\to\Sigma_1'}[P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1] \cap P^{-1}_{\Sigma_r\to\Sigma_2'}[P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2]\\
&\quad = P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1||\,P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2.
\end{aligned}$$

Então $[P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)||\,P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)]t \subseteq P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1||\,P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2$. Como tem-se que $P_{\Sigma\to\Sigma_r}(s)t \in [P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)||\,P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)]t$, pode-se dizer que

$$P_{\Sigma\to\Sigma_r}(s)t \in P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1||\,P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2$$

ou que

$$\theta(s)t \in \theta_1(P_{\Sigma\to\Sigma_1}s)t_1||\,\theta_2(P_{\Sigma\to\Sigma_2}s)t_2$$

que corresponde à hipótese *i*.

Para demonstrar a hipótese *ii*, parte-se da seguinte equação:

$$P_{\Sigma\to\Sigma_r}(s)t \in P_{\Sigma_1\to\Sigma_1'}(S_1)||\,P_{\Sigma_2\to\Sigma_2'}(S_2). \tag{5.2}$$

Aplicam-se as projeções $P_{\Sigma_r\to\Sigma_1'}$ e $P_{\Sigma_r\to\Sigma_2'}$, nos dois lados da equação (5.2), obtendo-se:

$$P_{\Sigma_r\to\Sigma_1'}(P_{\Sigma\to\Sigma_r}(s)t) \in P_{\Sigma_r\to\Sigma_1'}(P_{\Sigma_1\to\Sigma_1'}(S_1)||\,P_{\Sigma_2\to\Sigma_2'}(S_2)); \tag{5.3}$$

$$P_{\Sigma_r\to\Sigma_2'}(P_{\Sigma\to\Sigma_r}(s)t) \in P_{\Sigma_r\to\Sigma_2'}(P_{\Sigma_1\to\Sigma_1'}(S_1)||\,P_{\Sigma_2\to\Sigma_2'}(S_2)). \tag{5.4}$$

Considerando que o alfabeto após a projeção $P_{\Sigma_1\to\Sigma_1'}$ ($P_{\Sigma_2\to\Sigma_2'}$) é Σ_1' (Σ_2'), tem-se que

$P_{\Sigma_r\to\Sigma_1'}(P_{\Sigma_1\to\Sigma_1'}(S_1)||\ P_{\Sigma_2\to\Sigma_2'}(S_2)) \subseteq P_{\Sigma_1\to\Sigma_1'}(S_1)$. Da equação (5.3), tem-se que

$$\begin{aligned} P_{\Sigma_r\to\Sigma_1'}(P_{\Sigma\to\Sigma_r}(s))P_{\Sigma_r\to\Sigma_1'}(t) &\in P_{\Sigma_1\to\Sigma_1'}(S_1) && (5.5)\\ P_{\Sigma_r\to\Sigma_1'}(P_{\Sigma\to\Sigma_r}(s))t_1 &\in P_{\Sigma_1\to\Sigma_1'}(S_1), && (5.6)\end{aligned}$$

com $t_1 = P_{\Sigma_r\to\Sigma_1'}t$.

$\triangleright$
Além disso, tem-se que

$$\begin{aligned} P_{\Sigma_r\to\Sigma_1'}(P_{\Sigma\to\Sigma_r}(s))t_1 &= P_{\Sigma_r\to\Sigma_1'}(P_{\Sigma\to\Sigma_r}(P_{\Sigma\to\Sigma_1}s||P_{\Sigma\to\Sigma_2}s))t_1 && (5.7)\\ &= P_{\Sigma_r\to\Sigma_1'}(P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)||P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s))t_1.\end{aligned}$$

A substituição de $P_{\Sigma\to\Sigma_r}(P_{\Sigma\to\Sigma_1}s||P_{\Sigma\to\Sigma_2}s)$ na equação (5.7) por $P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)||P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)$ na equação (5.6) pode ser realizada devido à condição $\Sigma_1\cap\Sigma_2\subseteq\Sigma_r$ que permite aplicar a Proposição 5.2.1.
$\triangleleft$

Portanto,

$$P_{\Sigma_r\to\Sigma_1'}(P_{\Sigma\to\Sigma_r}(s))t_1 = P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1. \quad (5.8)$$

Na equação (5.6) substitui-se o termo mais à esquerda pelo termo mais à direita da equação (5.8), obtendo-se

$$P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1 \in P_{\Sigma_1\to\Sigma_1'}(S_1),$$

ou que

$$\theta_1(P_{\Sigma\to\Sigma_1}s)t_1 \in \theta_1(S_1). \qquad (ii)$$

Para mostrar *iii*, o mesmo procedimento (entre as equações (5.5) e (5.8)) pode ser aplicado à equação (5.4) gerando:

$$\theta_2(P_{\Sigma\to\Sigma_2}s)t_2 \in \theta_2(S_2). \qquad (iii)$$

A hipótese *iv*. decorre diretamente de *i*, *ii* e *iii*. Sendo esses demonstrados verdadeiros, pode-se afirmar que:

$$\theta_1(P_{\Sigma\to\Sigma_1}s)t_1||\theta_2(P_{\Sigma\to\Sigma_2}s)t_2 \subseteq \theta_1(S_1)||\theta_2(S_2).$$

□

Sejam $s \in \overline{S_1}||\overline{S_2}$, $t \in \Sigma_r^*$ tal que $\theta(s)t \in \theta_1(S_1)||\theta_2(S_2)$, $t_1 \in \Sigma_1'^*$ e $t_2 \in \Sigma_2'^*$ tais que $\theta_1(P_{\Sigma\to\Sigma_1}s)t_1 \in \theta_1(S_1)$ e $\theta_2(P_{\Sigma\to\Sigma_2}s)t_2 \in \theta_2(S_2)$. Sejam $u_1 \in \Sigma_1^*$ e $u_2 \in \Sigma_2^*$ tais que $\theta_1(u_1) =$

t_1 e $\theta_2(u_2) = t_2$ e $P_{\Sigma\to\Sigma_1}(s)u_1 \in S_1$ e $P_{\Sigma\to\Sigma_2}(s)u_2 \in S_2$. Há casos em que, sob todas as condições listadas acima, não existe u tal que $su \in S_1||S_2$. O exemplo apresentado na seqüência ilustra esse problema.

Exemplo 5.3.1. *Seja $S_1, S_2 \subseteq \Sigma^* = \{a, b, c, d\}^*$, cujos autômatos são apresentados na Figura 5.4 (a) e (b). As projeções de S_1 e S_2 no alfabeto $\Sigma' = \{a, c, d\}$, $\theta_1(S_1)$ e $\theta_2(S_2)$, são OP-abstrações apresentadas nas Figuras 5.5(a) e (b), respectivamente. Seja $s = a$. Então $P_{\Sigma\to\Sigma_1}s = a$ e $P_{\Sigma\to\Sigma_2}s = a$ e $t_1 = cd$ e $t_2 = cd$ tal que $P_{\Sigma\to\Sigma_1}(a)cd \in \theta_1(S_1)$ e $P_{\Sigma\to\Sigma_2}(a)cd \in \theta_2(S_2)$. Obtém-se então $u_1 = bcd$ e $u_2 = cbd$ tal que $\theta_1(u_1) = t_1$ e $\theta_2(u_2) = t_2$ e $P_{\Sigma\to\Sigma_1}(a)u_1 \in S_1$ e $P_{\Sigma\to\Sigma_2}(a)u_2 \in S_2$. Pode-se observar que, apesar de existirem u_1 e u_2, não existe $u \in \Sigma^*$ tal que $su \in S_1||S_2$, apresentado na Figura 5.4(c). O teste de não-conflito aplicado sobre as OP-abstrações (Figura 5.5(c)), nesse caso, não detectou a presença de conflito, detectado pelo teste original, na Figura 5.4(c). Pode-se observar que não foi respeitada a condição sobre os eventos relevantes, que diz que $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$. Essa condição garante que dados u_1 e u_2, existe $u \in u_1||u_2$ tal que $su \in S_1||S_2$.*

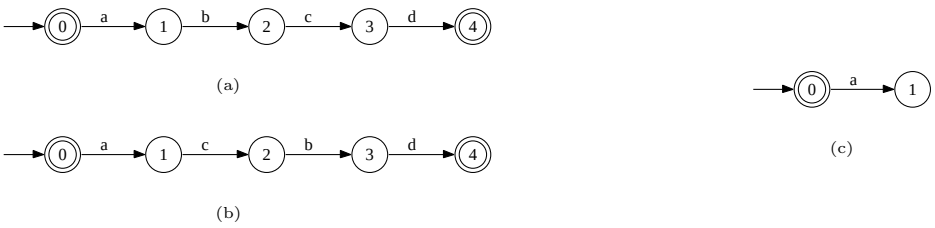


Figura 5.4: Exemplo 5.3.1: (a) S_1; (b) S_2; (c) $S_1||S_2$.

Figura 5.5: Exemplo 5.3.1: (a) $\theta_1(S_1)$; (b) $\theta_1(S_2)$; (c) $\theta_1(S_1)||\theta_1(S_2)$.

A existência de $u \in \Sigma^*$ tal que $su \in S_1||S_2$ é fundamental para que o teste possa ser aplicado com sucesso. Esse resultado é utilizado nas provas dos teoremas deste capítulo e, portanto, é apresentado no formato de um lema (Lema 5.3.2). Com relação às marcações $\triangleright$, $\triangleleft$, elas não serão utilizadas neste Lema porque será apresentado um novo lema considerando as novas condições no próximo capítulo.

Lema 5.3.2. *Sejam $\theta_1(S_1)$, $\theta_2(S_2)$ OP-abstrações. Se $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$ e $\exists s \in \Sigma^*$ e $t \in \Sigma'^*$, tal que $\theta(s)t \in \theta_1(S_1)||\theta_2(S_2)$ então $\exists u \in \Sigma'^*$ tal que $su \in S_1||S_2$.*

Demonstração. Para uniformizar a notação, as projeções θ_1, θ_2 e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_1 \approx P_{\Sigma_1\to\Sigma'_1}$, $\theta_2 \approx P_{\Sigma_2\to\Sigma'_2}$ e

$\theta \approx P_{\Sigma\to\Sigma_r}$. Se $\exists t \in \Sigma'^*$, tal que $P_{\Sigma\to\Sigma_r}(s)t \in P_{\Sigma_1\to\Sigma_1'}(S_1)||P_{\Sigma_2\to\Sigma_2'}(S_2)$, então $\exists t_1 \in \Sigma_1'^*$ e $\exists t_2 \in \Sigma_2'^*$ tal que $P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1 \in P_{\Sigma_1\to\Sigma_1'}(S_1)$ e $P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2 \in P_{\Sigma_2\to\Sigma_2'}(S_2)$ (pelo Lema 5.3.1). Como $P_{\Sigma_1\to\Sigma_1'}(S_1)$ e $P_{\Sigma_2\to\Sigma_2'}(S_2)$ são OP-abstrações, $\exists u_1 \in \Sigma_1'^*$ tal que $P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}(s)u_1) = P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}(s))t_1$ e $P_{\Sigma\to\Sigma_1}(s)u_1 \in S_1$. Analogamente, $\exists u_2 \in \Sigma_2'^*$ tal que $P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}(s)u_2) = P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}(s))t_2$ e $P_{\Sigma\to\Sigma_2}(s)u_2 \in S_2$. Pretende-se mostrar que $\exists u \in \Sigma'^*$ tal que $su \in S_1||S_2$.

Escolhe-se $u \in u_1||u_2$ e deve-se mostrar que $u_1||u_2 \neq \emptyset$. Sabe-se que $t_1||t_2 \neq \emptyset$. Portanto, $t_1||t_2 = P^{-1}_{\Sigma_r\to\Sigma_1'}t_1||P^{-1}_{\Sigma_r\to\Sigma_2'}t_2 \neq \emptyset$.

Outra forma de representar que $t_1||t_2 \neq \emptyset$ é utilizando $P_{\Sigma_1'\to(\Sigma_1'\cap\Sigma_2')}t_1 \cap P_{\Sigma_2'\to(\Sigma_1'\cap\Sigma_2')}t_2 \neq \emptyset$, ou seja, projetando t_1 e t_2 no conjunto de eventos comuns $\Sigma_1' \cap \Sigma_2'$. Então, se $t_1||t_2 \neq \emptyset$, significa que $P_{\Sigma_1'\to(\Sigma_1'\cap\Sigma_2')}t_1 \cap P_{\Sigma_2'\to(\Sigma_1'\cap\Sigma_2')}t_2 \neq \emptyset$. Como $t_1 = P_{\Sigma_1\to\Sigma_1'}(u_1)$ e $t_2 = P_{\Sigma_2\to\Sigma_2'}(u_2)$, obtém-se:

$$\begin{aligned} P_{\Sigma_1'\to(\Sigma_1'\cap\Sigma_2')}t_1 \cap P_{\Sigma_2'\to(\Sigma_1'\cap\Sigma_2')}t_2 &= P_{\Sigma_1'\to(\Sigma_1'\cap\Sigma_2')}P_{\Sigma_1\to\Sigma_1'}(u_1) \cap P_{\Sigma_2'\to(\Sigma_1'\cap\Sigma_2')}P_{\Sigma_2\to\Sigma_2'}(u_2) \\ &= P_{\Sigma_1\to(\Sigma_1'\cap\Sigma_2')}(u_1) \cap P_{\Sigma_2\to(\Sigma_1'\cap\Sigma_2')}(u_2). \end{aligned} \tag{5.9}$$

Como $P_{\Sigma_1'\to(\Sigma_1'\cap\Sigma_2')}t_1 \cap P_{\Sigma_2'\to(\Sigma_1'\cap\Sigma_2')}t_2 \neq \emptyset$ e $P_{\Sigma_1'\to(\Sigma_1'\cap\Sigma_2')}t_1 \cap P_{\Sigma_2'\to(\Sigma_1'\cap\Sigma_2')}t_2 = P_{\Sigma_1\to\Sigma_1'\cap\Sigma_2'}(u_1) \cap P_{\Sigma_2\to\Sigma_1'\cap\Sigma_2'}(u_2)$, então tem-se que $P_{\Sigma_1\to\Sigma_1'\cap\Sigma_2'}(u_1) \cap P_{\Sigma_2\to\Sigma_1'\cap\Sigma_2'}(u_2) \neq \emptyset$. Essas projeções apagam eventos que não estão em $\Sigma_1' \cap \Sigma_2'$, ou seja, apagam os eventos que não estão em:

$$\begin{aligned} \Sigma_1' \cap \Sigma_2' &= (\Sigma_1 \cap \Sigma_r) \cap (\Sigma_2 \cap \Sigma_r) \\ &= \Sigma_1 \cap \Sigma_2 \cap \Sigma_r. \end{aligned} \tag{5.10}$$

Como $\Sigma_1 \cap \Sigma_2 = \Sigma_s$ e $\Sigma_s \subseteq \Sigma_r$, substituindo-se na equação (5.10) tem-se que:

$$\begin{aligned} \Sigma_1' \cap \Sigma_2' &= \Sigma_s \cap \Sigma_r \\ &= \Sigma_s. \end{aligned}$$

Então equação (5.9) equivale a $P_{\Sigma_1\to\Sigma_s}(u_1)||P_{\Sigma_2\to\Sigma_s}(u_2) \neq \emptyset$ e portanto, $u_1||u_2 \neq \emptyset$. Consequentemente, $\exists u \in u_1||u_2$ tal que $su \in S_1||S_2$. □

O teorema que se segue apresenta condições suficientes para que o teste de não-conflito possa ser aplicado sobre as abstrações, reduzindo assim o espaço de estados e portanto a complexidade computacional do teste.

Teorema 5.3.1. *Sejam $S_1 \in \Sigma_1^*$, $S_2 \in \Sigma_2^*$, as projeções naturais θ_1, θ_2, θ, $P_{\Sigma\to\Sigma_1}$, $P_{\Sigma\to\Sigma_2}$, $P_{\Sigma_r\to\Sigma_1'}$, $P_{\Sigma_r\to\Sigma_2'}$, $P^{-1}_{\Sigma_r\to\Sigma_1'}$ e $P^{-1}_{\Sigma_r\to\Sigma_2'}$, como definidos anteriormente. Se as projeções naturais*

$\theta_1(S_1)$ e $\theta_2(S_2)$ forem OP-abstrações e $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$ então

$$\overline{\theta_1(S_1)||\theta_2(S_2)} = \overline{\theta_1(S_1)||\theta_2(S_2)} \iff \overline{S_1}||\overline{S_2} = \overline{S_1||S_2}.$$

Demonstração. A demonstração deste teorema é dividida em duas partes.

i. $\overline{S_1}||\overline{S_2} = \overline{S_1||S_2} \Rightarrow \overline{\theta_1(S_1)||\theta_2(S_2)} = \overline{\theta_1(S_1)||\theta_2(S_2)}$.

Tem-se que

$$\overline{\theta_1(S_1)||\theta_2(S_2)} = \theta_1(\overline{S_1})||\theta_2(\overline{S_2}).$$

▷

Como $\Sigma_s \subseteq \Sigma_r$, então $\theta_1(\overline{S_1})||\theta_2(\overline{S_2}) = \theta(\overline{S_1}||\overline{S_2})$, pela Proposição 5.2.1.

◁

Além disso, por hipótese, $\overline{S_1}||\overline{S_2} = \overline{S_1||S_2}$. Portanto,

$$\theta(\overline{S_1}||\overline{S_2}) = \theta(\overline{S_1||S_2}) = \overline{\theta(S_1||S_2)}. \tag{5.11}$$

▷

Usando novamente o fato de que $\Sigma_s \subseteq \Sigma_r$, pode-se aplicar a Proposição 5.2.1 sobre equação (5.11), obtendo-se

$$\overline{\theta(S_1||S_2)} = \overline{\theta_1(S_1)||\theta_2(S_2)}.$$

◁

Então,

$$\overline{\theta_1(S_1)||\theta_2(S_2)} = \overline{\theta_1(S_1)||\theta_2(S_2)}.$$

ii. $\overline{\theta_1(S_1)||\theta_2(S_2)} = \overline{\theta_1(S_1)||\theta_2(S_2)} \Rightarrow \overline{S_1}||\overline{S_2} = \overline{S_1||S_2}$.

Para uniformizar a notação de projeções nesta parte da demonstração, as projeções θ_1, θ_2 e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_1 \approx P_{\Sigma_1 \to \Sigma_1'}$, $\theta_2 \approx P_{\Sigma_2 \to \Sigma_2'}$ e $\theta \approx P_{\Sigma \to \Sigma_r}$. Então, $\overline{P_{\Sigma_1 \to \Sigma_1'}(S_1)||P_{\Sigma_2 \to \Sigma_2'}(S_2)} = \overline{P_{\Sigma_1 \to \Sigma_1'}(S_1)||P_{\Sigma_2 \to \Sigma_2'}(S_2)} \Rightarrow \overline{S_1}||\overline{S_2} = \overline{S_1||S_2}$.

▷

Como $P_{\Sigma_1 \to \Sigma_1'}(S_1)||P_{\Sigma_2 \to \Sigma_2'}(S_2) = P_{\Sigma \to \Sigma_r}(S_1||S_2)$, pela Proposição 5.2.1, e $P_{\Sigma \to \Sigma_r}(\overline{S}) = \overline{P_{\Sigma \to \Sigma_r}(S)}$, pode-se reescrever *ii.* como

$$P_{\Sigma \to \Sigma_r}(\overline{S_1}||\overline{S_2}) = P_{\Sigma \to \Sigma_r}(\overline{S_1||S_2}) \Longrightarrow \overline{S_1}||\overline{S_2} = \overline{S_1||S_2}. \tag{5.12}$$

◁

É suficiente mostrar que dado $s \in \overline{S_1}||\overline{S_2}$ tem-se que $s \in \overline{S_1||S_2}$. Por hipótese, $s \in \overline{S_1}||\overline{S_2}$. Então, $\theta(s) \in \theta(\overline{S_1}||\overline{S_2})$. Ainda, pelo lado esquerdo da equação (5.12), pode-se

dizer que, $P_{\Sigma\to\Sigma_r}(s) \in P_{\Sigma\to\Sigma_r}(\overline{S_1||S_2}) = \overline{P_{\Sigma\to\Sigma_r}(S_1||S_2)}$. Portanto, $\exists t \in \Sigma_r^*$ tal que $P_{\Sigma\to\Sigma_r}(s)t \in P_{\Sigma\to\Sigma_r}(S_1||S_2)$. Além disso, do Lema 5.3.1, $\exists t_1 \in \Sigma_1'^*$ e $\exists t_2 \in \Sigma_2'^*$, tal que

$$\begin{aligned} P_{\Sigma\to\Sigma_r}(s)t \in P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1||P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2 &\subseteq P_{\Sigma_1\to\Sigma_1'}(S_1)||P_{\Sigma_2\to\Sigma_2'}(S_2) \\ P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1 &\in P_{\Sigma_1\to\Sigma_1'}(S_1) \text{ e} \\ P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2 &\in P_{\Sigma_2\to\Sigma_2'}(S_2). \end{aligned}$$

Por hipótese, $P_{\Sigma\to\Sigma_1}s \in \overline{S_1}$ e $P_{\Sigma\to\Sigma_2}s \in \overline{S_2}$. Como $P_{\Sigma_1\to\Sigma_1'}(S_1)$ e $P_{\Sigma_2\to\Sigma_2'}(S_2)$ são OP-abstrações, pode-se dizer que $\exists u_1 \in \Sigma_1^*$ e $\exists u_2 \in \Sigma_2^*$ tal que $P_{\Sigma_1\to\Sigma_1'}(\ P_{\Sigma\to\Sigma_1}(s)u_1) = P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1$, $P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}(s)u_2) = P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2$ e $P_{\Sigma\to\Sigma_1}(s)u_1 \in S_1$, $P_{\Sigma\to\Sigma_2}(s)u_2 \in S_2$ (Definição 5.2.1). Então

$$P_{\Sigma\to\Sigma_1}(s)u_1||P_{\Sigma\to\Sigma_2}(s)u_2 \subseteq S_1||S_2.$$

$\triangleright$

Escolhendo $u \in u_1||u_2$, tem-se que $su \in s(u_1||u_2)$. Pelo Lema 5.3.2, tem-se que $u_1||u_2 \neq \emptyset$ e, portanto, $\exists u \in \Sigma^*$ tal que $u \in u_1||u_2$ e $su \in S_1||S_2$.

$\triangleleft$

Como $su \in S_1||S_2$, então $s \in \overline{S_1||S_2}$.

□

No Teorema 5.3.1, mostra-se que a propriedade de não-conflito é preservada quando se considera abstrações com as características mencionadas ao invés das linguagens originais, ou seja, os resultados dos dois testes são os mesmos se as condições não forem violadas.

As abstrações são projeções naturais. Em geral, o autômato mínimo que representa a projeção de uma linguagem L pode ter mais estados e transições que o autômato mínimo que representa a linguagem original. Como discutido anteriormente, o procedimento da projeção natural, quando aplicada a autômatos, possui dois passos. No primeiro passo, os estados ligados por cadeias de eventos que desaparecem na projeção são agrupados podendo gerar um autômato não-determinístico. Nesse caso, a operação de determinização pode gerar um autômato maior. No entanto, sabe-se que se a projeção natural tiver a propriedade do observador, o autômato obtido no primeiro passo do procedimento de projeção já é determinístico e, portanto, o passo de determinização não precisa ser executado [Wong, 1998]. O autômato obtido após agrupar estados do autômato original não é nunca maior que o autômato original, podendo ser do mesmo tamanho se nenhum evento é apagado. Pode-se dizer, portanto, que o espaço de estados do autômato original pode ser particionado. Cada estado do autômato da projeção corresponde a uma célula da partição e, portanto, há a garantia de que o autômato resultante não terá espaço de estados maior que o espaço de estados do autômato original.

Entretanto, a composição de abstrações com espaços de estados menores que os espaços de estados dos autômatos originais, obtidas como projeções naturais, não é, necessariamente,

menor que a composição dos supervisores originais.

Portanto, mesmo obtendo abstrações menores, não há a garantia de que o teste terá complexidade computacional reduzida. No entanto, sob as mesmas condições do Teorema 5.3.1, pode-se mostrar que a composição de OP-abstrações é também uma OP-abstração. Assim, o teste sobre abstrações não será maior que o teste original. Esse resultado é apresentado no Teorema 5.3.2 a seguir.

Teorema 5.3.2. *Sejam as linguagens $S_1 \in \Sigma_1^*$, $S_2 \in \Sigma_2^*$, as projeções naturais θ_1, θ_2, θ, $P_{\Sigma\to\Sigma_1}$, $P_{\Sigma\to\Sigma_2}$, $P_{\Sigma_r\to\Sigma_1'}$, $P_{\Sigma_r\to\Sigma_2'}$, $P^{-1}_{\Sigma_r\to\Sigma_1'}$ e $P^{-1}_{\Sigma_r\to\Sigma_2'}$, como definidas anteriormente. Se $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$ e as projeções naturais $\theta_1(S_1)$ e $\theta_2(S_2)$ são OP-abstrações de S_1 e S_2, respectivamente, então $\theta(S_1||\, S_2)$ é também uma OP-abstração.*

◇

Demonstração. Para uniformizar a notação, as projeções θ_1, θ_2 e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_1 \approx P_{\Sigma_1\to\Sigma_1'}$, $\theta_2 \approx P_{\Sigma_2\to\Sigma_2'}$ e $\theta \approx P_{\Sigma\to\Sigma_r}$. Seja $s \in \overline{S_1||\, S_2}$, tem-se que $s \in P_{\Sigma\to\Sigma_1}s||\, P_{\Sigma\to\Sigma_2}s$. Escolhe-se $t \in \Sigma_r^*$ tal que $P_{\Sigma\to\Sigma_r}(s)t \in P_{\Sigma\to\Sigma_r}(S_1||\, S_2)$. Tem-se que

$$\begin{aligned} P_{\Sigma\to\Sigma_r}(s) &\in P_{\Sigma\to\Sigma_r}(P_{\Sigma\to\Sigma_1}s||\, P_{\Sigma\to\Sigma_2}s) \\ P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(P_{\Sigma\to\Sigma_1}s||\, P_{\Sigma\to\Sigma_2}s)t. \end{aligned}$$

▷

Como $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$, pela Proposição 5.2.1, tem-se que

$$\begin{aligned} P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(P_{\Sigma\to\Sigma_1}s||\, P_{\Sigma\to\Sigma_2}s)t \\ &\in (P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)||\, P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s))t. \end{aligned}$$

◁

Então, considera-se $t_1 = P_{\Sigma_r\to\Sigma_1'}t$ e $t_2 = P_{\Sigma_r\to\Sigma_2'}t$. Para essas escolhas foi mostrado, no Lema 5.3.1, que $P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1 \in P_{\Sigma_1\to\Sigma_1'}(S_1)$ e $P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2 \in P_{\Sigma_2\to\Sigma_2'}(S_2)$. Pela Definição 5.2.1, como $P_{\Sigma_1\to\Sigma_1'}(S_1)$ e $P_{\Sigma_2\to\Sigma_2'}(S_2)$ são OP-abstrações, $\exists u_1 \in \Sigma_1^*$ e $\exists u_2 \in \Sigma_2^*$, tal que

$$\begin{aligned} P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}(s)u_1) &= P_{\Sigma_1\to\Sigma_1'}(P_{\Sigma\to\Sigma_1}s)t_1 \in P_{\Sigma_1\to\Sigma_1'}(S_1) \\ &\quad \text{e} \quad P_{\Sigma\to\Sigma_1}(s)u_1 \in S_1 \\ P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}(s)u_2) &= P_{\Sigma_2\to\Sigma_2'}(P_{\Sigma\to\Sigma_2}s)t_2 \in P_{\Sigma_2\to\Sigma_2'}(S_2) \\ &\quad \text{e} \quad P_{\Sigma\to\Sigma_2}(s)u_2 \in S_2. \end{aligned}$$

Pretende-se mostrar que $\exists u \in \Sigma^*$, tal que $P_{\Sigma\to\Sigma_r}(s)t = P_{\Sigma\to\Sigma_r}(su)$ e $su \in S_1||\,S_2$. Sabe-se que $t \in P^{-1}_{\Sigma_r\to\Sigma'_1}t_1 \cap P^{-1}_{\Sigma_r\to\Sigma'_2}t_2$. Além disso, $t_1 = P_{\Sigma_1\to\Sigma'_1}(u_1)$ e $t_2 = P_{\Sigma_2\to\Sigma'_2}(u_2)$. Então,

$$\begin{aligned} P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(s)[P^{-1}_{\Sigma_r\to\Sigma'_1}t_1 \cap P^{-1}_{\Sigma_r\to\Sigma'_2}t_2] \\ &= P_{\Sigma\to\Sigma_r}(s)[P^{-1}_{\Sigma_r\to\Sigma'_1}P_{\Sigma_1\to\Sigma'_1}(u_1) \cap P^{-1}_{\Sigma_r\to\Sigma'_2}P_{\Sigma_2\to\Sigma'_2}(u_2)] \\ &= P_{\Sigma\to\Sigma_r}(s)[P_{\Sigma_1\to\Sigma'_1}(u_1)||\,P_{\Sigma_2\to\Sigma'_2}(u_2)] \\ P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(s)[P_{\Sigma_1\to\Sigma'_1}(u_1)||\,P_{\Sigma_2\to\Sigma'_2}(u_2)] \end{aligned} \tag{5.13}$$

$\triangleright$

Usando a Proposição 5.2.1 e a equação (5.13),

$$\begin{aligned} P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(s)[P_{\Sigma_1\to\Sigma'_1}(u_1)||\,P_{\Sigma_2\to\Sigma'_2}(u_2)] = P_{\Sigma\to\Sigma_r}(s)P_{\Sigma\to\Sigma_r}(u_1||\,u_2) \\ P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}[s(u_1||\,u_2)] \end{aligned}$$

Como $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$, pode-se usar Lema 5.3.2 para garantir que $\exists u \in \Sigma^*$, tal que $u \in u_1||u_2$ e $su \in S_1||S_2$.

$\triangleleft$

Para completar as condições sobre as abstrações para que se enquadrem como OP-abstrações, deve-se mostrar que $t = P_{\Sigma\to\Sigma_r}(u)$. Essa igualdade é demonstrada por contradição, ou seja, parte-se de $t \neq P_{\Sigma\to\Sigma_r}(u)$ e deve-se chegar à uma contradição.

Aplicando-se $P_{\Sigma_r\to\Sigma'_1}$ sobre os dois lados, obtém-se que:

$$P_{\Sigma_r\to\Sigma'_1}t \neq P_{\Sigma_r\to\Sigma'_1}P_{\Sigma\to\Sigma_r}(u). \tag{5.14}$$

É sabido que $P_{\Sigma_r\to\Sigma'_1}t = t_1$ (do lado esquerdo), e pode-se manipular as projeções do lado direito de forma a obter:

$$\begin{aligned} P_{\Sigma_r\to\Sigma'_1}P_{\Sigma\to\Sigma_r}(u) &= P_{\Sigma_r\to\Sigma'_1}P_{\Sigma\to\Sigma_r}(u) = P_{\Sigma\to\Sigma'_1}u = P_{\Sigma_1\to\Sigma'_1}P_{\Sigma\to\Sigma_1}u \\ &= P_{\Sigma_1\to\Sigma'_1}u_1 = P_{\Sigma_1\to\Sigma'_1}(u_1) \end{aligned} \tag{5.15}$$

Na equação (5.14), substitui-se $P_{\Sigma_r\to\Sigma'_1}t = t_1$ no lado esquerdo e $P_{\Sigma_r\to\Sigma'_1}P_{\Sigma\to\Sigma_r}(u) = P_{\Sigma_1\to\Sigma'_1}(u_1)$ (equação (5.15)) no lado direito obtendo-se $t_1 \neq P_{\Sigma_1\to\Sigma'_1}(u_1)$, o que contradiz a definição de t_1 na hipótese. Portanto, é verdadeiro que $t = P_{\Sigma\to\Sigma_r}(u)$.

Se as projeções naturais $P_{\Sigma_1\to\Sigma'_1}(S_1)$ e $P_{\Sigma_2\to\Sigma'_2}(S_2)$, são OP-abstrações e $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$, para qualquer t escolhido, com $P_{\Sigma\to\Sigma_r}(s)t \in P_{\Sigma\to\Sigma_r}(S_1||S_2)$, pode-se obter $u \in \Sigma^*$ tal que $P_{\Sigma\to\Sigma_r}(s)t = P_{\Sigma\to\Sigma_r}(su)$ e $su \in S_1||S_2$ e finalmente concluir que $P_{\Sigma\to\Sigma_r}(S_1||S_2)$ e, portanto, $\theta(S_1||S_2)$ é uma OP-abstração. $\square$

O Teorema 5.3.2 mostra que, se $\theta_1(S_1)$ e $\theta_2(S_2)$ forem OP-abstrações, sua composição será também uma OP-abstração, desde que $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_r$. Com isso, sabe-se que o autômato mínimo que representa a linguagem $\theta(S_1||S_2)$ tem, no pior caso, o mesmo número de estados do autômato mínimo que representa $S_1||S_2$. Conseqüentemente, o teste de não-conflito aplicado sobre as abstrações tem, no pior caso, o mesmo tamanho, ou seja, a mesma complexidade computacional do teste de não-conflito original.

Os resultados apresentados nessa seção são estendidos, na próxima seção, para o caso de múltiplos supervisores. Uma vez que a seção seguinte apresenta extensões diretas dos resultados já apresentados, sua leitura é opcional.

5.3.2 Múltiplos Supervisores

Os resultados apresentados na Seção 5.3.1 são agora estendidos para o caso de m supervisores. De forma correspondente, as passagens das demonstrações desta seção que estiverem relacionadas com a condição sobre os eventos relevantes são marcadas com o par $\triangleright$, $\triangleleft$.

Sejam m supervisores S_j cujas linguagens implementadas são chamadas de $S_j \subseteq \Sigma_j^*$, $\forall j \in J = \{1, \ldots, m\}$, o conjunto de eventos relevantes $\Sigma_r \subseteq \Sigma = \bigcup_{j=1}^{m} \Sigma_j$, as projeções naturais $\theta_j : \Sigma_j^* \to (\Sigma_j \cap \Sigma_r)^*$ e $\theta : \Sigma^* \to \Sigma_r^*$. A condição sobre o conjunto Σ_r é a de que todos os eventos compartilhados por qualquer par S_k, S_l, com $k \neq l$, faça parte de Σ_r, ou seja, $\Sigma_s \subseteq \Sigma_r$ com $\Sigma_s = \bigcup_{k,l \in J}^{k \neq l} (\Sigma_k \cap \Sigma_l)$. Para simplificar a notação, usa-se $\Sigma'_j = \Sigma_j \cap \Sigma_r$. Sejam ainda a projeção $P_{\Gamma \to \Upsilon} : \Gamma^* \to \Upsilon^*$ e sua inversa $P^{-1}_{\Gamma \to \Upsilon} : \Upsilon^* \to \Gamma^*$, onde Γ e Υ podem assumir valores diversos.

O Lema 5.3.3 é uma extensão do Lema 5.3.1 apresentado na seção anterior.

Lema 5.3.3. *Sejam $S_j \subseteq \Sigma_j^*$, $s \in \Sigma^*$, $t \in \Sigma_r^*$ e as projeções naturais θ_j, θ, $P_{\Sigma \to \Sigma_j}$ como definidas anteriormente. Se $\Sigma_s \subseteq \Sigma_r$ com $\Sigma_s = \bigcup_{k,l \in J}^{k \neq l} (\Sigma_k \cap \Sigma_l)$ e $\theta(s)t \in \|_{j=1}^{m} \theta_j(S_j)$ então $\exists t_j \in \Sigma_j'^*$, $\forall j \in J$, tal que:*

i. $\theta(s)t \in \|_{j=1}^{m} \theta_j(P_{\Sigma \to \Sigma_j} s)t_j$;

ii. $\theta_j(P_{\Sigma \to \Sigma_j} s)t_j \in \theta_j(S_j)$, *para todo* $j \in J$;

iii. $\|_{j=1}^{m} \theta_j(P_{\Sigma \to \Sigma_j} s)t_j \subseteq \|_{j=1}^{m} \theta_j(S_j)$.

Demonstração. Para $s \in \overline{\|_{j=1}^{m} S_j}$, escolha $t_j = P_{\Sigma_r \to \Sigma'_j} t$, $\forall j \in J$. Então, $t \in \bigcap_{j=1}^{m} P^{-1}_{\Sigma_r \to \Sigma'_j} t_j$. Para provar hipótese *i.*, primeiro mostra-se que $\Big[\|_{j=1}^{m} \theta_j(P_{\Sigma \to \Sigma_j} s) \Big] t \subseteq \|_{j=1}^{m} \theta_j(P_{\Sigma \to \Sigma_j} s)t_j$. Para

uniformizar a notação, as projeções θ_j e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_j \approx P_{\Sigma_j \to \Sigma'_j}$ e $\theta \approx P_{\Sigma \to \Sigma_r}$. Portanto,

$$\begin{aligned}
\Big[\mathop{\Big\|}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s) \Big] t &= \Big[\mathop{\bigcap}_{j=1}^{m} P^{-1}_{\Sigma_r \to \Sigma'_j}\big(P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s)\big) \Big] t \\
&= \mathop{\bigcap}_{j=1}^{m} \Big[P^{-1}_{\Sigma_r \to \Sigma'_j}\big(P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s)\big) t \Big] \\
&\subseteq \mathop{\bigcap}_{j=1}^{m} \Big[P^{-1}_{\Sigma_r \to \Sigma'_j}\big(P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s)\big) P^{-1}_{\Sigma_r \to \Sigma'_j} t_j \Big] \\
&= \mathop{\bigcap}_{j=1}^{m} \Big[P^{-1}_{\Sigma_r \to \Sigma'_j}\big(P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s) t_j\big) \Big] \\
&= \mathop{\Big\|}_{j=1}^{m} \Big[P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s) t_j \Big].
\end{aligned}$$

Então,

$$\Big[\mathop{\Big\|}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s) \Big] t \subseteq \mathop{\Big\|}_{j=1}^{m} \Big[P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s) t_j \Big].$$

Da hipótese, tem-se que $P_{\Sigma \to \Sigma_r}(s)t \in \Big[\mathop{\Big\|}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s) \Big] t$ e pode-se dizer que:

$$P_{\Sigma \to \Sigma_r}(s)t \in \mathop{\Big\|}_{j=1}^{m} \Big[P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j} s) t_j \Big], \tag{5.16}$$

ou que

$$\theta(s)t \in \mathop{\Big\|}_{j=1}^{m} \Big[\theta_j(P_{\Sigma \to \Sigma_j} s) t_j \Big], \tag{5.17}$$

que corresponde a i..

Para demonstrar a hipótese ii., usa-se:

$$P_{\Sigma \to \Sigma_r}(s)t \in \mathop{\Big\|}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(S_j). \tag{5.18}$$

Aplica-se a projeção natural $P_{\Sigma_r \to \Sigma'_j}$, $\forall j \in J$, sobre os dois lados da equação (5.18), obtendo-se:

$$P_{\Sigma_r \to \Sigma'_j}\big(P_{\Sigma \to \Sigma_r}(s)t\big) \in P_{\Sigma_r \to \Sigma'_j}\big[\mathop{\Big\|}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(S_j) \big]. \tag{5.19}$$

Considerando que o alfabeto após a projeção $P_{\Sigma_j \to \Sigma'_j}$ é Σ'_j, tem-se que $P_{\Sigma_r \to \Sigma'_j}\Big[\mathop{\Big\|}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(S_j) \Big] \subseteq P_{\Sigma_j \to \Sigma'_j}(S_j)$. Da equação (5.19) tem-se, portanto, que

$$P_{\Sigma_r \to \Sigma'_j}\big(P_{\Sigma \to \Sigma_r}(s)t\big) \in P_{\Sigma_j \to \Sigma'_j}(S_j). \tag{5.20}$$

Reescrevendo a equação (5.20), tem-se

$$\begin{aligned} P_{\Sigma_r\to\Sigma'_j}\big(P_{\Sigma\to\Sigma_r}(s)\big)P_{\Sigma_r\to\Sigma'_j}t &\in P_{\Sigma_j\to\Sigma'_j}(S_j) \\ P_{\Sigma_r\to\Sigma'_j}\big(P_{\Sigma\to\Sigma_r}(s)\big)t_j &\in P_{\Sigma_j\to\Sigma'_j}(S_j) \end{aligned} \tag{5.21}$$

com $t_j = P_{\Sigma_r\to\Sigma'_j}t$. Além disso, tem-se que

$$P_{\Sigma_r\to\Sigma'_j}\big(P_{\Sigma\to\Sigma_r}(s)\big)t_j = P_{\Sigma_r\to\Sigma'_j}\big(P_{\Sigma\to\Sigma_r}(\mathop{||}_{j=1}^{m} P_{\Sigma\to\Sigma_j}s)\big)t_j.$$

$\triangleright$

Sob a condição $\Sigma_s \subseteq \Sigma_r$, tem-se que $P_{\Sigma\to\Sigma_r}(\mathop{||}_{j=1}^{m} L_j) = \mathop{||}_{j=1}^{m} P_{\Sigma_j\to\Sigma'_j}(L_j)$ (Proposição 5.2.1). Portanto,

$$P_{\Sigma_r\to\Sigma'_j}\big(P_{\Sigma\to\Sigma_r}(\mathop{||}_{j=1}^{m} P_{\Sigma\to\Sigma_j}s)\big)t_j = P_{\Sigma_r\to\Sigma'_j}\big(\mathop{||}_{j=1}^{m} P_{\Sigma_j\to\Sigma'_j}(P_{\Sigma\to\Sigma_j}s)\big)t_j = P_{\Sigma_j\to\Sigma'_j}(P_{\Sigma\to\Sigma_j}s))t_j.$$

$\triangleleft$

Então,

$$P_{\Sigma_r\to\Sigma'_j}\big(P_{\Sigma\to\Sigma_r}(s)\big)t_j = P_{\Sigma_j\to\Sigma'_j}(P_{\Sigma\to\Sigma_j}s)t_j. \tag{5.22}$$

Da equação (5.21) juntamente com a equação (5.22), tem-se que

$$P_{\Sigma_j\to\Sigma'_j}(P_{\Sigma\to\Sigma_j}s)t_j \in P_{\Sigma_j\to\Sigma'_j}(S_j), \forall j \in J,$$

ou

$$\theta_j(P_{\Sigma\to\Sigma_j}s)t_j \in \theta_j(S_j), \forall j \in J.$$

Sendo *i.* e *ii.* verdadeiras, pode-se garantir que *iii.* é verdadeira também. Ou seja, pode-se afirmar que

$$\mathop{||}_{j=1}^{m} \theta_j(P_{\Sigma\to\Sigma_j}s)t_j \subseteq \mathop{||}_{j=1}^{m} \theta_j(S_j).$$

$\square$

Da mesma forma que na seção anterior, deve-se mostrar que existe $u \in \Sigma^*$ tal que $su \in \mathop{||}_{j=1}^{m} S_j$. O Lema 5.3.4 generaliza o Lema 5.3.2 e mostra que, sob as condições utilizadas nos teoremas, dado $t \in \Sigma'^*$ tal que $\theta(s)t \in \mathop{||}_{j=1}^{m} \theta_j(S_j)$, existe $u \in \Sigma'^*$ tal que $su \in \mathop{||}_{j=1}^{m} S_j$.

Lema 5.3.4. *Seja $\theta_j(S_j)$, $\forall j \in J$, OP-abstrações. Se $\Sigma_s \subseteq \Sigma_r$ com $\Sigma_s = \mathop{\cup}_{k,l\in J}^{k\neq l} (\Sigma_k \cap \Sigma_l)$, $\exists s \in \Sigma^*$ e $\exists t \in \Sigma'^*$, tal que $\theta(s)t \in \mathop{||}_{j=1}^{m} \theta_j(S_j)$ então $\exists u \in \Sigma^*$ tal que $su \in \mathop{||}_{j=1}^{m} S_j$.*

Demonstração. Para uniformizar a notação, as projeções θ_j e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_j \approx P_{\Sigma_j \to \Sigma'_j}$ e $\theta \approx P_{\Sigma \to \Sigma_r}$. Se $\exists t \in \Sigma'^*$, tal que $P_{\Sigma \to \Sigma_r}(s)t \in \overset{m}{\underset{j=1}{||}} P_{\Sigma_j \to \Sigma'_j}(S_j)$, então $\exists t_j \in \Sigma'^*_j$ tal que $P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j}s)t_j \in P_{\Sigma_j \to \Sigma'_j}(S_j)$ (Lema 5.3.3). Como $P_{\Sigma_j \to \Sigma'_j}(S_j)$ são OP-abstrações, $\exists u_j \in \Sigma'^*_j$ tal que $P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j}(s)u_j) = P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j}(s))t_j$ e $P_{\Sigma \to \Sigma_j}(s)u_j \in S_j$. Para mostrar que $\exists u \in \Sigma'^*$ tal que $su \in \overset{m}{\underset{j=1}{||}} S_j$, escolhe-se $u \in \overset{m}{\underset{j=1}{||}} u_j$. Deve-se mostrar então que $\overset{m}{\underset{j=1}{||}} u_j \neq \emptyset$. Sabe-se que $\overset{m}{\underset{j=1}{||}} t_j \neq \emptyset$. Portanto, $\overset{m}{\underset{j=1}{||}} t_j = \overset{m}{\underset{j=1}{||}} P^{-1}_{\Sigma_r \to \Sigma'_j} t_j \neq \emptyset$.

Outra forma de representar que $\overset{m}{\underset{j=1}{||}} t_j \neq \emptyset$ consiste em projetar todas as cadeias no alfabeto comum e verificar sua interseção. Como nem todos os eventos de Σ_s estão em cada t_j, após realizar a projeção de t_j no conjunto de eventos comuns $(\Sigma'_j \cap \Sigma'_s)$, deve-se completar a cadeia resultante com cadeias formadas pelos eventos comuns restantes (colocar auto-laços com os eventos restantes nos estados do autômato correspondente). Portanto, deve-se projetar primeiro no alfabeto $\Sigma'_s \cap \Sigma_j$, com $\Sigma'_s = \overset{k \neq l}{\underset{k,l \in J}{\cup}} (\Sigma'_k \cap \Sigma'_l)$, e depois fazer a projeção inversa no alfabeto Σ'_s. Então, se $\overset{m}{\underset{j=1}{||}} t_j \neq \emptyset$, significa que $\overset{m}{\underset{j=1}{\cap}} P^{-1}_{\Sigma'_s \to (\Sigma'_j \cap \Sigma'_s)} P_{\Sigma'_j \to (\Sigma'_j \cap \Sigma'_s)} t_j \neq \emptyset$. Como $t_j = P_{\Sigma_j \to \Sigma'_j}(u_j)$, obtém-se:

$$\begin{aligned}
\overset{m}{\underset{j=1}{\cap}} P^{-1}_{\Sigma'_s \to (\Sigma'_j \cap \Sigma'_s)} P_{\Sigma'_j \to (\Sigma'_j \cap \Sigma'_s)} t_j &= \overset{m}{\underset{j=1}{\cap}} P^{-1}_{\Sigma'_s \to (\Sigma'_j \cap \Sigma'_s)} P_{\Sigma'_j \to (\Sigma'_j \cap \Sigma'_s)} P_{\Sigma_j \to \Sigma'_j}(u_j) \\
&= \overset{m}{\underset{j=1}{\cap}} P^{-1}_{\Sigma'_s \to (\Sigma'_j \cap \Sigma'_s)} P_{\Sigma'_j \to (\Sigma'_j \cap \Sigma'_s)} P_{\Sigma_j \to \Sigma'_j}(u_j) \\
&= \overset{m}{\underset{j=1}{\cap}} P^{-1}_{\Sigma'_s \to (\Sigma'_j \cap \Sigma'_s)} P_{\Sigma_j \to (\Sigma'_j \cap \Sigma'_s)} u_j. \qquad (5.23)
\end{aligned}$$

Os alfabetos das projeções são tratados a seguir. Inicialmente, trata-se $\Sigma'_s = \overset{k \neq l}{\underset{k,l \in J}{\cup}} (\Sigma'_k \cap \Sigma'_l)$.

$$\begin{aligned}
\Sigma'_s &= \overset{k \neq l}{\underset{k,l \in J}{\cup}} \Sigma'_k \cap \Sigma'_l \\
&= \overset{k \neq l}{\underset{k,l \in J}{\cup}} (\Sigma_k \cap \Sigma_r) \cap (\Sigma_l \cap \Sigma_r) \\
&= \overset{k \neq l}{\underset{k,l \in J}{\cup}} (\Sigma_k \cap \Sigma_l \cap \Sigma_r) \\
&= \overset{k \neq l}{\underset{k,l \in J}{\cup}} (\Sigma_k \cap \Sigma_l) \cap \Sigma_r \\
&= \Sigma_s \cap \Sigma_r = \Sigma_s,
\end{aligned}$$

pois $\overset{k \neq l}{\underset{k,l \in J}{\cup}} (\Sigma_k \cap \Sigma_l) = \Sigma_s$ e $\Sigma_s \subseteq \Sigma_r$.

Para o alfabeto $\Sigma'_j \cap \Sigma'_s$ tem-se:

$$\begin{aligned}\Sigma'_j \cap \Sigma'_s &= \Sigma_j \cap \Sigma_r \cap \Sigma_s \\ &= \Sigma_j \cap \Sigma_s\end{aligned}$$

pois $\Sigma_s \subseteq \Sigma_r$. Portanto, equação (5.23) pode ser reescrita como $\bigcap_{j=1}^{m} P^{-1}_{\Sigma_s \to (\Sigma_j \cap \Sigma_s)} P_{\Sigma_j \to (\Sigma_j \cap \Sigma_s)} u_j$ e:

$$\bigcap_{j=1}^{m} P^{-1}_{\Sigma'_s \to (\Sigma'_j \cap \Sigma'_s)} P_{\Sigma'_j \to (\Sigma'_j \cap \Sigma'_s)} t_j = \bigcap_{j=1}^{m} P^{-1}_{\Sigma_s \to (\Sigma_j \cap \Sigma_s)} P_{\Sigma_j \to (\Sigma_j \cap \Sigma_s)} u_j \neq \emptyset. \tag{5.24}$$

Usando raciocínio semelhante ao utilizado anteriormente, $\bigcap_{j=1}^{m} P^{-1}_{\Sigma_s \to (\Sigma_j \cap \Sigma_s)} P_{\Sigma_j \to (\Sigma_j \cap \Sigma_s)} u_j \neq \emptyset$ implica que $\|_{j=1}^{m} u_j \neq \emptyset$. Consequentemente, $\exists u \in \|_{j=1}^{m} u_j$ tal que $su \in \|_{j=1}^{m} S_j$. □

O teorema que se segue apresenta condições suficientes para que o teste de não-conflito possa ser aplicado sobre as abstrações, reduzindo assim a complexidade computacional do teste.

Teorema 5.3.3. *Sejam as linguagens $S_j \subseteq \Sigma_j^*$, $\forall j \in J$, as projeções naturais θ_j, θ, $P_{\Sigma \to \Sigma_j}$, $P_{\Sigma_r \to \Sigma'_j}$ e $P^{-1}_{\Sigma_r \to \Sigma'_j}$ como definidos anteriormente. Se $\Sigma_s \subseteq \Sigma_r$ com $\Sigma_s = \bigcup_{k,l \in J}^{k \neq l} (\Sigma_k \cap \Sigma_l)$ e as projeções naturais $\theta_j(S_j)$, $\forall j \in J$, são OP-abstrações então*

$$\|_{j=1}^{m} \overline{\theta_j(S_j)} = \overline{\|_{j=1}^{m} \theta_j(S_j)} \iff \|_{j=1}^{m} \overline{S_j} = \overline{\|_{j=1}^{m} S_j}.$$

Demonstração. A demonstração deste teorema é dividida em duas partes.

i. $\|_{i=1}^{m} \overline{S_j} = \overline{\|_{j=1}^{m} S_j} \Rightarrow \|_{j=1}^{m} \overline{\theta_j(S_j)} = \overline{\|_{j=1}^{m} \theta_j(S_j)}$.

Tem-se que

$$\|_{j=1}^{m} \overline{\theta_j(S_j)} = \|_{j=1}^{m} \theta_j(\overline{S_j}).$$

▷

Como $\Sigma_s \subseteq \Sigma_r$, então $\|_{j=1}^{m} \theta_j(\overline{S_j}) = \theta(\|_{j=1}^{m} \overline{S_j})$, pela Proposição 5.2.1.

◁

Além disso, por hipótese, $\|_{j=1}^{m} \overline{S_j} = \overline{\|_{j=1}^{m} S_j}$. Portanto,

$$\theta\big(\|_{j=1}^{m} \overline{S_j}\big) = \theta\big(\overline{\|_{j=1}^{m} S_j}\big) = \overline{\theta(\|_{j=1}^{m} S_j)}. \tag{5.25}$$

▷

Usando novamente o fato de que $\Sigma_s \subseteq \Sigma_r$, pode-se aplicar a Proposição 5.2.1 sobre

equação (5.25), obtendo-se

$$\overline{\theta(\mathop{||}_{j=1}^{m} S_j)} = \overline{\mathop{||}_{j=1}^{m} \theta_j(S_j)}.$$

$\lhd$

Então,

$$\mathop{||}_{j=1}^{m} \overline{\theta_j(S_j)} = \overline{\mathop{||}_{j=1}^{m} \theta_j(S_j)}.$$

ii. $\mathop{||}_{j=1}^{m} \overline{\theta_j(S_j)} = \overline{\mathop{||}_{j=1}^{m} \theta_j(S_j)} \Rightarrow \mathop{||}_{j=1}^{m} \overline{S_j} = \overline{\mathop{||}_{j=1}^{m} S_j}$.

Para uniformizar a notação de projeções nesta parte da demonstração, as projeções θ_j e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_j \approx P_{\Sigma_j \to \Sigma'_j}$ e $\theta \approx P_{\Sigma \to \Sigma_r}$.

$\rhd$

Como $\mathop{||}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(S_j) = P_{\Sigma \to \Sigma_r}(\mathop{||}_{j=1}^{m} S_j)$, pela Proposição 5.2.1, e $P_{\Sigma \to \Sigma_r}(\overline{S}) = \overline{P_{\Sigma \to \Sigma_r}(S)}$, pode-se reescrever *ii.* como

$$P_{\Sigma \to \Sigma_r}(\mathop{||}_{j=1}^{m} \overline{S_j}) = P_{\Sigma \to \Sigma_r}(\overline{\mathop{||}_{j=1}^{m} S_j}) \Longrightarrow \mathop{||}_{j=1}^{m} \overline{S_j} = \overline{\mathop{||}_{j=1}^{m} S_j}. \tag{5.26}$$

$\lhd$

É suficiente mostrar que dado $s \in \mathop{||}_{j=1}^{m} \overline{S_j}$ tem-se que $s \in \overline{\mathop{||}_{j=1}^{m} S_j}$. Por hipótese, $s \in \mathop{||}_{j=1}^{m} \overline{S_j}$. Então, $\theta(s) \in \theta(\mathop{||}_{j=1}^{m} \overline{S_j})$. Ainda, pelo lado esquerdo da equação (5.26), pode-se dizer que, $P_{\Sigma \to \Sigma_r}(s) \in P_{\Sigma \to \Sigma_r}(\overline{\mathop{||}_{j=1}^{m} S_j}) = \overline{P_{\Sigma \to \Sigma_r}(\mathop{||}_{j=1}^{m} S_j)}$. Portanto, $\exists t \in \Sigma_r^*$ tal que $P_{\Sigma \to \Sigma_r}(s)t \in P_{\Sigma \to \Sigma_r}(\mathop{||}_{j=1}^{m} S_j)$. Além disso, do Lema 5.3.3, $\exists t_j \in \Sigma_j'^*$, $\forall j \in J$, tal que

$$\begin{aligned} P_{\Sigma \to \Sigma_r}(s)t \in \mathop{||}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j}s)t_j &\subseteq \mathop{||}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(S_j) \\ P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j}s)t_j &\in P_{\Sigma_j \to \Sigma'_j}(S_j). \end{aligned} \tag{5.27}$$

Por hipótese, $P_{\Sigma \to \Sigma_j}s \in \overline{S_j}$. Como $P_{\Sigma_j \to \Sigma'_j}(S_j)$ é uma OP-abstração, pode-se dizer que $\exists u_j \in \Sigma_j^*$ tal que $P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j}(s)u_j) = P_{\Sigma_j \to \Sigma'_j}(P_{\Sigma \to \Sigma_j}s)t_j$ (Definição 5.2.1) e tal que $P_{\Sigma \to \Sigma_j}(s)u_j \in S_j$, então

$$\mathop{||}_{j=1}^{m} P_{\Sigma \to \Sigma_j}(s)u_j \subseteq \mathop{||}_{j=1}^{m} S_j.$$

$\rhd$

Escolhendo $u \in \mathop{||}_{j=1}^{m} u_j$, tem-se que $su \in s(\mathop{||}_{j=1}^{m} u_j)$. Pelo Lema 5.3.3, tem-se que $\mathop{||}_{j=1}^{m} u_j \neq \emptyset$ e, portanto, $\exists u \in \Sigma^*$ tal que $u \in \mathop{||}_{j=1}^{m} u_j$ e $su \in \mathop{||}_{j=1}^{m} S_j$.

$\lhd$
Como $su \in \overset{m}{\underset{j=1}{||}} S_j$, então $s \in \overline{\overset{m}{\underset{j=1}{||}} S_j}$.

□

Esse teorema é uma extensão do Teorema 5.3.1 para o caso de de m supervisores.

A seguir, o Teorema 5.3.2 é estendido para o caso de múltiplos supervisores, e seu enunciado e demonstração são apresentados a seguir.

Teorema 5.3.4. *Sejam as linguagens $S_j \subseteq \Sigma_j^*$, $\forall j \in J$, as projeções naturais θ_j, θ, $P_{\Sigma\to\Sigma_j}$, $P_{\Sigma_r\to\Sigma_j'}$ e $P^{-1}_{\Sigma_r\to\Sigma_j'}$, como definidos anteriormente. Se $\Sigma_s \subseteq \Sigma_r$ com $\Sigma_s = \overset{k\neq l}{\underset{k,l\in J}{\cup}} (\Sigma_k \cap \Sigma_l)$ e as projeções naturais $\theta_j(S_j)$, $\forall j \in J$, são OP-abstrações então $\theta(\overset{m}{\underset{j=1}{||}} S_j)$ é também uma OP-abstração.*

Demonstração. Para uniformizar a notação, as projeções θ_j e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_j \approx P_{\Sigma_j\to\Sigma_j'}$ e $\theta \approx P_{\Sigma\to\Sigma_r}$. Seja $s \in \overline{\overset{m}{\underset{j=1}{||}} S_j}$, tem-se que $s \in \overset{m}{\underset{j=1}{||}} P_{\Sigma\to\Sigma_j}s$. Escolhe-se um $t \in \Sigma_r^*$ tal que $P_{\Sigma\to\Sigma_r}(s)t \in P_{\Sigma\to\Sigma_r}(\overset{m}{\underset{j=1}{||}} S_j)$. Tem-se que

$$\begin{aligned} P_{\Sigma\to\Sigma_r}(s) &\in P_{\Sigma\to\Sigma_r}(\overset{m}{\underset{j=1}{||}} P_{\Sigma\to\Sigma_j}s) \\ P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(\overset{m}{\underset{j=1}{||}} P_{\Sigma\to\Sigma_j}s)t. \end{aligned}$$

$\rhd$
Como $\Sigma_s \subseteq \Sigma_r$, usando Proposição 5.2.1, tem-se que

$$\begin{aligned} P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(\overset{m}{\underset{j=1}{||}} P_{\Sigma\to\Sigma_j}s)t \\ &\in \big[\overset{m}{\underset{j=1}{||}} P_{\Sigma_j\to\Sigma_j'}(P_{\Sigma\to\Sigma_j}s)\big]t. \end{aligned}$$

$\lhd$

Então, considera-se $t_j = P_{\Sigma_r\to\Sigma_j'}t$, $\forall j \in J$. Para essas escolhas, foi mostrado, no Lema 5.3.3, que $P_{\Sigma_j\to\Sigma_j'}(P_{\Sigma\to\Sigma_j}s)t_j \in P_{\Sigma_j\to\Sigma_j'}(S_j)$. Pela Definição 5.2.1, como $P_{\Sigma_j\to\Sigma_j'}$ é uma OP-abstração, $\forall j \in J$, $\exists u_j \in \Sigma_j^*$, tal que

$$\begin{aligned} P_{\Sigma_j\to\Sigma_j'}(P_{\Sigma\to\Sigma_j}(s)u_j) &= P_{\Sigma_j\to\Sigma_j'}(P_{\Sigma\to\Sigma_j}s)t_j \in P_{\Sigma_j\to\Sigma_j'}(S_j) \quad \text{e} \\ P_{\Sigma\to\Sigma_j}(s)u_j &\in S_j. \end{aligned}$$

Pretende-se mostrar que $\exists u \in \Sigma^*$, tal que $P_{\Sigma\to\Sigma_r}(s)t = P_{\Sigma\to\Sigma_r}(su)$ e $su \in \mathop{||}_{j=1}^{m} S_j$. Tem-se que $t \in \bigcap_{j=1}^{m} P^{-1}_{\Sigma_r\to\Sigma'_j} t_j$, $\forall j \in J$. Além disso, $t_j = P_{\Sigma_j\to\Sigma'_j}(u_j)$. Então,

$$\begin{aligned} P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(s)\big[\bigcap_{j=1}^{m} P^{-1}_{\Sigma_r\to\Sigma'_j} t_j\big] \\ &= P_{\Sigma\to\Sigma_r}(s)\big[\bigcap_{j=1}^{m} P^{-1}_{\Sigma_r\to\Sigma'_j} P_{\Sigma_j\to\Sigma'_j}(u_j)\big] \\ &= P_{\Sigma\to\Sigma_r}(s)\big[\mathop{||}_{j=1}^{m} P_{\Sigma_j\to\Sigma'_j}(u_j)\big] \\ P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(s)\big[\mathop{||}_{j=1}^{m} P_{\Sigma_j\to\Sigma'_j}(u_j)\big]. \end{aligned} \tag{5.28}$$

$\triangleright$

Usando a Proposição 5.2.1 e equação (5.28), tem-se

$$\begin{aligned} P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}(s)P_{\Sigma\to\Sigma_r}\big(\mathop{||}_{j=1}^{m} u_j\big) \\ P_{\Sigma\to\Sigma_r}(s)t &\in P_{\Sigma\to\Sigma_r}\Big[s\big(\mathop{||}_{j=1}^{m} u_j\big)\Big] \end{aligned}$$

Como $\Sigma_s \subseteq \Sigma_r$, pode-se usar Lema 5.3.4 para garantir que $\exists u \in \Sigma^*$, tal que $u \in \mathop{||}_{j=1}^{m} u_j$ e $su \in \mathop{||}_{j=1}^{m} S_j$.

$\triangleleft$

Para completar as condições sobre as abstrações para que se enquadrem como OP-abstrações, deve-se mostrar que $t = P_{\Sigma\to\Sigma_r}(u)$. Essa igualdade é demonstrada por contradição. Parte-se de $t \neq P_{\Sigma\to\Sigma_r}(u)$.

Aplicando-se $P_{\Sigma_r\to\Sigma'_j}$ sobre os dois lados, obtém-se que:

$$P_{\Sigma_r\to\Sigma'_j} t \neq P_{\Sigma_r\to\Sigma'_j} P_{\Sigma\to\Sigma_r}(u) \tag{5.29}$$

É sabido que $P_{\Sigma_r\to\Sigma'_j} t = t_j$ (do lado esquerdo), e pode-se manipular as projeções do lado direito de forma a obter:

$$\begin{aligned} P_{\Sigma_r\to\Sigma'_j} P_{\Sigma\to\Sigma_r}(u) &= P_{\Sigma_r\to\Sigma'_j} P_{\Sigma\to\Sigma_r}(u) = P_{\Sigma\to\Sigma'_j} u = P_{\Sigma_j\to\Sigma'_j} P_{\Sigma\to\Sigma_j} u \\ &= P_{\Sigma_j\to\Sigma'_j} u_j = P_{\Sigma_j\to\Sigma'_j}(u_j). \end{aligned} \tag{5.30}$$

Na equação (5.29), substitui-se $P_{\Sigma_r\to\Sigma'_1} t = t_1$ no lado esquerdo e $P_{\Sigma_r\to\Sigma'_1} P_{\Sigma\to\Sigma_r}(u) = \theta_1(u_1)$ (equação (5.30)) no lado direito obtendo-se $t_j \neq P_{\Sigma_j\to\Sigma'_j}(u_j)$, o que contradiz a definição de

t_j na hipótese. Portanto, é verdadeiro que $t = P_{\Sigma\to\Sigma_r}(u)$.

Se as projeções naturais $P_{\Sigma_j\to\Sigma'_j}(S_j)$, $\forall j \in J$, são OP-abstrações e $\Sigma_s \subseteq \Sigma_r$, para qualquer t escolhido, com $P_{\Sigma\to\Sigma_r}(s)t \in P_{\Sigma\to\Sigma_r}(\mathop{||}_{j=1}^{m} S_j)$, pode-se obter $u \in \Sigma^*$ tal que $P_{\Sigma\to\Sigma_r}(s)t = P_{\Sigma\to\Sigma_r}(su)$ e $su \in \mathop{||}_{j=1}^{m} S_j$ e finalmente concluir que $P_{\Sigma\to\Sigma_r}(\mathop{||}_{j=1}^{m} S_j)$ e, portanto, $\theta(\mathop{||}_{j=1}^{m} S_j)$ é uma OP-abstração. □

Abstrações obtidas respeitando as condições dos Teoremas 5.3.3 e 5.3.4 são referidas como *W-abstrações*. A seguir, apresenta-se o procedimento a ser seguido para obter W-abstrações. No final da seção, apresenta-se um exemplo para ilustrar o procedimento.

5.4 Procedimento:

Para aplicar os resultados do Teorema 5.3.3 sobre um sistema de controle modular, propõe-se o procedimento a seguir.

Algoritmo 1:

1. Obtêm-se os supervisores locais;
2. Obtém-se o conjunto inicial de eventos relevantes, Σ_r^I;
3. Obtêm-se os conjuntos iniciais de eventos relevantes locais como $\Sigma_j'^I = \Sigma_j \cap \Sigma_r^I$, $\forall j \in J$;
4. Obtêm-se OP-abstrações $\theta_j(S_j)$ para cada supervisor S_j (pode ser necessário estender o conjunto de eventos Σ_r^I);
5. Usam-se as abstrações $\theta_j(S_j)$, $\forall j \in J$, no teste de não-conflito.

A discussão principal é realizada em torno do conjunto de eventos relevantes Σ_r, parâmetro das projeções naturais $\theta_j(S_j)$. O conjunto Σ_r deve ser tal que as condições dos Teoremas 5.3.3 e 5.3.4 sejam atendidas, ou seja,

c1. $\Sigma_s \subseteq \Sigma_r$ (onde Σ_s representa o conjunto de eventos compartilhados por mais de um supervisor) e

c2. $\theta_j(S_j)$ é uma *OP-abstração*, $\forall j \in J$.

Dado o conjunto inicial de eventos relevantes ($\Sigma_r^I = \Sigma_s$, neste caso) esse conjunto é expandido de forma a a gerar OP-abstrações para os supervisores, no passo 4.

O problema de encontrar uma extensão mínima para Σ_r^I é NP-difícil [Feng, 2006]. No entanto, Feng [2006] propõe um algoritmo polinomial (em relação a estados e transições)

que retorna uma extensão "razoável", não necessariamente mínima, para cada conjunto local de eventos relevantes, tal que a propriedade do observador seja obtida para cada projeção. No entanto, seu algoritmo não está implementado. O Capítulo 7 traz uma algoritmo de complexidade polinomial para verificação da propriedade do observador, dada a linguagem e o conjunto de eventos relevantes Σ_r.

De acordo com as condições do Teorema 5.3.3, tem-se que $\Sigma_r^I = \Sigma_s$ e o passo 2 do algoritmo pode ser substituído por:

Procedimento W para gerar Σ_r^I:

a. $\Sigma_r^I = \Sigma_s$, com $\Sigma_s = \bigcup_{k,l \in J}^{k \neq l} \Sigma_k \cap \Sigma_l$.

A combinação do Algoritmo 1 com o Procedimento W é chamada de **W-solução**. A seguir, apresenta-se um exemplo que ilustra a aplicação do algoritmo e a redução obtida pela aplicação do teste sobre as W-abstrações.

Exemplo 5.4.1. *Sistema de Manufatura Integrado -* ***W-solução***
Considera-se o Sistema Integrado de Manufatura (IMS) da Figura 5.6. O sistema é formado de sete componentes: dois tornos ($Torno_1$ e $Torno_2$), duas máquinas de pintura (MP_1 e MP_2), dois veículos auto-guiados (AGV_1 e AGV_2) e uma máquina de montagem (MM). Tornos 1 e 2 são alimentados por blocos que são processados e pintados por MP_1 e MP_2, respectivamente. Então, AGV_1 e AGV_2 ligam os depósitos B_3-B_5 e B_4-B_6 fornecendo as duas partes inacabados de MP_1 e MP_2 para MM. MM monta as duas partes. A interface entre as máquinas consiste de depósitos B_k, $k = \{1, \ldots, 6\}$, cada um com capacidade para 1 parte, e ainda um trilho que é compartilhado por AGV_1 e AGV_2. O problema de controle consiste em dar o maior grau de liberdade para o IMS, impedindo o "overflow" e "underflow" de partes nos depósitos. Além disso, os dois AGVs não devem utilizar os trilhos ao mesmo tempo. Os eventos controláveis são rotulados com números ímpares.

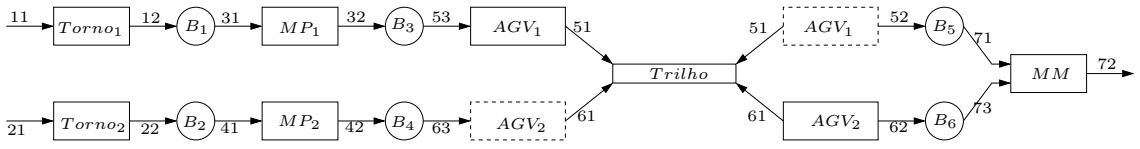
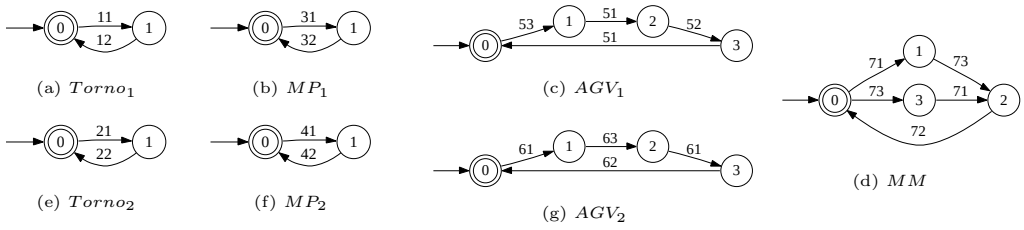


Figura 5.6: Exemplo 5.4.1 - Diagrama do IMS

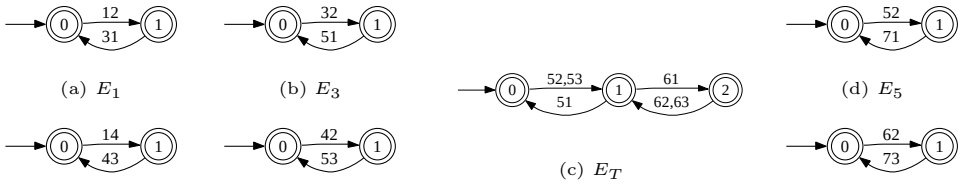
Os modelos das subplantas são apresentados na Figura 5.7 e as especificações são apresentadas na Figura 5.8. As plantas locais e suas respectivas especificações são:

- $G_1 = Torno_1 || MP_1$, *especificação* E_1;
- $G_2 = Torno_2 || MP_2$, *especificação* E_2;
- $G_3 = MP_1 || AGV_1$, *especificação* E_3;



(a) $Torno_1$ (b) MP_1 (c) AGV_1 (d) MM (e) $Torno_2$ (f) MP_2 (g) AGV_2

Figura 5.7: Exemplo 5.4.1 - Modelos das subplantas (a) $Torno_1$; (b) MP_1; (c) AGV_1; (d) MM; (e) $Torno_2$; (f) MP_2; (g) AGV_2



(a) E_1 (b) E_3 (c) E_T (d) E_5 (e) E_2 (f) E_4 (g) E_6

Figura 5.8: Exemplo 5.4.1 - Especificações E_j, $j = \{1, \ldots, 6, T\}$

- $G_4 = MP_2||AGV_2$, *especificação* E_4;
- $G_5 = AGV_1||MM$, *especificação* E_5;
- $G_6 = AGV_2||MM$, *especificação* E_6;
- $G_T = AGV_1||AGV_2$, *especificação* E_T.

As especificações locais K_j, *resultantes da composição de cada especificação genérica local com a planta local, são todas não-controláveis. As máximas sublinguagens controláveis de* K_j *com relação a* $\mathcal{L}_m(G_j)$, *denotadas por* $S_j = Sup\mathcal{C}(K_j, \mathcal{L}_m(G_j))$, *são obtidas; suas representações por autômatos são omitidas. O número de estados e transições (estados, transições) dos supervisores obtidos são:* S_1 *(6,8),* S_2 *(6,8),* S_3 *(12,18),* S_4 *(12,18),* S_5 *(28,51),* S_6 *(28,51) e* S_T *(12,16). O teste de não-conflito calculado sobre os supervisores gera um autômato de 12.960 estados e 49.968 transições e detecta o não-conflito.*

O conjunto de eventos que não são compartilhados por qualquer par de supervisores é $\{11, 12, 21, 22\}$. *Portanto, o conjunto inicial de eventos relevantes é* $\Sigma_r^I = \Sigma_s = \{31, 32, 41, 42, 51, 52, 53, 61, 62, 63, 71, 72, 73\}$. *Esse conjunto é estendido para incluir* $\{11, 21\}$ *para obter OP-abstrações para* S_1 *e* S_2. *Dessa forma, os eventos* $\{12, 22\}$ *podem ser apagados. A Figura 5.9 apresenta os dois supervisores e suas respectivas abstrações. Os supervisores* S_j, *com* $j \in \{4, 5, 6, T\}$, *não podem ser reduzidos usando esse método. Os supervisores originais são usados, considerando-os observadores de si mesmos. O teste de não-conflito deve portanto ser aplicado sobre as abstrações como se segue:*

$$\overline{\theta_1(S_1)}||\overline{\theta_2(S_2)}||\overline{S_3}||\overline{S_4}||\overline{S_5}||\overline{S_6}||\overline{S_T} \stackrel{?}{=} \overline{\theta_1(S_1)||\theta_2(S_2)||S_3||S_4||S_5||S_6||S_T}.$$

Conforme esperado, a igualdade é verificada. O autômato gerado pelo teste sobre as abstrações possui 5.760 estados e 20.928 transições e identifica a não-existência de conflito. As duas

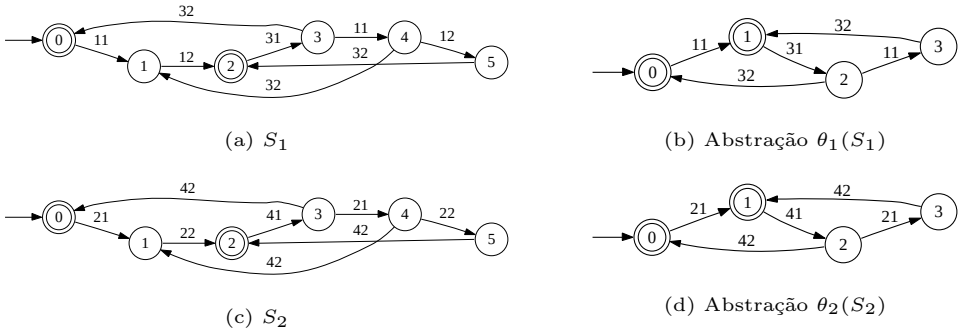


Figura 5.9: Exemplo 5.4.1 - Supervisores e abstrações gerados usando **W**: (a) Supervisor S_1; (b) $\theta_1(S_1)$; (c) Supervisor S_2 e (d) $\theta_2(S_2)$

abstrações foram responsáveis por uma redução de 55% *no número de estados e de* 58% *no número de transições no teste.*

A ferramenta utilizada para manipulação dos autômatos e cálculo dos supervisores nesta tese é o TCT, ferramenta desenvolvida pelo grupo deo Prof. Wonham [Feng e Wonham, 2006b].

5.5 Discussão

O presente capítulo apresentou um novo teste de não-conflito aplicado sobre abstrações das linguagens, obtidas pela operação de projeção natural. Condições suficientes para as abstrações foram apresentadas. A apresentação dos resultados foi feita em duas partes, em que primeiramente se apresentou o caso simples, com 2 supervisores, e posteriormente a extensão dos resultados para m supervisores. Em seguida, um procedimento para obtenção das abstrações e um exemplo ilustrativo foram apresentados.

Nesse trabalho, o problema da complexidade do teste de não-conflito de supervisores modulares é o foco principal. No entanto, pode-se perceber nas demonstrações, que propriedades estruturais dos supervisores não são utilizadas. A condição sobre os eventos mantidos pela abstração e a obtenção de OP-abstrações podem ser aplicados sobre qualquer conjunto de linguagens. Portanto, os resultados apresentados nesse capítulo podem ser aplicados à um conjunto arbitrário de linguagens, do qual que se queira verificar a existência de conflito.

No Capítulo 6, os resultados apresentados neste capítulo são validados para um novo conjunto Σ_r. Para que os resultados permaneçam válidos, as partes das demonstrações que são relacionadas ao conjunto Σ_r devem ser validadas para o novo conjunto Σ_r. Tem-se que cada passagem das demonstrações deste capítulo dependente da condição sobre os eventos relevantes foi demarcada com o par $\triangleright, \triangleleft$. Essas passagens utilizam a Proposição 5.2.1 ou um dos Lemas 5.3.2, 5.3.3. O Capítulo 6 apresenta demonstrações de duas Proposições (correspondente à Proposição 5.2.1 para o novo conjunto Σ_r quando se tem 2 e m supervisores) e de dois Lemas (correspondentes aos Lemas 5.3.2 e 5.3.3 para o novo conjunto Σ_r). Desta

forma, as referências à Proposição 5.2.1 e aos Lemas 5.3.2 e 5.3.3 poderão ser substituídas por referências a estas novas Proposições e Lemas e os resultados apresentados nos Teoremas 5.3.3 e 5.3.4 continuarão válidos para o novo conjunto de abstrações.

Capítulo 6

Verificação de Não-Conflito Utilizando Propriedades Estruturais dos Supervisores

No capítulo anterior, todos os eventos compartilhados por qualquer par de supervisores eram mantidos nas abstrações. O procedimento de construção dos supervisores possui uma etapa de composição de subplantas e especificações. Supervisores que compartilham algum evento de uma subplanta compartilham todos os eventos daquela subplanta. No entanto, muitas vezes, eventos que pertencem a uma determinada subplanta não são relevantes para o controle. No caso de não se utilizar este conhecimento estrutural dos supervisores, estes eventos teriam que ser mantidos nas abstrações, mesmo não agregando informação a respeito das possibilidades de ocorrência de conflito. Esta observação motivou o estudo de formas alternativas para construir as abstrações, explorando a estrutura do sistema e das especificações. Como as propriedades estruturais dos supervisores não vêm sendo utilizadas, os resultados até agora apresentados podem ser aplicados a qualquer conjunto de linguagens do qual se queira detectar a presença ou ausência de conflito. No entanto, dados como: as subplantas são assíncronas, as linguagens implementadas pelos supervisores são sublinguagens de suas plantas etc; podem ser usados para derivar outras condições suficientes para as abstrações, que não assumem $\Sigma_s \subseteq \Sigma_r$. Este capítulo apresenta um novo conjunto de condições suficientes para as abstrações que, em alguns casos, permite apagar na projeção eventos compartilhados. Estes resultados foram apresentados em [Pena et al., 2006b].

Duas condições foram extensivamente utilizadas no capítulo anterior, como sendo suficientes para que os lemas e teoremas fossem demonstrados. São elas:

- $\theta_j(S_j)$ são OP-abstrações;
- eventos compartilhados são incluídos no conjunto de eventos relevantes, ou seja, $\Sigma_s \subseteq \Sigma_r$ onde $\Sigma_s = \bigcup_{k,l \in J}^{k \neq l} (\Sigma_k \cap \Sigma_l)$.

A condição que muda neste capítulo é a condição sobre os eventos relevantes (Σ_r). Esta condição é usada em dois momentos para cada uma das demonstrações dos teoremas do capítulo anterior: (i) para utilização da propriedade de distributividade das projeções; (ii) para garantir que dado $t \in \Sigma'^*$ tal que $\theta(s)t \in \|_{j=1}^{m} \theta_j(S_j)$, existe $u \in \Sigma^*$ tal que $su \in \|_{j=1}^{m} S_j$. As passagens mencionadas estão delimitadas pelo par $\triangleright$,$\triangleleft$, nas demonstrações dos Lemas 5.3.1, 5.3.3 e dos Teoremas 5.3.1, 5.3.2 e 5.3.3, 5.3.4.

O objetivo deste capítulo é, como mencionado anteriormente, flexibilizar essa restrição sobre o conjunto de eventos relevantes. Para que os resultados apresentados no capítulo anterior continuem válidos, deve-se mostrar que, sob o novo conjunto de condições suficientes, as condições expressas em (i) e (ii) continuam válidas.

O novo conjunto de condições sobre as abstrações $\theta_j(S_j)$, sob as quais (i) e (ii) são válidas, é apresentado a seguir:

- $\theta(S_j)$ são OP-abstrações;
- K_j são controláveis em relação a $\mathcal{L}(G_j)$, $\forall j \in J$, portanto, $S_j = K_j$;
- $\bigcup_{j=1}^{m} \Sigma_{E_j} \subseteq \Sigma_r$;
- $S_j = \overline{S_j} \cap \mathcal{L}_m(G_j)$.

onde E_j representam as especificações genéricas a serem implementadas pelos supervisores S_j e K_j são as linguagens desejadas (ou especificações locais) para o sistema em malha fechada, ou seja, $K_j = E_j||\mathcal{L}_m(G_j)$. Os alfabetos Σ_{E_j} representam os conjuntos de eventos presentes nas especificações E_j.

Seguindo o padrão do último capítulo, apresentam-se os resultados para 2 supervisores e em seguida para m supervisores.

6.1 Dois Supervisores

Nas demonstrações dos Teoremas e Lemas do Capítulo 5, a condição sobre os eventos relevantes foi utilizada algumas vezes. Pode-se pervceber que essa condição sempre aparece associada à Proposição 5.2.1 ou ao Lema 5.3.2. Apresenta-se, portanto, uma proposição e um Lema sob as novas condições sobre os eventos relevantes, análogos à Proposição 5.2.1 e ao Lema 5.3.2.

Inicialmente, apresenta-se a Proposição 6.1.1, que estabelece que sob as novas condições apresentadas, a igualdade $\theta(S_1||\, S_2) = \theta_1(S_1)||\, \theta_2(S_2)$ se verifica.

As linguagens E_1 e E_2 representam as especificações genéricas a serem implementadas pelos supervisores S_1 e S_2, respectivamente, e K_1 e K_2 são as linguagens desejadas (ou

especificações locais) para o sistema em malha fechada, ou seja, $K_1 = E_1||\mathcal{L}_m(G_1)$ e $K_2 = E_2||\mathcal{L}_m(G_2)$. Os alfabetos Σ_{E_1} e Σ_{E_2} representam os conjuntos de eventos presentes nas especificações E_1 e E_2.

Proposição 6.1.1. *Sejam S_1 e S_2 linguagens implementadas pelos supervisores. A igualdade $\theta(S_1||\,S_2) = \theta_1(S_1)||\,\theta_2(S_2)$ se verifica se*

- *$\theta_1(S_1)$ e $\theta_2(S_2)$ são OP-abstrações;*
- *K_1 e K_2 são controláveis em relação a $\mathcal{L}(G_1)$ e $\mathcal{L}(G_2)$, respectivamente ($S_1 = K_1$ e $S_2 = K_2$);*
- *$(\Sigma_{E_1} \cup \Sigma_{E_2}) \subseteq \Sigma_r$;*
- *$S_1 = \overline{S_1} \cap \mathcal{L}_m(G_1)$ e $S_2 = \overline{S_2} \cap \mathcal{L}_m(G_2)$.*

Demonstração. $\theta(S_1||S_2) \subseteq \theta_1(S_1)||\theta_2(S_2)$ é sempre verdadeira. A prova da igualdade consiste portanto em mostrar que $\theta_1(S_1)||\theta_2(S_2) \subseteq \theta(S_1||S_2)$. Para uniformizar a notação, as projeções θ_1, θ_2 e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_1 \approx P_{\Sigma_1 \to \Sigma_1'}$, $\theta_2 \approx P_{\Sigma_2 \to \Sigma_2'}$ e $\theta \approx P_{\Sigma \to \Sigma_r}$.

$P_{\Sigma_1 \to \Sigma_1'}(S_1)||P_{\Sigma_2 \to \Sigma_2'}(S_2) \subseteq P_{\Sigma \to \Sigma_r}(S_1||S_2)$:

Seja

$$a \in P_{\Sigma_1 \to \Sigma_1'}(S_1)||P_{\Sigma_2 \to \Sigma_2'}(S_2) \subseteq \Sigma_r^*. \tag{6.1}$$

Então $P_{\Sigma_r \to \Sigma_1'} a \in P_{\Sigma_1 \to \Sigma_1'}(S_1)$ e $P_{\Sigma_r \to \Sigma_2'} a \in P_{\Sigma_2 \to \Sigma_2'}(S_2)$ e também $\exists s_a \in S_1 \subseteq \Sigma_1^*$ tal que $P_{\Sigma_1 \to \Sigma_1'}(s_a) = P_{\Sigma_r \to \Sigma_1'} a$ e $\exists s_b \in S_2 \subseteq \Sigma_2^*$ tal que $P_{\Sigma_2 \to \Sigma_2'}(s_b) = P_{\Sigma_r \to \Sigma_2'} a$.

Escreve-se $a = \sigma_1\sigma_2 \ldots \sigma_n$. Seja

$$s_1' \;=\; \mu_1^1\sigma_1\mu_2^1\sigma_2 \ldots \mu_n^1\sigma_n \subseteq (\Sigma_1 \cup \Sigma_r)^* \tag{6.2}$$

tal que

$$\begin{aligned} P_{(\Sigma_1 \cup \Sigma_r) \to \Sigma_1} s_1' &= s_1 \in \overline{S_1}, \\ P_{\Sigma_1 \to \Sigma_1'}(s_1) &= P_{\Sigma_1 \to \Sigma_1'}(P_{(\Sigma_1 \cup \Sigma_r) \to \Sigma_1} s_1') = P_{\Sigma_r \to \Sigma_1'} a, \\ P_{(\Sigma_1 \cup \Sigma_r) \to \Sigma_r} s_1' &= a, \end{aligned}$$

com $\mu_i^1 \in [\Sigma_1 - \Sigma_1']^*$ e $i = \{1 \ldots n\}$.

Hipótese: Para todo s_1' com as características acima, existe também um $s_2' = \mu_1^2\sigma_1\mu_2^2\sigma_2 \ldots \mu_i^2\sigma_i \ldots \mu_n^2\sigma_n$ tal que

$$\begin{aligned} P_{(\Sigma_2 \cup \Sigma_r) \to \Sigma_2} s_2' &= s_2 \in \overline{S_2} \\ P_{(\Sigma_2 \cup \Sigma_r) \to \Sigma_r} s_2' &= P_{(\Sigma_1 \cup \Sigma_r) \to \Sigma_r} s_1' = a. \end{aligned}$$

Demonstração: A demonstração usa a estratégia indutiva. Assume-se que para algum $i \geq 1$, $s_1'^{(i-1)} = \mu_1^1\sigma_1 \ldots \mu_{i-1}^1\sigma_{i-1}$, com $\mu_1^1 \ldots \mu_{i-1}^1$ definido de tal forma que $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r}s_1' = a$ e

$$s_1^{(i-1)} = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s_1'^{(i-1)} = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(\mu_1^1\sigma_1 \ldots \mu_{i-1}^1\sigma_{i-1}) \in \overline{S_1},$$

existe $s_2'^{(i-1)} = \mu_1^2\sigma_1 \ldots \ \mu_{i-1}^2\sigma_{i-1}$, com $\mu_1^2 \ldots \mu_{i-1}^2$ definido de tal forma que

$$\begin{aligned} P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_s}\mu_i^2 &= P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_s}\mu_i^1 \\ P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_r}s_2' &= a \end{aligned}$$

e

$$s_2^{(i-1)} = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_r}s_2'^{(i-1)} = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_r}(\mu_1^2\sigma_1 \ldots \mu_{i-1}^2\sigma_{i-1}) \in \overline{S_2}, \tag{6.3}$$

pretende-se mostrar que para qualquer $s_1'^{(i)}$ com as características desejadas, existe $s_2'^{(i)}$ tal que equação (6.3) é verdadeira.

Base: para $i = 1$, $s_1'^{(0)} = \epsilon$, $s_1^{(0)} = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s_1'^{(0)} \in \overline{S_1}$ e então há um $s_2'^{(0)} = \epsilon$, tal que $s_2^{(0)} = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2}s_2'^{(0)} \in \overline{S_2}$. Além disso, $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r}s_1'^{(0)} = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_r}s_2'^{(0)} = \epsilon \in \{\overline{a}\}$.

Passo de Indução: Para $2 \leq i \leq n$, se $s_1'^{(i)} = s_1'^{(i-1)}\mu_i^1\sigma_i = \mu_1^1\sigma_1 \ldots \mu_{i-1}^1\sigma_{i-1}\mu_i^1\sigma_i$, com $\mu_i^1 \in [\Sigma_1 - \Sigma_1']^*$ definidos de tal forma que $s_1^{(i)} = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s_1'^{(i)} \in \overline{S_1}$, então

$$\begin{aligned} s_1^{(i)} &= P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s_1'^{(i)} = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(s_1'^{(i-1)}\mu_i^1\sigma_i) \\ &= P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s_1'^{(i-1)}P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(\mu_i^1\sigma_i) \\ &= s_1^{(i-1)}P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(\mu_i^1\sigma_i) = s_1^{(i-1)}\mu_i^1P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(\sigma_i) \in \overline{S_1}. \end{aligned} \tag{6.4}$$

Pretende-se construir uma cadeia $\mu_i^2 \in [\Sigma_2 - \Sigma_2']^*$ tal que $P_{(\Sigma_1-\Sigma_1')\to\Sigma_s}\mu_i^1 = P_{(\Sigma_2-\Sigma_2')\to\Sigma_s}\mu_i^2$ e $P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2}\ s_2'^{(i-1)}\mu_i^2 \in \overline{S_2}$.

Da equação (6.4), $s_1^{(i-1)}\mu_i^1 \in \overline{S_1}$. Como (a) $s_2^{(i-1)} \in \overline{S_2}$; (b) eventos em $P_{(\Sigma_2-\Sigma_2')\to\Sigma_s}\mu_i^2$ não estão em E_2 (o que implica que eles nunca são desabilitados pelo supervisor S_2), pode-se então concluir que

$$s_2^{(i-1)}\mu_i^2 \in \overline{S_2}. \tag{6.5}$$

Para mostrar que $s_2'^{(i)} = s_2'^{(i-1)}\mu_i^2\sigma_i$ é tal que $s_2^{(i)} = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2}s_2'^{(i)} \in \overline{S_2}$, significando que, ao completar $s_2^{(i-1)}\mu_i$ com o sufixo σ_i não vai levar a cadeia a não pertencer a $\overline{S_2}$, considera-se os dois casos, em relação ao pertencimento de σ_i em Σ_2:

a. $\sigma_i \in \Sigma_2$ (σ_i é um evento de S_2).
Sabe-se que $s_2^{(i)} = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2}s_2'^{(i)} = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2}s_2'^{(i-1)}\mu_i^2\sigma_i = s_2^{(i-1)}\mu_i^2\sigma_i \in \mathcal{L}(G_2)$. Como $\overline{S_2} = \overline{K_2} = E_2||\mathcal{L}(G_2)$, se $s_2^{(i-1)}\mu_i^2\sigma_i \in E_2||s_2^{(i-1)}\mu_i^2\sigma_i \subseteq E_2||\mathcal{L}(G_2) = \overline{S_2}$, então $s_2^{(i-1)}\mu_i^2\sigma_i \in \overline{S_2}$. Mostrar $s_2^{(i)} \in \overline{S_2}$ é o mesmo que mostrar que $s_2^{(i)}||E_2 \neq$

$\emptyset$. Como $\Sigma_{E_2} \subseteq \Sigma'_2 \subseteq \Sigma_2$, então $s_2^{(i)}||E_2 \neq \emptyset$ é o mesmo que

$$P_{\Sigma_2 \to \Sigma'_2}(s_2^{(i)}) \cap P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2 \neq \emptyset.$$

Então,

$$\begin{aligned} P_{\Sigma_2 \to \Sigma'_2}(s_2^{(i)}) \cap P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2 &= P_{\Sigma_2 \to \Sigma'_2}(s_2^{(i-1)} \mu_i^2 \sigma_i) \cap P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2 \\ &= P_{\Sigma_2 \to \Sigma'_2}(s_2^{(i-1)}) P_{\Sigma_2 \to \Sigma'_2}(\mu_i^2 \sigma_i) \cap P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2 \\ &= P_{\Sigma_2 \to \Sigma'_2}(s_2^{(i-1)}) P_{\Sigma_2 \to \Sigma'_2}(\sigma_i) \cap P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2 \\ &= P_{\Sigma_2 \to \Sigma'_2}(s_2^{(i-1)}) \sigma_i \cap P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2 \\ &= P_{\Sigma_r \to \Sigma'_2}(\sigma_1 \dots \sigma_{i-1}) \sigma_i \cap P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2. \end{aligned}$$

Deve-se mostrar, portanto, que $P_{\Sigma_r \to \Sigma'_2}(\sigma_1 \dots \sigma_{i-1})\sigma_i \cap P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2 \neq \emptyset$ que é o mesmo que mostrar que $P_{\Sigma_r \to \Sigma'_2}(\sigma_1 \dots \sigma_{i-1})\sigma_i \subseteq P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2$.
Como $\overline{S_2} = \mathcal{L}(G_2)||E_2 = \mathcal{L}(G_2) \cap P^{-1}_{\Sigma_2 \to \Sigma_{E_2}} E_2$ pode-se dizer que $S_2 \subseteq P^{-1}_{\Sigma_2 \to \Sigma_{E_2}} E_2$. Substituindo $S_2 \subseteq P^{-1}_{\Sigma_2 \to \Sigma_{E_2}} E_2$, na equação (6.1),

$$\begin{aligned} a &\in P_{\Sigma_1 \to \Sigma'_1}(S_1)||P_{\Sigma_2 \to \Sigma'_2}(S_2) \subseteq P_{\Sigma_1 \to \Sigma'_1}(S_1)||P_{\Sigma_2 \to \Sigma'_2}(P^{-1}_{\Sigma_2 \to \Sigma_{E_2}} E_2) \\ a &\in P^{-1}_{\Sigma_r \to \Sigma'_1} P_{\Sigma_1 \to \Sigma'_1}(S_1) \cap P^{-1}_{\Sigma_r \to \Sigma'_2} P_{\Sigma_2 \to \Sigma'_2}(P^{-1}_{\Sigma_2 \to \Sigma_{E_2}} E_2) \end{aligned}$$

e pode-se derivar que

$$P_{\Sigma_r \to \Sigma'_2} a \in P_{\Sigma_2 \to \Sigma'_2}(P^{-1}_{\Sigma_2 \to \Sigma_{E_2}} E_2) \tag{6.6}$$

e, combinando as projeções $P_{\Sigma_2 \to \Sigma'_2}$ e $P^{-1}_{\Sigma_2 \to \Sigma_{E_2}}$ do lado direito da equação (6.6), tem-se

$$P_{\Sigma_r \to \Sigma'_2} a \in P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2.$$

Como E_2 é prefixo-fechada, se $P_{\Sigma_r \to \Sigma'_2} a \in P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2$, então $P_{\Sigma_r \to \Sigma'_2}(\sigma_1 \dots \sigma_{i-1})\sigma_i \in \{P_{\Sigma_r \to \Sigma'_2}\overline{a}\} \subseteq P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2$. Portanto, pode-se dizer que

$$P_{\Sigma_r \to \Sigma'_2}(\sigma_1 \dots \sigma_{i-1})\sigma_i \cap P^{-1}_{\Sigma'_2 \to \Sigma_{E_2}} E_2 \neq \emptyset,$$

e que $s_2^{(i)} = s_2^{(i-1)} \mu_i^2 \sigma_i \in \overline{S_2}$.
Então, $\exists s_2'^{(i)} \in [\Sigma_2 \cup \Sigma_r]^*$ tal que $s_2^{(i)} = P_{(\Sigma_2 \cup \Sigma_r) \to \Sigma_2} s_2'^{(i)} \in \overline{S_2}$ e $P_{(\Sigma_1 \cup \Sigma_r) \to \Sigma_r} s_1'^{(i)} = P_{(\Sigma_2 \cup \Sigma_r) \to \Sigma_r} s_2'^{(i)} = \sigma_1 \dots \sigma_i \in \{\overline{a}\}$.

b. $\sigma_i \notin \Sigma_2$ (σ_i não é um evento de S_2).

Nesse caso,

$$\begin{aligned} s_2^{(i)} &= P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2} s_2'^{(i)} = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2} s_2'^{(i-1)} \mu_i^2 \sigma_i \\ &= P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2} s_2'^{(i-1)} \mu_i^2 P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2} \sigma_i \\ &= s_2^{(i-1)} P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2} \mu_i^2 \epsilon = s_2^{(i-1)} \mu_i^2. \end{aligned} \tag{6.7}$$

Da equação (6.5), juntamente com a equação (6.7), pode-se dizer que $s_2^{(i)} \in \overline{S_2}$. Além disso, $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r} s_1'^{(i)} = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_r} s_2'^{(i)} = \sigma_1 \ldots \sigma_i \in \{\overline{a}\}$. Esse passo finaliza a indução.

$\diamond$

Seja s_1' (na equação (6.2)) tal que $s_1 = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1} s_1' \in S_1 \subseteq \overline{S_1}$. Como, pela hipótese, $a \in P_{\Sigma_1\to\Sigma_1'}(S_1)||P_{\Sigma_2\to\Sigma_2'}(S_2)$ mesmo que s_1' não esteja em S_1 mas apenas em $\overline{S_1}$, pode-se dizer que é possível obter uma cadeia s_1'' tal que $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r} s_1'' = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r} s_1'$ e $s_1 = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1} s_1'' \in S_1 \subseteq \overline{S_1}$.

Mostra-se que s_2' é tal que $s_2 = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2} s_2' \in \overline{S_2}$ e $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r} s_1'' = P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_r} s_2' = a$. Além disso, $P_{\Sigma_2\to\Sigma_2'}(s_2) = P_{\Sigma_2\to\Sigma_2'}(P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2} s_2') = P_{\Sigma_r\to\Sigma_2'} a \in P_{\Sigma_2\to\Sigma_2'}(S_2)$. Se $P_{\Sigma_2\to\Sigma_2'}(S_2)$ é uma OP-abstração então, escolhe-se $b = \epsilon$ na definição do observador. Então $\exists \mu \in \Sigma_2^*$ tal que $P_{\Sigma_2\to\Sigma_2'}(s_2\mu) = P_{\Sigma_2\to\Sigma_2'}(s_2)$ e $s_2\mu \in S_2$.

Como $P_{\Sigma_1\to\Sigma_1'}(s_1) = P_{\Sigma_r\to\Sigma_1'} a$ e $P_{\Sigma_2\to\Sigma_2'}(s_2) = P_{\Sigma_r\to\Sigma_2'} a$, qualquer continuação de s_2 induz uma continuação $\mu' = P_{(\Sigma_2-\Sigma_2')\to(\Sigma_s-(\Sigma_s\cap\Sigma_r))}\mu$ em s_1 se μ tem qualquer evento que seja comum ao conjunto de eventos de S_1. Para mostrar que $s_1\mu' \in S_1 = \overline{S_1} \cap \mathcal{L}_m(G_1)$, deve-se mostrar que:

a. $s_1\mu' \in \overline{S_1}$:
Pela mesma razão descrita acima, de que eventos em $P_{(\Sigma_2-\Sigma_2')\to\Sigma_s}\mu$ nunca são desabilitados pelos supervisores, se $s_2\mu = \mu_1^2\sigma_1\mu_2^2\sigma_2\mu_3^2 \ldots \mu_n^2\sigma_n\mu \in S_2$, $\exists \mu' \in [\Sigma_1 - \Sigma_1']^*$ com $P_{(\Sigma_1-\Sigma_1')\to\Sigma_s}\mu' = P_{(\Sigma_2-\Sigma_2')\to\Sigma_s}\mu$ tal que $s_1\mu' = \mu_1^1\sigma_1\mu_2^1\sigma_2 \ldots \mu_n^1\sigma_n\mu' \in \overline{S_1}$.

b. $s_1\mu' \in \mathcal{L}_m(G_1)$
Tendo-se que $s_2\mu \in S_2$ significa que depois da ocorrência de $s_2\mu$ todas as subplantas estão em estados marcados. Portanto, se uma cadeia está em S_2 ela está também em $\mathcal{L}_m(G_2)$ (de $S_2 = \overline{S_2} \cap \mathcal{L}_m(G_2)$).

Seja $\mu' = P_{\Sigma_2\to(\Sigma_s-(\Sigma_s\cap\Sigma_r))}\mu$. A ocorrência de μ' depois da ocorrência de s_1 causa evolução apenas na planta comum G_c. Devido à escolha de $\mu' = P_{\Sigma_2\to(\Sigma_s-(\Sigma_s\cap\Sigma_r))}\mu \subseteq (\Sigma_s - (\Sigma_s \cap \Sigma_r))^*$, o efeito em G_c pode ser previsto como movendo G_c para um estado marcado (pois $s_2\mu \in S_2$). Como $s_1 \in S_1$, a planta privada (representada por G_{1p}) está em um estado marcado e, como a concatenação de uma cadeia μ' moveu apenas G_c para um estado marcado, pode-se garantir que $G_1 = G_c||G_{1p}$ está num estado marcado e, portanto, pode-se dizer que $s_1\mu' \in \mathcal{L}_m(G_1)$.

Portanto, $s_1\mu' \in S_1$. Como $s_2\mu \in S_2$, $s_1\mu'||s_2\mu \subseteq S_1||S_2$ e $a \in P_{\Sigma\to\Sigma_r}(s_1\mu'||s_2\mu) \subseteq P_{\Sigma\to\Sigma_r}(S_1||S_2)$. Então,

$$P_{\Sigma_1\to\Sigma_1'}(S_1)||P_{\Sigma_2\to\Sigma_2'}(S_2) = P_{\Sigma\to\Sigma_r}(S_1||S_2)$$

e

$$\theta_1(S_1)||\theta_2(S_2) = \theta(S_1||S_2).$$

□

Portanto, Proposição 6.1.1 mostra que sob o novo conjunto de condições sobre as abstrações $\theta_1(S_1)$ e $\theta_2(S_2)$, a propriedade de distributividade pode ser aplicada. Portanto, nas demonstrações dos Teoremas 5.3.1 e 5.3.2, a propriedadade de distributividade de projeções pode ser usada sobre as novas abstrações.

Ao final das demonstrações dos Teoremas 5.3.1 e 5.3.2, o Lema 5.3.2 é utilizado para mostrar que se $t \in \Sigma'^*$ tal que $\theta(s)t \in \theta_1(S_1)||\theta_2(S_2)$ então $\exists u \in \Sigma^*$ tal que $su \in S_1||S_2$. O Lema 5.3.2 é válido se a condição $\Sigma_s \subseteq \Sigma_r$ é satisfeita. No entanto, essa condição, $\Sigma_s \subseteq \Sigma_r$, não é mais verdadeira e portanto o referido lema não pode ser usado neste novo contexto. Então, apresenta-se o Lema 6.1.1 que demonstra o mesmo resultado, sob as novas condições.

Lema 6.1.1. *Sejam $\theta_1(S_1)$ e $\theta_2(S_2)$ OP-abstrações e $\Sigma_{E_1} \cup \Sigma_{E_2} \subseteq \Sigma_r$. Se $\exists s \in \Sigma^*$ e $\exists t \in \Sigma'^*$, tal que $\theta(s)t \in \theta_1(S_1)||\theta_2(S_2)$ então $\exists u \in \Sigma^*$ tal que $su \in S_1||S_2$.*

Demonstração. A notação utilizada na Proposição 6.1.1 é usada. Portanto, sejam as cadeias a, s_j', μ_i^1, μ_i^2, μ' e μ', como definidos na demonstração da Proposição 6.1.1. Seja ainda a representação da projeção $\theta \approx P_{\Sigma\to\Sigma_r}$. Seja $P_{\Sigma\to\Sigma_r}(s)t = a$. Sem perda de generalidade, considera-se que $P_{\Sigma\to\Sigma_r}(s) = \epsilon$ e portanto $t = a = \sigma_1\sigma_2\ldots\sigma_n$. Pretende-se demonstrar que:

$$\exists u \in \Sigma^* \text{ tal que } su \in S_1||S_2 \text{ e } P_{\Sigma\to\Sigma_r}(su) = P_{\Sigma\to\Sigma_r}(s)t. \tag{6.8}$$

Tem-se, da Proposição 6.1.1, que $\exists s_1' \in [\Sigma_1 \cup \Sigma_r]^*$ e $\exists s_2' \in [\Sigma_2 \cup \Sigma_r]^*$ tais que $s_1 = \mu_1^1\sigma_1\mu_2^1\sigma_2\ldots\mu_n^1\sigma_n$ e $s_1 = \mu_1^2\sigma_1\mu_2^2\sigma_2\ldots\mu_n^2\sigma_n$. Ainda da Proposição 6.1.1, tem-se que $\exists\mu' \in [\Sigma_1 \cup \Sigma_r]^*$ e $\mu \in [\Sigma_2 \cup \Sigma_r]^*$ tal que $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s_1'\mu' \in S_1$ e $P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2}s_2'\mu \in S_2$.

Então, sejam $a_1 = s_1'\mu'$ e $a_2 = s_2'\mu$. Para demonstrar equação (6.8), mostra-se, inicialmente, que $\exists u \in \Sigma^*$ tal que $su \in a_1||a_2$. Em seguida, demonstra-se que a cadeia $su \in S_1||S_2$.

Para mostrar que $\exists su \in a_1||a_2$, basta mostrar que $a_1||a_2 \neq \emptyset$, que é o mesmo que mostrar que $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_{sa}}a_1 \cap P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_{sa}}a_2 \neq \emptyset$, onde Σ_{sa} é o conjunto de eventos compartilhados por a_1 e a_2. Tem-se que

$$\begin{aligned}\Sigma_{sa} &= (\Sigma_1 \cup \Sigma_r) \cap (\Sigma_2 \cup \Sigma_r)\\ &= (\Sigma_1 \cap \Sigma_2) \cup \Sigma_r\\ &= \Sigma_s \cup \Sigma_r\end{aligned}$$

Então, tem-se que

$$P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_{sa}}a_1 \cap P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_{sa}}a_2 \quad =$$

$$\begin{aligned}
&= P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}a_1 \cap P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}a_2 \\
&= P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\mu_1^1\sigma_1\ldots\sigma_n\mu' \cap P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\mu_1^2\sigma_1\ldots\sigma_n\mu \\
&= P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\mu_1^1 P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\sigma_1\ldots P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\sigma_n P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\mu' \\
&\quad \cap P_{(\Sigma_2\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\mu_1^2 P_{(\Sigma_2\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\sigma_1\ldots P_{(\Sigma_2\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\sigma_n P_{(\Sigma_2\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\mu.
\end{aligned} \tag{6.9}$$

Por construção tem-se que $\mu_i^1, \mu' \in [\Sigma_1 - \Sigma_1']^*$, $\mu_i^2, \mu \in [\Sigma_2 - \Sigma_2']^*$. Portanto,

- $P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\ \mu_1^1 = P_{(\Sigma_1-\Sigma_1')\to\Sigma_s}\mu_1^1$ e $P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\ \mu' = P_{(\Sigma_1-\Sigma_1')\to\Sigma_s}\mu'$;
- $P_{(\Sigma_2\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\ \mu_1^2 = P_{(\Sigma_2-\Sigma_2')\to\Sigma_s}\mu_1^2$ e $P_{(\Sigma_2\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\ \mu = P_{(\Sigma_2-\Sigma_2')\to\Sigma_s}\mu$.

Além disso, tem-se que:

- $P_{(\Sigma_1-\Sigma_1')\to\Sigma_s}\mu_1^1 = P_{(\Sigma_2-\Sigma_2')\to\Sigma_s}\mu_1^2$ e
- $P_{(\Sigma_1-\Sigma_1')\to\Sigma_s}\mu' = P_{(\Sigma_2-\Sigma_2')\to\Sigma_s}\mu$.

Por outro lado, os eventos $\sigma_i \in \Sigma_r$ sobrevivem às projeções, ou seja,

- $P_{(\Sigma_1\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\sigma_i = \sigma_i$;
- $P_{(\Sigma_2\cup\Sigma_r)\to(\Sigma_s\cup\Sigma_r)}\sigma_i = \sigma_i$.

Então, equação (6.9) torna-se:

$$P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_{sa}}a_1 \cap P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_{sa}}a_2 \quad = \quad P_{(\Sigma_1-\Sigma_1')\to\Sigma_s}\mu_1^1\sigma_1\ldots\sigma_n P_{(\Sigma_1-\Sigma_1')\to\Sigma_s}\mu' \neq \emptyset. \tag{6.10}$$

Portanto, $\exists su \in a_1||a_2$. Para completar a demonstração, mostra-se que $su \in S_1||S_2$.

Por construção de a_1 e a_2 tem-se que $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}a_1 \in S_1$ e $P_{(\Sigma_2\cup\Sigma_r)\to\Sigma_2}a_2 \in S_2$. Portanto, $a_1 \in S_1||\Sigma_r^*$ e $a_2 \in S_2||\Sigma_r^*$. Então, $a_1||a_2 \subseteq (S_1||\Sigma_r^*)||(S_2||\Sigma_r^*) = S_1||S_2$. Tem-se portanto que $a_1||a_2 \subseteq S_1||S_2$. Como $su \in a_1||a_2$ pode-se concluir que $su \in S_1||S_2$ e $P_{\Sigma\to\Sigma_r}(su) = P_{\Sigma\to\Sigma_r}(s)t = a$.

□

Na seqüência, apresenta-se a extensão da Proposição 6.1.1 e do Lema 6.1.1 para o caso de múltiplos supervisores.

6.1.1 Múltiplos supervisores

Os dois resultados na seção anterior devem ser estendidos para o caso de múltiplos supervisores. Estes resultados são também apresentados sob a forma de uma proposição e um lema.

Inicialmente, apresenta-se a Proposição 6.1.2 que estabelece que, sob as condições apresentadas, a igualdade $\theta(\mathop{||}_{j=1}^{m} S_j) = \mathop{||}_{j=1}^{m} \theta_j(S_j)$ se verifica.

As linguagens E_j representam as especificações genéricas a serem implementadas pelos supervisores S_j e K_j são as linguagens desejadas (ou especificações locais) para o sistema em malha fechada, ou seja, $K_j = E_j||\mathcal{L}_m(G_j)$, $\forall j \in J$. Os alfabetos Σ_{E_j} representam os conjuntos de eventos presentes nas especificações E_j.

A Proposição 6.1.1 é, portanto, reescrita para o caso de m supervisores.

Proposição 6.1.2. *Sejam S_j, $\forall j \in J$, linguagens implementadas pelos supervisores. A igualdade $\theta(\mathop{||}_{j=1}^{m} S_j) = \mathop{||}_{j=1}^{m} \theta_j(S_j)$ se verifica se, $\forall j \in J$,*

- *$\theta_j(S_j)$ são OP-abstrações;*
- *K_j são controláveis em relação a $\mathcal{L}(G_j)$ ($S_j = K_j$);*
- *$\mathop{\cup}_{j=1}^{m} (\Sigma_{E_j}) \subseteq \Sigma_r$;*
- *$S_j = \overline{S_j} \cap \mathcal{L}_m(G_j)$.*

Demonstração. $\theta(\mathop{||}_{j=1}^{m} S_j) \subseteq \mathop{||}_{j=1}^{m} \theta_j(S_j)$ é sempre verdadeira. Para provar a igualdade, é suficiente mostrar que $\mathop{||}_{j=1}^{m} \theta_j(S_j) \subseteq \theta(\mathop{||}_{j=1}^{m} S_j)$. Para uniformizar a notação, as projeções θ_j e θ são substituídas pela representação explícita dos alfabetos envolvidos, ou seja, $\theta_j \approx P_{\Sigma_j \to \Sigma'_j}$ e $\theta \approx P_{\Sigma \to \Sigma_r}$.

$\mathop{||}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(S_j) \subseteq P_{\Sigma \to \Sigma_r}(\mathop{||}_{j=1}^{m} S_j)$:

Seja

$$a \in \mathop{||}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(S_j) \subseteq \Sigma_r^*. \tag{6.11}$$

Então $P_{\Sigma_r \to \Sigma'_j} a \in P_{\Sigma_j \to \Sigma'_j}(S_j)$ e também $\exists s_{x_j} \in S_j \subseteq \Sigma_j^*$ tal que $P_{\Sigma_j \to \Sigma'_j}(s_{x_j}) = P_{\Sigma_r \to \Sigma'_j} a$.

Escreve-se $a = \sigma_1 \sigma_2 \ldots \sigma_n \in \Sigma_r^*$. Seja

$$s'_1 \quad = \quad \mu_1^1 \sigma_1 \mu_2^1 \sigma_2 \ldots \mu_n^1 \sigma_n \in (\Sigma_1 \cup \Sigma_r)^* \tag{6.12}$$

tal que

$$\begin{aligned} P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s'_1 &= s_1 \in \overline{S_1}, \\ \theta_1(s_1) &= \theta_1(P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s'_1) = P_{\Sigma_r\to\Sigma'_1}a, \\ P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r}s'_1 &= a, \end{aligned}$$

com $\mu_i^1 \in [\Sigma_1 - \Sigma'_1]^*$ e $i = \{1 \dots n\}$.

Hipótese: Para todo s'_1 com as características acima, existem também $s'_j = \mu_1^j\sigma_1\mu_2^j\sigma_2 \dots \mu_i^j \sigma_i \dots \mu_n^j\sigma_n \in (\Sigma_j \cup \Sigma_r)^*$, $\forall j \in (J - \{1\})$, tal que

$$\begin{aligned} P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j}s'_j &= s_j \in \overline{S_j} \\ P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_r}s'_j &= P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r}s'_1 = a. \end{aligned}$$

Demonstração: A estratégia indutiva é utilizada nesta demonstração. Assume-se que para algum $i \geq 1$, $s_1^{\prime(i-1)} = \mu_1^1\sigma_1 \dots \mu_{i-1}^1\sigma_{i-1}$, com $\mu_1^1 \dots \mu_{i-1}^1$ definido de tal forma que $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r}s'_1 = a$ e

$$s_1^{(i-1)} = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s_1^{\prime(i-1)} = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(\mu_1^1\sigma_1 \dots \mu_{i-1}^1\sigma_{i-1}) \in \overline{S_1}$$

existem, para cada $j \in J$, $s_j^{\prime(i-1)} = \mu_1^j\sigma_1 \dots \mu_{i-1}^j\sigma_{i-1}$, com $\mu_1^j \dots \mu_{i-1}^j$ definidos de tal forma que

$$\begin{aligned} P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_j\cap\Sigma_k)}\mu_i^j &= P_{(\Sigma_k\cup\Sigma_r)\to(\Sigma_k\cap\Sigma_j)}\mu_i^k, \qquad \forall k \in J \\ P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_r}s'_j &= a \end{aligned}$$

e

$$s_j^{(i-1)} = P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_r}s_j^{\prime(i-1)} = P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_r}(\mu_1^j\sigma_1 \dots \mu_{i-1}^j\sigma_{i-1}) \in \overline{S_j} \quad (6.13)$$

pretende-se mostrar que para qualquer $s_1^{\prime(i)}$ com as características desejadas existem $s_j^{\prime(i)}$ tal que a equação (6.13) é verdadeira.

Base: para $i = 1$, $s_1^{\prime(0)} = \epsilon$, $s_1^{(0)} = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}s_1^{\prime(0)} \in \overline{S_1}$ existem $s_j^{\prime(0)} = \epsilon$, tais que $s_j^{(0)} = P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j}s_j^{\prime(0)} \in \overline{S_j}$. Além disso, $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r}s_1^{\prime(0)} = P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_r}s_j^{\prime(0)} = \epsilon \in \{\overline{a}\}$.

Passo de Indução: Para $2 \leq i \leq n$, se $s_1^{(i)} = s_1^{(i-1)}\mu_i^1\sigma_i = \mu_1^1\sigma_1 \dots \mu_{i-1}^1\sigma_{i-1}\mu_i^1\sigma_i$, com

$\mu_i^1 \in [\Sigma_1 - \Sigma_1']^*$ definidos de forma que $s_1^{(i)} = P_{(\Sigma_1 \cup \Sigma_r)\to\Sigma_1} s_1'^{(i)} \in \overline{S_1}$, então

$$\begin{aligned} s_1^{(i)} &= P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1} s_1'^{(i)} = P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(s_1'^{(i-1)}\mu_i^1\sigma_i) \\ &= P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1} s_1'^{(i-1)} P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(\mu_i^1\sigma_i) \\ &= s_1^{(i-1)} P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(\mu_i^1\sigma_i) \\ &= s_1^{(i-1)}\mu_i^1 P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_1}(\sigma_i) \in \overline{S_1}. \end{aligned} \tag{6.14}$$

Pretende-se construir $\mu_i^j \in [\Sigma_j - \Sigma_j']^*$ tal que $P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j} s_j'^{(i-1)}\mu_i^j \in \overline{S_j}$. Como Σ_s é a união das interseções dos conjuntos de eventos dos supervisores, o processo de construção de μ_i^j tem que ser feito interativamente. Cada cadeia μ_i^j tem que ser tal que:

1. $P_{\Sigma_j\to(\Sigma_j\cap\Sigma_k)}\mu_i^j = P_{\Sigma_k\to(\Sigma_k\cap\Sigma_j)}\mu_i^k$, $\forall k$ tal que $\Sigma_k \cap \Sigma_j \neq \emptyset$;
2. $s_j^{(i-1)}\mu_i^j \in \overline{S_j}$.

Como todos os eventos comuns que não estão em Σ_r nunca são desabilitados, se um destes evento aparece em uma cadeia da linguagem implementada pelo supervisor, isso signica que o evento é possível na planta (no estado correspondente). Então, o evento está habilitado no estado correspondente em todos os supervisores que compartilham a planta.

Portanto, a construção de $\mu_i^j, \forall j \in J$, é apenas uma questão de encontrar uma ordenação para os eventos tal que as seqüências de eventos em Σ_s sejam consistentes entre as cadeias e μ_i^j seja uma continuação válida para a cadeia $s_j^{(i-1)}$.

Da equação (6.14), $s_1^{(i-1)}\mu_i^1 \in \overline{S_1}$. Como (a) $s_j^{(i-1)} \in \overline{S_j}$; (b) eventos em $P_{(\Sigma_j-\Sigma_j')\to\Sigma_s}\mu_i^j$ não estão em E_j (o que implica que nunca são desabilitados pelo supervisor S_j), pode-se concluir que:

$$s_j^{(i-1)}\mu_i^j \in \overline{S_j}. \tag{6.15}$$

Para mostrar que $s_j'^{(i)} = s_j'^{(i-1)}\mu_i^j\sigma_i$ é tal que $s_j^{(i)} = P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j} s_j'^{(i)} \in \overline{S_j}$, ou seja, que adicionando o sufixo σ_i à cadeia $s_j^{(i-1)}\mu_i$ não vai gerar uma cadeia que não está em $\overline{S_j}$, considera-se os dois casos seguintes, relacionados ao pertencimento de σ_i em Σ_j:

a. $\sigma_i \in \Sigma_j$ (σ_i é um evento de S_j).
Sabe-se que $s_j^{(i)} = P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j} s_j'^{(i)} = P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j} s_j'^{(i-1)}\mu_i^j\sigma_i = s_j^{(i-1)}\mu_i^j\sigma_i \in \mathcal{L}(G_j)$. Como $\overline{S_j} = \overline{K_j} = E_j||\mathcal{L}(G_j)$, se $s_j^{(i-1)}\mu_i^j\sigma_i \in E_j||s_j^{(i-1)}\mu_i^j\sigma_i \subseteq E_j||\mathcal{L}(G_j) = \overline{S_j}$, então $s_j^{(i-1)}\mu_i^j\sigma_i \in \overline{S_j}$. Mostrar $s_j^{(i)} \in \overline{S_j}$ é o mesmo que mostrar que $s_j^{(i)}||E_j \neq \emptyset$. Como $\Sigma_{E_j} \subseteq \Sigma_j' \subseteq \Sigma_j$, $s_j^{(i)}||E_j \neq \emptyset$ é o mesmo que

$$P_{\Sigma_j\to\Sigma_j'}(s_j^{(i)}) \cap P^{-1}_{\Sigma_j'\to\Sigma_{E_j}} E_j \neq \emptyset.$$

Então,

$$\begin{aligned} P_{\Sigma_j\to\Sigma'_j}(s_j^{(i)}) \cap P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j &= P_{\Sigma_j\to\Sigma'_j}(s_j^{(i-1)}\mu_i^j\sigma_i) \cap P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j \\ &= P_{\Sigma_j\to\Sigma'_j}(s_j^{(i-1)}) P_{\Sigma_j\to\Sigma'_j}(\mu_i^j\sigma_i) \cap P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j \\ &= P_{\Sigma_j\to\Sigma'_j}(s_j^{(i-1)})\sigma_i \cap P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j \\ &= P_{\Sigma_r\to\Sigma'_j}(\sigma_1\ldots\sigma_{i-1})\sigma_i \cap P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j. \end{aligned}$$

Deve-se mostrar portanto que $P_{\Sigma_r\to\Sigma'_j}(\sigma_1\ldots\sigma_{i-1})\sigma_i \cap P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j \neq \emptyset$ ou que $P_{\Sigma_r\to\Sigma'_j}(\sigma_1\ldots\sigma_{i-1})\sigma_i \subseteq P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j$.
Como $\overline{S_j} = \mathcal{L}(G_j)||E_j = \mathcal{L}(G_j) \cap P^{-1}_{\Sigma_j\to\Sigma_{E_j}} E_j$ pode-se dizer que $S_j \subseteq P^{-1}_{\Sigma_j\to\Sigma_{E_j}} E_j$. Substituindo $S_j \subseteq P^{-1}_{\Sigma_j\to\Sigma_{E_j}} E_j$ na equação (6.11)

$$\begin{aligned} a &\in \mathop{||}_{j=1}^{m} P_{\Sigma_j\to\Sigma'_j}(S_j) \subseteq \mathop{||}_{j=1}^{m} P_{\Sigma_j\to\Sigma'_j}(P^{-1}_{\Sigma_j\to\Sigma_{E_j}} E_j) \\ a &\in \mathop{\cap}_{j=1}^{m} P^{-1}_{\Sigma_r\to\Sigma'_j} P_{\Sigma_j\to\Sigma'_j}(P^{-1}_{\Sigma_j\to\Sigma_{E_j}} E_j) \end{aligned}$$

e pode-se obter

$$P_{\Sigma_r\to\Sigma'_j} a \in P_{\Sigma_j\to\Sigma'_j}(P^{-1}_{\Sigma_j\to\Sigma_{E_j}} E_j) \tag{6.16}$$

e combinando as projeções $P_{\Sigma_j\to\Sigma'_j}$ e $P^{-1}_{\Sigma_j\to\Sigma_{E_j}}$ do lado direito da equação (6.16), tem-se

$$P_{\Sigma_r\to\Sigma'_j} a \in P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j.$$

Como E_j é prefixo-fechado, se $P_{\Sigma_r\to\Sigma'_j} a \in P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j$, então $P_{\Sigma_r\to\Sigma'_j}(\sigma_1\ldots\sigma_{i-1})\sigma_i \in \{P_{\Sigma_r\to\Sigma'_j}\overline{a}\} \subseteq P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j$. Portanto, pode-se dizer que

$$P_{\Sigma_r\to\Sigma'_j}(\sigma_1\ldots\sigma_{i-1})\sigma_i \cap P^{-1}_{\Sigma'_j\to\Sigma_{E_j}} E_j \neq \emptyset,$$

e $s_j^{(i)} = s_j^{(i-1)}\mu_i^j\sigma_i \in \overline{S_j}$.
Então, $\exists s_j'^{(i)} \in [\Sigma_j \cup \Sigma_r]^*$ tal que $s_j^{(i)} = P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j} s_j'^{(i)} \in \overline{S_j}$ e $P_{(\Sigma_1\cup\Sigma_r)\to\Sigma_r} s_1'^{(i)} = P_{\Sigma_j\cup\Sigma_r\to\Sigma_r} s_j'^{(i)} = \sigma_1\ldots\sigma_i \in \{\overline{a}\}$, $\forall j \in J$.

b. $\sigma_i \notin \Sigma_j$ (σ_i não é um evento de S_j).
Nesse caso,

$$\begin{aligned} s_j^{(i)} &= P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j} s_j'^{(i)} = P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j} s_j'^{(i-1)}\mu_i^j\sigma_i \\ &= P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j} s_j'^{(i-1)}\mu_i^j P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j}\sigma_i \\ &= s_j^{(i-1)} P_{(\Sigma_j\cup\Sigma_r)\to\Sigma_j}\mu_i^j\epsilon = s_j^{(i-1)}\mu_i^j. \end{aligned} \tag{6.17}$$

Das equações (6.15) e (6.17), pode-se dizer que $s_j^{(i)} \in \overline{S_j}$, $\forall j \in J$. Além disso, $P_{(\Sigma_1 \cup \Sigma_r)\to\Sigma_r} s_1'^{(i)} = P_{(\Sigma_j \cup \Sigma_r)\to\Sigma_r} s_j'^{(i)} = \sigma_1 \ldots \sigma_i \in \{\overline{a}\}$. Esse passo finaliza a indução.

$\diamond$

Mostrou-se acima que para qualquer $s_1' \in \overline{S_1}$ existe também s_j' tal que $s_j = P_{(\Sigma_j \cup \Sigma_r)\to\Sigma_j} s_j' \in \overline{S_j}$ e $P_{(\Sigma_j \cup \Sigma_r)\to\Sigma_r} s_1' = P_{(\Sigma_j \cup \Sigma_r)\to\Sigma_r} s_j' = a$, $\forall j \in J$.

Sabe-se que $P_{\Sigma_j\to\Sigma_j'}(s_j) = P_{\Sigma_j\to\Sigma_j'}(P_{(\Sigma_j \cup \Sigma_r)\to\Sigma_j} s_j') = P_{\Sigma_r\to\Sigma_j'} a \in P_{\Sigma_j\to\Sigma_j'}(S_j)$ e que $P_{\Sigma_j\to\Sigma_j'}$ é uma OP-abstração. Então, escolhe-se $b = \epsilon$ (na definição do observador) e pode-se concluir que $\exists \eta_j \in \Sigma_j^*$ tal que $P_{\Sigma_j\to\Sigma_j'}(s_j\eta_j) = P_{\Sigma_j\to\Sigma_j'}(s_j)$ e $s_j\eta_j \in S_j$, $\forall j \in J$. Entretanto, os η_j para os diferentes js devem ser obtidos de forma que as cadeias $s_j\eta_j$ nos diferentes supervisors não bloqueiem umas às outras.

Cada supervisor tem uma planta local que é formada por dois tipos de subplantas: (i) a planta privada (G_{jp}) composta de todas as subplantas que são exclusivas de cada supervisor S_j; (ii) as subplantas compartilhadas ($G_{c\{j,m\}}$) que são compostas do conjunto de subplantas que são compartilhadas por S_j e S_m, $\forall m \in M_j = \{m \in J | G_{c\{j,m\}}$ é uma subplanta de $S_m\}$. Um supervisor S_j pode ter mais de uma subplanta compartilhada se ela compartilha subplantas diferentes com diferentes subconjuntos de supervisores, por exemplo $G_{c\{j,m,n\}}$ que é uma planta compartilhada por S_j, S_m e S_n.

Como $P_{\Sigma_j\to\Sigma_j'}(S_j)$ é uma OP-abstração, $\exists \eta_j \in [\Sigma_j - (\Sigma_j - \Sigma_r)]^*$ tal que $s_j\eta_j \in S_j$ e $P_{\Sigma_j\to\Sigma_j'}(s_j\eta_j) = P_{\Sigma_j\to\Sigma_j'}(s_j)$, isto significa que existe uma seqüência de eventos não-relevantes que leva a cadeia s_j para um estado marcado. Como η_j é composto de eventos não-relevantes, uma seqüência de eventos não-relevantes que leva a planta local para um estado marcado leva também o supervisor para um estado marcado. Um estado marcado na planta local é equivalente a alcançar estados marcados tanto na planta privada como em todas as plantas compartilhadas. Então, obter η_j é equivalente a obter a seqüência de eventos não-relevantes que leva todas as subplantas da planta local para estados marcados. Para cada subplanta compartilhada $G_{c\{j,m\}}$ é gerada uma seqüência de eventos não-relevantes que completam $P_{\Sigma_j\to\Sigma_{G_{c\{j,m\}}}} s_j = P_{\Sigma_m\to\Sigma_{G_{c\{j,m\}}}} s_m \in \mathcal{L}(G_{c\{j,m\}})$ para uma cadeia na linguagem marcada. Então, a cadeia obtida é nomeada η_{jm} (e η_{mj}) que representa a subseqüência que deve estar no η_j (η_m) final para garantir que a planta compartilhada $G_{c\{j,m\}}$ alcançe um estado marcado.

Depois de gerar as subseqüências de todos as subplantas compartilhadas por quaisquer dois ou mais supervisores, obtém-se um conjunto de cadeias disjuntas $\Xi_j = \{\eta_{jm} | \forall m \in M_j\}$ para cada supervisor S_j. A cadeia η_j é então obtida intercalando as cadeias em Ξ_j, $\forall j \in J$. Se $s_j\eta_j \notin S_j$, significa que a subplanta privada não está em um estado marcado. Uma seqüência de eventos não-relevantes que completam a planta privada para um estado marcado pode ser obtida. O η_j final é obtido intercalando o η_j com a seqüência de eventos privados. Esse passo final pode ser realizado independentemente e não afeta as cadeias dos outros supervisores.

Agora, pretende-se mostrar que os η_j construídos são tais que $s_j\eta_j \in S_j = \overline{S_j} \cap \mathcal{L}_m(G_j)$, $\forall j \in J$. Então, mostra-se que: (a) $s_j\eta_j \in \overline{S_j}$; (b) $s_j\eta_j \in \mathcal{L}_m(G_j)$ a seguir.

(a) $s_j\eta_j \in \overline{S_j}$:
Os eventos comuns em cada η_j nunca são desabilitados pelos supervisores. Então, sempre que eles estão habilitados na planta eles estão habilitados em todos os supervisores que controlam a planta. Se $s_j \in \overline{S_j}$, acrescentar a continuação η_j, gerada intercalando as cadeias de suas subplantas, não vai tirar a cadeia de $\overline{S_j}$, $\forall j \in J$.

(b) $s_j\eta_j \in \mathcal{L}_m(G_j)$
Como

$$\begin{aligned} P_{\Sigma_j \to \Sigma_{G_{c\{j,m\}}}} s_j\eta_j &\in \mathcal{L}_m(G_{c\{j,m\}}), \forall m \in M_j \\ P_{\Sigma_j \to \Sigma_{G_{jp}}} s_j\eta_j &\in \mathcal{L}_m(G_{jp}), \end{aligned}$$

onde G_{jp} denota a planta privada de S_j e $G_{c\{j,m\}}$ denota a planta (ou composição de subplantas) compartilhada por S_j e S_m. Então

$$\begin{aligned} s_j\eta_j &\in \mathop{||}_{m \in M_j} P_{\Sigma_j \to \Sigma_{G_{c\{j,m\}}}} s_j\eta_j || P_{\Sigma_j \to \Sigma_{G_{jp}}} s_j\eta_j \\ &\subseteq (\mathop{||}_{m \in M_j} (\mathcal{L}_m(G_{c\{j,m\}})) || \mathcal{L}_m(G_{jp})) \\ s_j\eta_j &\in \mathcal{L}_m(G_j). \end{aligned}$$

Portanto, para qualquer s_j existe um η_j tal que $s_j\eta_j \in S_j$, $\forall j \in J$. Como $\mathop{||}_{j=1}^{m} s_j\eta_j \subseteq \mathop{||}_{j=1}^{m} S_j$ é verdade que $a \in P_{\Sigma \to \Sigma_r}(\mathop{||}_{j=1}^{m} s_j\eta_j) \subseteq P_{\Sigma \to \Sigma_r}(\mathop{||}_{j=1}^{m} S_j)$. Então,

$$\mathop{||}_{j=1}^{m} P_{\Sigma_j \to \Sigma'_j}(S_j) \subseteq P_{\Sigma \to \Sigma_r}(\mathop{||}_{j=1}^{m} S_j)$$

□

Com isso, completa-se o primeiro passo, de garantir que, sob as novas condições, a propriedade de distributividade das projeções pode ser aplicada sobre as abstrações $\theta_j(S_j)$. O segundo passo do procedimento consiste em mostrar que dado $t \in \Sigma'^*$ tal que $\theta(s)t \in \mathop{||}_{j=1}^{m} \overline{\theta_j(S_j)}$ existe $u \in \Sigma^*$ tal que $su \in \mathop{||}_{j=1}^{m} S_j$, correspondente ao Lema 5.3.4 do Capítulo 5.

O Lema 6.1.2 é análogo ao Lema 6.1.1 generalizado para o caso de múltiplos supervisores.

Lema 6.1.2. *Sejam $\theta_j(S_j)$ OP-abstrações e $\bigcup_{j=1}^{m} \Sigma_{E_j} \subseteq \Sigma_r$. Se $\exists s \in \Sigma^*$ e $\exists t \in \Sigma'^*$, tal que $\theta(s)t \in \mathop{||}_{j=1}^{m} \theta_j(S_j)$ então $\exists u \in \Sigma^*$ tal que $su \in \mathop{||}_{j=1}^{m} S_j$.*

Demonstração. A notação utilizada na Proposição 6.1.2 é usada. Portanto, sejam as cadeias a, s'_j, μ_i^j e η_j, $\forall j \in J$, como definidos na demonstração da Proposição 6.1.2. Seja ainda a representação da projeção $\theta \approx P_{\Sigma\to\Sigma_r}$. Seja $P_{\Sigma\to\Sigma_r}(s)t = a$. Sem perda de generalidade, considera-se que $P_{\Sigma\to\Sigma_r}(s) = \epsilon$ e portanto $t = a = \sigma_1\sigma_2\ldots\sigma_n$. Pretende-se demonstrar que:

$$\exists u \in \Sigma^* \text{ tal que } su \in \mathop{||}_{j=1}^{m} S_j \text{ e } P_{\Sigma\to\Sigma_r}(su) = P_{\Sigma\to\Sigma_r}(s)t. \tag{6.18}$$

Tem-se, da demonstração da Proposição 6.1.2, que $\exists s'_j \in [\Sigma_j \cup \Sigma_r]^*$ tal que $s'_j = \mu_1^j\sigma_1\mu_2^j\ldots \mu_n^j\sigma_n$ para todo $j \in J$. Seja $a_j = s'_j\eta_j = \mu_1^j\sigma_1\mu_2^j\ldots\ \mu_n^j\sigma_n\eta_j$. Para demonstrar equação (6.18), mostra-se inicialmente que $\exists u \in \Sigma^*$ tal que $su \in \mathop{||}_{j=1}^{m} a_j$. Em seguida, mostra-se que $su \in \mathop{||}_{j=1}^{m} S_j$.

Para mostrar que $\exists su \in \mathop{||}_{j=1}^{m} a_j$, basta mostrar que $\mathop{||}_{j=1}^{m} a_j \neq \emptyset$. Verificar $\mathop{||}_{j=1}^{m} a_j \neq \emptyset$ é o mesmo que verificar se $\mathop{||}_{j=1}^{m} P^{-1}_{\Sigma_{sa}\to(\Sigma_{sa}\cap\Sigma_j)}P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}a_j \neq \emptyset$, onde Σ_{sa} é o conjunto de eventos compartilhados por dois ou mais a_j.

$$\mathop{||}_{j=1}^{m} P^{-1}_{\Sigma_{sa}\to(\Sigma_{sa}\cap\Sigma_j)}P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}a_j \quad =$$

$$\begin{aligned}
&= \mathop{||}_{j=1}^{m} P^{-1}_{\Sigma_{sa}\to(\Sigma_{sa}\cap\Sigma_j)}P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}\mu_1^j\sigma_1\mu_2^j\ldots\mu_n^j\sigma_n\eta_j \\
&= \mathop{||}_{j=1}^{m} P^{-1}_{\Sigma_{sa}\to(\Sigma_{sa}\cap\Sigma_j)}[P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}\mu_1^j P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}\sigma_1 P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}\cdots \\
&\qquad P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}\mu_n^j P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}\sigma_n P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}\eta_j].
\end{aligned}$$

Tem-se que:

$$\begin{aligned}
\Sigma_{sa} &= \mathop{\cup}_{k,l\in J}^{k\neq l}(\Sigma_{a_k}\cap\Sigma_{a_l}) = \mathop{\cup}_{k,l\in J}^{k\neq l}[(\Sigma_k\cup\Sigma_r)\cap[\Sigma_l\cup\Sigma_r)] \\
&= \mathop{\cup}_{k,l\in J}^{k\neq l}(\Sigma_k\cap\Sigma_l)\cup\Sigma_r \\
&= \Sigma_s\cup\Sigma_r,
\end{aligned}$$

então $\mathop{||}_{j=1}^{m} P^{-1}_{\Sigma_{sa}\to(\Sigma_{sa}\cap\Sigma_j)}P_{(\Sigma_j\cup\Sigma_r)\to(\Sigma_{sa}\cap\Sigma_j)}a_j$ torna-se:

$$\mathop{||}_{j=1}^{m} P^{-1}_{(\Sigma_r\cup\Sigma_s)\to((\Sigma_r\cup\Sigma_s)\cap\Sigma_j)}P_{(\Sigma_j\cup\Sigma_r)\to((\Sigma_r\cup\Sigma_s)\cap\Sigma_j)}a_j.$$

Os eventos $\sigma_i \in \Sigma_r$ estão presentes em todos os a_j, $\forall j \in J$, na mesma ordem que aparecem

em a. Por isso, $\forall i \in I$ e $\forall k, l \in J$ tem-se que:

$$P_{(\Sigma_k \cup \Sigma_r)\rightarrow((\Sigma_r \cup \Sigma_s)\cap\Sigma_k)}\sigma_i = P_{(\Sigma_l \cup \Sigma_r)\rightarrow((\Sigma_r \cup \Sigma_s)\cap\Sigma_l)}\sigma_i = \sigma_i.$$

Para que $\mathop{||}_{j=1}^{m} a_j \neq \emptyset$ deve ser verdade que:

$$P_{(\Sigma_k \cup \Sigma_r)\rightarrow((\Sigma_r \cup \Sigma_s)\cap\Sigma_k)}\mu_i^k = P_{(\Sigma_l \cup \Sigma_r)\rightarrow((\Sigma_r \cup \Sigma_s)\cap\Sigma_l)}\mu_i^l \quad (6.19)$$

$$P_{(\Sigma_k \cup \Sigma_r)\rightarrow((\Sigma_r \cup \Sigma_s)\cap\Sigma_k)}\eta_k = P_{(\Sigma_l \cup \Sigma_r)\rightarrow((\Sigma_r \cup \Sigma_s)\cap\Sigma_l)}\eta_l, \quad (6.20)$$

$\forall i \in I$ e $\forall k, l \in J$ tal que $\Sigma_k \cap \Sigma_l \neq \emptyset$. Ou seja, para todo par de cadeias μ_i^k e μ_i^l (η_k e η_l) que compartilham eventos, as subcadeias de eventos compartilhados devem ser sincronizadas. Como $\mu_i^j, \eta_j \in [\Sigma_j - (\Sigma_j \cap \Sigma_r)]^*$, tem-se que:

$$\begin{aligned} P_{(\Sigma_j \cup \Sigma_r)\rightarrow((\Sigma_r \cup \Sigma_s)\cap\Sigma_j)}\mu_i^j &= P_{\Sigma_j\rightarrow(\Sigma_s\cap\Sigma_j)}\mu_i^j, \\ P_{(\Sigma_j \cup \Sigma_r)\rightarrow((\Sigma_r \cup \Sigma_s)\cap\Sigma_j)}\eta_j &= P_{\Sigma_j\rightarrow(\Sigma_s\cap\Sigma_j)}\eta_j \end{aligned}$$

e equações (6.19) e (6.20) tornam-se:

$$\begin{aligned} P_{\Sigma_k\rightarrow(\Sigma_s\cap\Sigma_k)}\mu_i^k &= P_{\Sigma_l\rightarrow(\Sigma_s\cap\Sigma_l)}\mu_i^l \\ P_{\Sigma_k\rightarrow(\Sigma_s\cap\Sigma_k)}\eta_k &= P_{\Sigma_l\rightarrow(\Sigma_s\cap\Sigma_l)}\eta_l, \end{aligned}$$

que são verdadeiras por construção de μ_i^k, μ_i^l, η_k e η_l. Com a sincronização das subcadeias de eventos compartilhados garantida, tem-se que $\mathop{||}_{j=1}^{m} P^{-1}_{\Sigma_{sa}\rightarrow(\Sigma_{sa}\cap\Sigma_j)}P_{(\Sigma_j\cup\Sigma_r)\rightarrow(\Sigma_{sa}\cap\Sigma_j)}a_j \neq \emptyset$ e portanto que $\exists u \in \Sigma^*$ tal que $su \in \mathop{||}_{j=1}^{m} a_j$.

Para completar a demonstração, precisa-se mostrar que $su \in \mathop{||}_{j=1}^{m} S_j$. Por construção de a_j, pode-se dizer que a_j é tal que $P_{(\Sigma_j\cup\Sigma_r)\rightarrow\Sigma_j}a_j \in S_j$. Portanto, $a_j \in S_j||\Sigma_r^*$. Assim, $\mathop{||}_{j=1}^{m} a_j \subseteq \mathop{||}_{j=1}^{m} (S_j||\Sigma_r^*) = (\mathop{||}_{j=1}^{m} S_j)||\Sigma_r^* = \mathop{||}_{j=1}^{m} S_j$. Então, como $su \in \mathop{||}_{j=1}^{m} a_j$, tem-se que $su \in \mathop{||}_{j=1}^{m} S_j$ e $P_{\Sigma\rightarrow\Sigma_r}(s)t = P_{\Sigma\rightarrow\Sigma_r}(su) = a$.

□

Cada passagem das demonstrações do Capítulo 5 dependente da condição sobre os eventos relevantes foi demarcada com o par $\triangleright, \triangleleft$. Pode-se observar que cada uma dessas passagens utiliza a Proposição 5.2.1 ou um dos Lemas 5.3.2 e 5.3.3. Nos resultados apresentados no Capítulo 6, as referências à Proposição 5.2.1 e aos Lemas 5.3.2 e 5.3.3 devem ser substituídas por referências à Proposição 6.1.2 e aos Lemas 6.1.2 e 6.1.1, respectivamente. Desta forma, os teoremas do capítulo anterior são válidos para as novas abstrações.

Antes de reescrever os teoremas com as novas condições, apresenta-se, na próxima seção, uma discussão a respeito.

6.2 Abstrações obtidas Usando as Condições Estruturais

As novas abstrações são obtidas atendendo às condições apresentadas nesse capítulo e reapresentadas a seguir. Elas são:

- $\theta_j(S_j)$ são OP-abstrações;
- K_j são controláveis ($S_j = K_j$);
- $\overset{m}{\underset{j=1}{\cup}} (\Sigma_{E_j}) \subseteq \Sigma_r$;
- $S_j = \overline{S_j} \cap \mathcal{L}_m(G_j)$.

A segunda e terceira condições refletem o fato que K_j é controlável, $\forall j \in J$. Como $K_j = E_j||\mathcal{L}_m(G_j)$ é controlável, então os eventos da especificação genérica E_j (que gerou K_j) são relevantes e devem estar em Σ_r. Pode-se observar que a condição sobre os eventos não requer que todos os eventos compartilhados estejam em Σ_r, mas isso pode acontecer, se as especificações genéricas contiverem todos os eventos compartilhados. A quarta condição estabelece que S_j deve ser $\mathcal{L}_m(G_j)$-fechada, ou seja, que a marcação do autômato que representa o supervisor (e portanto da especificação) seja consistente com a marcação da planta.

As especificações genéricas, em geral, utilizam um número reduzido de eventos das plantas locais. Dessa forma, um número reduzido de eventos será considerado relevante. Isso evidencia o potencial do trabalho de obter abstrações de tamanho reduzido.

A condição que diz que as linguagens desejadas devem ser controláveis pode parecer restritiva. Propõe-se uma solução para lidar com a situação em que se tenha K_js não-controláveis. Nesse caso, a máxima sublinguagem controlável ($S_j = Sup\mathcal{C}(K_j, \mathcal{L}_m(G_j))$) pode ser obtida. A idéia consiste em recuperar a especificação genérica (E'_j) que gera a linguagem desejada (K'_j) que é controlável e igual à máxima sublinguagem controlável: $E'_j||\mathcal{L}_m(G_j) = K'_j = S_j$. Em outras palavras, o procedimento consiste em gerar uma especificação genérica "fictícia" que, quando composta com a planta, resulta em $K'_j = Sup\mathcal{C}(K_j, G_j)$. Então, a linguagem desejada (K'_j) obtida pela composição de E'_j com G_j é necessariamente controlável.

Muitas formas de adaptar a especificação original para obter uma que gere uma linguagem desejada controlável podem ser usadas. Neste trabalho, sugere-se o uso de algoritmos de redução de supervisores. É comum que o supervisor tenha muita informação redundante que pode ser inferida da estrutura da planta. Seja M_1 um supervisor que controla a planta G. Um supervisor M_2 é considerado equivalente a M_1 se $\mathcal{L}(M_1||G) = \mathcal{L}(M_2||G)$ e $\mathcal{L}_m(M_1||G) = \mathcal{L}_m(M_2||G)$. O supervisor M_2 é um supervisor reduzido de M_1 se é equivalente e tem menos estados que M_1. Apesar de Su e Wonham [2004] terem mostrado que o problema de computar um supervisor com espaço de estados mínimo é NP-difícil, autores como [Su e Wonham, 2004] e [Minhas, 2002] apresentam algoritmos de redução heurísticos, de tempo polinomial, que atingem, freqüentemente, redução razoável no espaço de estados para problemas de tamanho

moderado. Ferramentas computacionais desenvolvidas para TCS possuem rotinas de redução de supervisores implementadas [Sivolella et al., 2006], [Reiser et al., 2006], [Feng e Wonham, 2006b]. As rotinas implementadas retornam também estimativas para o número de estados do supervisor reduzido mínimo.

O supervisor reduzido é então usado para evidenciar quais eventos devem estar em Σ_r. Ele substitui a especificação genérica E'_j pois, quando composto com a planta, gera a linguagem desejada $K'_j = E'_j||\mathcal{L}_m(G_j)$ que é controlável e igual a S_j. Os eventos que aparecem em autolaços em todos os estados devem ser removidos do autômato que representa a especificação genérica (supervisor reduzido). Então, os eventos restantes da especificação genérica fictícia são incluídos no conjunto inicial de eventos relevantes, $\Sigma_r^I = \bigcup_{j=1}^{m} \Sigma_{E_j}$. Os conjuntos de eventos relevantes locais são obtidos fazendo $\Sigma_j'^I = \Sigma_j \cap \Sigma_r^I$, $\forall j \in J$.

As especificações genéricas, assim como os supervisores reduzidos, em geral utilizam um número reduzido de eventos das plantas locais. Isso evidencia a potencialidade das condições estabelecidas neste capítulo.

6.3 Resultados

Nesta seção, apresenta-se o Lema 5.3.1 e os Teoremas 5.3.3 e 5.3.4, reescritos com as novas condições.

Lema 6.3.1. *Sejam $S_j \subseteq \Sigma_j^*$, $s \in \Sigma^*$, $t \in \Sigma_r^*$, as projeções naturais θ_j, θ, $P_{\Sigma\to\Sigma_j}$ e as linguagens K_j e E_j como definidas anteriormente. Se*

- *K_j são controláveis em relação a $\mathcal{L}(G_j)$ ($S_j = K_j$);*
- *$\bigcup_{j=1}^{m} (\Sigma_{E_j}) \subseteq \Sigma_r$;*
- *$S_j = \overline{S_j} \cap \mathcal{L}_m(G_j)$.*

e se $\theta(s)t \in \|_{j=1}^{m} \theta_j(S_j)$ então $\exists t_j \in \Sigma_j'^*$, $\forall j \in J$, tal que:

i. $\theta(s)t \in \|_{j=1}^{m} \theta_j(P_{\Sigma\to\Sigma_j}s)t_j$;

ii. $\theta_j(P_{\Sigma\to\Sigma_j}s)t_j \in \theta_j(S_j)$, *para todo* $j \in J$;

iii. $\|_{j=1}^{m} \theta_j(P_{\Sigma\to\Sigma_j}s)t_j \subseteq \|_{j=1}^{m} \theta_j(S_j)$.

Demonstração. A demonstração deste lema decorre diretamente das demonstrações do Lema 5.3.3 e Proposição 6.1.2, onde as referências à Proposição 5.2.1 devem ser substituídas por referências à Proposição 6.1.2. □

Teorema 6.3.1. *Sejam $S_j \in \Sigma_j^*$, $\forall j \in J$, as linguagens implementadas pelos supervisores S_j, as projeções naturais θ_j, θ, $P_{\Sigma \to \Sigma_j}$, $P_{\Sigma_r \to \Sigma_j'}$ e $P^{-1}_{\Sigma_r \to \Sigma_j'}$ como definidos anteriormente. Se as projeções naturais $\theta_j(S_j)$, $\forall j \in J$, são OP-abstrações e se*

- *K_j são controláveis em relação a $\mathcal{L}(G_j)$ ($S_j = K_j$);*
- *$\bigcup_{j=1}^{m} (\Sigma_{E_j}) \subseteq \Sigma_r$;*
- *$S_j = \overline{S_j} \cap \mathcal{L}_m(G_j)$.*

então

$$\mathop{||}_{j=1}^{m} \overline{\theta_j(S_j)} = \overline{\mathop{||}_{j=1}^{m} \theta_j(S_j)} \iff \mathop{||}_{j=1}^{m} \overline{S_j} = \overline{\mathop{||}_{j=1}^{m} S_j}.$$

Demonstração. A demonstração deste teorema decorre diretamente das demonstrações do Teorema 5.3.3, Proposição 6.1.2 e Lema 6.1.2, onde as referências à Proposição 5.2.1 e ao Lema 5.3.4 devem ser substituídas por referências à Proposição 6.1.2 e ao Lema 6.1.2, respectivamente. □

Teorema 6.3.2. *Sejam $S_j \in \Sigma_j^*$, $\forall j \in J$, as linguagens implementadas pelos supervisores, as projeções naturais θ_j, θ, $P_{\Sigma \to \Sigma_j}$, $P_{\Sigma_r \to \Sigma_j'}$ e $P^{-1}_{\Sigma_r \to \Sigma_j'}$ como definidos anteriormente. Se $\forall j \in J$, as projeções naturais $\theta_j(S_j)$ são OP-abstrações e se*

- *K_j são controláveis em relação a $\mathcal{L}(G_j)$ ($S_j = K_j$);*
- *$\bigcup_{j=1}^{m} (\Sigma_{E_j}) \subseteq \Sigma_r$;*
- *$S_j = \overline{S_j} \cap \mathcal{L}_m(G_j)$.*

então $\theta\big(\mathop{||}_{j=1}^{m} S_j\big)$ é também uma OP-abstração.

Demonstração. A demonstração deste teorema decorre diretamente das demonstrações do Teorema 5.3.4, Proposição 6.1.2 e Lema 6.1.2, onde as referências à Proposição 5.2.1 e ao Lema 5.3.4 devem ser substituídas por referências à Proposição 6.1.2 e ao Lema 6.1.2, respectivamente. □

6.4 Procedimento

Para aplicar os resultados do Teorema 6.3.1 sobre um sistema de controle modular, utiliza-se o mesmo algoritmo apresentado na seção 5.4. Esse algoritmo é repetido a seguir.

Algoritmo 1:

1. Obtêm-se os supervisores locais;
2. Obtém-se o conjunto inicial de eventos relevantes, Σ_r^I;
3. Obtêm-se os conjuntos iniciais locais de eventos relevantes como $\Sigma_j'^I = \Sigma_j \cap \Sigma_r^I$, $\forall j \in J$;
4. Obtêm-se OP-abstrações $\theta(S_j)$ para cada supervisor S_j (pode ser necessário estender o conjunto de eventos Σ_r^I);
5. Usam-se as abstrações $\theta_j(S_j)$, $\forall j \in J$, no teste de não-conflito.

Como discutido anteriormente, a diferença entre a abordagem apresentada neste capítulo e a abordagem do capítulo anterior está no conjunto inicial de eventos relevantes. Desta forma, o item 2 do algoritmo é substituído pelo procedimento a seguir.

Procedimento T para gerar Σ_r^I:

a. Aplica-se algoritmo de redução de supervisores a todos os supervisores locais, obtendo-se S_{Rj};

b. Removem-se os eventos que aparecem em auto-laços em todos os estados de cada supervisor reduzido e renomeia-se cada supervisor reduzido S_{Rj} como E_j';

c. $\Sigma_r^I = \bigcup_{j=1}^{m} \Sigma_{E_j'}$.

O procedimento T é aplicado apenas sobre supervisores que não são gerados através da composição de uma especificação genérica com sua planta local, ou seja, para o caso da especificação local ser não-controlável. No caso de $S_j = E_j||G_j$, o supervisor não precisa ser reduzido, $E_j' = E_j$ e $\Sigma_{E_j'} = \Sigma_{E_j}$.

A combinação do Algoritmo 1 com o Procedimento T é chamada de **T-solução**. A seguir, apresenta-se o mesmo exemplo utilizado anteriormente (Exemplo 5.4.1), que ilustra a aplicação do algoritmo e os ganhos obtidos pela aplicação do teste sobre as T-abstrações.

Exemplo 6.4.1. *Sistema de Manufatura Integrado - **T-solução***
O exemplo apresentado no capítulo anterior é resolvido usando a abordagem apresentada neste capítulo.
Considera-se o IMS, cujo diagrama é reapresentado na Fig. 6.1, cujos modelos de suas subplantas são reapresentados na Fig. 6.2 e as especificações na Fig. 6.3. As abstrações são

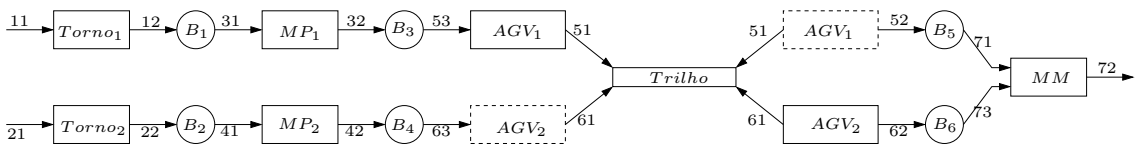


Figura 6.1: Diagrama do IMS

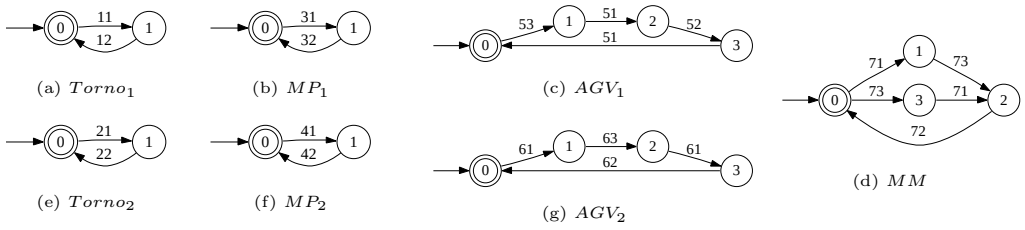


Figura 6.2: Exemplo 6.4.1 - Modelos das subplantas: (a) $Torno_1$; (b) MP_1; (c) AGV_1; (d) MM; (e) $Torno_2$; (f) MP_2; (g) AGV_2

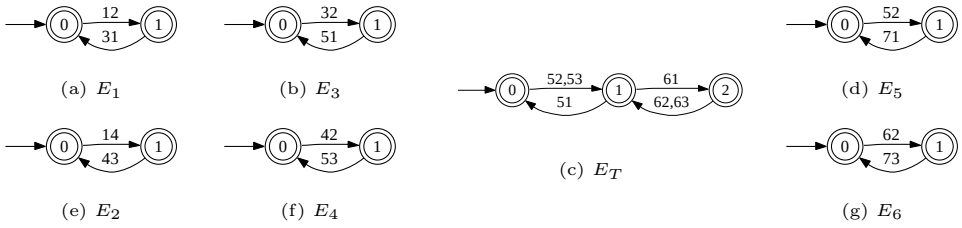


Figura 6.3: Exemplo 6.4.1 - Especificações E_j, $j = \{1, \ldots, 6, T\}$

geradas a partir dos supervisores obtidos no exemplo anterior.
As especificações locais K_j, $j \in \{1 \ldots 6, T\}$ são todas não-controláveis. De acordo com o procedimento T, apresentado na Seção 6.4, obtém-se especificações equivalentes E'_j tais que a linguagem desejada $K'_j = G_j||E'_j$ gerada é controlável e igual a S_j, para $j = \{1, \ldots, 6, T\}$. O supervisor reduzido resultante (S_{Rj}) para cada supervisor S_j, neste exemplo, é apresentado na Fig. 6.4. Como S_j pode ser reconstruído através da composição do supervisor reduzido S_{Rj} com a planta G_j, então S_{Rj} pode ser considerado a especificação fictícia E'_j. Os algoritmos de redução de supervisores não retornam o supervisor mínimo em todos os casos. Entretanto, eles retornam uma estimativa do número de estados do autômato do supervisor mínimo. Se esta estimativa coincidir com o número de estados do autômato reduzido, indica que esse supervisor reduzido é mínimo. Neste exemplo, todos os supervisores reduzidos são mínimos. Quando um evento aparece em auto-laços em todos os estados do supervisor reduzido, significa que sua ocorrência não é importante para os objetivos de controle e, portanto, suas ocorrências podem ser apagadas sem causar perda de informação. A Tabela 6.1 apresenta os eventos retidos e removidos para cada especificação. Todos os eventos mantidos nas es-

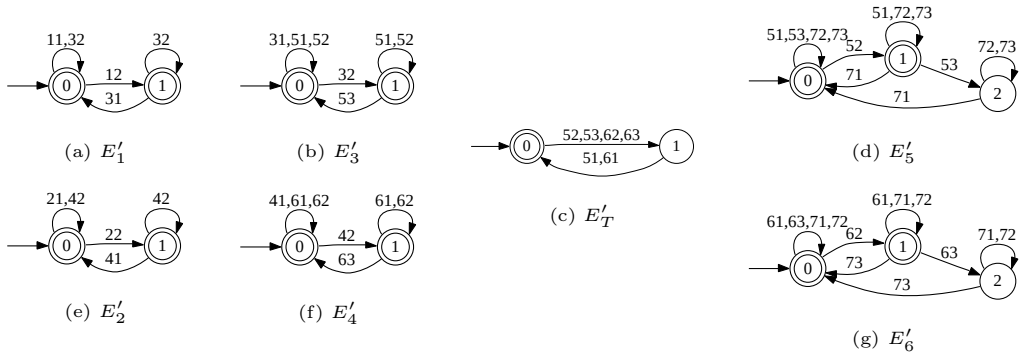


Figura 6.4: Exemplo 6.4.1 - Especificações fictícias E'_j, $j = \{1, \ldots, 6, T\}$

Tabela 6.1: Eventos das especificações E'_j, $\forall j \in \{1 \ldots 6, T\}$

Espec.	Removidas	Retidos
$E'_1 = S_{R1}$	$\{32\}$	$\{11, 12, 31\}$
$E'_2 = S_{R2}$	$\{42\}$	$\{21, 22, 41\}$
$E'_3 = S_{R3}$	$\{51, 52\}$	$\{31, 32, 53\}$
$E'_4 = S_{R4}$	$\{61, 62\}$	$\{41, 42, 63\}$
$E'_5 = S_{R5}$	$\{72, 73\}$	$\{51, 52, 53, 71\}$
$E'_6 = S_{R6}$	$\{71, 72\}$	$\{61, 62, 63, 73\}$
$E'_T = S_{RT}$	none	$\{51, 52, 53, 61, 62, 63\}$

pecificações constituem o conjunto inicial de eventos relevantes: $\Sigma_r^I = \{11, 12,\ 21, 22,\ 31, 32\ 41, 42,\ 51, 52, 53, 54,\ 61, 62, 63, 71\}$. *Nesse caso, o único evento que pode ser apagado é o* 72, *que faz parte dos supervisores* S_5 *e* S_6. *Em seguida, deve-se buscar OP-abstrações através da extensão de* Σ_r^I. *Nesse exemplo, a projeção de* S_5 *e* S_6, *apagando o evento* 72 *gera OP-abstrações. Os supervisores originais possuem 28 estados e 51 transições cada um. Por outro lado, suas abstrações possuem 21 estados e 38 transições cada. Os outros supervisores são mantidos intactos e o teste de não-conflito consiste em verificar a seguinte igualdade:*

$$\overline{S_1}||\overline{S_2}||\overline{S_3}||\overline{S_4}||\overline{\theta_5(S_5)}||\overline{\theta_6(S_6)}||\overline{S_T} = \overline{S_1||S_2||S_3||S_4||\theta_5(S_5)||\theta_6(S_6)||S_T}.$$

O autômato gerado pelo teste do não-conflito sobre os supervisores originais possui 12.960 estados e 15.264 transições e o autômato do teste sobre as T-abstrações $\theta_5(S_5)$ *e* $\theta_6(S_6)$ *possui 9.720 estados e 36.504 transições e detectou o não-conflito no sistema. As duas abstrações foram responsáveis por uma redução de 25% no número de estados e de 27% no número de transições do teste e, como esperado, identificaram a não-existência de conflito no sistema.*

6.5 Estratégia Combinada

Tendo desenvolvido dois procedimentos independentes e incomparáveis para obtenção das abstrações, surgiu o interesse de combinar os dois procedimentos de forma a obter um terceiro procedimento que produza resultados ainda melhores. Portanto, o conjunto inicial de eventos relevantes é obtido por uma combinação da W-solução com T-solução de forma apropriada. Essa seção discute essa combinação, apontando em quais situações e como as soluções podem ser combinadas.

Para uma solução combinada, pode-se a princípio pensar em duas possibilidades:

- W $\circ$ T, onde W é aplicada sobre as abstrações resultantes de T;
- T $\circ$ W, onde T é aplicada sobre as abstrações resultantes de W.

A estratégia combinada W $\circ$ T (T $\circ$ W) consiste em quatro passos: (1) obtenção do conjunto de eventos relevantes usando T (W), Σ_{rT}^I (Σ_{rW}^I); (2) geração das abstrações; (3) uso

das abstrações obtidas como linguagens originais para W (T); (4) obtenção de um conjunto de eventos relevantes usando W (T). No entanto, o caso T $\circ$ W não funciona. Para obter Σ_{rT}^{I}, é necessário ter conhecimento da estrutura do sistema. Na verdade, T é aplicado sobre linguagens obtidas por composições de especificações genéricas com plantas locais. Isso não é verdadeiro, quando aplicado às W-abstrações. Baseado nessa observação, apresenta-se o procedimento combinado, W $\circ$ T.

Algoritmo 2 - W $\circ$ T:

1. Obtém-se os supervisores locais;
2. Obtém-se o conjunto inicial de eventos relevantes Σ_{rT}^{I} usando T;
3. Obtém-se o conjunto inicial local de eventos relevantes como $\Sigma_{jT}^{\prime I} = \Sigma_j \cap \Sigma_{rT}^{I}$, $\forall j \in J$;
4. Obtém-se OP-abstrações $\theta(S_j)$ para os supervisores S_j. Renomeia-se o conjunto final de eventos relevantes para Σ_{rT};
5. Obtém-se o conjunto inicial de eventos relevantes $\Sigma_{rW\circ T}^{I} = \Sigma_{sT} \subseteq \Sigma_{rT}$, onde $\Sigma_{sT} = \Sigma_s \cap \Sigma_{rT} \subseteq \Sigma_s$. Em palavras, Σ_{sT} é o conjunto de eventos compartilhados por quaisquer duas ou mais T-abstrações;
6. Obtém-se o conjunto inicial local de eventos relevantes como $\Sigma_{jW\circ T}^{\prime I} = (\Sigma_j \cap \Sigma_{rT}) \cap \Sigma_{rW\circ T}^{I}$, $\forall j \in J$;
7. Obtém-se OP-abstrações sobre cada T-abstração obtida no passo 4.. Renomeia-se o conjunto final de eventos relevantes como $\Sigma_{rW\circ T}$;
8. Usam-se as abstrações resultantes do passo 7 para realizar o teste de não-conflito.

Pode-se perceber que os 4 primeiros passos correspondem à aplicação do procedimento T sobre os supervisores e os passos 5-8 correspondem à aplicação de W sobre as abstrações resultantes dos passos 1-4.

Em muitos casos, supervisores são reduzidos apenas por um dos dois métodos. Há casos em que determinado supervisor não será reduzido por nenhum dos dois métodos. De acordo com a redução obtida, um supervisor pode ser classifcado em: SW e ST. Supervisores do tipo ST são aqueles que são reduzidos nos primeiros 4 passos do Algoritmo 2, independente deles serem também reduzidos nos passos 5-8 do algoritmo. Supervisores do tipo SW são aqueles que não podem ser reduzidos nos primeiros 4 passos do Algoritmo 2. Apresenta-se na seqüência um exemplo da classificação de supervisores em ST e SW.

Exemplo 6.5.1. *Supervisores tipo ST e SW*
Sejam $\Sigma_1 = \{a, b, c, d\}$, $\Sigma_2 = \{b, c, e, f, g, h\}$, $\Sigma_3 = \{g, h, i, j\}$ *e* $\Sigma_4 = \{j, l, m\}$ *os conjuntos de eventos dos supervisores* $S_k \subseteq \Sigma_k^*$, *para* $k \in \{1 \ldots 4\}$, *como mostrado na Fig. 6.5.*

Depois de aplicar T (passos 1-4), assume-se que o conjunto de eventos relevantes é $\Sigma_{rT} = \{a, b, c, d, e, f, h, i, j, l, m\}$. *Nesse caso, o único evento apagado é* $\{g\}$. *A partir do conjunto*

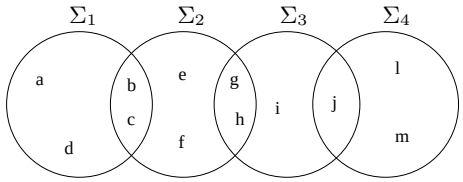


Figura 6.5: Diagramas de Venn dos alfabetos dos supervisores.

de eventos relevantes pode se dizer quais supervisores são reduzidos e como são classificados. Nos primeiros 4 passos do procedimento, os supervisores S_2 e S_3 são reduzidos e classificados como sendo do tipo ST. Os supervisores restantes são então classificados como sendo do tipo SW. Passos 5 a 8 são aplicados sobre o conjunto de supervisores SW (S_1 e S_4) mais o conjunto de abstrações obtidas nos passos anteriores ($\theta_2(S_2)$ e $\theta_3(S_3)$). Seja o conjunto de eventos relevantes, obtido a partir dos passos 5-8, igual a $\Sigma_{rW} = \{b, c, d, e, h, i, j, l, m\}$. Nesse caso, os eventos $\{a, f\}$ são apagados e os supervisores S_1 e S_2 são abstraídos. O conjunto SW= $\{S_1, S_4\}$ é composto de um supervisor que só pode ser reduzido por W e outro supervisor que não é reduzido por nenhum dos métodos.

Para comparar as 3 abordagens (W,T e W$\circ$T) e as reduções obtidas através delas, deve-se considerar os supervisores do tipo SW e ST separadamente.

- Supervisor do tipo SW: Aplicação de W gera o mesmo resultado que a aplicação de W $\circ$ T. Este tipo de supervisor não é reduzido por T.
- Supervisor do tipo ST: Este tipo de supervisor é reduzido por T. Pode ser o caso que os passos 5-8 (W) gerem redução também em alguma abstração. Pode também ser o caso dos passos 5-8 não reduzirem qualquer abstração. De forma geral, não há a garantia de que o procedimento combinado vai gerar reduções maiores do que a aplicação de apenas W.

Não é sempre verdade que quanto mais eventos apagados, maior a redução obtida. Pode ser o caso de a abstração resultante de apagar apenas um evento ser menor que a abstração resultante de apagar 2 ou mais eventos. Por isso, para um supervisor que possa ser reduzido por mais de um dos três métodos disponíveis, não é possível saber qual deve ser usado para obter o melhor resultado.

A maior vantagem que o método combinado traz é a possibilidade de abstrair supervisores diferentes usando métodos diferentes (W ou T). Para ilustrar essa idéia, utiliza-se o Exemplo 6.5.1. Ao aplicar T, apenas os supervisores S_2 e S_3 podem ser reduzidos. Aplicando W, os supervisores S_2 e S_3 podem ser abstraídos. Usando W $\circ$ T, pode-se reduzir S_2 e S_3 nos primeiros 4 passos e S_1 pode ser reduzido na segunda metade do procedimento. Pode ainda ser o caso de $\theta_2(S_2)$ ser reduzido nos passos 5-8 do procedimento W $\circ$ T. Portanto, ao aplicar W $\circ$ T, o teste de não-conflito será aplicado sobre $\theta_1(S_1)$, $\theta_2(S_2)$, $\theta_3(S_3)$ e S_4, obtendo maior redução que se aplicasse W ou T separadamente, já que 3 dos 4 supervisores foram reduzidos.

Um exemplo da estratégia combinada é apresentada a seguir.

Exemplo 6.5.2. *Sistema de Manufatura Integrado - $W \circ T$-solução*
A abordagem combinada, $W \circ T$ é aplicada sobre o Exemplo 5.4.1. A primeira parte, que consiste nos passos 1-4 do Algoritmo 2, foi realizado no Exemplo 6.4.1. Tem-se que $\Sigma_{rT} = \{11, 12, 21, 22,\ 31, 32, 41, 42,\ 51, 52, 53,\ 61, 62, 63,\ 71, 73\}$ e os supervisores S_5 e S_6 são reduzidos, sendo o evento $\{72\}$ apagado. Os supervisores podem então ser classificados como sendo do tipo ST= $\{S_5, S_6\}$ ou SW= $\{S_1, S_2, S_3, S_4, S_T\}$. Depois de obter as OP-abstrações para os supervisores S_5 e S_6, aplica-se W sobre os conjunto de supervisores do tipo SW e também sobre as abstrações obtidas, ou seja, sobre $\{S_1, S_2, S_3, S_4, \theta_5(S_5), \theta_6(S_6), S_T\}$. O conjunto inicial de eventos relevantes é composto do conjunto de eventos compartilhados, $\Sigma_c = \{31, 32, 41, 42,\ 51, 52, 53, 61,\ 62, 63, 71, 73\}$. Para obter OP-abstrações, esse conjunto de eventos relevantes é estendido, para incluir eventos $\{11, 21\}$. Então, $\Sigma_r = \{11, 21, 31, 32, 41, 42, 51, 52, 53, 61, 62,\ 63, 71, 73\}$ e $\{12, 22\}$ são apagados. Os supervisores S_1 e S_2 são abstraídos e as abstrações resultantes são apresentadas na Fig. 5.9.

Então, as abstrações obtidas nos passos 1-4 e aquelas obtidas nos passos 5-8 do Algoritmo 2 são usados para realização do teste de não-conflito:

$$\overline{\theta_1(S_1)}||\overline{\theta_2(S_2)}||\overline{S_3}||\overline{S_4}||\overline{\theta_5(S_5)}||\overline{\theta_6(S_6)}||\overline{S_T} = \overline{\theta_1(S_1)||\theta_2(S_2)||S_3||S_4||\theta_5(S_5)||\theta_6(S_6)||S_T}$$

Pode-se perceber que 4 dos 6 supervisores são abstraídos usando $W \circ T$. Aplicando T ou W separadamente, seria possível obter abstrações para apenas 2 de 6 supervisores.

O autômato gerado pelo teste de não-conflito sobre as OP-abstrações de S_1, S_2, S_5 e S_6 é não-bloqueante, como esperado, e tem 4.320 estados e 15.264 transições. Portanto, a solução combinada causou uma redução de 67% no número de estados e 69% no número de transições do teste.

6.6 Conflitos e a Propriedade do Observador

Os resultados apresentados neste capítulo e no Capítulo 5 (e previamente publicadas em [Pena et al., 2006c] e [Pena et al., 2006b]) possuem características similares ao trabalho de Flordal e Malik [2006] pois ambos utilizam abstrações de subsistemas para verificar existência de bloqueio do sistema global formado pela composição desses subsistemas.

Nos resultados apresentados nessa tese usa-se a projeção natural das linguagens reconhecidas pelos subsistemas em um conjunto de eventos relevantes Σ_r. Esse conjunto deve ser escolhido tal que a projeção tenha a propriedade do observador. Dois métodos para obtenção desse conjunto mínimo de eventos a serem mantidos na projeção foram apresentados nos capítulos 5 e 6. O que acontece nessa abordagem, em geral, é a necessidade da expansão desse conjunto mínimo de eventos para que a projeção possua a propriedade do observador.

Na abordagem introduzida por Flordal e Malik [2006], considera-se o conjunto de eventos compartilhados como os eventos que devem ser mantidos nas abstrações e o restante dos

eventos é renomeado para τ sendo que cada τ pode ser "escondido" (projetado) independentemente das outras ocorrências. Essa idéia de renomear eventos utilizando τ com essas características foi inspirada na teoria de álgebra de processos. Alguns critérios para a escolha de quais ocorrências de τ podem ser escondidas são apresentadas em [Flordal e Malik, 2006].

Flordal et al. [2007] apresentam uma discussão a respeito das duas abordagens mencionadas e estabelecem relações entre seus conceitos e resultados. Entre as conclusões apresentadas, destaca-se a que diz que uma abstração obtida pela propriedade do observador é a melhor abstração possível (em termos de número de estados) que permite concluir sobre a existência de conflitos. Este trabalho abre novas frentes para os dois grupos de pesquisadores. Pretende-se combinar os dois métodos para produzir algoritmos mais eficientes para verificação de não-bloqueio. Mais especificamente, como as projeções naturais com a propriedade do observador podem ser incorporadas ao conceito de conflito-equivalência e, por outro lado, como a propriedade do observador pode ser estendida incorporando conceitos, como a renomeação de eventos, visando obter reduções ainda maiores para as abstrações.

6.7 Discussão

Este capítulo apresentou uma variação do teste de não-conflito, em que um outro conjunto de condições suficientes para as abstrações é apresentado. Nesse novo conjunto, propriedades estruturais dos supervisores são utilizadas, tornando possível, em alguns casos, apagar eventos compartilhados pelos supervisores. Para ilustrar os conjuntos de eventos utilizados, apresentam-se diagramas Venn na Figura 6.6.

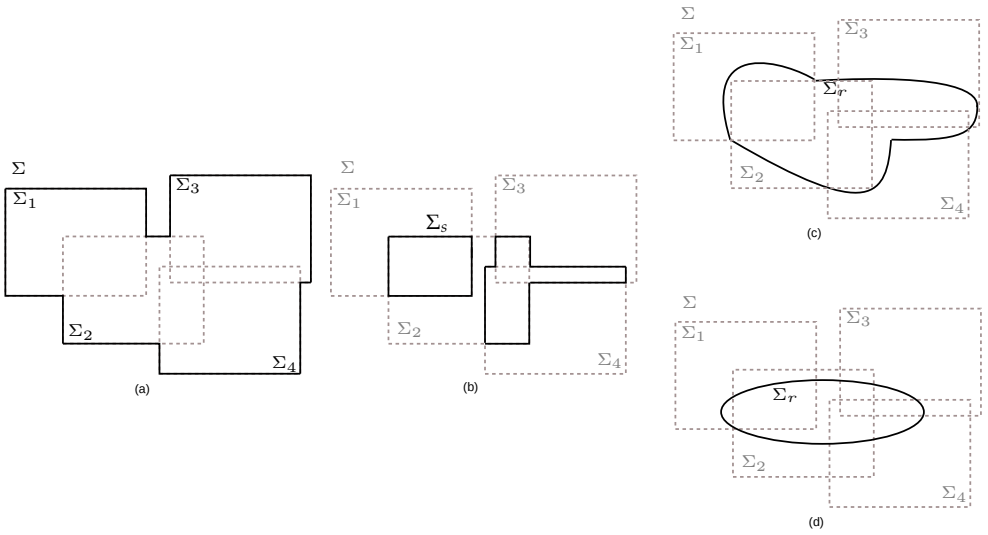


Figura 6.6: Diagramas de Venn dos conjuntos de eventos para $m = 4$.

Se $m = 4$ quer dizer que há 4 supervisores ($S_1 \subseteq \Sigma_1^*$, $S_2 \subseteq \Sigma_2^*$, $S_3 \subseteq \Sigma_3^*$ e $S_4 \subseteq \Sigma_4$). Consideram-se os alfabetos Σ_1, Σ_2, Σ_3, Σ_4, como mostrados na Figura 6.6(a). A superposição de alfabetos representa a existência de eventos comuns aos alfabetos superpostos.

Seja o conjunto $\Sigma_s = \bigcup_{k,l\in J}^{k\neq l} (\Sigma_k \cap \Sigma_l)$, formado dos eventos compartilhados por dois ou mais supervisores, mostrado na Figura 6.6(b). O conjunto Σ_r obtido usando W é apresentado na Figura 6.6(c) e Σ_r obtido usando T é apresentado na Figura 6.6(d). Pode-se observar que $\Sigma_s \subseteq \Sigma_r$, na Figura 6.6(c), enquanto que, na Figura 6.6(d), o conjunto Σ_r não contém, necessariamente, o conjunto Σ_s, já que é formado da união dos conjuntos de eventos das especificações controláveis, $\bigcup_{j=1}^{m} \Sigma_{E'_j} \subseteq \Sigma_r$, que não necessariamente contém todos os eventos comuns.

Os teoremas apresentados no Capítulo 5 foram todos reescritos sob as novas condições apresentadas. Apenas as partes das demonstrações dos teoremas que foram afetadas pela mudança no conjunto de eventos relevantes foram demonstradas sob a forma da Proposição 6.1.2 e Lema 6.1.2.

Capítulo 7

Algoritmo para Verificação da Propriedade do Observador

O problema de obter projeções com a propriedade do observador é tratado em [Wong e Wonham, 2004], [Feng, 2006], [Schmidt e Moor, 2006]. Feng [2006] propõe um algoritmo polinomial no número de estados e transições que retorna, a partir de um conjunto inicial de eventos relevantes, uma extensão razoável para o conjunto, não necessariamente mínima, de forma a obter uma projeção natural com a propriedade do observador. O trabalho de Feng [2006] é uma extensão do trabalho de Wong e Wonham [2004] que gera mapas-repórteres com a propriedade do observador, pela renomeação de eventos, ou seja, ao detectar que um dado conjunto de eventos relevantes não gera uma abstração com a propriedade do observador, renomeia-se a ocorrência do evento que viola a propriedade. Segundo Schmidt e Moor [2006], o método para escolha do conjunto inicial de eventos relevantes não é óbvio em [Wong e Wonham, 2004]. Eles então apresentam um procedimento para modificar uma projeção natural de forma a torná-la um observador, chamado pelos autores de MSA-observador[1]. A projeção inicial, que é modificada pelo procedimento, consiste na projeção da linguagem no conjunto de eventos compartilhados por pelo menos um par de supervisores. O algoritmo usa renomeação de eventos, como Wong e Wonham [2004].

Em procedimentos de síntese, a renomeação de eventos para fins de obtenção de mapas-repórteres com a propriedade do observador pode ser utilizada, desde que seguida de uma reavaliação das especificações de forma que o supervisor leve em conta estas modificações. Já no contexto em que este trabalho se insere, de detectar o conflito de forma eficiente, não cabe a utilização deste tipo de abordagem. Portanto, entre os trabalhos citados, apenas o trabalho de Feng [2006] poderia ser aplicado diretamente nos procedimentos apresentados neste tese.

Com relação ao algoritmo apresentado por Feng [2006], sua implementação não está disponível para utilização. Como os resultados apresentados nesta tese dependem fortemente da propriedade do observador, julgou-se adequado desenvolver um algoritmo para verificação

[1] do inglês *Marked-String Acceptance Observer*

dessa propriedade. Este algoritmo deverá ser implementado na ferramenta *Grail para Controle Supervisório* [Reiser et al., 2006] que é desenvolvida pelo grupo do Professor José Cury.

Obtiveram-se abstrações, nos resultados apresentados nos Capítulos 5 e 6, com duas características, sendo uma relacionada ao conjunto de eventos relevantes e a outra relacionada à propriedade do observador. Há diferenças entre as condições sobre as abstrações de cada conjunto de resultados. No entanto, a condição de possuir a propriedade do observador está presente em ambos. Dadas as linguagens sobre as quais pretende-se verificar não-conflito, deve-se:

1. Determinar o conjunto inicial de eventos relevantes.
2. Obter as OP-abstrações pela projeção natural das linguagens no conjunto de eventos relevantes.
3. Aplicar o teste de não-conflito sobre as abstrações obtidas.

O conjunto inicial de eventos relevantes poderá ser estendido para incluir outros eventos, de forma a obter a projeção com a propriedade do observador. Sabe-se que a complexidade da operação de projeção natural pode ser exponencial, no caso dela não possuir a propriedade do observador. É interessante, portanto, ser capaz de classificar uma projeção como possuindo ou não a propriedade do observador sem efetivamente computar a projeção. Este capítulo apresenta um teste de complexidade polinomial aplicado sobre linguagens para verificar se uma projeção possui a propriedade do observador sem, no entanto, computá-la.

Dada uma linguagem e um conjunto de eventos relevantes, o teste é aplicado sobre pares de cadeias da linguagem que possuem a mesma projeção. Em seguida, apresenta-se uma estrutura de transição, chamada de OP-verificador. A construção do verificador coincide com a aplicação do teste, evidenciando a existência ou não da propriedade do observador.

Dado um autômato G obtém-se uma estrutura M a partir da qual constrói-se o verificador V_G. O OP-verificador é uma estrutura de transição cujos estados são formados por pares de estados de M alcançados por pelo menos um par de cadeias de mesma projeção. Sua construção baseia-se no verificador introduzido por Yoo e Lafortune [2002a] para verificar diagnosticabilidade de sistemas a eventos discretos.

Neste capítulo, distingue-se a notação utilizada para representar os autômatos G, M e suas linguagens $\mathcal{L}_m(G)$, $\mathcal{L}(G)$, $\mathcal{L}(M)$.

7.1 Estrutura de Transição Auxiliar M

Dado um autômato e um conjunto de eventos relevantes Σ_r, obtém-se, inicialmente, uma estrutura de transição $M = (Q^M, \Sigma^\tau, \delta^\tau, q_0^M)$ tal que o teste apresentado é aplicado sobre a linguagem reconhecida por esta estrutura.

A estrutura de transição M é definida a partir de $G = (Q^G, \Sigma, \delta^\tau, q_0^G, F^G)$ como segue. Os estados marcados de G são mapeados em estados de M que possuem um auto-laço rotulado com o evento especial (e relevante) τ, ou seja, $M = (Q^M, \Sigma^\tau, \delta^\tau, q_0^M)$, onde:

- $Q^M = Q^G$ é o conjunto de estados (igual ao conjunto de estados do autômato G);
- $\Sigma^\tau = \Sigma \cup \{\tau\}$ sendo o evento τ considerado relevante ($\tau \in \Sigma_r$);
- a função de transição é definida a seguir:

$$\delta^\tau(q,\sigma) \quad = \quad \begin{cases} \delta^G(q,\sigma) & \text{if } q \in Q^M, \sigma \neq \tau \\ q & \text{if } q \in F^G, \sigma = \tau \end{cases}$$

  ou seja, a função de transição δ^τ de M possui as mesmas transições de G acrescidas de auto-laços rotulados com o evento τ nos estados correspondentes a estados marcados de G;

- $q_0^M = q_0^G$ é o estado inicial, corresponde ao estado inicial de G.

Dessa forma, a linguagem gerada por M é:

$$\mathcal{L}(M) = \{s' \in \Sigma^{\tau *} | \delta^\tau(q_0^M, s') \text{ é definida}\}. \tag{7.1}$$

A linguagem $\mathcal{L}_m(G)$ pode ser redefinida em função das cadeias de $\mathcal{L}(M)$ como:

$$\mathcal{L}_m(G) = \{s \in \Sigma^* | s' = s\tau \text{ com } s' \in \mathcal{L}(M)\}.$$

Define-se ainda a linguagem $N \subseteq \mathcal{L}(M)$ como sendo a sublinguagem de $\mathcal{L}(M)$ formada por cadeias em Σ^*, ou seja,

$$N = \mathcal{L}(M) \cap \Sigma^*. \tag{7.2}$$

É trivial verificar que $N = \mathcal{L}(G)$.

A seguir apresenta-se um exemplo da obtenção de M a partir de G.

Exemplo 7.1.1. *Seja $G = (Q^G, \Sigma, \delta^\tau, q_0^G, F^G)$ como apresentado na Figura 7.1(a). O estado 6 é marcado neste exemplo. A estrutura de transição $M = (Q^M, \Sigma^\tau, \delta^\tau, q_0^M)$ correspondente é apresentada na Figura 7.1(b). Pode-se observar que a única diferença entre as estruturas está na representação do estado marcado. No autômato original ele é representado por dois círculos cocêntricos e em M é representado pela existência de um auto-laço rotulado com o evento τ.*

Esta representação é bastante conveniente, pois permite representar o estado marcado de G como uma transição especial habilitada nos estados correspondentes em M.

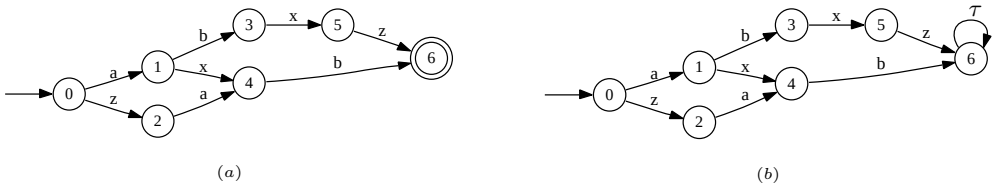


Figura 7.1: Exemplo 7.1.1: (a) G; (b) M.

Em seguida, define-se $T : \Sigma^* \rightarrow 2^{\Sigma^{\tau *}}$ como sendo:

$$T(s) = \{t\tau \in \Sigma^{\tau *} | st\tau \in \mathcal{L}(M)\}, \tag{7.3}$$

ou seja, $T(s)$ é o conjunto de cadeias $t\tau \in \Sigma^{\tau *}$ tal que $st\tau \in \mathcal{L}(M)$.

Duas restrições são impostas sobre o autômato G, para que os resultados deste capítulo sejam válidos:

(i) G não pode possuir ciclos formados apenas por eventos não-relevantes.

(ii) G deve ser não-bloqueante.

Como será discutido à frente, a condição (i) simplifica o algoritmo de construção do OP-verificador. Em uma continuação do trabalho, pretende-se eliminar tal restrição. A razão para G ser não-bloqueante é que a propriedade do observador lida apenas com cadeias contidas no prefixo da linguagem, não se interessando pelas cadeias bloqueantes do autômato. Se G for bloqueante, deve-se apará-lo antes da aplicação do algoritmo. O resultado dos testes aplicados sobre o verificador obtido para o G aparado é válido para G, ou seja, se a projeção de G aparado possuir a propriedade do observador, a projeção do G original também a possuirá e vice-versa.

7.2 OP-Teste

Apresenta-se nesta seção um teste a ser aplicado sobre M, tal que seja possível determinar se $\theta(\mathcal{L}_m(S))$ possui a propriedade do observador. O OP-Teste é descrito a seguir.

OP-Teste Se $s_1, s_2 \in N$ tal que $\theta(s_1) = \theta(s_2)$ então $\theta(T(s_2)) = \theta(T(s_1))$.

Em palavras, para todo par $s_1, s_2 \in N$ tal que $\theta(s_1) = \theta(s_2)$, $\forall t \in \Sigma^*$ tal que $s_2 t\tau \in \mathcal{L}(M)$, $\exists v \in \Sigma^*$ tal que $s_1 v\tau \in \mathcal{L}(M)$ e $\theta(t) = \theta(v)$. Nesse caso, pode-se dizer que M satisfaz o OP-Teste.

No Teorema 7.2.1 demonstra-se que $\theta(G)$ é uma OP-abstração se e somente se a condição apresentada acima pode ser verificada para todo par $s_1, s_2 \in N$, com $\theta(s_1) = \theta(s_2)$.

Teorema 7.2.1. *Sejam o autômato G, a estrutura de transição M, as linguagens $\mathcal{L}(M)$ e N, a projeção natural $\theta : \Sigma^* \to \Sigma_r^*$, OP-Teste, como apresentados anteriormente. Tem-se que:*

$$M \text{ satisfaz o OP-Teste} \iff \theta(G) \text{é uma } \textbf{\textit{OP-abstração}} \tag{7.4}$$

Demonstração. Esta demonstração é dividida em duas partes: (i) supõe-se M não satisfaz o OP-Teste e $\theta(\mathcal{L}_m(G))$ possui a propriedade do observador; (ii) supõe-se que $\theta(\mathcal{L}_m(G))$ não possui a propriedade do observador e M satisfaz o OP-Teste. Se for possível chegar a contradições nos dois casos, a demonstração estará completa.

Caso (i): supõe-se que M não satisfaz o OP-Teste e $\theta(\mathcal{L}_m(G))$ possui a propriedade do observador. Se M não satisfaz o OP-Teste, então $\theta(T(s_2)) \neq \theta(T(s_1))$. A expressão $\theta(T(s_2)) \neq \theta(T(s_1))$ é verdadeira se: $\theta(T(s_2)) \nsubseteq \theta(T(s_1))$ e/ou $\theta(T(s_1)) \nsubseteq \theta(T(s_2))$. Deve-se mostrar que em cada um dos subcasos chega-se a uma contradição.

$\theta(T(s_2)) \nsubseteq \theta(T(s_1))$ Neste caso, para algum par de cadeias $s_1, s_2 \in N$, com $\theta(s_1) = \theta(s_2)$, $\exists t \in \Sigma^*$ com $s_2 t\tau \in \mathcal{L}(M)$, mas $\nexists v \in \Sigma^*$ tal que $s_1 v\tau \in \mathcal{L}(M)$ e $\theta(s_1) = \theta(s_2)$. Pela construção de M, tem-se qe $s_1, s_2 \in \mathcal{L}(G)$, pois $N = \mathcal{L}(G)$. Tem-se também que $\mathcal{L}_m(G) = \{x \in \Sigma^* | x\tau \in \mathcal{L}(M)\}$, o que significa que $s_2 t \in \mathcal{L}_m(G)$ e:

$$\nexists v \in \Sigma^* \text{ tal que } s_1 v \in \mathcal{L}_m(G) \tag{7.5}$$

com $\theta(t) = \theta(v)$.

Como $s_2 t \in \mathcal{L}_m(G)$, tem-se que $\theta(s_2 t) \in \theta(\mathcal{L}_m(G))$. Considere a escolha $m = s_1 \in \mathcal{L}(G)$ e $n = \theta(t)$. Então, tem-se que $\theta(m)n \in \theta(\mathcal{L}_m(G))$, uma vez que $\theta(s_1) = \theta(s_2)$. Se $\theta(\mathcal{L}_m(G))$ possui a propriedade do observador, então é verdade que existe uma cadeia $p \in \Sigma^*$, daqui para frente chamada v para unificar a notação, tal que $\theta(mv) = \theta(m)n \in \theta(\mathcal{L}_m(G))$ e $mv \in \mathcal{L}_m(G)$. Se substitui-se m por s_1 e n por $\theta(t)$ percebe-se claramente a contradição, ou seja: $\exists v \in \Sigma^*$ tal que $\theta(s_1 v) = \theta(s_2)\theta(t) = \theta(s_2 t)$ e $s_1 v \in \mathcal{L}_m(G)$. Então:

$$\exists v \in \Sigma^* \text{ tal que } s_1 v \in \mathcal{L}_m(G) \tag{7.6}$$

com $\theta(t) = \theta(v)$ e equação (7.6) contradiz equação (7.5).

$\theta(T(s_1)) \nsubseteq \theta(T(s_2))$ A mesma argumentação pode ser aplicada após substituir todas as ocorrências de s_1 por s_2 e de t por v, e vice-versa, levando também a uma contradição.

Caso (ii): Supõe-se que M satisfaz o OP-Teste e $\theta(\mathcal{L}_m(G))$ não possui a propriedade do observador. M satisfaz o OP-Teste se $\theta(T(s_2)) = \theta(T(s_1))$, ou seja, se $\forall s_1, s_2 \in N$, com $\theta(s_2) = \theta(s_1)$:

1. $\forall t \in \Sigma^*$ com $s_2 t\tau \in \mathcal{L}(M)$, $\exists v \in \Sigma^*$ tal que $s_1 v\tau \in \mathcal{L}(M)$ e $\theta(t) = \theta(v)$ e
2. $\forall v \in \Sigma^*$ com $s_1 v\tau \in \mathcal{L}(M)$, $\exists t \in \Sigma^*$ tal que $s_2 t\tau \in \mathcal{L}(M)$ e $\theta(v) = \theta(t)$

Considerando as relações entre N, $\mathcal{L}(M)$, $\mathcal{L}(G)$ e $\mathcal{L}_m(G)$, tem-se que $\forall s_1, s_2 \in \mathcal{L}(G)$ e

1. $\forall t \in \Sigma^*$ com $s_2 t \in \mathcal{L}_m(G)$,

$$\exists v \in \Sigma^* \text{ tal que } s_1 v \in \mathcal{L}_m(G) \tag{7.7}$$

com $\theta(t) = \theta(v)$. Considerando o descrito acima como hipótese, usa-se a propriedade do observador para chegar a uma contradição.

Se $\mathcal{L}_m(G)$ não possui a propriedade do observador, então deve existir pelo menos uma cadeia que viola a propriedade. Considere que esta cadeia é $s_1 \in \mathcal{L}(G)$. Então, deve existir outra cadeia $s_2 t \in \mathcal{L}_m(G)$ $(m = s_2 t)$ tal que $\theta(s_1) = \theta(s_2)$. Desta forma, $\theta(s_2 t) \in \theta(\mathcal{L}_m(G))$ e $n = \theta(t)$. Então $\theta(s_2)n \in \theta(\mathcal{L}_m(G))$. Como $\theta(s_2) = \theta(s_1)$, é verdade que $\theta(s_1)n \in \theta(\mathcal{L}_m(G))$. Se $\theta(\mathcal{L}_m(G))$ não possui a propriedade do observador, então não existe $p \in \Sigma^*$, daqui para frente chamada de v para unificar notação, tal que $\theta(s_1)n = \theta(s_1 v)$ e $s_1 v \in \mathcal{L}_m(G)$. Ou seja:

$$\nexists v \in \Sigma^* \text{ tal que } s_1 v \in \mathcal{L}_m(G) \tag{7.8}$$

com $\theta(v) = \theta(t)$. Se M satisfaz o OP-Teste, é também verdade que $\forall v \in \Sigma^*$ com $s_1 v \in \mathcal{L}_m(G)$, $\exists t \in \Sigma^*$ tal que $s_2 t \in \mathcal{L}_m(G)$.

2. $\forall v \in \Sigma^*$ com $s_1 v \in \mathcal{L}_m(G)$,

$$\exists t \in \Sigma^* \text{ tal que } s_2 t \in \mathcal{L}_m(G) \tag{7.9}$$

com $\theta(v) = \theta(t)$. A mesma argumentação pode ser aplicada após substituir todas as ocorrências de s_1 por s_2 e de t por v, e vice-versa, levando à seguinte contradição:

$$\nexists t \in \Sigma^* \text{ tal que } s_2 t \in \mathcal{L}_m(G) \tag{7.10}$$

com $\theta(v) = \theta(t)$.

Os dois passos apresentados mostram que se, e somente se, M satisfaz o OP-Teste, $\theta(\mathcal{L}_m(G))$ possui a propriedade do observador □

O OP-Teste aplicado sobre M é equivalente a um teste que define se $\theta(G)$ é uma OP-abstração, ou seja, a falha no OP-Teste implica a violação da propriedade do observador e, por outro lado, o sucesso no OP-Teste implica a projeção com a propriedade do observador.

O exemplo 7.2.1 ilustra como o OP-Teste detecta a violação da propriedade do observador.

Exemplo 7.2.1. *O autômato G e a estrutura de transição M são apresentados nas Figuras 7.2(a) e (b), respectivamente. Seja a linguagem N como definida anteriormente e $\Sigma_u = \{x\}$. Para mostrar que M não satisfaz o OP-Teste, verificamos que para algum par $s_1, s_2 \in N$, tal que $\theta(s_2) = \theta(s_1)$ existe alguma continuação que viola a propriedade $\theta(T(s_1)) = \theta(T(s_2))$.*

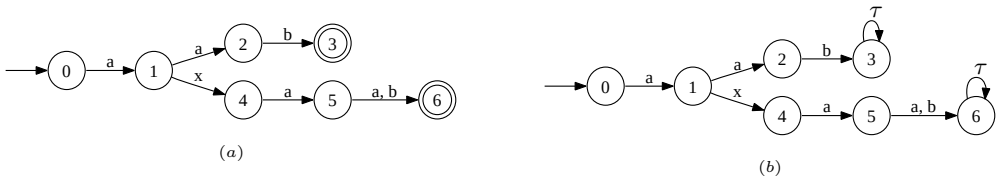


Figura 7.2: Exemplo 7.2.1: (a) G; (b) M.

O primeiro passo é definir as linguagens $\mathcal{L}(M)$ *e* N*. Nesse caso:*

$$\mathcal{L}(M) = \{aab\tau^*, axaa\tau^*, axab\tau^*\} \tag{7.11}$$

$$N = \{\epsilon, a, aa, ax, aab, axa, axaa, axab\} \tag{7.12}$$

Em seguida, deve-se obter todos os pares s_1, s_2 *que atendem à propriedade* $\theta(s_1) = \theta(s_2)$ *e verificar se* $\theta(T(s_1)) = \theta(T(s_2))$*. Os pares* (s_1, s_2) *que obedecem à propriedade são discriminados a seguir:*

$$\{(a, ax), (aa, axa), (aab, axab)\}. \tag{7.13}$$

$T(a) = \{ab\tau, xaa\tau, xab\tau\}$	$\theta(T(a)) = \{ab\tau, aa\tau\}$
$T(ax) = \{ab\tau, aa\tau\}$	$\theta(T(ax)) = \{ab\tau, aa\tau\}$
$T(aa) = \{b\tau\}$	$\theta(T(aa)) = \{b\tau\}$
$T(axa) = \{a\tau, b\tau\}$	$\theta(T(axa)) = \{a\tau, b\tau\}$
$T(aab) = \{\tau\}$	$\theta(T(aab)) = \{\tau\}$
$T(axab) = \{\tau\}$	$\theta(T(axab)) = \{\tau\}$

Pode-se perceber que o par de cadeias (aa,axa) faz com que o OP-Teste falhe, uma vez que $\theta(T(aa)) \neq \theta(T(axa))$*.*

O OP-Teste não é uma forma efetiva de se verificar a propriedade do observador. Na próxima seção, apresenta-se a estrutura de transição, chamada de OP-verificador, que traduz o OP-Teste em um problema de alcançabilidade de um estado chamado *Dead*.

[Yoo e Lafortune, 2002a] apresentam uma estrutura não-determinística chamada de *verificador* cujo objetivo é diagnosticar a ocorrência de falhas pela observação das cadeias geradas pelo sistema. Os eventos de falha são não-observáveis e, portanto, não podem ser observados diretamente. Pode ser o caso de se ter duas cadeias de mesma projeção onde uma delas possui um evento de falha sendo executado e a outra não. Portanto, o diagnóstico de falhas é feito observando cadeias que possuem a mesma projeção no conjunto de eventos observáveis e a informação de falha é atualizada à medida que as cadeias evoluem [Yoo e Lafortune, 2002a]. O fato do verificador ter a função de 'vigiar' cadeias com a mesma projeção faz com que aspectos de sua construção possam ser utilizados na construção de um verificador da propriedade do observador. Na próxima seção, apresenta-se o OP-verificador, como sendo uma estrutura de transição cujos estados representam pares de estados de M que são alcançados por cadeias de mesma projeção.

7.3 OP-Verificador

O OP-verificador é uma estrutura de transição determinística obtida a partir de M cujos estados representam pares de estados de M que possuem a mesma projeção. O verificador construído a partir de M será utilizado para auxiliar na classificação de $\theta(G)$ como sendo (ou não) uma OP-abstração, sem computar a projeção $\theta(G)$. Tem-se, pelo Teorema 7.2.1, que se M satisfaz o OP-Teste, então $\theta(\mathcal{L}_m(G))$ possui a propriedade do observador, sendo G o autômato que gerou M.

O verificador permite, portanto, que só se calcule a projeção $\theta(G)$ quando já se souber que ela será uma OP-abstração, evitando o custo computacional de se obter a projeção natural (que pode ter complexidade exponencial, no pior caso) que não possua tal característica.

Apresenta-se, nesta seção, um algoritmo de verificação da propriedade do observador que constrói o verificador, a partir de M e Σ_r e que é capaz de verificar se M satisfaz o OP-Teste.

Após uma inspeção do OP-Teste, pode-se perceber que o verificador deve ser capaz de destacar cadeias tal que tenham a mesma projeção. Para atender tais requisitos, cada estado do verificador é obtido pela associação de um par de estados de M alcançados por cadeias de mesma projeção.

Seja $M = (Q^M, \Sigma^\tau, \delta^\tau, q_0^M)$ como definido anteriormente e $En^M(q) = \{\sigma | \delta^\tau(q, \sigma)!\}$ o conjunto de eventos habilitados no estado q de M. A execução do algoritmo retorna uma estrutura $V_G = (Q, \Sigma^\tau, \delta, q_0)$, onde

- $Q \in Q^M \times Q^M \cup \{Dead\} \Rightarrow$ conjunto de estados;
- $\Sigma^\tau = \Sigma \cup \{\tau\} \Rightarrow$ alfabeto;
- $q_0 = (q_0^M, q_0^M) \Rightarrow$ estado inicial;
- $\delta : Q \times \Sigma \rightarrow Q \Rightarrow$ função de transição estendida, cuja obtenção é apresentada adiante.

A construção de V_G é feita segundo o algoritmo abaixo.

```
    Main
1   Q_{T+1} = {(0,0)}
2   Q_T = { }
3   ∀q ∈ (Q_{T+1} − Q_T)
4       Q_T = Q_T ∪ q
5       δ(q)
6       if Dead ∈ Q_{T+1}
7           quit
8       end
9   end
10  V_G = (Q_T, Σ^τ, δ, q_0)
```

Apenas o estado inicial é conhecido a princípio. Os outros estados são enumerados à medida que são alcançados a partir do estado inicial de V_G. Cada estado alcançado é adicionado ao conjunto Q_{T+1} de estados enumerados e cada estado de Q_{T+1} que é analisado pelo algoritmo é adicionado ao conjunto Q_T.

A função de transição δ é aplicada sobre cada estado de Q_{T+1} que não é de Q_T, para gerar o novo conjunto de estados alcançados a serem incluídos em Q_{T+1}. Esse procedimento é aplicado iterativamente, até que $Q_{T+1} = Q_T$. Ao final, tem-se o conjunto de estados alcançáveis em V_G. Se o estado *Dead* está entre os estados de Q, então pode-se dizer que a propriedade do observador foi violada em $\mathcal{L}_m(G)$. O algoritmo pára a execução no caso de um estado *Dead* ser alcançado.

Como o verificador apresentado em Yoo e Lafortune [2002a], o OP-verificador possui sua função de transição definida diferentemente para cada tipo de evento: σ relevante ($\sigma \in \Sigma_r$) e σ não-relevante ($\sigma \in \Sigma_u$).

A função de transição $\delta(q)$ é apresentada a seguir.

$\delta(q)$
$q_1 = q(1)$
$q_2 = q(2)$
$\forall \sigma \in En(q) = En^M(q_1) \cup En^M(q_2)$
    if $\sigma \in \Sigma_r$
        if $\delta^\tau(q_1, \sigma)!$ & $\delta^\tau(q_2, \sigma)!$
            $\delta((q_1, q_2), \sigma) = (\delta^\tau(q_1, \sigma), \delta^\tau(q_2, \sigma))$
            $Q_{T+1} = Q_{T+1} \cup \{(\delta^\tau(q_1, \sigma), \delta^\tau(q_2, \sigma))\}$
        elseif $\delta^\tau(q_1, \sigma)! \,\&\, En^M(q_2) \cap \Sigma_u = \emptyset$ ou $\delta^\tau(q_2, \sigma)! \,\&\, En^M(q_1) \cap \Sigma_u = \emptyset$
            $\delta((q_1, q_2), \sigma) = Dead$
            $Q_{T+1} = Q_{T+1} \cup \{Dead\}$
        end
    else
        if $\delta^\tau(q_1, \sigma)!$
            $\delta((q_1, q_2), \sigma) = (\delta^\tau(q_1, \sigma), q_2)$
            $Q_{T+1} = Q_{T+1} \cup \{(\delta^\tau(q_1, \sigma), q_2)\}$
        end
        if $\delta^\tau(q_2, \sigma)!$
            $\delta((q_1, q_2), \sigma) = (q_1, \delta^\tau(q_2, \sigma))$
            $Q_{T+1} = Q_{T+1} \cup \{(q_1, \delta^\tau(q_2, \sigma))\}$
        end
    end
end

O primeiro estado a ser analisado é o estado $(0, 0)$, onde $q_1 = 0$ e $q_2 = 0$. A função $\delta(q)$ é então aplicada ao estado inicial. Os eventos habilitados no estado q_1 e q_2 são analisados para

definir quais estados de V_G são alcançáveis a partir do estado atual. O mesmo procedimento é repetido até que todos os estados enumerados tenham sido analisados.

Seja o par de cadeias $s_1, s_2 \in N$ tal que $\theta(s_1) = \theta(s_2)$. Essas cadeias levam M do estado inicial aos estados q_1 e q_2, respectivamente. A propriedade do observador, reapresentada em termos das cadeias de mesma projeção e seus sufixos na linguagem marcada (Teorema 7.2.1), estabelece que $\theta(T(s_1) = \theta(T(s_2)$. Ou seja, que o conjunto de projeções dos sufixos de s_1 é igual ao conjunto de projeções dos sufixos de s_2, para todo par $s_1, s_2 \in N$ tal que $\theta(s_1) = \theta(s_2)$.

O estado $q \in Q$ é passado como parâmetro para $\delta(q)$ que retorna o conjunto de estados alcançados Q_{T+1} e a estrutura de transição entre q e os novos estados alcançados.

Este procedimento constrói o OP-verificador, cujos estados representam pares de estados de M tal que as cadeias que alcançam tais estados de V_G são cadeias de M com a mesma projeção. O estado *Dead* é alcançado quando um evento relevante σ está definido em um estado (seja q_1 do par (q_1, q_2)) de M e σ não está definido no outro estado (q_2) e também não há continuações não-relevantes em q_2. O estado *Dead* também será alcançado quando o dual ocorrer, ou seja, um evento relevante σ está definido em um estado (seja q_2 do par (q_1, q_2)) de M e σ não está definido no outro estado (q_1) e também não há continuações não-relevantes em q_1. Estas duas situações caracterizam $\theta(T(s_1) \neq \theta(T(s_2)$.

A seguir, procede-se a análise do algoritmo $\delta(q)$. Em linha gerais:

- o estado $q \in Q$ é mapeado em termos dos estados $q_1, q_2 \in Q^M$ (**Linhas 11 e 12**);
- evento σ habilitado em q (evento possível depois da cadeia s_1 ou s_2) a ser analisado (**Linha 13**);
- trata o caso de $\sigma \in En(q)$ ser relevante, ou seja, $\sigma \in \Sigma_r$ (**Linhas 14 a 21**);
- trata o caso de $\sigma \in En(q)$ ser não-relevante, ou seja, $\sigma \notin \Sigma_r$ (**Linhas 22 a 31**).

A seguir, detalha-se as linhas 14 a 21 do algoritmo, que tratam o caso em que $\sigma \in \Sigma_r$. Dois casos podem ocorrer:

- Se $\sigma \in En^M(q_1)$ e $\sigma \in En^M(q_2)$, os dois estados de q (q_1 e q_2) transitam através de σ gerando o novo estado $(\delta^\tau(q_1, \sigma), \delta^\tau(q_2, \sigma))$ que possui a característica $\theta(s_1\sigma) = \theta(s_2\sigma)$ que qualifica o par a ser adicionado ao conjunto Q_{T+1} de estados enumerados **(Linhas 15 a 17)**.
- se o algoritmo executa essa parte, significa que (i) o evento σ está habilitado em q_1 mas não está habilitado em q_2 e não existe continuação não-relevante para q_2, (ii) o evento σ está habilitado em q_2 mas não está habilitado em q_1 e não existe continuação não-relevante para q_1. Dessa forma, é possível concluir que $\theta(T(s_2)) \neq \theta(T(s_2))$. Nesse caso,

considera-se que V_G transita para um estado chamado *Dead*. A existência de um estado *Dead* em V_G implica violação da propriedade do observador, como será demonstrado nesta seção (**Linhas 18 a 20**)

Quando $\sigma \in \Sigma_u$, as linhas 22 a 31 são executadas. Quando um $\sigma \in \Sigma_u$, a ocorrência de um σ não é detectada pela projeção, e então considera-se que os estados q_1 e q_2 evoluem dessincronizadamente de tal forma a explicitar os pares de estados (q_1', q_2) e (q_1, q_2') tal que s_1, s_2, $s_1\sigma$ e $s_2\sigma$ possuem a mesma projeção. De forma semelhante, dois casos podem ocorrer:

- Se $\sigma \in En^M(q_1)$, então procede-se a evolução de q_1, mantendo-se q_2. É possível notar que ao evoluir q_1 através do evento σ, tem-se que $\theta(s_1\sigma) = \theta(s_2)$ e o estado $(\delta^\tau(q_1, \sigma), q_2)$ é alcançável em V_G e deve ser adicionado ao conjunto Q_{T+1} (**Linhas 23 a 25**)
- Se $\sigma \in En^M(q_2)$, então procede-se a evolução de q_2, mantendo-se q_1. É possível notar que ao evoluir q_2 através do evento σ, tem-se que $\theta(s_2\sigma) = \theta(s_1)$ e o estado $(q_1, \delta^\tau(q_2, \sigma))$ é alcançável em V_G e deve ser adicionado ao conjunto Q_{T+1} (**Linhas 27 a 29**)

Pode-se observar que se o estado *Dead* for alcançado a partir do estado inicial $(0, 0)$, o algoritmo pára, significando falha no OP-Teste e, consequentemente, que a projeção $\theta(\mathcal{L}_m(G))$ não possui a propriedade do observador.

Esse resultado é demonstrado no Teorema 7.3.1 a seguir.

Teorema 7.3.1. *Sejam M, V_G, θ e o OP-Teste como apresentados anteriormente. Tem-se que o estado Dead é alcançável em V_G se, e somente se, M satisfaz o OP-Teste.*

Demonstração. A estratégia usada nesta demonstração consiste de dois passos: (i) assume-se que M satisfaz o OP-Teste e o estado *Dead* é alcançável em V_G; (ii) assume-se que o estado *Dead* não é alcançável em V_G e M não satisfaz o OP-Teste. Se for possível obter contradições em ambos os casos, a demonstração estará completa.

Caso (i): assume-se que *Dead* é alcançável. Então, existe um par $s_1, s_2 \in N$, tal que $\theta(s_1) = \theta(s_2)$, que alcança o estado (q_1', q_2') em V_G, a partir do qual o estado *Dead* é alcançável. Então, pelo algoritmo, tem-se que $\exists\sigma \in \Sigma_r$ tal que:

i. $s_2\sigma \in N$;

ii. $s_1\sigma \notin N$;

iii. $s_1\alpha \notin N$, $\forall\alpha \in \Sigma_u$;

ou

iv. $s_1\sigma \in N$;

v. $s_2\sigma \notin N$;

vi. $s_2\alpha \notin N$, $\forall \alpha \in \Sigma_u$.

Toda a argumentação utilizada a partir deste ponto sobre o caso dos items *i.*, *ii.* e *iii.* é válido para o caso dos items *iv.*, *v.* e *vi.*, apenas substituindo cada ocorrência de s_2 por s_1 e cada ocorrência de t por v, e vice-versa.

Considera-se o primeiro subcaso: $s_2\sigma \in N$, $s_1\sigma \notin N$ e $s_1\alpha \notin N$, $\forall \alpha \in \Sigma_u$.

Como $s_2\sigma \in N$, pode-se dizer que $\exists w \in \Sigma^*$ tal que $s_2\sigma w\tau \in \mathcal{L}(M)$. Pela hipótese, M satisfaz o OP-Teste. Então, $\forall s_1, s_2 \in N$, tal que $\theta(s_1) = \theta(s_2)$, $\forall t \in \Sigma^*$ tal que $s_2 t\tau \in \mathcal{L}(M)$, existe $v \in \Sigma^*$ tal que $s_1 v\tau \in \mathcal{L}(M)$. Seja $t = \sigma w$, com $\sigma \in \Sigma_r$ e $w \in \Sigma^*$, então $s_2\sigma w\tau \in \mathcal{L}(M)$. Pode-se dizer que existe uma cadeia v tal que $\theta(v) = \theta(t) = \theta(\sigma w)$ e $s_1 v\tau \in \mathcal{L}(M)$. Divide-se a cadeia v em duas partes, $v = v_a v_b$, $\theta(v_a v_b) = \theta(\sigma w)$ e

$$s_1 v_a, s_1 v_a v_b \in N. \tag{7.14}$$

Dua igualdades devem ser satisfeitas:

1. $\theta(v_a) = \theta(\sigma) = \sigma$ então $v_a \in \Sigma_u^* \sigma \Sigma_u^*$;
2. $\theta(v_b) = \theta(w)$.

De 1. e a equação (7.14) tem-se que $s_1 v_a \in s_1 \Sigma_u^* \sigma \Sigma_u^* \cap N$. Então, pode-se concluir que:

$$s_1 \Sigma_u^* \sigma \Sigma_u^* \cap N \neq \emptyset. \tag{7.15}$$

No entanto, pela hipótese de que o estado *Dead* é alcançável, sabe-se que $\nexists \alpha \in \Sigma_u$ tal que $s_1\alpha \in N$ e $s_1\sigma \notin N$. Como N é prefixo-fechado, pode-se concluir que:

$$s_1 \Sigma_u^* \sigma \Sigma_u^* \cap N = \emptyset. \tag{7.16}$$

As equações (7.15) e (7.16) são contraditórias: pode-se concluir que se M satisfaz o OP-Teste então o estado *Dead* não pode ser alcançado em V_G.

Considera-se o segundo subcaso, em que o estado *Dead* é alcançado devido a: $s_1\sigma \in N$, $s_2\sigma \notin N$ e $s_2\alpha \notin N$, $\forall \alpha \in \Sigma_u$. O mesmo desenvolvimento apresentado acima pode ser aplicado para este subcaso, apenas trocando cada ocorrência de s_1 por s_2 e de t por v, e vice-versa.

Caso (**ii**): A segunda parte da demonstração consiste em considerar que M não satisfaz o OP-Teste e o estado *Dead* é alcançável em V_G. Pelo resultado do OP-Teste, tem-se que para algum par $s_1, s_2 \in N$, com $\theta(s_1) = \theta(s_2)$:

a. existe uma cadeia $t \in \Sigma^*$ tal que $s_2 t\tau \in \mathcal{L}(M)$ mas $\nexists v \in \Sigma^*$ tal que $s_1 v\tau \in \mathcal{L}(M)$ e $\theta(v) = \theta(t)$ ou

b. existe uma cadeia $v \in \Sigma^*$ tal que $s_1 v\tau \in \mathcal{L}(M)$ mas $\nexists t \in \Sigma^*$ tal que $s_2 t\tau \in \mathcal{L}(M)$ e $\theta(t) = \theta(v)$.

De forma semelhante ao caso anterior, apenas o subcaso *a.* será tratado. A argumentação utilizada pode ser extendida para o subcaso dual (b.), apenas substituindo-se cada ocorrência de s_2 por s_1 e de t por v, e vice-versa. Estas substituições levam também a uma contradição.

Seja $t = \sigma w$, tal que $s_2 t\tau = s_2\sigma w\tau \in \mathcal{L}(M)$, onde $\sigma \in \Sigma_r$ e $w \in \Sigma^*$. Não existe uma cadeia $v \in \Sigma^*$ tal que $s_1 v\tau \in \mathcal{L}(M)$ e $\theta(v) = \theta(\sigma w)$. Pode-se assumir, sem perda de generalidade, que s_1 é tal que não há continuações não-relevantes, ou seja, $s_1\alpha \notin N$, $\forall \alpha \in \Sigma_u$. Se este não for o caso, pode-se fazer $s_1' = s_1\alpha$ e s_1' continua a ser uma cadeia em N com a propriedade $\theta(s_1') = \theta(s_2)$. Como v não existe, tem-se que $s_2\sigma \in N$, $s_1 \in N$ mas

$$s_1\alpha \notin N, \forall \alpha \in \Sigma_u; \tag{7.17}$$

$$s_1\sigma \notin N. \tag{7.18}$$

Como tem-se que *Dead* não é alcançável, para todos os estados $q = (q_1, q_2)$ de V_G e $\forall \sigma \in \Sigma_r$ tal que $\sigma \in En^M(q_2)$:

1. $\sigma \in En^M(q_1)$ or
2. $\exists \alpha \in \Sigma_u$ tal que $\alpha \in En^M(q_1)$.

Sejam q_1, q_2' estados alcançados por s_1, s_2, respectivamente. Como $s_2\sigma \in N$, então $\sigma \in En^M(q_2')$. Considerando-se que o estado *Dead* não é alcançável, pode-se dizer que:

i. $\sigma \in En^M(q_1)$ e $s_1\sigma \in N$, que contradiz a equação (7.18), ou

ii. $\exists \alpha \in \Sigma_u$ tal que $\alpha \in En^M(q_1)$, implicando a existência da cadeia $s_1\alpha \in N$, o que contradiz a equação (7.17).

Pode ser observado que contradições são obtidas em ambos os casos. Então, se M não satisfaz o OP-Teste, o estado *Dead* é alcançado.

Tendo sido obtidas contradições para os dois passos da prova, pode-se concluir que M satisfaz os OP-Teste se e somente se o estado *Dead* não for alcançado a partir do estado inicial de V_G.

□

O corolário 7.3.1.1 decorre dos Teoremas 7.2.1 e 7.3.1 e é apresentado a seguir.

Corolário 7.3.1.1. *$\theta(G)$ é uma OP-abstração $\Longleftrightarrow$ estado Dead não é alcançável no OP-verificador V_G.*

Pode-se portanto resumir o procedimento de verificação da propriedade do observador em uma seqüência de três itens. Dados G e Σ_r:

- obtém-se a estrutura de transição M a partir de G, pela introdução de auto-laços rotulados com o evento τ nos estados marcados de G e a retirada da representação dos estados marcados através de círculos concêntricos;
- contrói-se o verificador V_G a partir de M;
- classifica $\theta(G)$ como uma OP-abstração se o o estado *Dead* não for alcançável em V_G e $\theta(G)$ como não sendo uma OP-abstração se o o estado *Dead* for alcançável em V_G.

A complexidade envolvida neste algoritmo é polinomial no número de estados de M, no pior caso. O pior caso ocorre quando o número de estados de V_G for igual ao número de possíveis pares de estados de M, ou seja, n^2 (n é o número de estados de M). Em geral, esse número será menor que n^2. Além disso, no caso da propriedade ser violada, o algoritmo pára a execução antes de completar a construção de V_G.

Para ilustrar a construção de V_G, apresentam-se quatro exemplos a seguir. No Exemplo 7.3.1 ilustra-se um caso em que, dados G e Σ_r, pode-se concluir que $\theta(G)$ é uma OP-abstração. Nos Exemplos 7.3.2, 7.3.3 e 7.3.4 ilustram-se três casos em que $\theta(G)$ não é OP-abstração.

Exemplo 7.3.1. *Sejam o autômato $G = (Q^G, \Sigma, \delta^\tau, q_0^G, F^G)$ e sua projeção apresentados, respectivamente, nas Figuras 7.3(a) e 7.3(b), e o conjunto de eventos não-relevantes $\Sigma_u = \Sigma - \Sigma_r = \{x, z\}$.*

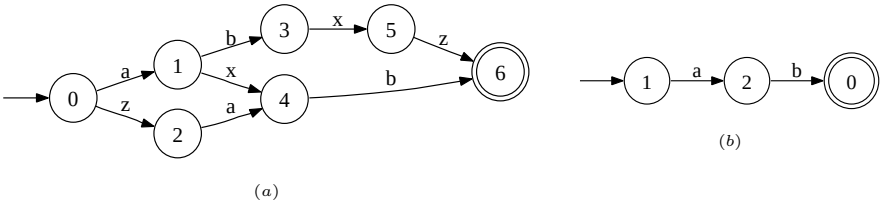


Figura 7.3: Exemplo 7.3.1: (a) G; (b) $\theta(G)$.

Para obter M, acrescenta-se o auto-laço rotulado com o evento τ nos estados marcados de G, como mostrado na Figura 7.4.

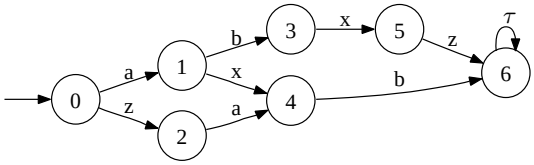


Figura 7.4: Exemplo 7.3.1: M.

A seguir, procede-se a construção de V_G a partir de M.

A execução do algoritmo leva a alcançar as seguintes transições:

$(0,0) - a - (1,1) \rightarrow$ ***Linha 15***
$(0,0) - z - (2,0) \rightarrow$ ***Linha 23***
$(0,0) - z - (0,2) \rightarrow$ ***Linha 27***
$(0,2) - a - (1,4) \rightarrow$ ***Linha 15***
$(0,2) - z - (2,2) \rightarrow$ ***Linha 23***
$(2,2) - a - (4,4) \rightarrow$ ***Linha 15***
$(4,4) - b - (6,6) \rightarrow$ ***Linha 15***
$(6,6) - \tau - (6,6) \rightarrow$ ***Linha 15***
$(1,4) - b - (3,6) \rightarrow$ ***Linha 15***
$(1,4) - x - (4,4) \rightarrow$ ***Linha 23***
$(3,6) - x - (5,6) \rightarrow$ ***Linha 23***
$(5,6) - z - (6,6) \rightarrow$ ***Linha 23***
$(2,0) - a - (4,1) \rightarrow$ ***Linha 15***
$(2,0) - z - (2,2) \rightarrow$ ***Linha 27***
$(4,1) - b - (6,3) \rightarrow$ ***Linha 15***
$(4,1) - x - (4,4) \rightarrow$ ***Linha 27***
$(6,3) - x - (6,5) \rightarrow$ ***Linha 27***
$(6,5) - z - (6,6) \rightarrow$ ***Linha 27***
$(1,1) - b - (3,3) \rightarrow$ ***Linha 15***
$(1,1) - x - (4,1) \rightarrow$ ***Linha 23***
$(1,1) - x - (1,4) \rightarrow$ ***Linha 27***
$(3,3) - x - (5,3) \rightarrow$ ***Linha 23***
$(3,3) - x - (3,5) \rightarrow$ ***Linha 27***
$(5,3) - x - (5,5) \rightarrow$ ***Linha 27***
$(5,3) - z - (6,3) \rightarrow$ ***Linha 23***
$(3,5) - x - (5,5) \rightarrow$ ***Linha 23***
$(3,5) - z - (3,6) \rightarrow$ ***Linha 27***
$(5,5) - z - (6,5) \rightarrow$ ***Linha 23***
$(5,5) - z - (5,6) \rightarrow$ ***Linha 27***

O verificador V_G resultante é apresentado na Figura 7.5.

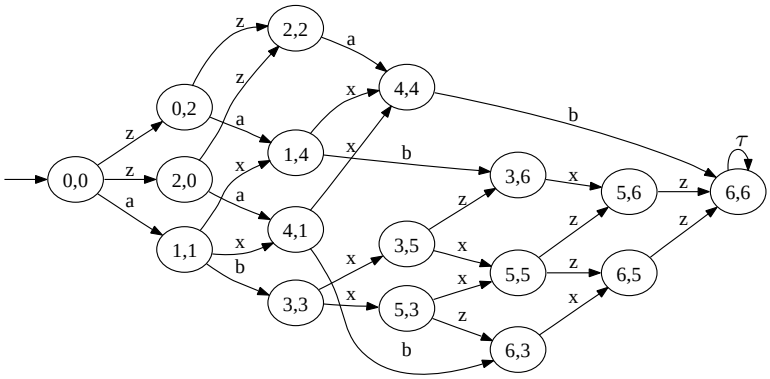


Figura 7.5: Exemplo 7.3.1 - Verificador V_G obtido utilizando o algoritmo apresentado

Na Figura 7.5 o estado Dead não é alcançável. Portanto, pode-se concluir que o $\theta(G)$ é uma OP-abstração.

No próximo exemplo ilustra-se o caso em que projeção $\theta(G)$ é classificada como não tendo a propriedade do observador devido ao evento τ.

Exemplo 7.3.2. *Considera-se o autômato G_a como uma versão modificada do autômato G usado no Exemplo 7.3.1. Os estados $1, 3, 6$ são marcados (Figura 7.6(a)) e a projeção $\theta(G_a)$ no conjunto de eventos $\Sigma_r = \{a, b\}$ é apresentado na Figura 7.6(b).*

Assim como no exemplo anterior, obtém-se M_a acrescentando auto-laços rotulados com o evento τ nos estados marcados de G_a, como mostrado na Figura 7.7.

A execução do algoritmo sobre M_a é detalhada a seguir:

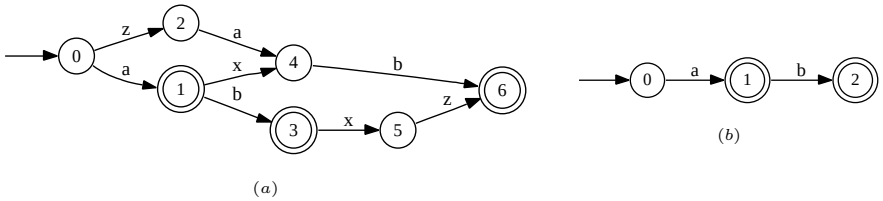


Figura 7.6: Exemplo 7.3.2: (a) G_a; (b) $\theta(G_a)$;

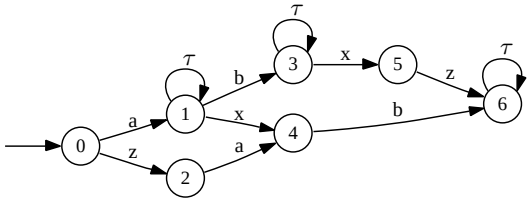


Figura 7.7: Exemplo 7.3.2: M_a.

$(0,0) - a - (1,1) \rightarrow$ ***Linha 15***
$(0,0) - z - (2,0) \rightarrow$ ***Linha 23***
$(0,0) - z - (0,2) \rightarrow$ ***Linha 27***
$(0,2) - a - (1,4) \rightarrow$ ***Linha 15***
$(0,2) - z - (2,2) \rightarrow$ ***Linha 23***
$(2,2) - a - (4,4) \rightarrow$ ***Linha 15***
$(4,4) - b - (6,6) \rightarrow$ ***Linha 15***
$(6,6) - \tau - (6,6) \rightarrow$ ***Linha 15***
$(1,4) - b - (3,6) \rightarrow$ ***Linha 15***
$(1,4) - x - (4,4) \rightarrow$ ***Linha 23***
$(1,4) - \tau - Dead \rightarrow$ ***Linha 18***
$(3,6) - x - (5,6) \rightarrow$ ***Linha 23***
$(3,6) - \tau - (3,6) \rightarrow$ ***Linha 15***
$(5,6) - z - (6,6) \rightarrow$ ***Linha 23***
$(2,0) - a - (4,1) \rightarrow$ ***Linha 15***
$(2,0) - z - (2,2) \rightarrow$ ***Linha 27***
$(4,1) - b - (6,3) \rightarrow$ ***Linha 15***
$(4,1) - x - (4,4) \rightarrow$ ***Linha 27***
$(4,1) - \tau - Dead \rightarrow$ ***Linha 18***
$(6,3) - x - (6,5) \rightarrow$ ***Linha 27***
$(6,3) - \tau - (6,3) \rightarrow$ ***Linha 15***
$(6,5) - z - (6,6) \rightarrow$ ***Linha 27***
$(1,1) - b - (3,3) \rightarrow$ ***Linha 15***
$(1,1) - x - (4,1) \rightarrow$ ***Linha 23***
$(1,1) - x - (1,4) \rightarrow$ ***Linha 27***
$(1,1) - \tau - (1,1) \rightarrow$ ***Linha 15***
$(3,3) - x - (5,3) \rightarrow$ ***Linha 23***

A construção completa de V_{G_a} (Figura 7.14) é realizada para fins de ilustração. No entanto, uma vez alcançado o estado Dead, a execução do algoritmo é interrompida.

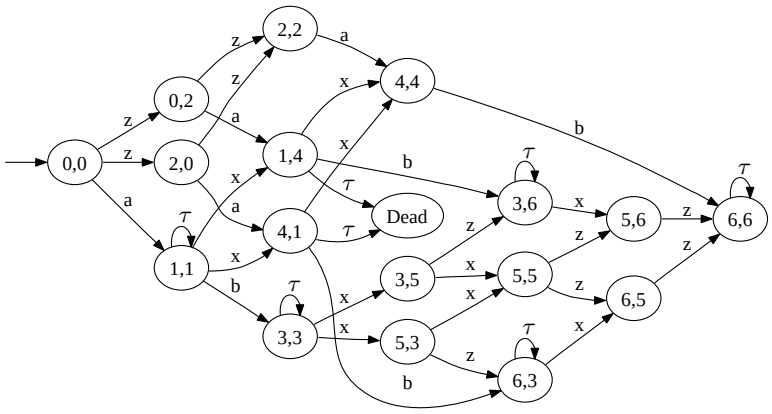


Figura 7.8: Exemplo 7.3.2: Verificador V_{G_a} completo.

O próximo exemplo ilustra outro caso em que o OP-Teste falha.

Exemplo 7.3.3. *Supõe-se a seguinte variação do autômato G, em que os estados 4 e 6 são marcados e o rótulo da transição a entre os estados 1 e 2 é substituída por b. Esse novo autômato, apresentado na Figura 7.9(a), é chamado de G_b e sua projeção é apresentada na Figura 7.9(b).*

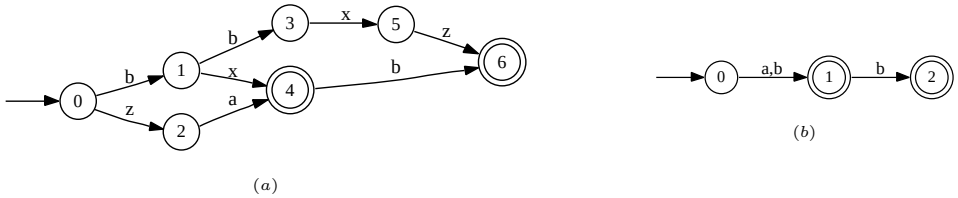


Figura 7.9: Exemplo 7.3.3: (a) G_b; (b) $\theta(G_b)$.

Assim como nos outros exemplos, acrescenta-se auto-laços rotulados com o evento τ nos estados marcados, como mostrado na Figura 7.10, obtendo-se M_b .

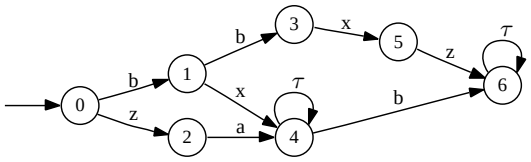


Figura 7.10: Exemplo 7.3.3: M_b.

A execução do algoritmo sobre M_b é detalhada a seguir:

$(0,0) - b - (1,1) \rightarrow$ ***Linha 15***
$(0,0) - z - (2,0) \rightarrow$ ***Linha 23***
$(0,0) - z - (0,2) \rightarrow$ ***Linha 27***
$(0,2) - b - Dead \rightarrow$ ***Linha 18***
$(0,2) - z - (2,2) \rightarrow$ ***Linha 23***
$(2,2) - a - (4,4) \rightarrow$ ***Linha 15***
$(4,4) - b - (6,6) \rightarrow$ ***Linha 15***
$(4,4) - \tau - (4,4) \rightarrow$ ***Linha 15***
$(6,6) - \tau - (6,6) \rightarrow$ ***Linha 15***
$(2,0) - b - Dead \rightarrow$ ***Linha 18***
$(2,0) - z - (2,2) \rightarrow$ ***Linha 27***
$(1,1) - b - (3,3) \rightarrow$ ***Linha 15***
$(1,1) - x - (4,1) \rightarrow$ ***Linha 23***
$(1,1) - x - (1,4) \rightarrow$ ***Linha 27***
$(1,4) - b - (3,6) \rightarrow$ ***Linha 15***
$(1,4) - x - (4,4) \rightarrow$ ***Linha 23***
$(3,6) - x - (5,6) \rightarrow$ ***Linha 23***
$(5,6) - z - (6,6) \rightarrow$ ***Linha 23***
$(3,3) - x - (5,3) \rightarrow$ ***Linha 23***
$(3,3) - x - (3,5) \rightarrow$ ***Linha 27***
$(3,5) - x - (5,5) \rightarrow$ ***Linha 23***
$(3,5) - z - (3,6) \rightarrow$ ***Linha 27***
$(5,5) - z - (6,5) \rightarrow$ ***Linha 23***
$(5,5) - z - (5,6) \rightarrow$ ***Linha 27***
$(4,1) - b - (6,3) \rightarrow$ ***Linha 15***
$(4,1) - x - (4,4) \rightarrow$ ***Linha 27***
$(5,3) - x - (5,5) \rightarrow$ ***Linha 27***
$(5,3) - z - (6,3) \rightarrow$ ***Linha 23***
$(6,3) - x - (6,5) \rightarrow$ ***Linha 27***
$(6,5) - z - (6,6) \rightarrow$ ***Linha 27***

A construção completa de V_{G_b} (Figura 7.11) é realizada para fins de ilustração. No entanto, uma vez alcançado o estado Dead, a execução do algoritmo é interrompida.

Finalmente, ilustra-se um caso em que o algoritmo para verificação da propriedade do

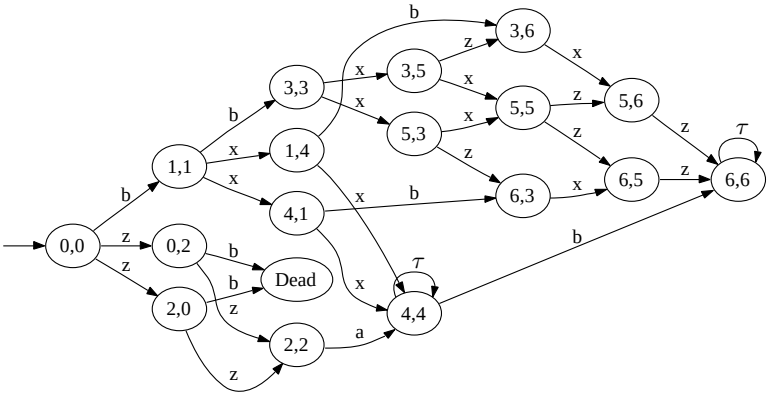


Figura 7.11: Exemplo 7.3.3: Verificador V_{G_b} completo.

observador é aplicado sobre uma estrutura um pouco mais complexa, onde há transições com eventos relevantes e não-relevantes entre um determinado par de estados.

Exemplo 7.3.4. *Considera-se o autômato G_c como uma versão modificada do autômato G usado no Exemplo 7.3.1, onde introduz-se a transição rotulada com o evento x entre os estados* 2 *e* 4*, a entre os estados* 4 *e* 6 *e os eventos a e b em auto-laços no estado* 6*. O conjunto de eventos relevantes é $\Sigma_r = \{a, b\}$, portanto $\Sigma_u = \Sigma - \Sigma_r = \{x, z\}$.*

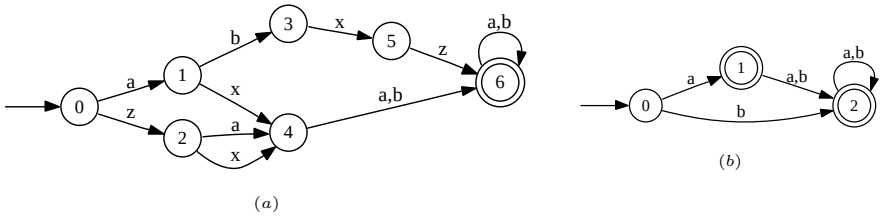


Figura 7.12: Exemplo 7.3.4: (a) G_c; (b) $\theta(G_c)$.

Para obter M_c, acrescenta-se o auto-laço rotulado com o evento τ no estado marcado 6 *de G_c, como mostrado na Figura 7.13.*

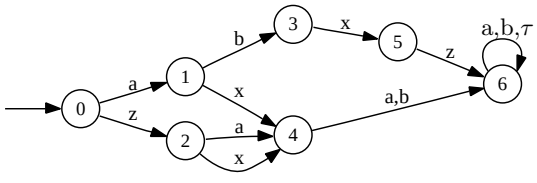


Figura 7.13: Exemplo 7.3.4: M_c.

A execução do algoritmo sobre M_c é detalhada a seguir:

$(0,0) - a - (1,1) \rightarrow$ ***Linha 15***
$(0,0) - z - (2,0) \rightarrow$ ***Linha 23***
$(0,0) - z - (0,2) \rightarrow$ ***Linha 27***
$(0,2) - a - (1,4) \rightarrow$ ***Linha 15***
$(0,2) - x - (0,4) \rightarrow$ ***Linha 27***
$(0,2) - z - (2,2) \rightarrow$ ***Linha 23***
$(2,2) - a - (4,4) \rightarrow$ ***Linha 15***
$(2,2) - x - (4,2) \rightarrow$ ***Linha 23***
$(2,2) - x - (2,4) \rightarrow$ ***Linha 27***
$(2,4) - a - (4,6) \rightarrow$ ***Linha 15***
$(2,4) - x - (4,4) \rightarrow$ ***Linha 23***
$(4,6) - a - (6,6) \rightarrow$ ***Linha 15***
$(4,6) - b - (6,6) \rightarrow$ ***Linha 15***
$(4,6) - \tau - Dead \rightarrow$ ***Linha 18***
$(6,6) - a - (6,6) \rightarrow$ ***Linha 15***
$(6,6) - b - (6,6) \rightarrow$ ***Linha 15***
$(6,6) - \tau - (6,6) \rightarrow$ ***Linha 15***
$(4,2) - a - (6,4) \rightarrow$ ***Linha 15***
$(4,2) - x - (4,4) \rightarrow$ ***Linha 27***
$(6,4) - a - (6,6) \rightarrow$ ***Linha 15***
$(6,4) - b - (6,6) \rightarrow$ ***Linha 15***
$(6,4) - \tau - Dead \rightarrow$ ***Linha 18***
$(0,4) - a - (1,6) \rightarrow$ ***Linha 15***
$(0,4) - z - (2,4) \rightarrow$ ***Linha 23***
$(1,6) - b - (3,6) \rightarrow$ ***Linha 15***
$(1,6) - x - (4,6) \rightarrow$ ***Linha 23***
$(3,6) - x - (5,6) \rightarrow$ ***Linha 23***

$(2,0) - a - (4,1) \rightarrow$ ***Linha 15***
$(2,0) - x - (4,0) \rightarrow$ ***Linha 23***
$(2,0) - z - (2,2) \rightarrow$ ***Linha 27***
$(4,0) - a - (6,1) \rightarrow$ ***Linha 15***
$(4,0) - z - (4,2) \rightarrow$ ***Linha 27***
$(1,1) - b - (3,3) \rightarrow$ ***Linha 15***
$(1,1) - x - (4,1) \rightarrow$ ***Linha 23***
$(1,1) - x - (1,4) \rightarrow$ ***Linha 27***
$(4,1) - b - (6,3) \rightarrow$ ***Linha 15***
$(4,1) - x - (4,4) \rightarrow$ ***Linha 27***
$(6,1) - b - (6,3) \rightarrow$ ***Linha 15***
$(6,1) - x - (6,4) \rightarrow$ ***Linha 27***
$(3,3) - x - (5,3) \rightarrow$ ***Linha 23***
$(3,3) - x - (3,5) \rightarrow$ ***Linha 27***
$(5,3) - x - (5,5) \rightarrow$ ***Linha 27***
$(5,3) - z - (6,3) \rightarrow$ ***Linha 23***
$(6,3) - x - (6,5) \rightarrow$ ***Linha 27***
$(1,4) - b - (3,6) \rightarrow$ ***Linha 15***
$(1,4) - x - (4,4) \rightarrow$ ***Linha 23***
$(4,4) - a - (6,6) \rightarrow$ ***Linha 15***
$(4,4) - b - (6,6) \rightarrow$ ***Linha 15***
$(3,5) - x - (5,5) \rightarrow$ ***Linha 23***
$(3,5) - z - (3,6) \rightarrow$ ***Linha 27***
$(5,5) - z - (6,5) \rightarrow$ ***Linha 23***
$(5,5) - z - (5,6) \rightarrow$ ***Linha 27***
$(6,5) - z - (6,6) \rightarrow$ ***Linha 27***
$(5,6) - z - (6,6) \rightarrow$ ***Linha 23***

A construção completa de V_{G_c} (Figura 7.14) é realizada para fins de ilustração.

Em todos os exemplos apresentados, pode-se perceber que estados duais são alcançados. Este estados duais são simétricos, ou seja, um estado $q = (q_1, q_2)$ possui o mesmo conjunto de eventos habilitados $En(q)$ que o estado dual $q' = (q_2, q_1)$ e alcança os mesmos estados q'' ou seus duais. Uma modificação no algoritmo pode ser feita de forma a tratar apenas um destes estados. Essa é mais uma sugestão a ser levada em conta no momento da implementação computacional da ferramenta.

7.4 Discussão

Neste capítulo, foi apresentado um teste para classificar uma abstração obtida pela operação de projeção natural como uma OP-abstração, sem efetivamente calcular a projeção natural.

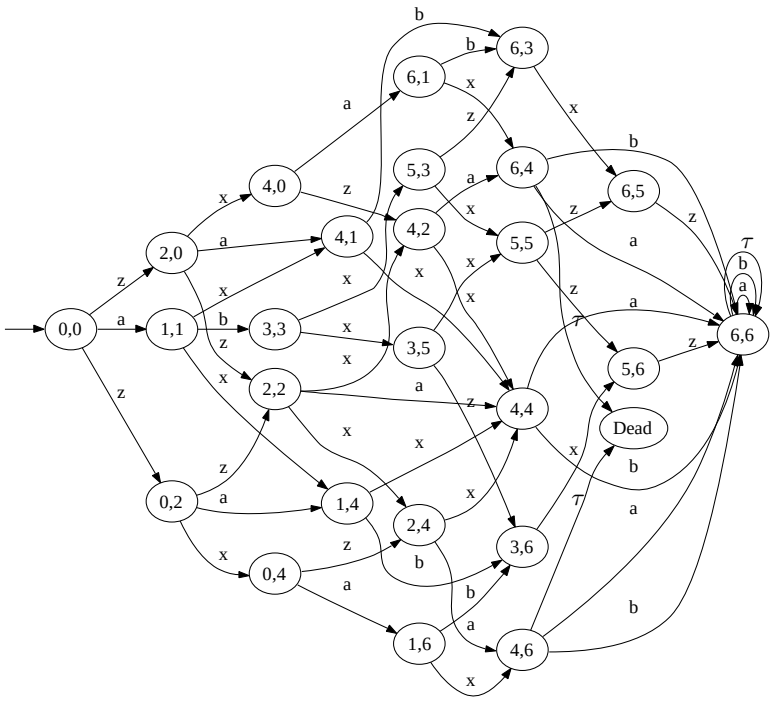


Figura 7.14: Exemplo 7.3.2: Verificador V_{G_c} completo.

A construção de uma estrutura de transição auxiliar V_G, chamada de OP-verificador, é realizada e o OP-Teste sobre G traduz-se na verificação da existência do estado *Dead* em V_G. O verificador V_G é construído a partir de M, que é obtido de G, e o conjunto de eventos relevantes. Conclui-se sobre $\theta(G)$ ser OP-abstração, sem efetivamente calcular a projeção $\theta(G)$. A alcançabilidade do estado *Dead* implica a violação da propriedade do observador.

A complexidade envolvida na construção de V_G é polinomial no número de estados de M, e portanto de G, no pior caso. No caso de a condição do observador ser violada, o algoritmo pára sua execução e o verificador identifica que $\theta(G)$ não será uma OP-abstração sem completar a construção de V_G, reduzindo ainda mais a complexidade. No caso de $\theta(G)$ ser uma OP-abstração, o algoritmo completa a construção de V_G. No pior caso, V_G terá n^2 estados, onde n é o número de estados de M (igual ao número de estados de G). Em geral, o número de estados de V_G será menor que n^2.

Portanto, utilizando o método apresentado neste capítulo, com um algoritmo polinomial verifica-se se a projeção de um dado G no conjunto Σ_r, possui a propriedade do observador sem computar a projeção. Note que o procedimento de obtenção de projeções naturais pode ter complexidade exponencial. Usando o algoritmo apresentado, a projeção natural de G não precisa ser calculada para que se conclua sobre ela ter ou não a propriedade do observador. Ao evitar o cálculo da projeção para casos em que não se tem a propriedade verificada, obtém-se uma redução da complexidade dos procedimentos intermediários de busca da OP-abstração.

Ainda não há um procedimento automático para auxiliar na definição do conjunto de eventos a ser considerado relevante na projeção.

Capítulo 8

Conclusões e Perspectivas

Esta tese trata da detecção eficiente de conflito entre supervisores. Como mencionado no Capítulo 1 e mais especificamente no Capítulo 4, o conflito é um problema inerente quando se tem diversos supervisores atuando simultaneamente sobre a planta. Para garantir um sistema livre de conflito, diferentes opções estão disponíveis. A primeira delas consiste no uso do controle monolítico (centralizado), onde se tem apenas um supervisor responsável por garantir o comportamento controlado do sistema, determinado pelas especificações. Essa solução não é adequada para sistemas de grande porte devido ao problema da explosão do espaço de estados que pode ocorrer ainda no procedimento de modelagem do sistema. Com o objetivo de evitar o problema da explosão de estados, deve-se ser capaz de realizar o controle do sistema sem realizar a composição de suas subplantas. Considerando que a maioria de sistemas práticos são formados de subsistemas e que a restrição do comportamento da planta global consiste, em geral, na coordenação de subconjuntos destes subsistemas, o uso de abordagens não-centralizadas torna-se natural. Desta forma, o problema da explosão do espaço de estados dos procedimentos de modelagem e síntese de supervisores fica reduzido. No entanto, há a necessidade de verificar a existência de conflito entre esses supervisores.

Outra possibilidade consiste em obter supervisores não-conflitantes por construção. Neste caso, a condição de ser não-conflitante é colocada como condição durante o procedimento de obtenção dos supervisores. Este não é o enfoque dado ao problema nesta tese. Uma outra possibilidade é modelar os subsistemas utilizando linguagens prefixo-fechadas. Esta abordagem também não é adequada para modelar os sistemas reais, pois ao invés de tratar o problema do conflito, mascara a sua existência.

Nesta tese, analisa-se o problema do conflito sob o enfoque de que há diversos supervisores. Portanto, o conflito pode existir. Para lidar com o problema, deve-se ser capaz de detectar sua existência de forma eficiente. Uma vez detectado, o conflito deve ser tratado, seja através do uso de coordenadores, seja através de uma mudança no procedimento de síntese utilizado. O ponto abordado nesta tese é a detecção eficiente do conflito.

A verificação da existência de conflito pode ser realizada através de um teste onde todos os supervisores são compostos. Se o autômato obtido (representando a linguagem do sistema

global sob controle) for bloqueante, tem-se que os supervisores são conflitantes. Sabe-se que o número de estados do autômato resultante do teste é exponencial no número de componentes, no pior caso. Desta forma, este teste torna-se inviável para sistemas de grande porte.

Sendo a existência do conflito uma característica global do sistema, não há como decompor o teste em testes menores a serem aplicados sobre subconjuntos dos supervisores. A solução vislumbrada foi, portanto, obter abstrações dos supervisores e aplicar o teste sobre tais abstrações. O desafio tornou-se a obtenção de abstrações tais que: *i* as propriedades relacionadas ao conflito fossem mantidas e a informação irrelevante fosse descartada; *ii* as abstrações tivessem espaço de estados menor que o supervisor, de forma a obter redução na verificação da propriedade.

Para tanto, haveria de se determinar qual o conjunto de eventos a ser considerado de forma a obter a abstração com tais características. O estudo de vários exemplos indicou que, em geral, os eventos não-compartilhados por nenhum par de supervisores não são relevantes para o conflito, ou seja, podem ser apagados da linguagem. Esta suspeita foi formalizada e demonstrada no Capítulo 5. Todos os resultados são apresentados para o caso simples, em que se tem apenas 2 supervisores e em seguida estendidos para o caso de múltiplos supervisores.

O primeiro resultado da tese, apresentado no Capítulo 5, consiste em um novo teste de não-conflito, aplicado sobre abstrações dos supervisores e cujo resultado é o mesmo do teste aplicado sobre os supervisores originais (Teoremas 5.3.1 e 5.3.3). Além disso, há garantias que o teste sobre as abstrações não terá nunca espaço de estados maior que o teste original (Teoremas 5.3.2 e 5.3.4). Apresenta-se ainda um procedimento, chamado de **W-solução**, para aplicação dos resultados deste capítulo. Neste procedimento, define-se o conjunto de eventos relevantes como sendo composto de todos os eventos compartilhados por qualquer par de supervisores. Este conjunto é, então, estendido de tal forma que se obtenham OP-abstrações pela projeção da linguagem implementada neste conjunto de eventos relevantes.

As linguagens testadas neste capítulo são referidas como sendo as linguagens implementadas pelos supervisores. No entanto, as propriedades das abstrações não têm relação com a forma de obtenção das linguagens. Portanto, os resultados do Capítulo 5 são aplicáveis a qualquer conjunto de linguagens a partir do qual se queira detectar conflito, sejam elas implementadas por supervisores ou não.

Ao se estabelecer que as linguagens testadas são aquelas implementadas por supervisores pode-se começar a analisar aspectos estruturais destas linguagens. Se os supervisores são obtidos pela abordagem modular local (revista no Capítulo 4) sabe-se que a representação do sistema é realizada de tal forma que suas subplantas são assíncronas (representação por sistema produto). Se o supervisor é obtido pelo cálculo do $Sup\mathcal{C}$, então sabe-se que ele implementa uma sublinguagem da linguagem da planta.

Quando um evento é compartilhado por um par de supervisores, significa que a planta que o contém é compartilhada por aquele par de supervisores. Percebeu-se, no entanto, que o fato de um evento ser compartilhado por mais de um supervisor não implica, necessaria-

mente, que ele é relevante para o conflito. Isso ocorre quando o evento não é desabilitado por nenhum dos supervisores, significando que sempre que o evento é possível na planta, ele também é possível no sistema controlado, se os supervisores implementarem os comportamentos expressos nas especificações. Além disso, tal evento não deve definir a desabilitação de nenhum outro evento, ou seja, ele não deve indicar que depois de sua ocorrência, um outro evento é desabilitado. Os eventos que não aparecem nas especificações genéricas (caso sejam controláveis) são aqueles que além de não serem desabilitados pelos supervisores, não definem a desabilitação de nenhum outro evento. Para o caso de especificações não-controláveis, deve-se obter uma especificação controlável fictícia, como será mencionado adiante. Dessa forma, um evento pode ser compartilhado e não ser necessário para caracterizar a existência de conflito, podendo portanto ser apagado nas abstrações. Explorando estas condições estruturais dos supervisores apresentou-se, no Capítulo 6, outro conjunto de condições suficientes para as abstrações para que o teste de não-conflito possa ser aplicado sobre o mesmo. Estas contribuições foram formalizadas nos Teoremas 6.3.1 e 6.3.2, apresentados no Capítulo 6.

Ainda neste capítulo, apresentam-se dois procedimentos para utilização dos resultados apresentados. No primeiro procedimento, apresenta-se a **T-solução**, que consiste em obter abstrações a partir de um conjunto de eventos relevantes composto por todos os eventos presentes nas especificações genéricas (que devem ser controláveis) do sistema de controle. Se estas especificações não forem controláveis, devem-se obter especificações fictícias (dadas pela aplicação de rotinas de redução de supervisores, por exemplo) de forma a determinar os eventos relevantes. A partir do conjunto de eventos relevantes, são obtidas OP-abstrações pela extensão, se necessário, deste conjunto. As duas soluções podem ser combinadas, gerando a **W** $\circ$ **T-solução**. Nesta solução, utiliza-se uma combinação das duas soluções previamente apresentadas, onde primeiro aplica-se a **T-solução** e em seguida a **W-solução** é aplicada sobre o conjunto de abstrações obtido pela primeira parte do procedimento.

Em geral, os três procedimentos apresentados são incomparáveis, ou seja, não há um método que gere abstrações menores sempre, sendo que essa comparação depende do sistema. No entanto é fato que: (1) a **W-solução** pode gerar abstrações menores que a **T-solução** e a **W** $\circ$ **T-solução**; (2) a **T-solução** e a **W** $\circ$ **T-solução** podem gerar abstrações menores que a **W-solução**; (3) a **W** $\circ$ **T-solução** nunca gera abstrações maiores que a **T-solução**.

Pode-se perceber que a propriedade do observador é utilizada nos dois conjuntos de resultados (apresentados no Capítulo 5 e Capítulo 6), ou seja, que esta propriedade das abstrações é utilizada indepedentemente de se utilizarem as condições estruturais ou não. Em visita ao grupo do Professor Knut Åkesson, da *Chalmers University of Technology* na cidade de Gotemburgo, na Suécia, realizaram-se discussões com os pesquisadores Hugo Flordal (doutorando do grupo) e Robi Malik (Professor da *University of Waikato*- Waikato - Nova Zelândia), que também visitava o grupo no mesmo período. Estes pesquisadores possuem resultados para detecção eficiente de conflito entre linguagens, usando o formalismo de álgebra de processos. O objetivo desta visita foi discutir a relação entre os conceitos utilizados por cada um, para obtenção dos resultados na detecção do conflito. Entre os resultados desta discussão, destaca-se a conclusão de que a propriedade do observador é a propriedade que gera a melhor

abstração possível que permite concluir sobre a existência de conflitos (usando abstrações obtidas pela aplicação da projeção natural).

A verificação da propriedade do observador é realizada inúmeras vezes durante os procedimentos apresentados na tese, até que se consiga obter OP-abstrações. A obtenção de uma projeção que não possua a propriedade do observador pode ter, em geral, complexidade exponencial [Wong, 1998]. Então, torna-se interessante evitar o cálculo da projeção a menos que se saiba que tal projeção será uma OP-abstração. Há alguns procedimentos para obtenção de projeções com a propriedade do observador, mas nenhum deles está implementado nas ferramentas de controle supervisório disponíveis. Tendo-se tomado conhecimento de uma estrutura em árvore, chamada de verificador de diagnosticabilidade [Yoo e Lafortune, 2002a], percebeu-se que a idéia básica utilizada neste verificador poderia ser também aplicada para verificação da propriedade do observador. A idéia principal consiste em monitorar cadeias de mesma projeção pela evolução dessincronizada através de eventos não-observáveis, avaliando, em paralelo, a ocorrência de faltas no sistema. Essa idéia foi adaptada e um verificador da propriedade do observador, chamado de OP-verificador, foi obtido e é apresentado no Capítulo 7. O OP-verificador constrói um autômato que possui o estado *Dead* alcançável, no caso em que a propriedade do observador é violada. Se este estado é alcançado, o algoritmo pára a execução e retorna falha na verificação da propriedade. Este algoritmo é apresentado e demonstrado correto no Capítulo 7. Exemplos ilustrativos de sua aplicação são apresentados.

Apesar do algoritmo não ter sido efetivamente implementado na ferramenta Grail para Controle Supervisório, desenvolvida pelo grupo do Professor José Cury, há a expectativa de que essa implementação se concretize.

Um passo na direção da efetiva aplicação de técnicas formais a problemas reais foi dado com este trabalho. No entanto, muitas questões permanecem em aberto. A seguir são elencadas algumas delas, levantadas no desenvolvimento da tese.

Uma vez detectada a existência de conflito, deve-se ser capaz de resolvê-lo. A resolução do conflito é realizada sobre o sistema global, ou seja, não há o conhecimento da localização do conflito de forma a tratá-lo localmente. Acredita-se que é possível obter resultados neste sentido, pelo uso da abordagem multitarefa. Esta abordagem permite diferenciar classes de tarefas e introduz duas gradações de não-conflito, o não-conflito fraco e o não-conflito forte. A introdução destes novos conceitos permite analisar aspectos do conflito entre linguagens ainda não explorados. Acredita-se que o desenvolvimento de pesquisas nesta área significa mais um passo na direção de tornar a teoria do controle supervisório viável para aplicação em sistemas reais, pois será possível tomar medidas locais para solução do conflito.

Especificamente sobre os resultados obtidos, algumas sugestões de continuidade do trabalho são discutidas.

i. A renomeação de ocorrências de eventos locais (não compartilhados entre supervisores) que estejam violando a propriedade do observador pode ser utilizada. Para tanto, deve-se ser capaz de identificar tais eventos e em seguida renomeá-los de tal forma que estas

ocorrências sejam mantidas nas abstrações enquanto as ocorrência do evento que não violam a propriedade podem ser apagadas no procedimento de projeção. Esta extensão trará reduções ainda maiores para o procedimento de verificação de não-conflito.

ii. Acredita-se ser possível obter outros conjuntos de eventos relevantes de forma a obter abstrações com as características desejadas para detecção do conflito através da combinação dos dois conjuntos de eventos relevantes apresentados na tese.

iii. Na abordagem multitarefa deve-se verificar a existência de conflito (forte e fraco) entre os supervisores. Os resultados apresentados nesta tese podem, portanto, ser estendidos para tratar destas duas classes de conflito. Para tanto, deve-se redefinir a propriedade do observador para o caso de se ter múltiplas tarefas e adequar as demonstrações para esta nova realidade.

iv. O procedimento de obtenção de OP-abstrações passa pela extensão do conjunto de eventos relevantes de forma a garantir a propriedade do observador na projeção. A forma como esta extensão do conjunto de eventos é feita, é baseada em heurísticas. Acredita-se ser possível obter um método que indique caminhos para esta extensão.

Todas estas questões ficam em aberto, para pesquisa futura.

Referências Bibliográficas

Abdelwahed, S. *Interacting Discrete Event Systems: Modelling, Verification, and Supervisory Control*. PhD thesis, University of Toronto, Abdelwahed, 2002.

Abdelwahed, S. e Wonham, W. "Supervisory Control of Interacting Discrete-Event Systems". In *Proceedings of the 41st IEEE Conference on Decision and Control, CDC'02*, pp. 1175–1180, Las Vegas, USA, Dec. 2002.

Abdelwahed, S. e Wonham, W. "Blocking Detection in Discrete Event Systems". In *Proceedings of the 2003 American Control Conference, ACC'03*, volume 2, pp. 1673– 1678, June 2003a.

Abdelwahed, S. e Wonham, W. "Interacting DES: Modelling and Analysis". In *IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pp. 4222– 4229, Oct. 2003b.

Asarin, E., Maler, O., e Pnueli, A. *Symbolic Controller Synthesis for Discrete and Timed Systems*, volume 999, pp. 1–20. Lecture Notes in Computer Science, Hybrid Systems II edition, 1995. Editors P. Antsaklis, W. Kohn, A. Nerode and S. Sastry.

Baccelli, F., Cohen, G., Olsder, G. J., e Quadrat, J.-P. *Synchronization and Linearity*. Wiley, 1992.

Balemi, S. *Control of Discrete Event Systems: Theory and Application*. PhD thesis, Swiss Federal Institute of Technology, Zurich, 1992.

Balemi, S., Hoffmann, G., Gyugyi, P., Wong-Toi, H., e Franklin, G. "Supervisory Control of a Thermal Multiprocessor". *IEEE Transactions on Automatic Control*, 38(7):1040–1059, July 1993.

Barrett, G. e Lafortune, S. "A Novel Framework for Decentralized Supervisory Control with Communication". In *Proceedings of 1998 IEEE International Conference on Systems, Man, and Cybernetics Conference*, pp. 617– 620, Oct. 1998.

Bherer, H., Desharnais, J., e St-Denis, R. "Synthesis of State Feedback Controllers for Parameterized Discrete Event Systems Under Partial Observation". In *Proceedings of 44th IEEE Conference on Decision and Control, CDC'05*, Seville, Spain, December 2005.

Bherer, H., Desharnais, J., e St-Denis, R. "On the Reachability and Nonblocking Properties for Parameterized Discrete Event Systems". In *Proceedings of 8th International Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, Michigan, USA, July 2006.

Brandin, B., Malik, R., e Dietrich, P. "Incremental System Verification and Synthesis of Minimally Restrictive Behaviours". In *Proceedings of the American Control Conference, ACC'00*, pp. 4056–4061, Chicago, Illinois, June 2000.

Brandin, B., Malik, R., e Malik, P. "Incremental Verification and Synthesis of Discrete-Event Systems Guided by Counter Examples". *IEEE Transactions on Control Systems Technology*, 12(3):387–401, May 2004.

Bryant, R. "Graph-Based Algorithms for Boolean Function Manipulations". *IEEE Transactions on Computers*, 35(8), 1986.

Byröd, M., Lennartson, B., Vahidi, A., e Åkesson, K. "Efficient Reachability Analysis of Modular Discrete-Event Systems using Bynary Decision Diagrams". In *Proceedings of 8th Internation Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, MI, USA, July 2006.

Cassandras, C. e Lafortune, S. *Introdution to Discrete Event Systems.* Kluwer Academic Publishers, 1999.

Chen, E. e Lafortune, S. "On Nonconflicting Languages that Arise in Supervisory Control of Discrete Event Systems". *Systems & Control Letters*, 17:105–113, 1991.

Chen, Y.-L. e Lafortune, S. "Design of Nonblocking Modular Supervisors Using Event Priority Functions". *IEEE Transactions on Automatic Control*, 45(3):432–452, March 2000.

Chen, Y. e Lin, F. "Modeling of Discrete Event Systems Using Finite State Machines with Parameters". In *Proceedings of the 2000 IEEE International Conference on Control Applications*, pp. 941–946, Anchorage, Alaska, USA, Sep. 2000.

Chen, Y. e Lin, F. "Hierarchical Modeling and Abstraction of Discrete Event Systems Using Finite State Machines with Parameters". In *Proceedings of the 40th IEEE Conference on Decision and Control, CDC'01*, pp. 4110–4115, Orlando, Florida, USA, Dec. 2001a.

Chen, Y. e Lin, F. "Safety Control of Discrete Event Systems Using Finite State Machines with Parameters". In *Proceedings of the American Control Conference, ACC'01*, pp. 975–980, Arlington, VA, June 2001b.

Cunha, A. e Cury, J. "Hierarchical Supervisory Control Based on Discrete Event Systems with Flexible Marking". *IEEE Transactions on Automatic Control*, p. aceito, 2007.

Cury, J. "Teoria de Controle Supervisório de Sistemas a Eventos Discretos". Minicurso no V Simpósio Brasileiro de Automação Inteligente, Nov. 2001. `http://www.das.ufsc.br/~cury/cursos/apostila.pdf`.

Cury, J., Torrico, C., e Cunha, A. "Supervisory Control of Discrete Event Systems with Flexible Marking". *European Journal Of Control*, 10(1):47–60, 2004.

Cury, J., Torrico, C., e da Cunha, A. "A New Approach for Supervisory Control of Discrete Event Systems". In *Proceedings of European Control Conference, ECC'01*, Porto, Portugal, 2001.

de Queiroz, M. "Controle Supervisório Modular de Sistemas de Grande Porte". Master's thesis, Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil, 2000.

de Queiroz, M. e Cury, J. "Modular Supervisory Control of Large Scale Discrete Event Systems". In *Proceedings of the 5th Workshop on Discrete Event Systems, WODES'00*, volume 1, pp. 103–110., Ghent, Belgium, Aug. 2000.

de Queiroz, M. e Cury, J. "Controle Supervisório Modular de Sistemas de Manufatura". *Revista Controle & Automação*, 13(2):115–125, Aug. 2002a.

de Queiroz, M. e Cury, J. "Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell". In *Proceedings of the 6th Workshop on Discrete Event Systems, WODES'02*, Zaragoza, Sep. 2002b.

de Queiroz, M., Cury, J., e Wonham, W. "Multitasking Supervisory Control of Discrete-Event Systems". *Discrete Event Dynamic Systems: Theory and Applications*, 15:375–395, 2005.

Eyzell, J. *Aspectos de Síntese de Supervisores para Sistemas a Eventos Discretos e Sistemas Híbridos*. PhD thesis, Universidade Federal de Santa Catarina, UFSC, Florianópolis, SC, Brasil, 2000.

Eyzell, J. e Cury, J. "Exploiting Symmetry in the Synthesis of Supervisors for Discrete Event Systems". *IEEE Transactions on Automatic Control*, 46(9):1500–1505, Sep. 2001.

Feng, L. "On the Computation of Natural Observers in Discrete-Event Systems". Technical report, Systems Control Group Report, University of Toronto, Jan. 2006.

Feng, L. e Wonham, W. "Computationally Efficient Supervisor Design: Abstraction and Modularity". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 3–8, Ann Arbor, MI, USA, July 2006a.

Feng, L. e Wonham, W. "TCT: A Computation Tool for Supervisory Control Synthesis". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 388–389, Ann Arbor, MI, USA, July 2006b.

Flordal, H. e Malik, R. "Nonblocking Verification Using Conflict Equivalence". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 100–106, Ann Arbor, MI, USA, July 2006.

Flordal, H., Malik, R., e Pena, P. "Conflicts and Observers". In *Aceito para 1st IFAC Workshop on Dependable Control of Discrete Systems*, Cachan-Paris, França, June 2007.

Gaudin, B. "Efficient Solution for the State Avoidance Control Problem on Concurrent Systems using a Disjunctive Architecture". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 70–75, Ann Arbor, MI, USA, July 2006.

Gaudin, B. e Marchand, H. "Modular Supervisory Control of Asynchronous and Hierarchical Finite State Machines". In *Proceedings of European Control Conference, ECC'03*, Sep. 2003a.

Gaudin, B. e Marchand, H. "Supervisory Control of Product and Hierarchical Discrete Event Systems". In *Versão estendida do artigo em Proceedings of European Control Conference*, 2003b.

Gaudin, B. e Marchand, H. "Supervisory Control of Structured Discrete Event Systems". Publication interne, Institut de Recherche en Informatique et Systèmes Aléatoires, Campus Universitaire de Beaulieu 35042 Rennes Cedex-France, Nov. 2003c.

Gaudin, B. e Marchand, H. "Modular Supervisory Control of a Class of Concurrent Discrete Event Systems". In *Proceedings of the 7th International Workshop on Discrete Event Systems, WODES'04*, pp. 181–186, Sep. 2004.

Gaudin, B. e Marchand, H. "Efficient Computation of Supervisors for Loosely Synchronous Discrete Event Systems: A State Based Approach". In *Proceedings of 16th Triennial IFAC World Congress, IFAC'05*, Prague, Czech Republic, July 2005.

Gohari, P. e Wonham, W. "On the Complexity of Supervisory Control Design in the RW Framework". *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 30(5):643–652, Oct. 2000.

Gunnarsson, J., Plantin, J., e Germundsson, R. "Verification of a Large Discrete System Using Algebraic Methods". In *Proceedings of the 3rd Workshop on Discrete Event Systems, WODES'96*, 1996.

Heymann, M. "Concurrency and Discrete Event Control". *IEEE Control Systems Magazine*, pp. 103–112, June 1990.

Hill, R. e Tilbury, D. "Modular Supervisory Control of Discrete Event Systems with Abstraction and Incremental Hierarchical Construction". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 399–406, Ann Arbor, MI, USA, July 2006.

Hoffman, G. e Wong-Toi, H. "Symbolic Synthesis of Supervisory Controllers". In *Proceedings of the 1992 American Control Conference, ACC'92*, Chicago, IL, USA, June 1992.

Hopcroft, J., Motwani, R., e Ullman, J. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, second edition edition, 2001.

Huang, Y., Rudie, K., e Lin, F. "Decentralized Control of Discrete-Event Systems when Supervisors Observe Particular Event Occurrences". In *Proceedings of the 2006 American Control Conference, ACC'06*, Minneapolis, MN, USA, June 2006.

Çinclair, E. *Introduction to Stochastic Processes*. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1975.

Jiang, S., Chandra, V., e Kumar, R. "Decentralized Control of Discrete Event Systems with Multiple Local Specifications". In *Proceedings of the 2001 American Control Conference, ACC'01*, number 2, pp. 959–964, Arlington, USA, June 2001.

Åkesson, K., Flordal, H., e Fabian, M. "Exploiting Modularity for Synthesis and Verification of Supervisors". In *Proceedings of 15th Triennial World Congress of the IFAC*, Barcelona, Spain, July 2002.

Kleinrock, L. *Queueing Systems, Volume I: Theory*. John Wiley & Sons, Canada, 1975.

Komenda, J. e van Schuppen, J. "Supremal Sublanguages of General Specification Languages Arising in Modular Control of Discrete-Event Systems". In *Proceedings of the 44th IEEE Conference on Decision and Control CDC'05, and the European Control Conference ECC'05*, pp. 2775–2780, Seville, Dec. 2005.

Komenda, J. e van Schuppen, J. "Optimal Solutions of Modular Supervisory Control Problems With Indecomposable Specification Languages". In *Proceedings of the 8th Internation Workshop on Discrete Event Systems, WODES'06*, pp. 143–148, Ann Arbor, MI, USA, July 2006.

Komenda, J., van Schuppen, J., Gaudin, B., e Marchand, H. "Modular Supervisory Control with General Indecomposable Specification Languages". In *Proceedings of the 44th IEEE Conference on Decision and Control CDC'05, and the European Control Conference ECC'05*, pp. 3474–3479, Seville, Dec. 2005.

Krishnan, P. "Decomposing Controllers into Non-Conflicting Distributed Controllers.". In *International Colloquium on Theoretical Aspects of Computing, ICTAC'04*, pp. 511–526, Guiyang, China, 2004.

Kumar, R. e Takai, S. "Inference-based Ambiguity Management in Decentralized Decision-Making: Decentralized Control of Discrete Event Systems". In *Proceedings of 44th IEEE Conference on Decision and Control, CDC'05*, Seville, Spain, December 2005.

Lawesson, D., Nilsson, U., e Klein, I. "Fault Isolation in Discrete Event Systems by Observational Abstraction". In *Proceedings of the 42nd IEEE Conference on Decision and Control, CDC'03*, Maui, Hawaii, Dec. 2003.

Leduc, R., Brandin, B., Lawford, M., e Wonham, W. "Hierarchical Interace-Based Supervisory Control - Part I: Serial Case". *IEEE Transactions on Automatic Control*, 50(9): 1322–1335, Sep. 2005a.

Leduc, R., Lawford, M., e Dai, P. "Hierarchical Interace-Based Supervisory Control of a Flexible Manufacturing System". *IEEE Transactions on Control Systems Technology*, 14 (4):654–668, July 2006.

Leduc, R., Lawford, M., e Wonham, W. "Hierarchical Interace-Based Supervisory Control - Part II: Parallel Case". *IEEE Transactions on Automatic Control*, 50(9):1336–1348, Sep. 2005b.

Lee, S. e Wong, K. "Decentralised Control of Concurrent Discret-Event Systems with Non-Prefix Closed Local Specification". In *Proceedings of the 36th IEEE Conference on Decision and Control, CDC'97*, San Diego, CA, USA, Dec. 1997.

Lin, F. e Wonham, W. "Decentralized Control and Coordination of Discrete-Event Systems". In *Proceedings of the 27th IEEE Conference on Decision and Control, CDC'88*, pp. 1125–1130, Austin, Texas - USA, Dec. 1988.

Lin, F. e Wonham, W. "Decentralized Control and Coordination of Discrete-Event Systems with Partial Observation". *IEEE Transactions on Automatic Control*, 35(12):1330–1337, Dec. 1990.

Lin, F. e Wonham, W. "Verification of Nonblocking in Decentralized Supervision". *Control Theory and Advanced Technology*, 7(1):19–29, Mar. 1991.

Ma, C. *Nonblocking Supervisory Control of State Tree Structures*. PhD thesis, University of Toronto, Toronto, Canada, 2004.

Ma, C. e Wonham, W. "Control of State Tree Structures". In *Proceedings of 11th Mediterranean Conference on Control and Automation*, Rhodes, Greece, June 2003.

Ma, C. e Wonham, W. "A Symbolic Approach to the Supervision of State Tree Structures". In *Proceedings of 13th Mediterranean Conference on Control and Automation*, Limassol, Cyprus, June 2005.

Ma, C. e Wonham, W. "Nonblocking Supervisory Control of State Tree Structures". *IEEE Transactions on Automatic Control*, 51(5):782–793, May. 2006.

Malik, R. "On the Set of Certain Conflicts of a Given Language". In *Proceedings of the 7th Workshop on Discrete Event Systems, WODES'04*, Sep. 2004.

Malik, R., Streader, D., e Reeves, S. "Fair Testing Revisited: A Process-Algebraic Characterisation of Conflicts". In *Proceedings of 2nd International Symposium on Automated Technology for Verification and Analysis, ATVA'04*, volume 3299, pp. 120–134, Taipei, Taiwan, 2004. Springer LNCS.

Minhas, R. *Complexity Reduction in Discrete Event Systems*. PhD thesis, University of Toronto, Toronto, Canada, 2002.

Murata, T. "Petri nets: Properties, Analysis and Applications". In *Proceedings of IEEE*, volume 77, pp. 541–580, 1989.

Oliveira, C. e Cury, J. "Supervisory Control Problem for Parameterized and Non-Regular Discrete Event Systems". *Submetido para IEEE Transactions on Automatic Control*, 2006.

Oliveira, C., Cury, J., e Kaestner, C. "Discrete Event Systems with Guards". In *Proceedings of the 11th IFAC Symposium on Information Control Problems in Manufacturing, INCOM'2004,*, pp. 90–95, Salvador, Brasil, April 2004.

Pena, P., Cury, J., e Lafortune, S. "Detecção de Conflito na Síntese Modular de Supervisores: Uma Abordagem Baseada em Abstrações". In *Anais do XVI Congresso Brasileiro de Automática, CBA'06*, pp. 977–982, Salvador, Brasil, Oct. 2006a.

Pena, P., Cury, J., e Lafortune, S. "New Results on Testing Modularity of Local Supervisors using Abstractions". In *11th IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 950–956, Sep. 2006b.

Pena, P., Cury, J., e Lafortune, S. "Testing Modularity of Local Supervisors: An Approach Based on Abstractions". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 107–112, Ann Arbor, MI, USA, July 2006c.

Pena, P., Cury, J., e Lafortune, S. "Verification of Nonconflict of Supervisors Using Abstractions". *Submetido para IEEE Transactions on Automatic Control*, 2006d.

Ramadge, P. e Wonham, W. "Supervisory Control of a Class of Discrete Event Systems". Technical Report 8311, Systems Control Group Report, University of Toronto, Oct. 1983.

Ramadge, P. e Wonham, W. "Supervisory Control of a Class of Discrete Event Processes". *SIAM Journal Control and Optimization*, Jan. 1987.

Ramadge, P. e Wonham, W. "The Control of Discrete Event Systems". In *Proceedings IEEE, Special Issue on Discrete Event Dynamic Systems*, volume 77, pp. 81–98, Jan. 1989.

Reiser, C., da Cunha, A. E., e E.R.Cury, J. "The Environment Grail for Supervisory Control of Discrete Event Systems". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 390–391, Ann Arbor, MI, USA, July 2006.

Rohloff, K. e Lafortune, S. "The Control and Verification of Similar Agents Operating in a Broadcast Network Environment". In *Proceedings of the 42nd IEEE Conference on Decision and Control, CDC'03*, Maui, Hawaii, Dec. 2003a.

Rohloff, K. e Lafortune, S. "Symmetry Reductions for a Class of Discrete-Event Systems". In *Proceedings of the 43rd IEEE Conference on Decision and Control, CDC'04*, Atlantis, Paradise Island, Bahamas, Dec. 2004a.

Rohloff, K. e Lafortune, S. "Symmetry Reductions for a Class of Discrete-Event Systems". Technical Report CGR04-02, Department of Electrical Engineering and Computer Science, University of Michigan, http://www.eecs.umich.edu/umdes/sym_tKrep.ps, 2004b.

Rohloff, K. e Lafortune, S. "Supervisor Existence for Modular Discrete-Event Systems". In *Proceedings of IFAC Conference on Control Systems Design*, Bratislava, Slovak Republic, Sep. 2003b.

Rudie, K. e Wonham, W. "Think Globally, Act Locally: Decentralized Supervisory Control". *IEEE Transactions on Automatic Control*, 37(11):1692–1708, Nov. 1992.

Schmidt, K. *Hierarchical Control of Decentralized Discrete Event Systems Theory and Application*. PhD thesis, Universität Erlangen-Nürnberg, Erlangen, Alemanha, 2005.

Schmidt, K., Marchand, H., e Gaudin, B. "Modular and Decentralizd Supervisory Control of Concurrent Discrete Event Systems Using Reduced Systems Models". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 149–154, Ann Arbor, MI, USA, July 2006.

Schmidt, K. e Moor, T. "Marked-String Accepting Observers for the Hierarchical and Decentralized Control Discrete Event Systems". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 413–418, Ann Arbor, MI, USA, July 2006.

Sivolella, L., Cunha, A., e Ades, R. "Redução de Supervisores como ferramenta para Implementação de Supervisores em Controladores Discretos". In *Anais do XVI Congresso Brasileiro de Automática, CBA'06*, pp. 2778–2783, Salvador, Brasil, Oct. 2006.

Su, R. e Thistle, J. "A Distributed Supervisor Synthesis Approach Based on Weak Bisimulation". In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, pp. 64–69, Ann Arbor, MI, USA, July 2006.

Su, R. e Wonham, W. "Supervisor Reduction for Discrete-Event Systems". *Discrete Event Dynamic Systems: Theory and Applications*, 14:31–53, 2004.

Sztipanovits, J. e Misra, A. "Diagnosis of Discrete Event Systems Using Ordered Binary Decision Diagrams". In *Proceedings of 7th Intl. Workshop on Principles of Diagnosis*, 1996.

Tilbury, D. e Khargonekar, P. "Challenges and Opportunities in Logic Control for Manufacturing Systems". Report of the NSF Workshop held at the University of Michigan, Ann Arbor, MI, Aug. 2000.

Torrico, C. *Controle Supervisório Hierárquico de Sistemas a Eventos Discretos: Uma Abordagem Baseada na Agregação de Estados*. PhD thesis, Universidade Federal de Santa Catarina, UFSC, Florianópolis, SC, Brasil, 2003.

Vahidi, A. *Efficient Analysis of Discrete Event Systems*. PhD thesis, Departament of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden, 2004.

Vahidi, A., Lennartson, B., e Fabian, M. "Efficient Analysis of Large Discrete-Event Systems with Bynary Decision Diagrams". In *Proceedings of 44th IEEE Conference on Decision and Control, CDC'05*, Seville, Spain, December 2005.

Willner, Y. e Heymann, M. "Supervisory Control of Concurrent Discrete-Event Systems". *International Journal of Control*, 54:1143–1169, 1991.

Wong, K. "On the Complexity of Projections of Discrete-Event Systems". In *Proceedings of the 4th Workshop on Discrete Event Systems, WODES'98*, Aug. 1998.

Wong, K., Thistle, J., Hoang, H.-H., e Malhamé, R. "Conflict Resolution in Modular Control with Applications to Feature Interaction". In *Proceedings of the 34th IEEE Conference on Decision and Control, CDC'95*, New Orleans, LA, Dec. 1995.

Wong, K., Thistle, J., Malhamé, R., e Hoang, H.-H. "Supervisory Control of Distributed Systems: Conflict Resolution". *Discrete Event Dynamic Systems: Theory and Applications*, 10:131–186, 2000.

Wong, K. e Wonham, W. M. "Hierarchical Control of Discrete-Event Systems". *Discrete Event Dynamic Systems: Theory and Applications*, 6(3):241–273, 1996.

Wong, K. e Wonham, W. "Modular Control and Coordination of Discrete-Event Systems". *Discrete Event Dynamic Systems: Theory and Applications*, 8(3):247–297, 1998.

Wong, K. e Wonham, W. "On the Computation of Observers in Discrete-Event Systems". *Discrete Event Dynamical Systems*, 14(1):55–107, Jan. 2004.

Wonham, W. "Supervisory Control of Discrete-Event Systems". Dept. of Electrical & Computer Engineering, University of Toronto, 2004.

Wonham, W. "Supervisory Control of Discrete-Event Systems". Dept. of Electrical & Computer Engineering, University of Toronto, 2005.

Wonham, W. "Supervisory Control of Discrete-Event Systems". Dept. of Electrical & Computer Engineering, University of Toronto, 2006.

Wonham, W. e Ramadge, P. "On the Supremal Controllable Sublanguage of a Given Language". *SIAM Journal Control and Optimization*, May 1987.

Wonham, W. e Ramadge, P. "Modular Supervisory Control of Discrete Event Systems". *Math. Control, Signals Systems*, 1:13–30, 1988.

Yoo, T. e Lafortune, S. "New Results On Decentralized Supervisory Control of Discrete-Event Systems". In *Proceedings of the 39th IEEE Conference on Decision and Control, CDC'00*, pp. 12–15, Sydney, Australia, Dec. 2000a.

Yoo, T. e Lafortune, S. "Polynomial-time verification of diagnosability of partially observed discrete-event systems". *IEEE Transactions on Automatic Control*, 47(9):1491– 1495, Sep. 2002a.

Yoo, T. e Lafortune, S. "Decentralized Supervisory Control with Conditional Decisions: Supervisor Existence". *IEEE Transactions on Automatic Control*, 49(11):1886–1904, Nov. 2004.

Yoo, T. e Lafortune, S. "Decentralized Supervisory Control with Conditional Decisions: Supervisor Realization". *IEEE Transactions on Automatic Control*, 50(8):1205–1211, Aug. 2005.

Yoo, T.-S. e Lafortune, S. "A General Architecture for Decentralized Supervisory Control of Discrete-Event Systems". In *Proceedings of the 5th Workshop on Discrete Event Systems, WODES'00*, Ghent, Belgium, Aug. 2000b.

Yoo, T.-S. e Lafortune, S. "A General Architecture for Decentralized Supervisory Control of Discrete-Event Systems". *Discrete Event Dynamic Systems: Theory and Applications*, 12 (3):335–377, July 2002b.

Yoo, T. e Lafortune, S. "Decentralized Supervisory Control: A New Architecture with a Dynamic Decision Fusion Rule". In *Proceedings of the 6th Workshop on Discrete Event Systems, WODES'02*, pp. 11–17, Zaragoza, Sep. 2002c.

Zhong, H. e Wonham, W. "On the Consistency of Hierarchical Supervision in Discrete-Event Systems". *IEEE Transactions on Automatic Control*, 35(10):1125–1134, Oct. 1990.