FRANCISCO DE ASSIS CARVALHO DA SILVA NETO

REDUÇÃO DE SUPERVISORES UTILIZANDO MARCAÇÃO POR EVENTOS E MÉTODOS DE OTIMIZAÇÃO

FLORIANÓPOLIS
2008

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

REDUÇÃO DE SUPERVISORES UTILIZANDO MARCAÇÃO POR EVENTOS E MÉTODOS DE OTIMIZAÇÃO

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

FRANCISCO DE ASSIS CARVALHO DA SILVA NETO

Florianópolis, Setembro de 2008.

REDUÇÃO DE SUPERVISORES UTILIZANDO MARCAÇÃO POR EVENTOS E MÉTODOS DE OTIMIZAÇÃO

FRANCISCO DE ASSIS CARVALHO DA SILVA NETO

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, Área de Concentração em *Controle, Automação e Informática Industrial*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Max Hering de Queiroz, Doutor
Orientador

Guilherme Bittencourt, Doutor
Co-Orientador

Kátia Campos de Almeida, Doutora
Coordenadora do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Guilherme Bittencourt, Doutor
Presidente

Antonio Eduardo Carrilho da Cunha, Doutor

Jean-Marrie Farines, Doutor

Eduardo Camponogara, Doutor

Aos queridos Mãe, Pai, Dona Maria, Dona Ana, Pedro, Mariana, Marina e Maria Clara

AGRADECIMENTOS

Agradeço primeiramente e principalmente aos meus pais. Agradeço também a minhas queridas avós Dona Ana e Dona Maria e, postumamente a meu avô Assis. A minhas amadas irmãs Marina e Mariana (sem esquecer de meu afilhado Pedro e meu cunhado Daniel), a minha família como um todo e à indispensável companheira Maria Clara. Aos professores Dr. Max Hering de Queiroz e Dr. Guilherme Bittencourt pelo empenho e tempo desprendido na excelente orientação. Ao professor Dr. Jean-Marie Farines que me estimulou a iniciar o mestrado. Ao professor Dr. Eduardo Camponogara por sua essencial ajuda em alguns momentos da dissertação. Ao professor Dr. José Eduardo Ribeiro Cury que muito me ajudou durante a minha trajetória na área de Sistemas a Eventos Discretos. E a todos os professores e funcionários do DAS. Aos grandes amigos Adriano Naspolini, Alfredo Américo, André Maciel, Antônio Emygdio, Antônio Meirelles, Bernardo de Castro, Bruno Selva, Carlos Eduardo Mauad, Carlos Gonzaga, Diogo Barbosa, Guilherme Stein, Gustavo Raposo, João Lourenço, Lucas Maciel, Luiz Stival, Marco San Martin, Rafael Larcher, Rodrigo Saad e Sílvio Ary. A todos os amigos do Sindicato do LCMI.

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

REDUÇÃO DE SUPERVISORES UTILIZANDO MARCAÇÃO POR EVENTOS E MÉTODOS DE OTIMIZAÇÃO

FRANCISCO DE ASSIS CARVALHO DA SILVA NETO

Setembro/2008

Orientador: Max Hering de Queiroz, Doutor
Co-orientador: Guilherme Bittencourt, Doutor
Área de Concentração: Controle, Automação e Informática Industrial
Palavras-chave: Sistemas a eventos discretos, redução de supervisores, marcação de estados, otimização, programação linear inteira mista, algoritmos genéticos.
Número de Páginas: 80

Este trabalho tem como objetivo enunciar organizadamente as diferentes abordagens de síntese de supervisores em sistemas a eventos discretos (supervisor não-marcador, supervisor marcador e supervisor desmarcador) e reduzí-los utilizando métodos de otimização como Programação Linear Inteira Mista (PLIM) e Algoritmos Genéticos (AG). Para tal, propõe-se uma metodologia original para se representar a marcação de estados por meio de eventos de marcação e reduzir supervisores por meio de coberturas de controle independentemente do tipo de supervisor. Como resultado, primeiramente demonstra-se matematicamente a equivalência entre a marcação de estados tradicional e a marcação por eventos de marcação. Então, substitui-se a condição de consistência da marcação durante a redução de supervisores por um problema de controlabilidade dos eventos de marcação. Além disso, e ainda utilizando eventos de marcação, o problema da redução de supervisores é modelado em PLIM e AG, chegando-se a reduzir supervisores de 384 estados (por meio de PLIM) e de 96 estados (por meio de AG). Ao final, concluem-se as metodologias propostas e estabelecem-se perspectivas para trabalhos futuros.

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

SUPERVISORY REDUCTION USING MARKING EVENTS AND OPTIMIZATION METHODS

FRANCISCO DE ASSIS CARVALHO DA SILVA NETO

September/2008

Primary Advisor: Max Hering de Queiroz, PhD
Secondary Advisor: Guilherme Bittencourt, PhD
Area of Concentration: Control, Automation and Industrial Informatics
Key words: Discrete-Event Systems, Supervisory Reduction, States Marking, Optimization, Mixed Integer Programming, Genetic Algorithms.
Number of Pages: 80

This work has as objectives to provide an organized description of different approaches on synthesizing supervisors on Discrete-Event Systems (not-marking supervisors, marking supervisors and unmarking supervisors), and to reduce them by using optimization methods such Mixed Integer Programming (MIP) and Genetic Algorithms (GA). Thus it enounces a new methodology to represent marker states by the use of marking events, and to reduce supervisors by control set coverings, independently of supervisors' approaches. As results, at first it mathematically proves equivalence between traditional state-marking and the proposed event-marking. Then one replaces the marking states condition regarding supervisory reduction with a marking events controllability problem. Further, and still using marking events, the supervisory reduction problem is modeled in MIP and GA, achieving successful reductions of 384 states supervisors (using MIP) and 96 states supervisors (using GA). Finally, it concludes on the proposed methodologies and states perspective for further work.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Não é de hoje que o homem procura controlar sistemas a sua volta. Por exemplo, ao se cobrir ou descobrir o corpo com um cobertor, procura-se controlar a temperatura do ar à sua volta de forma a auxiliar o controle biológico da temperatura do corpo. Este tipo de sistema é considerado contínuo, pois suas variáveis (neste caso a temperatura) estão constantemente variando em relação ao tempo e são descritas por equações diferenciais. Assim como este exemplo da temperatura, existem outros exemplos clássicos, como o controle do nível de um líqüido em um tanque ou o controle da velocidade ou trajetória de um veículo. Porém, há sistemas que são considerados discretos, isto é, não possuem uma evolução explícitamente dependente do tempo e, por conseguinte, não podem ser modelados por equações diferenciais e nem utilizar a teoria clássica de controle. Tais sistemas, denominados Sistemas a Eventos Discretos (SED), caracterizam-se por possuírem conjuntos finitos de *estados* e *eventos*, sendo que transições entre estados são causadas por ocorrências de eventos, que não necessariamente estão ligadas ao tempo. Por exemplo, sistemas de filas, controle de acesso, protocolos de comunicação, coordenação de sistemas de manufatura são tratados como SEDs.

Sistemas a Eventos Discretos se caracterizam por perceberem as ocorrências a sua volta por meio de estímulos, denominados *eventos*. Estes eventos são considerados instantâneos, de maneira que sistemas que evoluem por meio deles têm caráter discreto no tempo. Por exemplo, pode-se considerar uma porta automática de um ônibus como sendo um SED a ser controlado (denominado *planta*). A abertura e o fechamento seriam os eventos, enquanto que os possíveis *estados* deste sistema seriam: porta aberta e porta fechada. Neste exemplo, a abertura e o fechamento da porta devem ser considerados instantâneos, fato que não condiz com a realidade. Porém para uma análise do problema somente do ponto de vista da ordem

em que os eventos acontecem, o fato de que para a porta abrir seja necessário que todo um sistema pneumático seja acionado não necessita ser levado em conta.

Considerando somente o exemplo anterior, pode parecer fácil controlar um SED empiricamente ou por meio de tentativa e erro, porém, para se controlar sistemas complexos ou aplicações críticas, tal metodologia é muito custosa e não garante que a ação do controle é correta. A Teoria de Controle Supervisório (TCS) [55] tem como objetivo encontrar uma solução formal, que garanta um comportamento controlável, não-bloqueante e minimamente restritivo a um SED, respeitando uma dada especificação. Ela fornece um algoritmo de complexidade polinomial, em relação ao produto entre o número de estados da planta e o número de estados da especificação, que permite sintetizar um supervisor. Embora o algoritmo seja de ordem polinomial, o número de estados da planta cresce exponencialmente em relação ao número de subsistemas. Além disso, o número de estados da especificação cresce exponencialmente com o número de restrições. Logo, o tamanho do problema cresce exponencialmente (fato conhecido como *explosão de estados*), o que resulta em problemas computacionais na síntese de supervisores para problemas de grande porte. A Teoria de Controle Supervisório Modular Local [43] propõe uma alternativa para a síntese modular de supervisores, reduzindo drasticamente seu custo computacional, porém necessita de uma etapa de verificação que enfrenta os mesmos problemas computacionais da síntese tradicional de supervisores.

Uma vez sintetizados os supervisores, é necessário implementá-los para que a solução seja efetivamente utilizada em problemas reais. A implementação dos supervisores ótimos obtidos pela TCS nem sempre é viável devido ao seu grande número de estados. Uma alternativa para contornar tal problema é a redução de supervisores [50], que busca encontrar supervisores reduzidos que possuam ação de controle equivalente ao supervisor ótimo, obtido pela TCS, com relação à planta, isto é, impondo o mesmo comportamento. Segundo Vaz e Wonham [50], existe um supervisor reduzido cujo número de estados é mínimo (embora tal supervisor não seja único) e a tarefa de encontrar tal supervisor é conhecida como Problema da Minimização de Supervisores. A redução de supervisores pode, em vários casos, resultar em grandes reduções no espaço de estados do supervisor, como por exemplo em [14, 43, 51], viabilizando sua implementação. Porém a resolução do Problema da Minimização de Supervisores é comprovadamente NP-difícil [11], conforme explicitado por [49]. Naquele trabalho, Su e Wonham propõem um algoritmo de complexidade polinomial em relação ao número de estados do supervisor para se reduzí-los, porém consideram somente parte do possível espaço de soluções, não garantindo que a solução encontrada seja ótima, além de que

não há registros de reduções de supervisores com mais de 2000 estados.

Na síntese de supervisores, a marcação de estados é de fundamental importância, pois é por meio dela que as condições de bloqueio são estabelecidas. No que diz respeito à redução, a marcação de estados é tratada diferentemente em trabalhos como [48, 49, 50]. Cada trabalho estabelece diferentes condições para a redução de supervisores ao considerar a marcação. Todos os trabalhos consideram que a marcação do sistema supervisionado deve ser preservada na redução.

1.1 Objetivos

O presente trabalho tem como objetivo apresentar de forma organizada os diferentes tipos de supervisores, as diferentes metodologias de redução de supervisores e suas relações com a marcação de estados. Além disso, contribuir com uma metodologia original que substitui a marcação de estados por eventos e que possibilita a síntese e a redução de supervisores independentemente do tipo de supervisor utilizado e das condições de marcação para a redução. Baseando-se nestes resultados, propõem-se metodologias originais para a redução de supervisores por meio de Programação Linear Inteira Mista [11, 54], que foi publicada em congresso nacional [15], e Algoritmos Genéticos [6, 25].

1.2 Organização do documento

O Capítulo 2 apresenta conceitos preliminares sobre a TCS, tipos de supervisores, marcação e a redução de supervisores. No Capítulo 3 propõe-se a marcação de estados por eventos, que é um alternativa à marcação tradicional. Em seguida, o Capítulo 4 expõe a metodologia utilizada para solucionar o problema por meio de Programação Linear Inteira Mista. O Capítulo 5 explicita formas de aplicar a teoria de Algoritmos Genéticos ao problema da Minimização de Supervisores e, finalmente, no Capítulo 6 apresentam-se conclusões e perspectivas para trabalhos futuros.

Capítulo 2

Redução de supervisores

Este capítulo apresenta conceitos básicos e notações utilizados na Teoria de Controle Supervisório e na redução de supervisores de Sistemas a Eventos Discretos. A Seção 2.1 apresenta conceitos elementares sobre a teoria de linguagens e geradores (uma descrição mais aprofundada pode ser encontrada em [7, 26]). A Seção 2.2 apresenta conceitos sobre a Teoria de Controle Supervisório que são necessários para se entender o problema da redução de supervisores, maiores detalhes podem ser encontrados em [7, 55]. Na Seção 2.3, apresentam-se considerações sobre a marcação para diferentes tipos de supervisores. Na Seção 2.4, o problema da redução de supervisores é detalhado. E, finalmente, a Seção 2.5 apresenta detalhes sobre a influência da marcação na redução de supervisores.

2.1 Linguagens e geradores

O comportamento lógico de Sistemas a Eventos Discretos pode ser modelado por linguagens. A seguir, os conceitos básicos são apresentados.

Definição 2.1.1 (Linguagem) *Uma linguagem L definida sobre um conjunto de eventos Σ é um conjunto de cadeias finitas formadas por eventos pertencentes a Σ.*

O conjunto de eventos Σ é denominado *alfabeto* e $\varepsilon \notin \Sigma$ representa uma cadeia vazia, isto é a não ocorrência de um evento[1]. Σ^* denota o conjunto de todas as possíveis cadeias finitas formadas pela concatenação de elementos de Σ, incluindo também a cadeia vazia ε. Dada esta definição, é possível perceber que qualquer linguagem L definida sobre Σ é necessariamente um subconjunto de Σ^*, isto é, $L \subseteq \Sigma^*$. Em particular, $\emptyset$ e Σ^* são linguagens.

Para uma dada cadeia $tu = s$, com $t, u, s \in \Sigma^*$, denomina-se t como sendo *prefixo* de s.

Além das tradicionais operações entre conjuntos, outras operações são definidas sobre linguagens.

Definição 2.1.2 (Prefixo-fechamento) *Seja $L \subseteq \Sigma^*$, então seu prefixo-fechamento resulta em um conjunto $\overline{L}$ formado por todas as cadeias em Σ^* que são prefixo das cadeias em L. Formalmente, $\overline{L} := \{s \in \Sigma^* : \exists t \in \Sigma^*(st \in L)\}$[2].*

Nota-se então que $L \subseteq \overline{L}$ e uma linguagem L é dita *prefixo-fechada* se $L = \overline{L}$. Logo L é prefixo-fechada se qualquer prefixo de qualquer cadeia em L também pertence a L.

Um gerador $\boldsymbol{G}$ é um modelo matemático capaz de representar um SED, conforme a definição abaixo.

Definição 2.1.3 (Gerador) *Um gerador é uma tupla $\boldsymbol{G} = (Q, \Sigma, \delta, q_0, Q_m)$, onde Σ é o alfabeto, Q é o conjunto de estados, $q_0 \in Q$ é o estado inicial, $Q_m \subseteq Q$ é o conjunto de estados marcados (ou estados finais), e $\delta : Q \times \Sigma \to Q$ é a função parcial (fnp) de transição, que é definida em cada estado $q \in Q$ somente para um subconjunto de elementos do alfabeto.*

A notação $\delta(q, \sigma)!$ significa que $\delta(q, \sigma)$ está definida, para $q \in Q$ e $\sigma \in \Sigma$. Já a notação $\neg\delta(q, \sigma)!$ indica que $\delta(q, \sigma)$ não está definida. Pode-se estender δ para cadeias por meio da

[1] A cadeia vazia é muito importante em problemas de observabilidade [7].
[2] "$:=$" *denota* igual por definição.

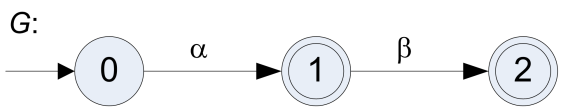

Figura 2.1: Exemplo de um gerador.

função $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ (fnp) da seguinte forma:

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

sempre que $q' := \hat{\delta}(q, s)!$ e $\delta(q', \sigma)!$, com $s \in \Sigma^*$. Para simplificar a notação, utiliza-se δ como $\hat{\delta}$ no restante deste documento.

Geradores podem ser representados por diagramas de transição de estados, que são grafos direcionados nos quais cada nó representa um estado do sistema $q \in Q$ e cada arco direcionado uma transição $\delta(q, \sigma) = m$, que, a partir de um estado $q \in Q$, com a ocorrência de um evento $\sigma \in \Sigma$, leva a outro estado $m \in Q$. Cada arco é rotulado com o evento $\sigma \in \Sigma$ de sua transição correspondente. O estado inicial $q_0 \in Q$ é identificado por ser destino de um arco sem rótulo nem origem. Os estados marcados pertencentes a Q_m são identificados por nós com duplos círculos.

Exemplo 2.1.1 (Exemplo de um gerador) A Figura 2.1 exemplifica um gerador $\boldsymbol{G}$, com $\Sigma = \{\alpha, \beta\}$, $Q = \{0, 1, 2\}$, $\delta(0, \alpha) = 1$, $\delta(1, \beta) = 2$, $q_0 = 0$ e $Q_m = \{1, 2\}$.

□

A *linguagem gerada* por $\boldsymbol{G}$, denotada por $L(\boldsymbol{G})$, representa o conjunto de cadeias $s \in \Sigma^*$ definidas a partir do estado inicial.

Definição 2.1.4 (Linguagem gerada por um gerador)

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

O *linguagem marcada* de $\boldsymbol{G}$, denotada por $L_m(\boldsymbol{G})$, representa o sub-conjunto das cadeias geradas que leva a um estado marcado. Formalmente:

Definição 2.1.5 (Linguagem marcada por um gerador)

$$L_m(\boldsymbol{G}) := \{s \in L(\boldsymbol{G}) : \delta(q_0, s) \in Q_m\}$$

De acordo com as definições acima, tem-se que $L_m(\boldsymbol{G}) \subseteq L(\boldsymbol{G}) = \overline{L(\boldsymbol{G})}$. Isto é, todo gerador $\boldsymbol{G}$ pode está associado a uma linguagem $L = \overline{L}$, gerada por ele.

A marcação de estados em geradores é uma etapa muito importante no processo de modelagem de sistemas a eventos discretos. Isto porque ela indica um conjunto de estados $Q_m \subseteq Q$ que representam *tarefas completas*, isto é, estados que se deseja que o sistema alcance. Juntamente com o conceito de marcação, vem o conceito de *bloqueio*. Pois considera-se que um gerador está em situação de bloqueio quando não há como atingir um estado marcado, a partir de algum estado do gerador.

Um estado $q \in Q$ é *acessível* se existe uma cadeia $s \in \Sigma^*$ com $\delta(q_0, s)!$ e $\delta(q_0, s) = q$. $\boldsymbol{G}$ é considerado *acessível* se q for acessível para todo $q \in Q$. Um estado $q \in Q$ é *co-acessível* se existe $s \in \Sigma^*$ tal que $\delta(q, s) \in Q_m$, e $\boldsymbol{G}$ é considerado *co-acessível* se q é co-acessível para todo $q \in Q$. $\boldsymbol{G}$ é *não-bloqueante* se todo estado acessível é também co-acessível, ou, equivalentemente, $L(\boldsymbol{G}) = \overline{L_m(\boldsymbol{G})}$. E finalmente, $\boldsymbol{G}$ é considerada *trim* caso seja acessível e co-acessível.

2.2 Teoria de Controle Supervisório

Na Teoria de Controle Supervisório [55], deseja-se *controlar* uma *planta* $\boldsymbol{G}$, representando o comportamento em malha aberta de um Sistema a Eventos Discreto (SED) [7], por meio de um *supervisor* $\mathbb{S}$ que é responsável por habilitar eventos em função de seqüências de eventos geradas pela planta. A Figura 2.2 ilustra a estrutura de controle supervisório. O problema de controle supervisório consiste em encontrar um supervisor que controle a planta de acordo com uma *especificação* $K \subseteq L_m(\boldsymbol{G})$ de forma minimamente restritiva e sem causar bloqueio. A solução ótima para tal problema pode ser calculada automaticamente por um algoritmo de complexidade polinomial em relação ao produto entre o número de estados da planta e o número de estados da especificação.

Definição 2.2.1 (Planta) *Sistema a eventos discreto a ser controlado. É modelada por meio de um gerador* $\boldsymbol{G} = (Q, \Sigma, \delta, q_0, Q_m)$.

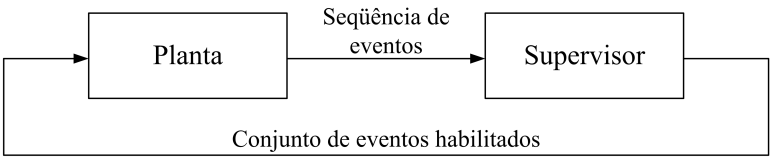

Figura 2.2: Estrutura de controle supervisório.

O alfabeto Σ da planta é dividido em dois conjuntos disjuntos de eventos $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$, onde Σ_u contém todos os eventos *não-controláveis*, isto é, eventos cuja ocorrência não pode ser evitada pelo supervisor, enquanto que Σ_c contém todos os eventos *controláveis*, que são eventos cuja ocorrência pode ser evitada por um supervisor. A distinção entre eventos controláveis e não-controláveis é representada nos grafos dos geradores, respectivamente, por arcos cortados por um traço e arcos não cortados.

Um conjunto de eventos $\gamma \in 2^{\Sigma}$, denominado *opção de controle*, indica quais eventos da planta estão habilitados a ocorrer. O conjunto de todas as possíveis opções de controle Γ, denominado *estrutura de controle*, é definido da seguinte forma:

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

sendo que a condição $\gamma \supseteq \Sigma_u$ simplesmente indica que os eventos não-controláveis devem sempre estar habilitados.

Definição 2.2.2 (Supervisor não-marcador) *Um supervisor não-marcador para $\boldsymbol{G}$ é um mapa $\mathbb{S} : L(\boldsymbol{G}) \rightarrow \Gamma$ que mapeia as cadeias geradas pela planta $s \in L(\boldsymbol{G})$ a uma opção de controle $\gamma \in \Gamma$.*

Definição 2.2.3 (Linguagem gerada $L(\mathbb{S}/\boldsymbol{G})$) *A linguagem gerada por $\mathbb{S}/\boldsymbol{G}$ é $L(\mathbb{S}/\boldsymbol{G}) \subseteq L(\boldsymbol{G})$ e é definida recursivamente como:*

1. $\varepsilon \in L(\mathbb{S}/\boldsymbol{G})$, *e*
2. $[(s \in L(\mathbb{S}/\boldsymbol{G})) \wedge (s\sigma \in L(\boldsymbol{G})) \wedge (\sigma \in \mathbb{S}(s))] \iff [s\sigma \in L(\mathbb{S}/\boldsymbol{G})]$

em palavras, inicia-se $L(\mathbb{S}/\boldsymbol{G}) = \{\varepsilon\}$ e, em seguida, adicionam-se cadeias a $L(\mathbb{S}/\boldsymbol{G})$ sempre que forem definidas na planta $\boldsymbol{G}$ e permitidas pelo supervisor $\mathbb{S}$. Uma definição mais informal seria: o subconjunto de cadeias possíveis em $\boldsymbol{G}$ que são permitidas pelo supervisor $\mathbb{S}$.

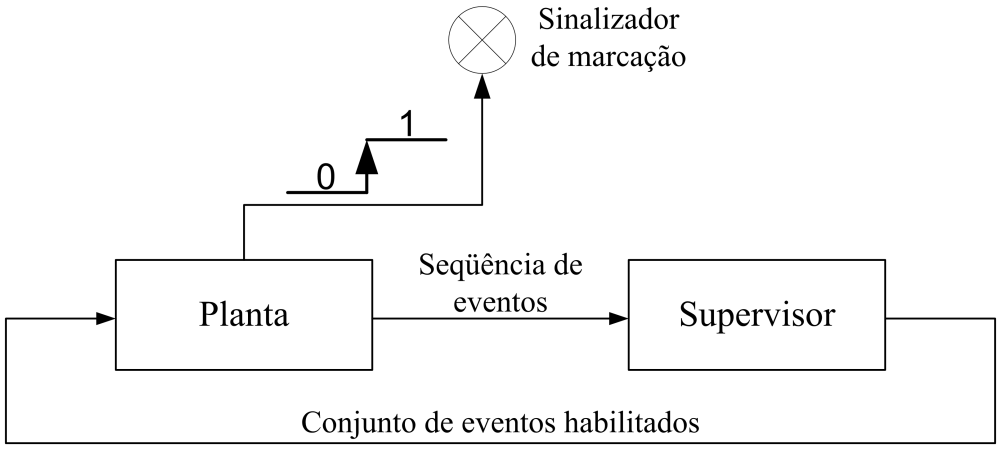

Figura 2.3: Sinalizador de marcação para um supervisor não-marcador.

Definição 2.2.4 (Linguagem Marcada $L_m(\mathbb{S}/\boldsymbol{G})$) *A linguagem marcada de* $\mathbb{S}/\boldsymbol{G}$ *é*

$$L_m(\mathbb{S}/\boldsymbol{G}) := L(\mathbb{S}/\boldsymbol{G}) \cap L_m(\boldsymbol{G})$$

isto é, $L_m(\mathbb{S}/\boldsymbol{G})$ contém todas as cadeias da linguagem marcada da planta $L_m(\boldsymbol{G})$ que restaram após a supervisão de $\mathbb{S}$. A marcação do sistema em malha-fechada, no caso de supervisores não-marcadores, independe da marcação do supervisor. Conforme [55], essa idéia pode ser entendida imaginando-se que, na planta, haja um *sinalizador de marcação*, que é acionado sempre que o sistema supervisionado atinge um estado marcado, conforme ilustrado na Figura 2.3.

Definição 2.2.5 (Supervisor não-bloqueante) *Diz-se que* $\mathbb{S}$ *é não-bloqueante, com relação a* $\boldsymbol{G}$*, se*

$$\overline{L_m(\mathbb{S}/\boldsymbol{G})} = L(\mathbb{S}/\boldsymbol{G})$$

ou seja, a partir de qualquer cadeia de $L(\mathbb{S}/\boldsymbol{G})$ é possível atingir uma cadeia da linguagem marcada $L_m(\mathbb{S}/\boldsymbol{G})$. Claramente percebe-se que a condição de bloqueio depende diretamente da marcação do sistema em malha-fechada.

É conveniente representar supervisores por meio de um par $\mathbb{S} = (\boldsymbol{S}, \Psi)$, sendo $\boldsymbol{S} = (X, \Sigma, \xi, x_0, X_m)$ um gerador e $\Psi : X \times \Sigma \to \{0, 1, dc\}$ uma lei de controle. $\boldsymbol{S}$ é considerado como sendo dirigido externamente por uma cadeia $s \in L(\boldsymbol{G})$ de eventos $\sigma \in \Sigma$ gerados pela planta $\boldsymbol{G}$. Em contrapartida, enquanto $\boldsymbol{S}$ está no estado $x \in X$ e $\boldsymbol{G}$ no estado $q \in Q$, se a

transição $\delta(q,\sigma)$ está definida ($\delta(q,\sigma)!$), então ela está sujeita à ação de controle Ψ definida como

$$\Psi(x,\sigma) = 1, \qquad \sigma \in \Sigma_u, x \in X$$
$$\Psi(x,\sigma) \in \{0,1,dc\}, \qquad \sigma \in \Sigma_c, x \in X$$

se $\Psi(x,\sigma) = 0$ (desabilitado) então a transição $\delta(\sigma,q)$ está proibida de ocorrer, se $\Psi(x,\sigma) = 1$ (habilitado) então a transição $\delta(\sigma,q)$ está permitida, mas não forçada, a ocorrer. Para os casos em que $\Psi(x,\sigma) = dc$ (*don't care*) então o fato de se atribuir $\Psi(x,\sigma) = 0$ ou $\Psi(x,\sigma) = 1$ não faz diferença no comportamento de $\boldsymbol{G}$, pois a transição $\delta(q,\sigma)$ não está definida ($\neg\delta(q,\sigma)!$).

Pode-se dizer que o par $\mathbb{S} = (\boldsymbol{S}, \Psi)$ é um mapa $\mathbb{S} : L(\boldsymbol{G}) \rightarrow \Gamma$ definindo-se

$$\mathbb{S}(s) := \{\sigma : \Psi(\xi(x_0,s),\sigma) \neq 0\}, \text{ para } s \in L(\boldsymbol{G})$$

isto é, $\mathbb{S}$ mapeia as cadeias $s \in L(\boldsymbol{G})$ de acordo com o estado atingido por $\boldsymbol{S}$ por meio da cadeia s e pela lei de controle deste mesmo estado em um conjunto de eventos habilitados.

Quando a planta está sendo controlada pelo supervisor, considera-se que sua função de transição $\delta : Q \times \Sigma \rightarrow Q$ é substituída por $\delta_\Psi : X \times Q \times \Sigma \rightarrow Q$ onde

$$\delta_\Psi(x,q,\sigma) = \delta(q,\sigma)$$

sempre que $\Psi(x,\sigma) = 1$ e é indefinida caso contrário e o comportamento em malha-fechada do sistema $\mathbb{S}/\boldsymbol{G}$ (lê-se $\boldsymbol{G}$ supervisionado por $\mathbb{S}$) é representado por outro gerador

$$\mathbb{S}/\boldsymbol{G} = (X \times Q, \Sigma, \xi \times \delta_\Psi, (x_0,q_0), X \times Q_m)$$

onde a função parcial de transição é definida por

$$(\xi \times \delta_\Psi)((x,q),\sigma) := (\xi(x,\sigma), \delta_\Psi(x,q,\sigma))$$

sempre que ambos os componentes da direita estejam definidos.

Graficamente, um supervisor $\mathbb{S}$ pode ser representado por meio do grafo de seu gerador $\boldsymbol{S}$, adicionado de informações que indicam a lei de controle Ψ em cada estado. Caso $\Psi(x,\sigma) = 1$ então o estado x é origem de um arco rotulado com σ e que tem como destino $\xi(x,\sigma)$. Se $\Psi(x,\sigma) = 0$, então o estado x deve ser origem de um arco tracejado rotulado com σ e

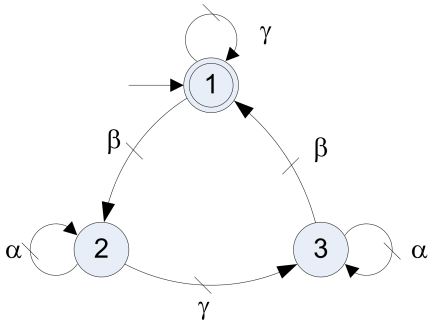

Figura 2.4: Planta $\boldsymbol{G}$.

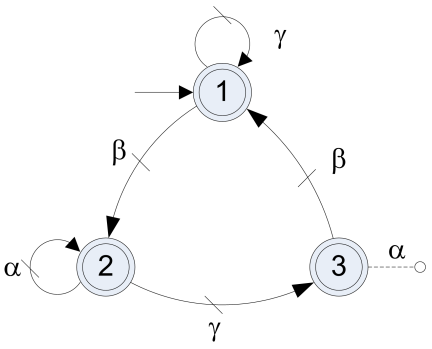

Figura 2.5: Representação de $\mathbb{S}$.

com um pequeno círculo em seu final. Finalmente, quando $\Psi(x, \sigma) = dc$ então x não é origem de nenhum arco rotulado com σ. Sistemas supervisionados $\mathbb{S}/\boldsymbol{G}$ também podem ser representados de forma equivalente.

Exemplo 2.2.1 (Exemplo de um supervisor $\mathbb{S}$) Um exemplo de supervisor $\mathbb{S} = (\boldsymbol{S}, \Psi)$ para a planta $\boldsymbol{G}$ da Figura 2.4 é mostrado na Figura 2.5. Neste caso, tem-se que a lei de controle Ψ desabilita a ocorrência do evento α no estado 3 da planta. A representação do sistema em malha-fechada $\mathbb{S}/\boldsymbol{G}$ é semelhante à de $\mathbb{S}$, com exceção da marcação de estados, pois $\mathbb{S}/\boldsymbol{G}$ possui somente o estado 1 como marcado, uma vez que a planta impõe sua marcação ao sistema em malha-fechada. O supervisor claramente é não-bloqueante, pois sempre permite a evolução do sistema para um estado marcado, neste caso o estado 1.

□

Definição 2.2.6 (Especificação) *Uma especificação é uma linguagem $K \subseteq L_m(\boldsymbol{G}) \subseteq \Sigma^*$ contendo o comportamento que se deseja impor à planta $\boldsymbol{G}$.*

Normalmente, a especificação $K \neq \emptyset$ é obtida a partir da intersecção $K = L_m(\boldsymbol{E}) \cap L_m(\boldsymbol{G})$, sendo que $\boldsymbol{E}$ é um gerador que modela o comportamento desejável do sistema, sem considerar o modelo da planta.

Definição 2.2.7 (Controlabilidade de K com relação a G) *Dada uma linguagem $K \in \Sigma^*$, diz-se que K é controlável com relação a $\boldsymbol{G}$ se*

$$\overline{K}\Sigma_u \cap L(\boldsymbol{G}) = \overline{K}$$

em palavras, para qualquer cadeia s que é prefixo de alguma cadeia em K, se, a partir dela, é possível que ocorra um evento não-controlável σ na planta $\boldsymbol{G}$, então esta ocorrência também

deve ser possível na linguagem $\overline{K}$. Isto é, a linguagem $\overline{K}$ não pode inibir diretamente a ocorrência de um evento não-controlável que seja possível na planta.

Definição 2.2.8 (Problema de controle supervisório) *Dados uma planta $\boldsymbol{G}$ e uma especificação $K \subseteq L_m(\boldsymbol{G})$, o problema de controle supervisório é encontrar um supervisor $\mathbb{S}$ não-bloqueante tal que $L_m(\mathbb{S}/\boldsymbol{G}) = K$.*

Ou seja, deseja-se encontrar um supervisor que não cause bloqueio ao sistema em malha-fechada e que respeite completamente a especificação $K \subseteq L_m(G)$[3].

Definição 2.2.9 ($L_m(\boldsymbol{G})$-fechamento) *Seja $K \subseteq L_m(\boldsymbol{G}) \subseteq \Sigma^*$, então K é $L_m(\boldsymbol{G})$-fechada caso K possua todos os seus prefixos que também pertençam a $L_m(\boldsymbol{G})$. Formalmente, $K = \overline{K} \cap L_m(\boldsymbol{G})$.*

A propriedade de $L_m(\boldsymbol{G})$-fechamento é essencial para o teorema [55] que apresenta as condições de existência de um supervisor, enunciado a seguir.

Teorema 2.2.1 (Solução do problema de controle supervisório) *Seja $\emptyset \subset K \subseteq L_m(\boldsymbol{G})$, então existe um supervisor não-bloqueante $\mathbb{S}$ para $\boldsymbol{G}$, com $L_m(\mathbb{S}/\boldsymbol{G}) = K$ se e somente se*

(i) K for controlável com relação a $\boldsymbol{G}$, e

(ii) K for $L_m(\boldsymbol{G})$-fechada.

É importante observar cuidadosamente a condição (ii) do teorema acima. Ela impõe que a existência de uma solução depende diretamente do fato de K conter qualquer um de seus prefixos que pertençam à linguagem marcada da planta $L_m(\boldsymbol{G})$. Isto é, K não pode *desmarcar* nenhum estado da planta. Por outro lado, a noção de controlabilidade não sofre influência da marcação da planta.

Exemplo 2.2.2 (Carro no trânsito) O gerador $\boldsymbol{G}$ ilustrado na Figura 2.6 representa uma situação típica do dia a dia de várias pessoas, modelando a possibilidade de ocorrência de um acidente de trânsito uma vez que se está dirigindo. Sendo que o estado 0 indica que o carro está parado na garagem, o estado 1 indica que o carro está andando no trânsito e o estado 2 representa que o carro sofreu um acidente. Os eventos controláveis α e β

[3] É importante perceber que, no caso do problema de controle supervisório conforme enunciado em [55] tem-se que $K \subseteq L_m(G)$. Esta consideração nem sempre é feita para casos em que a marcação é interpretada diferentemente, conforme discutido, a posteriori, nas seções 2.3.

representam a saída e o retorno do carro à garagem, respectivamente. Enquanto que o evento não-controlável γ representa a ocorrência de um acidente de trânsito. A marcação indica que é desejável que o carro possa sempre chegar à garagem. A linguagem gerada pela planta é $L(\boldsymbol{G}) = \{\varepsilon, \alpha, \alpha\gamma, \alpha\beta, \alpha\beta\alpha, \alpha\beta\alpha\gamma, \dots\}$, enquanto que a linguagem marcada é $L_m(\boldsymbol{G}) = \{\varepsilon, \alpha\beta, \alpha\beta\alpha\beta, \dots\}$.

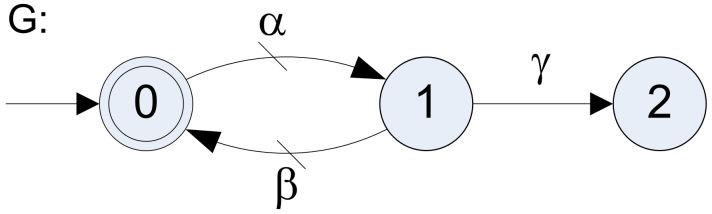

Figura 2.6: Carro no trânsito.

Uma linguagem $K' = \{\alpha\beta\}$, que impõe uma única saída do carro da garagem e inibe a ocorrência de um acidente, é não-controlável. Pois a cadeia $\alpha \in \overline{K'}$ seguida do evento $\gamma \in \Sigma_u$ é possível na planta $\boldsymbol{G}$ (uma vez que se está na rua, não se pode garantir que um acidente não ocorrerá), porém $\alpha\gamma$ não está contida em $\overline{K'}$. Já uma linguagem $K'' = \{\varepsilon, \alpha\gamma\}$ é controlável, pois não inibe diretamente, em nenhum momento, a ocorrência de γ. Outro exemplo de linguagem controlável seria a linguagem $K''' = \{\varepsilon\}$ que inibe a saída do carro da garagem e, por conseguinte, a ocorrência de um acidente. No caso de K''', o evento γ não é inibido diretamente, pois $\overline{K'''} = \{\varepsilon\}$ e $\varepsilon\gamma \notin L(\boldsymbol{G})$.

O supervisor $\mathbb{S}''$, obtido a partir da linguaguem $K'' = \{\varepsilon, \alpha\gamma\}$ é tal que $\mathbb{S}(\varepsilon) = \{\alpha, \beta, \gamma\}$, $\mathbb{S}(\alpha) = \{\alpha, \gamma\}$, $\mathbb{S}(\alpha\gamma) = \{\alpha, \beta, \gamma\}$. Logo, por meio das definições 2.2.3 e 2.2.4 tem-se $L(\mathbb{S}''/\boldsymbol{G}) = \{\varepsilon, \alpha, \alpha\gamma\}$ e $L_m(\mathbb{S}''/\boldsymbol{G}) = \{\varepsilon\}$. Ao se analisar $L(\mathbb{S}''/\boldsymbol{G})$ é possível encontrar uma situação de bloqueio, pois $(\alpha\gamma \in L(\mathbb{S}''/\boldsymbol{G})) \wedge (\alpha\gamma \notin \overline{L_m(\mathbb{S}''/\boldsymbol{G})})$, logo a linguagem K'' é controlável, porém causa uma situação de bloqueio (situação em que acontece um acidente e o carro não pode retornar à garagem). Já o supervisor $\mathbb{S}'''$, obtido a partir da linguaguem $K''' = \{\varepsilon\}$ é tal que $\mathbb{S}'''(\varepsilon) = \{\beta, \gamma\}$. Logo $L(\mathbb{S}'''/\boldsymbol{G}) = \{\varepsilon\}$ e $L_m(\mathbb{S}'''/\boldsymbol{G}) = \{\varepsilon\}$ e K''' é controlável e não-bloqueante.

□

Exemplo 2.2.3 (Aceleração de um carro) A Figura 2.7 ilustra um gerador com conjunto de estados $Q = \{0, 1, 2\}$, alfabeto $\Sigma = \Sigma_c = \{\alpha, \beta, \gamma\}$, estado inicial $q_0 = 0$ e conjunto de estados marcados $Q_m = \{0, 2\}$. Este gerador modela um automóvel, no que diz respeito à sua aceleração, da seguinte forma: o estado 0 (*parado*) indica que o carro está parado, o

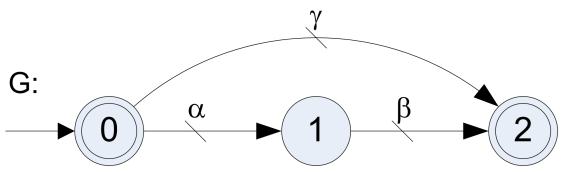

Figura 2.7: Aceleração de um carro.

estado 1 (*andando*) representa o carro andando em velocidade média e o estado 2 (*correndo*) indica que o carro encontra-se correndo em alta velocidade. O evento α representa uma pequena aceleração, saindo de *parado* para *andando*. O evento β indica um pequena aceleração que leva o carro de *andando* a *correndo*. Já o evento γ indica uma arrancada brusca, levando o carro do estado *parado* ao estado *correndo*. A marcação de estados indica que é desejável que sempre seja possível que o carro fique ou *parado* ou *correndo*. O gerador exemplificado possui linguagem gerada $L(\boldsymbol{G}) = \{\varepsilon, \alpha, \alpha\beta, \gamma\}$ e linguagem marcada $L_m(\boldsymbol{G}) = \{\varepsilon, \alpha\beta, \gamma\}$.

Considere uma linguagem $K' = \{\alpha\beta\} \subseteq L_m(\boldsymbol{G})$, que impõe ao sistema que o carro não deve arrancar bruscamente (inibindo o evento γ), não pode ficar permanentemente parado (estado 0) e nem pode ficar permanentemente em velocidade média (estado 1). K' é controlável com relação a $\boldsymbol{G}$ uma vez que $\Sigma = \Sigma_c$. Porém $K' \neq \overline{K'} \cap L_m(G)$, pois $\varepsilon \in (\overline{K'} \cap L_m(G))$ e $\varepsilon \notin K'$, logo K' não é $L_m(\boldsymbol{G})$-fechada e, segundo o Teorema 2.2.1, não existe um supervisor $\mathbb{S}'$ tal que $L_m(\mathbb{S}'/\boldsymbol{G}) = K$. Percebe-se que, da forma como o problema é posto na teoria tradicional [55], não se pode impor ao sistema que um de seus estados considerados como *tarefa completada* não mais o seja após a supervisão.

□

Caso uma dada linguagem $K \subseteq \Sigma^*$ não seja controlável em relação a uma planta $\boldsymbol{G}$, é possível encontrar a *maior* sublinguagem de K que seja controlável, sendo ela *maior* em termos de ser a menos restritiva possível com relação à planta. Tal linguagem é denominada a *máxima linguagem controlável* e é denotada por $supC(K, \boldsymbol{G})$. A existência da máxima linguagem controlável é muito importante, pois torna possível encontrar uma sublinguagem de $supC(K, \boldsymbol{G})$ que satisfaça a condição (i) do Teorema 2.2.1, desde que $supC(K, \boldsymbol{G}) \neq \emptyset$. A complexidade do algoritmo [7] que calcula a máxima linguagem controlável para qualquer linguagem regular $K = L_m(\boldsymbol{E})$, onde $\boldsymbol{E}$ é um gerador com $m \in \mathbb{N}$ estados, é $O(n^2m^2|\Sigma|)$, sendo $n \in \mathbb{N}$ o número de estados da planta e $|\Sigma|$ o número de elementos do alfabeto Σ da planta. O supervisor $\mathbb{S}_{sup}$, calculado a partir de $supC(K, \boldsymbol{G})$, é denominado *supervisor supremo*.

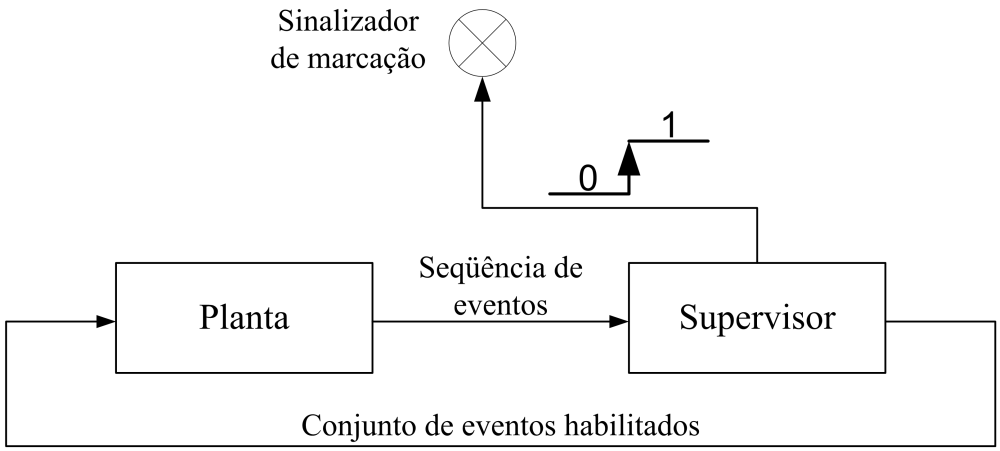

Figura 2.8: Supervisor marcador.

2.3 Supervisores Marcadores e Desmarcadores

Além da idéia de tarefa completa, a marcação é de fundamental importância na síntese de supervisores, pois é por intermédio dela que as considerações sobre bloqueio são feitas. Para o caso tradicional, enunciado na Seção 2.2, a marcação é ditada pela planta $\boldsymbol{G}$. Porém existem outras situações que fazem considerações diferentes com relação à marcação do sistema em malha-fechada. Em algumas situações, considera-se que um supervisor, além da ação de controle, possui também uma ação de marcação, possibilitando que o supervisor decida um comportamento marcado $L_m(\mathbb{S}/\boldsymbol{G}) \subseteq L(\boldsymbol{G})$ para o sistema em malha-fechada, partindo de uma linguagem $K = L_m(\boldsymbol{S})$. Para tal, deve-se encontrar um supervisor marcador $\mathbb{S}$, definido como:

Definição 2.3.1 (Supervisor marcador) *Um supervisor marcador para $\boldsymbol{G}$ é um par $\mathbb{S} = (\boldsymbol{S}, \Psi)$, sendo $\boldsymbol{S} = (X, \Sigma, \xi, x_0, X_m)$ um gerador e $\Psi : X \times \Sigma \rightarrow \{0, 1, dc\}$ uma lei de controle, sendo que o comportamento de $\mathbb{S}/\boldsymbol{G}$ é tal que*

$$L_m(\mathbb{S}/\boldsymbol{G}) := L(\mathbb{S}/\boldsymbol{G}) \cap L_m(\boldsymbol{S})$$

O Teorema 2.2.1 pode ser relaxado para este caso, eliminando-se a condição de $L_m(\boldsymbol{G})$-fechamento, bastando simplesmente que uma linguagem $K = L_m(\boldsymbol{S})$, gerada a partir de um gerador $\boldsymbol{E}$, seja controlável com relação à planta $\boldsymbol{G}$ para que a existência de $\mathbb{S}$ seja garantida. Assim, o Teorema 2.2.1 é reescrito [55] para o caso em que o supervisor possui ação de marcação como:

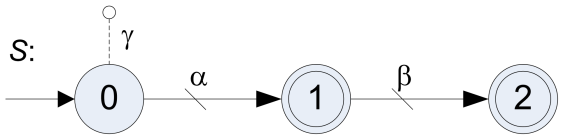

Figura 2.9: Supervisor marcador $\mathbb{S}$.

Teorema 2.3.1 *Seja $\emptyset \subset K \subseteq L(\boldsymbol{G})$, existe um supervisor marcador não-bloqueante $\mathbb{S}$ para $\boldsymbol{G}$, com $L(\mathbb{S}/\boldsymbol{G}) = K$ se e somente se K for controlável em relação a $\boldsymbol{G}$.*

Como quem carrega a informação da marcação do sistema em malha-fechada é o supervisor, então o conjunto de estados marcados do sistema supervisionado $\mathbb{S}/\boldsymbol{G}$ é definido como $X_m \times Q$.

Exemplo 2.3.1 (Supervisor marcador) Fazendo uso novamente da planta $\boldsymbol{G}$ do Exemplo 2.2.3 da aceleração de um carro, considere a especificação $K = \{\alpha, \alpha\beta\} \nsubseteq L_m(\boldsymbol{G})$. Percebe-se que K é controlável com relação a $\boldsymbol{G}$ e, conseqüentemente, o problema de controle supervisório possui solução, sendo que sua solução é o supervisor ilustrado na Figura 2.9, que *marca* os estados 1 e 2 do sistema em malha-fechada. Este supervisor inibe uma arrancada brusca (ocorrência do evento γ) e impõe ao sistema, por meio da marcação, que o carro não deve permanecer parado.

□

Outro tipo de supervisor existente é o supervisor desmarcador, que, conjuntamente com a planta $\boldsymbol{G}$, decide o comportamento marcado para o sistema em malha-fechada e é definido da seguinte forma:

Definição 2.3.2 (Supervisor desmarcador) *Um supervisor desmarcador para $\boldsymbol{G}$ é um par $\mathbb{S} = (\boldsymbol{S}, \Psi)$, sendo $\boldsymbol{S} = (X, \Sigma, \xi, x_0, X_m)$ um gerador e $\Psi : X \times \Sigma \to \{0, 1, dc\}$ uma lei de controle, sendo que o comportamento de $\mathbb{S}/\boldsymbol{G}$ é tal que*

$$L_m(\mathbb{S}/\boldsymbol{G}) := L(\mathbb{S}/\boldsymbol{G}) \cap L_m(\boldsymbol{G}) \cap L_m(\boldsymbol{S})$$

O supervisor desmarcador $\mathbb{S}$ pode *desmarcar* eventuais tarefas que estejam em $L_m(\boldsymbol{G})$, mas não estejam em $K = L_m(\boldsymbol{S})$. O Teorema 2.2.1 também pode ser relaxado para este

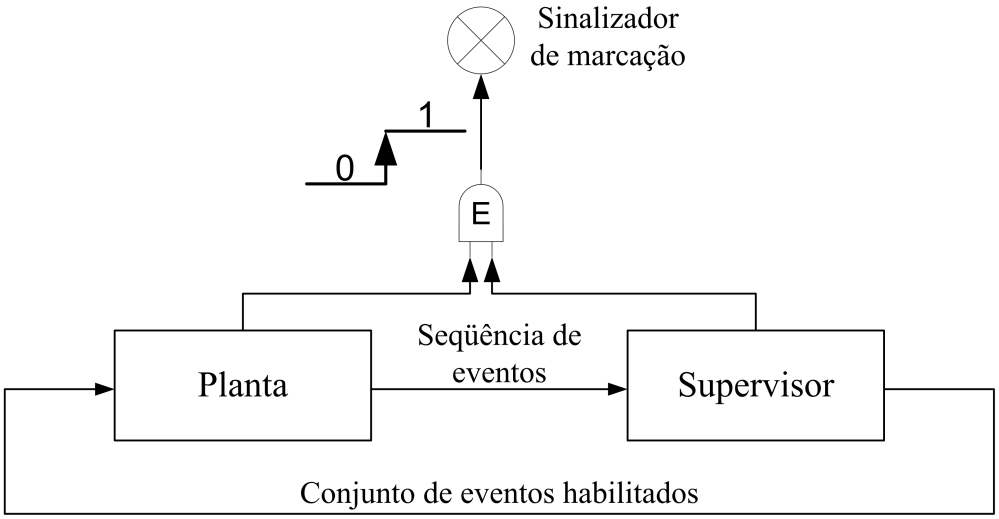

Figura 2.10: Supervisor desmarcador.

caso, eliminando-se a condição de $L_m(\boldsymbol{G})$-fechamento. Para isso, a linguagem K deve ser tal que $K \subseteq L_m(\boldsymbol{S})$. Além disso, K deve ser controlável com relação à planta $\boldsymbol{G}$ para que a existência de $\mathbb{S}$ seja garantida. Assim, o Teorema 2.2.1 pode ser reescrito [55] para o caso do supervisor desmarcador como:

Teorema 2.3.2 *Seja $\emptyset \subset K \subseteq L_m(\boldsymbol{G})$, existe um supervisor desmarcador não-bloqueante $\mathbb{S}$ para $\boldsymbol{G}$, com $L_m(\mathbb{S}/\boldsymbol{G}) = K$ se e somente se K for controlável com relação a $\boldsymbol{G}$.*

Como a marcação do sistema em malha-fechada é definida pelo supervisor e pela planta, tem-se que seu conjunto de estados marcados é $X_m \times Q_m$.

O *sinalizador da marcação* do sistema em malha-fechada é ilustrado na Figura 2.10. Sendo que o sistema somente atinge estados marcados em situações nas quais tanto a planta quanto o supervisor estão em estados marcados simultaneamente.

Exemplo 2.3.2 (Supervisor desmarcador) Novamente considere o Exemplo 2.2.3 da aceleração de um carro com a planta $\boldsymbol{G}$, ilustrada na Figura 2.7, e com o supervisor mostrado na Figura 2.9. Considerando-se $\mathbb{S}$ como sendo desmarcador, tem-se que o gerador que representa o comportamento em malha-fechada de $\mathbb{S}/\boldsymbol{G}$ está na Figura 2.11. É possível perceber que o supervisor obtido a partir de K *desmarcou* o estado 0 da planta, enquanto que a planta *desmarcou* o estado 1 do supervisor, restando como tarefas completas apenas aqueles estados que são marcados tanto em $\boldsymbol{G}$ quanto em $\mathbb{S}$.

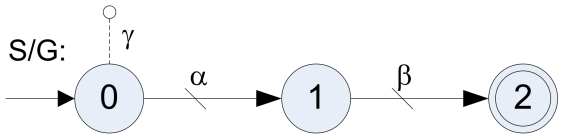

Figura 2.11: Comportamento em malha-fechada $\mathbb{S}/\boldsymbol{G}$.

2.4 Redução de Supervisores Não-Marcadores

Uma vez que se tenha computado o supervisor supremo $\mathbb{S}_{sup} = (\boldsymbol{S}_{sup}, \Psi_{sup})$, sendo $\boldsymbol{S}_{sup} = (X, \Sigma, \xi, x_0, X_m)$ um gerador e $\Psi_{sup} : X \times \Sigma \rightarrow \{0, 1, dc\}$ uma lei de controle, para uma planta $\boldsymbol{G}$ e uma especificação $K = L_m(\boldsymbol{E}) \cap L_m(\boldsymbol{G})$, é possível encontrar um supervisor reduzido $\hat{\mathbb{S}} = (\hat{\boldsymbol{S}}, \hat{\Psi})$ cuja ação de controle sobre a planta é a mesma que a de $\mathbb{S}_{sup}$, porém com o supervisor reduzido possuindo um gerador $\hat{\boldsymbol{S}}$ com um número de estados[4] muito menor. Ou seja, busca-se um supervisor reduzido tal que

$$|\hat{\boldsymbol{S}}| \ll |\boldsymbol{S}_{sup}|$$

e que as seguintes propriedade de *controle-equivalência* [50] sejam respeitadas.

Definição 2.4.1 (Controle-equivalência) *Diz-se que um supervisor reduzido $\hat{\mathbb{S}}$ é controle-equivalente a $\mathbb{S}_{sup}$, em relação a $\boldsymbol{G}$ se*

$$\begin{aligned} L(\hat{\boldsymbol{S}}/\boldsymbol{G}) &= L(\boldsymbol{S}_{sup}/\boldsymbol{G}) \\ L_m(\hat{\boldsymbol{S}}/\boldsymbol{G}) &= L_m(\boldsymbol{S}_{sup}/\boldsymbol{G}) \end{aligned}$$

Em palavras, um supervisor $\hat{\mathbb{S}}$ é controle-equivalente ao supervisor supremo $\mathbb{S}_{sup}$ de uma planta $\boldsymbol{G}$ sempre que a supervisão de $\boldsymbol{G}$ por meio de $\mathbb{S}_{sup}$ implica os mesmos resultados que a supervisão de $\boldsymbol{G}$ por meio de $\hat{\mathbb{S}}$.

Exemplo 2.4.1 Para a planta e supervisor supremo apresentados no Exemplo 2.2.1, pode-se construir um supervisor reduzido que é controle equivalente, conforme a Figura 2.12.

□

[4] A notação $|\boldsymbol{G}|$ indica o número de estados do gerador $\boldsymbol{G}$.

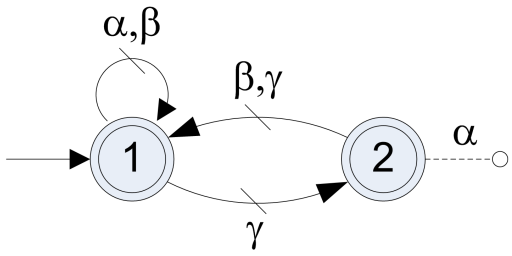

Figura 2.12: $\hat{\mathbb{S}}$

Existem diversos supervisores reduzidos, com diferentes tamanhos de $\hat{\boldsymbol{S}}$, e é interessante obter um tal que o número de estados de $\hat{\boldsymbol{S}}$ seja o menor possível. Muitas vezes, quanto menor é o número de estados do gerador de um supervisor, mais fácil se torna compreender sua ação de controle sobre a planta e também mais fácil, em termos computacionais, é implementá-lo.

Problema 2.4.1 (Supervisor mínimo) *Dados uma planta $\boldsymbol{G}$ e um supervisor supremo $\mathbb{S}_{sup}$, encontrar um supervisor reduzido $\hat{\mathbb{S}} = (\hat{\boldsymbol{S}}, \hat{\Psi})$, tal que $\hat{\boldsymbol{S}}$ possua o número mínimo de estados e seja controle-equivalente a $\mathbb{S}_{sup}$, com relação a $\boldsymbol{G}$.*

No entanto, o Problema 2.4.1 é NP-Difícil [49], o que o torna intratável em várias de suas instâncias. Não há registros de algoritmos tradicionais que tenham minimizado supervisores com mais de dois mil estados, que é um tamanho de supervisor que pode ser obtido facilmente para sistemas compostos reais.

A Figura 2.13 ilustra as relações entre linguagens [49], onde Σ^* contém todas as linguagens. Dadas a linguagem marcada $L_m(\boldsymbol{G})$ da planta e uma especificação $K = L_m(\boldsymbol{G}) \cap L_m(\boldsymbol{E})$, tem-se que K não necessariamente é controlável. Contida em K, está a máxima linguagem controlável $supC(K, \boldsymbol{G})$. Um supervisor reduzido $\hat{\mathbb{S}}$ possui uma linguagem marcada $L_m(\hat{\boldsymbol{S}})$, tal que sua intersecção com $L_m(\boldsymbol{G})$ resulta na máxima linguagem controlável, isto é, em termos de comportamento, a planta supervisionada por $\hat{\mathbb{S}}$ respeita a máxima linguagem controlável.

Em seu artigo, Vaz e Wonham [50] definiram *cobertura de controle* (definida a seguir) e obtiveram dois importantes resultados:

- A partir de qualquer cobertura de controle é possível se obter um supervisor reduzido.
- Qualquer supervisor reduzido pode ser obtido por meio de uma cobertura de controle.

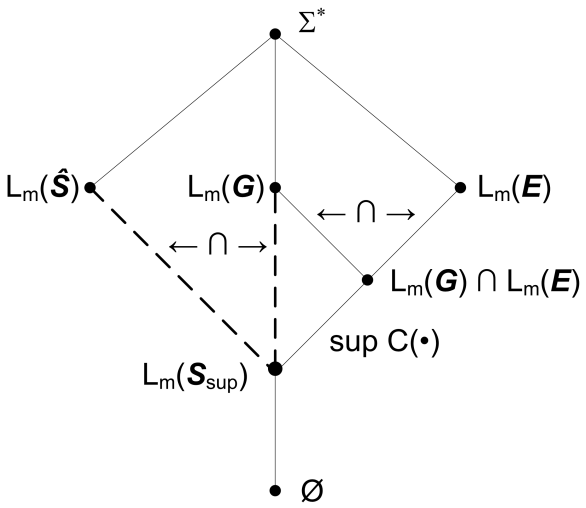

Figura 2.13: Diagrama Hasse de Linguagens [49].

Uma cobertura [Apêndice A] sobre o conjunto de estados do gerador $\boldsymbol{S}_{sup}$ do supervisor supremo $\mathbb{S}_{sup} = (\boldsymbol{S}_{sup}, \Psi)$ é uma *cobertura de controle* $C = \{X_i, i \in I\}$, tal que $X_i \subseteq X$ e I é um conjunto de índices, desde que as seguintes propriedades sejam respeitadas:

1. $\forall (i \in I) X_i \neq \emptyset$.
2. $\bigcup_{i \in I} X_i = X$ (*Cobertura completa*).
3. $\forall (i \in I, \sigma \in \Sigma) \exists y \in X_i : \xi(y, \sigma)! \rightarrow \exists j \in I : \forall x \in X_i \big(\xi(x, \sigma)! \rightarrow \xi(x, \sigma) \in X_j\big)$ (*Determinismo*).
4. $\forall (i \in I, \sigma \in \Sigma) \forall (x, y \in X_i) \big((\Psi(x, \sigma) \neq dc \neq \Psi(y, \sigma))\big) \rightarrow \big(\Psi(x, \sigma) = \Psi(y, \sigma)\big)$ (*Consistência da ação de controle*).

As propriedades 1 e 2 garantem que uma *cobertura de controle* seja realmente uma cobertura.

A Propriedade 3 (*determinismo*) garante que não sejam agrupados estados que, ao se construir o supervisor reduzido, impliquem que a ocorrência de um evento a partir de um estado leve a múltiplos estados. A Figura 2.14 ilustra um caso no qual a propriedade de determinismo é respeitada, pois foi possível agrupar, em um elemento da cobertura X_j, todos os estados que são destinos das transições originadas a partir de um elemento da cobertura X_i com um evento σ. Ou seja, a ocorrência do evento σ, a partir de qualquer estado em X_i, leva

o sistema a algum estado em X_j. Para que a propriedade de determinismo seja respeitada, o mesmo deve acontecer para qualquer elemento da cobertura e qualquer evento do sistema.

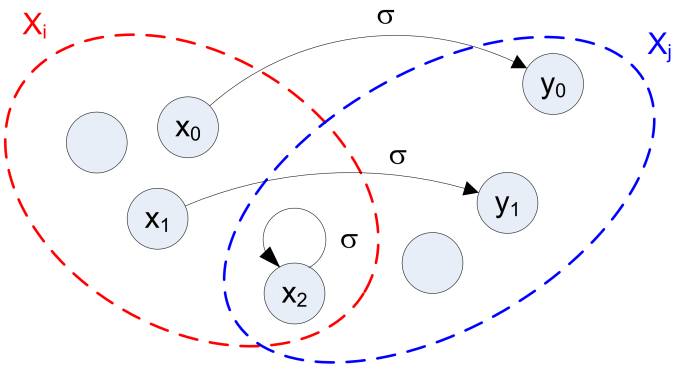

Figura 2.14: Consistência de determinismo.

A Propriedade 4 (*consistência da ação de controle*) assegura que todos os estados agrupados possuam ação de controle consistente, isto é, que não habilitem e desabilitem simultaneamente o mesmo evento.

A partir de uma *cobertura de controle* C tal que $|C| < |Q|$, onde $|\bullet|$ representa o número de elementos de seu argumento, é possível se obter um supervisor reduzido $\hat{\mathbb{S}} = (\hat{\boldsymbol{S}}, \hat{\Psi})$ da seguinte forma:

- $\hat{\boldsymbol{S}} := (I, \Sigma, \hat{\xi}, i_0, I_m)$;
- seleciona-se $i_0 \in I$ tal que $x_0 \in X_{i_0}$;
- define-se $\hat{\xi} : I \times \Sigma \rightarrow I$ (função parcial) como: $\forall (i \in I, \sigma \in \Sigma)$, se $\xi(x, \sigma)!$ para algum $x \in X_i$, seleciona-se $j \in I$ tal que $\xi(x, \sigma) \in X_j$ para tal x. Define-se então $\hat{\xi}(i, \sigma) := j$;
- $I_m = I$[5].
- $\hat{\Psi} : I \times \Sigma \rightarrow \{0, 1, dc\}$ é definida como:
  para $i \in I, \sigma \in \Sigma$, se existe $x \in X_i$ tal que $\Psi(x, \sigma) \neq dc$ então $\hat{\Psi}(i, \sigma) := \Psi(x, \sigma)$, caso contrário $\hat{\Psi}(i, \sigma) := dc$;

Percebe-se pela definição acima que existem etapas na construção do supervisor reduzido nas quais há mais de uma possibilidade de escolha a ser tomada, por exemplo o

[5] A marcação, neste caso, é imposta pela planta. Sempre que a planta atingir um estado marcado $q \in Q_m$ deve-se considerar uma tarefa completada.

estado inicial $i_0 \in I$ pode ser qualquer um desde que $x_0 \in X_{i_0}$. Isto demonstra que uma mesma cobertura de controle pode gerar diferentes supervisores, no que diz respeito a $i_0, \hat{\xi}$ e $\hat{\Psi}$, mas não ao número de estados, que será $|C|$.

Exemplo 2.4.2 O supervisor reduzido exibido na Figura 2.12 pode ser obtido a partir da cobertura de controle $C = \{\{1,2\},\{1,3\}\}$ sobre o supervisor supremo ilustrado no Exemplo 2.2.1.

□

2.5 Redução com marcação

Durante a síntese de supervisores, existem basicamente três abordagens definindo o papel da marcação: *supervisor não-marcador*, *supervisor marcador* e *supervisor desmarcador*. Em todos os casos de síntese, as considerações sobre bloqueio e existência de um supervisor variam, conforme explicitado nas seções anteriores. Porém, uma vez calculado o supervisor para uma dada especificação utilizando-se uma das três abordagens anteriores, a marcação pode passar a ser desnecessária para efeitos de implementação.

A influência da marcação na implementação de um supervisor é um aspecto importante e deve ser levado em conta. Existem diversos casos [14, 43, 51] em que a marcação somente é útil no processo de síntese, sendo completamente ignorada na implementação. Pois, uma vez realizada a síntese, obtendo-se um sistema com comportamento em malha-fechada controlável e não-bloqueante, a informação de que um estado do supervisor é ou não marcado pouco importa. Casos como este possuem uma *implementação não-sensível à marcação*. Porém pode haver casos, denominados *implementação sensível à marcação*, nos quais é necessária a informação de quais estados do supervisor são marcados.

2.5.1 Redução não-sensível à marcação

Quando a informação de quais estados são marcados no supervisor é desnecessária, tem-se o caso de redução não-sensível à marcação. O caso do supervisor não-marcador claramente se enquadra nesta categoria, pois a marcação do supervisor pouco importa para determinar a marcação do sistema em malha-fechada. Além disso, para os demais tipos de supervisores, tem-se que, caso a implementação seja não-sensível à marcação, a redução de supervisores

pode ser feita conforme explicitado na seção 2.4, baseada somente no supervisor supremo calculado a priori e ignorando-se a marcação.

Ao se ignorar a marcação durante a redução, pode-se obter supervisores menores do que aqueles com condições adicionais para a agregação de estados em coberturas.

2.5.2 Redução sensível à marcação

Quando a implementação necessita a informação de quais estados do sistema em malha-fechada são marcados, não se pode perder esta informação durante a redução e, dependendo do tipo de supervisor que se tem, não se podem agregar estados marcados com estados não-marcados descriteriosamente.

Vaz e Wonham, em [50], fazem considerações para o caso em que o supervisor é marcador. Naquele trabalho, consideram que a informação da marcação do sistema em malha-fechada deve ser preservada após a redução e propõem uma condição para a formação de coberturas de controle que inibe qualquer agregação de estados que não possuam marcação equivalente, segundo a propriedade 2.5.1.

Propriedade 2.5.1 (Consistência da marcação)

$$\exists(I_m \subseteq I) : \left[(X_m = \bigcup_{i \in I_m} X_i) \wedge (X - X_m = \bigcup_{i \in (I - I_m)} X_i)\right]$$

Esta propriedade considera que não se podem agregar quaisquer estados que sejam marcados no gerador $\boldsymbol{S}_{sup}$ do supervisor supremo $\mathbb{S}_{sup}$ com estados não-marcados, independentemente de sua marcação na planta $\boldsymbol{G}$. Pois, uma vez que o supervisor é marcador, seus estados levam sozinhos a informação da marcação e não podem ser agregados entre si caso possuam marcações diferentes.

Exemplo 2.5.1 (Marcação segundo Vaz e Wonham [50]) Considere a planta $\boldsymbol{G}$, com $\Sigma = \Sigma_u = \{\alpha, \beta, \gamma, \tau\}$, representada na Figura 2.15 e o supervisor $\mathbb{S}$ da Figura 2.16. Note que o supervisor não inibe a ocorrência de nenhum evento não-controlável e atua somente sobre a marcação do sistema em malha-fechada. Considerando $\mathbb{S}$ um supervisor marcador, tem-se que a marcação do sistema em malha-fechada é igual a do supervisor. Para formar coberturas de controle e reduzir o supervisor, pode-se agregar somente o estado $0 \in X_m$ em conjunto com o estado $1 \in X_m$ e o estado $2 \in X - X_m$ em conjunto com o estado $3 \in X - X_m$, de forma

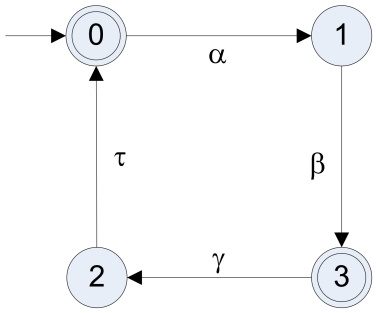

Figura 2.15: Planta $\boldsymbol{G}$.

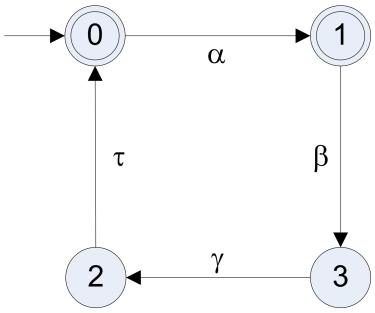

Figura 2.16: Supervisor $\mathbb{S}$.

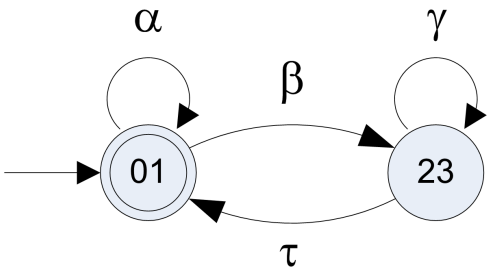

Figura 2.17: Supervisor reduzido $\hat{\mathbb{S}}$ (Caso Vaz e Wonham).

a respeitar a propriedade de consistência da marcação. A Figura 2.17 ilustra o supervisor reduzido $\hat{\mathbb{S}}$ obtido a partir da cobertura de controle $C = \{\{0,1\},\{2,3\}\}$.

□

Em um trabalho posterior [49], Su e Wonham propõem condições de agregação de estados para o caso de supervisores desmarcadores, isto é $L_m(\mathbb{S}_{sup}/\boldsymbol{G}) := L(\mathbb{S}_{sup}/\boldsymbol{G}) \cap L_m(\boldsymbol{G}) \cap L_m(\boldsymbol{S})$. Eles consideram que, ao se agregar estados de $\boldsymbol{S}_{sup}$ em elementos X_i para formar uma cobertura de controle C, deve haver consistência na marcação dos estados $x, x' \in X$ agregados sempre que $T(x) = T(x')$, sendo $T : X \to \{0,1\}$ definida da seguinte forma:

$$T(x) = \begin{cases} 1 & \text{se } \exists s \in \Sigma^*(\xi(x_0, s) = x \wedge \delta(q_0, s) \in Q_m) \\ 0 & \text{caso contrário} \end{cases}$$

Isto é, a consistência de marcação dos estados $x, x' \in X$ de $\boldsymbol{S}_{sup}$ agregados deve respeitar sua relação com a marcação da planta $\boldsymbol{G}$. Estados $x_m \in X_m$ podem ser agregados em dois grupos: um grupo que é acessível por cadeias que levam a planta a estados marcados e outro

que é acessível por cadeias que levam a planta somente a estados não-marcados. O mesmo vale para os estados $x \in X - X_m$. Logo, ao invés de somente duas relações de agrupamento de estados com relação à consistência de marcação, conforme Vaz e Wonham [50] propõem, Su e Wonham [49] permitem quatro relações de agrupamento para que a consistência de marcação seja preservada.

Exemplo 2.5.2 (Marcação segundo Su e Wonham [49]) Analisando ainda a mesma planta e o mesmo supervisor do Exemplo 2.5.1, porém considerando o supervisor $\mathbb{S}$ como sendo desmarcador. Tem-se que $\mathbb{S}/\boldsymbol{G}$ é semelhante a $\boldsymbol{G}$, porém somente com o estado 0 marcado, que é marcado tanto em $\boldsymbol{G}$ quanto em $\mathbb{S}$. O valor da propriedade $T(x)$ para cada estado $x \in X$ é: $T(0) = 1$, $T(1) = 0$, $T(2) = 0$ e $T(3) = 1$. Logo não é possível agregar nenhum estado, pois ao se tentar agregar os estados $0, 1 \in X_m$, tem-se que $T(0) \neq T(1)$ e para o caso de $2, 3 \in X - X_m$, tem-se que $T(2) \neq T(3)$. Logo, tem-se que a única cobertura de controle possível é $C = \{\{0\}, \{1\}, \{2\}, \{3\}\}$ e que o supervisor reduzido $\hat{\mathbb{S}}$ obtido a partir de C é tal que $\hat{\mathbb{S}} = \mathbb{S}$.

□

Posteriormente, Sivolella et. al. [48] propõem um relaxamento para a condição de agregação de estados de supervisores desmarcadores. Naquele trabalho, os autores introduzem o atributo de ação de marcação $A : X \to \{-1, 0, 1\}$ de um estado de um supervisor $\mathbb{S}$ como

$$A(x) = \begin{cases} 1, & \text{se } T(x) = 1 \wedge x \in X_m, \quad \text{estado marcador} \\ 0, & \text{se } T(x) = 0, \quad \text{estado indiferente} \\ -1, & \text{se } T(x) = 1 \wedge x \in X - X_m, \quad \text{estado desmarcador} \end{cases}$$

e restringem a agregação de estados, com relação à marcação, permitindo que estados marcadores sejam agregados somente com estados marcadores ou indiferentes e que estados desmarcadores o sejam somente com estados desmarcadores ou indiferentes. A novidade nesta abordagem é a flexibilidade concedida a estados indiferentes que podem ser agregados a estados marcadores, desmarcadores ou indiferentes.

Exemplo 2.5.3 (Marcação segundo Sivolella et. al. [48]) Considerando o mesmo caso exposto no Exemplo 2.5.2, tem-se que o valor do atributo da ação de marcação A para cada estado de X é: $A(0) = 1$, $A(1) = 0$, $A(2) = 0$, $A(3) = -1$. Logo é possível agregar entre si os estados $0, 1, 2 \in X$ ou $1, 2, 3 \in X$. Os estados $1, 2 \in X$ podem ser agregados tanto ao estado marcador 0 quanto ao estado desmarcador 3, porém os estados 1 e 3 não podem ser

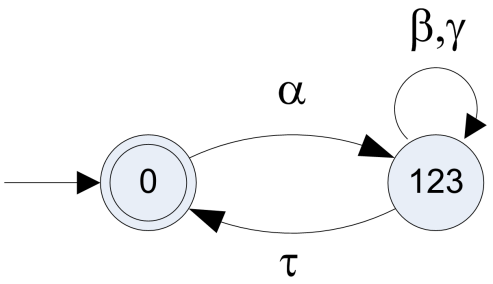

Figura 2.18: Supervisor reduzido $\hat{\mathbb{S}}$ (Caso Sivolella et. al.).

agregados entre si por possuírem ações de marcação conflitantes. A Figura 2.18 ilustra o supervisor reduzido $\hat{\mathbb{S}}$ obtido a partir da cobertura de controle $C = \{\{0\}, \{1, 2, 3\}\}$.

□

A proposta de Vaz e Wonham se aplica a supervisores marcadores, enquanto que as seguintes se aplicam a supervisores desmarcadores, dentre as quais, a última é mais flexível, sem restringir excessivamente a agregação de estados no que diz respeito à marcação. Porém, a utilização de qualquer uma destas abordagens introduz a necessidade da implementação de mais uma condição em um algoritmo de minimização de supervisores que considere a marcação. O próximo capítulo apresenta uma metodologia para evitar a utilização explícita da marcação nos casos de redução sensível à marcação, facilitando a elaboração e implementação de algoritmos de minimização.

Capítulo 3

Marcação de estados por eventos

A marcação de estados, dependendo do tipo de supervisor utilizado, pode tornar necessária a utilização de uma condição para verificar a consistência da marcação durante a redução, conforme visto na Seção 2.5. Além disso, pode exigir a verificação da propriedade de $L_m(\boldsymbol{G})$-fechamento para uma especificação K durante a síntese de um supervisor do tipo tradicional (Seção 2.2). Este capítulo apresenta uma contribuição original desta dissertação de mestrado, propondo uma abordagem alternativa para substituir a marcação de estados por um evento de marcação $\mu \notin \Sigma$, conservando o significado da marcação, eliminando a necessidade de se verificar a condição de $L_m(\boldsymbol{G})$-fechamento durante o cálculo de um supervisor e a necessidade da verificação de condições de consistência da marcação durante a redução de qualquer tipo de supervisor.

3.1 O evento de marcação μ

Para representar a marcação de estados de um gerador $\boldsymbol{G} = (Q, \Sigma, \delta, q_0, Q_m)$ por meio do evento de marcação $\mu \notin \Sigma$, constrói-se um novo gerador $\boldsymbol{G}^\mu := (Q, \Sigma^\mu, \delta^\mu, q_0, Q)$, onde $\Sigma^\mu = \Sigma \cup \{\mu\}$ e

$$\delta^\mu(q,\sigma) := \begin{cases} \delta(q,\sigma) & \text{se } \sigma \neq \mu \\ q & \text{se } \sigma = \mu \wedge q \in Q_m \\ \text{indefinido} & \text{caso contrário} \end{cases}$$

Desta forma, a cada estado marcado de $\boldsymbol{G}$ é adicionado um auto-laço do evento μ e em seguida todos os seus estados são marcados, resultando em $\boldsymbol{G}^\mu$. Claramente $L(\boldsymbol{G}) \subseteq L(\boldsymbol{G}^\mu)$, $L(\boldsymbol{G}^\mu) = L_m(\boldsymbol{G}^\mu)$ e é possível *retornar* de Σ^μ para Σ por meio de projeções naturais $P_\Sigma : \Sigma^{\mu *} \to \Sigma^*$, que são definidas como:

$$\begin{aligned} P_\Sigma(\varepsilon) &:= \varepsilon \\ P_\Sigma(s\sigma) &:= \begin{cases} P_\Sigma(s) & \text{se } \sigma \notin \Sigma \\ P_\Sigma(s)\sigma & \text{se } \sigma \in \Sigma \end{cases} \end{aligned}$$

A projeção natural $P_\Sigma(s)$ *apaga* todos os eventos $\sigma \notin \Sigma$ da cadeia $s \in \Sigma^{\mu *}$. Considera-se que quando o argumento da projeção é uma linguagem, o resultado é uma linguagem com a projeção aplicada a cada um de seus elementos. Logo, tem-se que

$$\begin{aligned} L(\boldsymbol{G}) &= P_\Sigma(L(\boldsymbol{G}^\mu)) \\ L_m(\boldsymbol{G}) &= P_\Sigma(\{s\mu \in L(\boldsymbol{G}^\mu) : s \in \Sigma^{\mu *}\}) \end{aligned}$$

O evento de marcação μ pode ser considerado controlável ou não-controlável, dependendo do tipo de supervisor que se esteja utilizando, conforme exposto nas próximas seções.

Como a informação da marcação em G^μ está na ocorrência dos eventos de marcação μ, deve-se considerar a μ-acessibilidade ao invés da co-acessibilidade (conforme definida na Seção 2.1).

Definição 3.1.1 (μ-acessibilidade) *Um estado $q \in Q$ é μ-acessível se existe $s \in \Sigma^{\mu *}$ tal que $\delta^\mu(q,s) = y$ e $\delta^\mu(y,\mu)!$. Ademais, um gerador $\boldsymbol{G}^\mu$ é considerado μ-acessível se q é μ-acessível para todo $q \in Q$.*

Em palavras, um estado é μ-acessível se, a partir dele, é possível se chegar a um estado

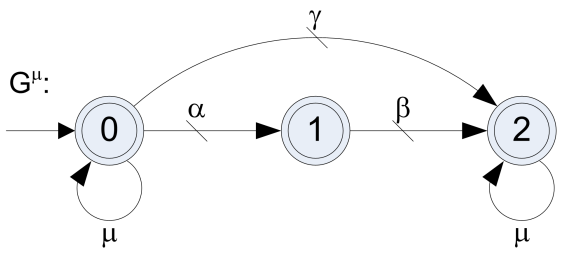

Figura 3.1: Aceleração de um carro com o evento de marcação.

tal que o evento de marcação μ esteja definido. Um gerador é μ-acessível se todos os seus estados são μ-acessíveis. Além disso, G^μ é considerada μ-*trim* caso seja acessível e μ-acessível.

Exemplo 3.1.1 (Construindo G^μ) Considere o Exemplo 2.2.3 da aceleração de um carro com a planta $\boldsymbol{G}$, ilustrada na Figura 2.7. Ao seguir os procedimentos citados anteriormente obtém-se o gerador $\boldsymbol{G}^\mu$, ilustrado na Figura 3.1. Claramente todas as cadeias de $L(\boldsymbol{G})$ estão presentes em $L(\boldsymbol{G}^\mu)$ e, ainda, as cadeias de $L(\boldsymbol{G}^\mu)$ terminadas com o evento μ levam a um estado que era marcado em $\boldsymbol{G}$, logo a projeção em Σ destas cadeias resulta em $L_m(\boldsymbol{G})$. Além disso, $\boldsymbol{G}^\mu$ é μ-acessível, pois, a partir de qualquer estado, é possível atingir um estado tal que o evento de marcação μ está definido.

□

3.2 Supervisores não-marcadores

Uma das vantagens de se utilizar eventos μ de marcação sobre a marcação tradicional é a eliminação da necessidade de verificar a condição de $L_m(\boldsymbol{G})$-fechamento durante a síntese de supervisores tradicional. Neste caso, ao invés de se utilizar um planta $\boldsymbol{G}$, utiliza-se $\boldsymbol{G}^\mu$ e, no lugar de uma especificação (Definição 2.2.6) $K = L_m(\boldsymbol{E})$, faz-se uso de $K^\mu := L(\boldsymbol{E}^\mu)$. Ao se utilizar K^μ e $\boldsymbol{G}^\mu$, considerando μ como um evento não-controlável, basta que K^μ seja controlável com relação à planta $\boldsymbol{G}^\mu$ para que a existência de um supervisor $\mathbb{S}$ seja garantida.

Proposição 3.2.1 *Dados um planta $\boldsymbol{G}$, uma especificação $K \subseteq L_m(\boldsymbol{G})$, então K é $L_m(\boldsymbol{G})$-fechada e controlável com relação a $\boldsymbol{G}$ se e somente se K^μ é controlável com relação a $\boldsymbol{G}^\mu$, sendo μ um evento não-controlável.*

Demonstração:

A demonstração desta proposição é feita por contradição, primeiramente demostrando-se a parte da implicação, baseando-se na Definição 2.2.7 de controlabilidade e na Definição 2.2.9 de $L_m(\boldsymbol{G})$-fechamento:

$$[(K = \overline{K} \cap L_m(\boldsymbol{G})) \wedge (\overline{K}\Sigma_u \cap L(\boldsymbol{G}) \subseteq \overline{K})] \rightarrow (\overline{K^\mu}\Sigma_u^\mu \cap L(\boldsymbol{G}^\mu) \subseteq \overline{K^\mu})$$

Assume-se que:
$(K = \overline{K} \cap L_m(\boldsymbol{G})) \wedge (\overline{K}\Sigma_u \cap L(\boldsymbol{G}) \subseteq \overline{K}) \wedge (\overline{K^\mu}\Sigma_u^\mu \cap L(\boldsymbol{G}^\mu) \subsetneq \overline{K^\mu})$
Logo:
$\exists(t \in \overline{K^\mu})$ e $(\sigma \in \Sigma_u^\mu)$ tais que $(t\sigma \in L(\boldsymbol{G}^\mu)) \wedge (t\sigma \notin \overline{K^\mu})$
Seja $(r = P_\Sigma(t) \in \overline{K^\mu}) \rightarrow r \in \overline{K}$, pois, de acordo com a definição de δ^μ, é possível perceber que eventuais ocorrências do evento μ não causam mudança de estados. Logo, os casos em que a ocorrência de μ importa são cobertos durante a análise de $\sigma = \mu$.

Primeiramente analisa-se o caso em que $\sigma \neq \mu$, ou seja, $\sigma \in \Sigma_u$:
Como $t\sigma \in L(\boldsymbol{G}^\mu) \rightarrow P_\Sigma(t\sigma) \in P_\Sigma(L(\boldsymbol{G}^\mu)) \rightarrow r\sigma \in L(\boldsymbol{G})$
Por contradição, $r \in \overline{K} \wedge \sigma \in \Sigma_u \wedge r\sigma \in L(\boldsymbol{G}) \rightarrow r\sigma \in \overline{K}$
Mas $r\sigma \in \overline{K} \rightarrow t\sigma \in \overline{K}^\mu$, o que contradiz a hipótese para $\sigma \neq \mu$.

Caso $\sigma = \mu$ tem-se que, para todo $s \in \overline{K}$:
$s \in \overline{K} \rightarrow s \in \overline{K^\mu}$
$s \in L_m(\boldsymbol{G}) \rightarrow s\mu \in L(\boldsymbol{G}^\mu)$
$(s \in \overline{K}) \wedge (s \in L_m(\boldsymbol{G})) \rightarrow s \in K \rightarrow s\mu \in K^\mu \rightarrow s\mu \in \overline{K^\mu}$, que contradiz $\exists t\mu \notin \overline{K^\mu}$.

Agora demonstra-se a replicação, também por contradição:

$$(\overline{K^\mu}\Sigma_u^\mu \cap L(\boldsymbol{G}^\mu) \subseteq \overline{K^\mu}) \rightarrow [(K = \overline{K} \cap L_m(\boldsymbol{G})) \wedge (\overline{K}\Sigma_u \cap L(\boldsymbol{G}) \subseteq \overline{K})]$$

Assume-se que:
$(\overline{K^\mu}\Sigma_u^\mu \cap L(\boldsymbol{G}^\mu) \subseteq \overline{K^\mu}) \wedge [(K \neq \overline{K} \cap L_m(\boldsymbol{G})) \vee (\overline{K}\Sigma_u \cap L(\boldsymbol{G}) \subsetneq \overline{K})]$
Logo, uma das possibilidades seria:
$\exists t\sigma \notin \overline{K}$, tal que $t \in \overline{K}, \sigma \in \Sigma_u, t\sigma \in L(\boldsymbol{G})$
Mas:
$t \in \overline{K} \rightarrow t \in \overline{K^\mu}$
$\sigma \in \Sigma_u \rightarrow \sigma \in \Sigma_u^\mu$
$t\sigma \in L(\boldsymbol{G}) \rightarrow t\sigma \in L(\boldsymbol{G}^\mu)$

$t\sigma \notin \overline{K} \rightarrow t\sigma \notin \overline{K}^{\mu}$
Que contradiz com $\forall s \in \overline{K^{\mu}}, \sigma \in \Sigma_u^{\mu}, s\sigma \in L(\boldsymbol{G}^{\mu}) \rightarrow s\sigma \in \overline{K^{\mu}}$.
Outra possibilidade seria: $\exists(t \in \overline{K})$ tal que $(t \in L_m(\boldsymbol{G})) \wedge (t \notin K)$
Porém, para todo $s \in \Sigma^*$:
$s \in \overline{K} \rightarrow s \in \overline{K^{\mu}}$
$s \in L_m(\boldsymbol{G}) \rightarrow s\mu \in L(\boldsymbol{G}^{\mu})$
$(s \in \overline{K^{\mu}}) \wedge (s\mu \in L(\boldsymbol{G}^{\mu})) \wedge (\mu \in \Sigma_u^{\mu}) \rightarrow s\mu \in \overline{K^{\mu}}$
Mas $s \in \overline{K} \wedge s\mu \in \overline{K^{\mu}} \rightarrow s \in K$, que contradiz a hipótese.

□

A demonstração acima mostra que a propriedade da controlabilidade de K com relação a $\boldsymbol{G}$ é *transmitida* para K^{μ} com relação a $\boldsymbol{G}^{\mu}$ para todos os eventos diferentes do evento marcador μ. Por outro lado, eventuais falhas na controlabilidade em $\boldsymbol{G}^{\mu}$ do evento μ representam violações de $L_m(\boldsymbol{G})$-fechamento em $\boldsymbol{G}$ e vice-versa.

Utilizando-se o evento de marcação μ como um evento não-controlável e os geradores $\boldsymbol{G}^{\mu}$ e $\boldsymbol{E}^{\mu}$ é possível ignorar a marcação[1] dos estados para calcular um supervisor $\mathbb{S}^{\mu}$, do qual é possível obter $\mathbb{S}$. Para tal, basta que se calcule a componente μ-trim do supervisor $\mathbb{S}^{\mu}$, com $L_m(\mathbb{S}^{\mu}/\boldsymbol{G}^{\mu}) = K^{\mu}$, considerando $K^{\mu} \subseteq L(\boldsymbol{G}^{\mu})$. A informação da marcação de $\mathbb{S}$ pode ser recuperada por meio dos estados em $\boldsymbol{S}^{\mu}$ que possuírem uma transição do evento μ. Tais estados representam os estados marcados.

Para efeitos de redução, no caso do supervisor não-marcador, a marcação do supervisor é ignorada, conforme explicitado na Seção 2.4. Logo, a vantagem em se utilizar o evento de marcação μ é a possibilidade de não ser necessário verificar a propriedade de $L_m(\boldsymbol{G})$-fechamento diretamente. Além disso, é possível encontrar a máxima linguagem controlável e $L_m(\boldsymbol{G})$-fechada para uma especificação $K = L_m(\boldsymbol{E})$.

Exemplo 3.2.1 (Controlabilidade de $\mu \in \Sigma_u^{\mu}$) Considere novamente o Exemplo 2.2.3 da aceleração de um carro. A Figura 3.2 ilustra o gerador $\boldsymbol{E}^{\mu}$, cuja linguagem gerada equivale à especificação K^{μ} que deve ser aplicada à planta $\boldsymbol{G}^{\mu}$ (Figura 3.1). Ao se tentar computar o supervisor $\mathbb{S}^{\mu}$, há uma violação de controlabilidade no estado 0, pois o gerador $\boldsymbol{E}^{\mu}$ inibe a ocorrência do evento não-controlável μ enquanto que ele é possível na planta $\boldsymbol{G}^{\mu}$. Há também uma violação de restrições no estado zero no cálculo do supervisor $\mathbb{S}$ de $K = L_m(\boldsymbol{E})$ com

[1] Ignorar a marcação, ou considerar todos os estados de $\boldsymbol{G}^{\mu}$ e $\boldsymbol{E}^{\mu}$ como estados marcados.

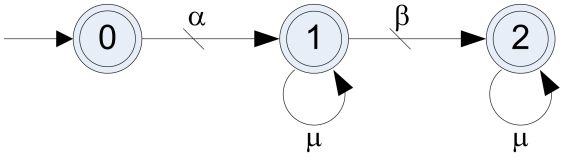

Figura 3.2: Gerador de especificação $\boldsymbol{E}^{\mu}$ com o evento de marcação.

relação a $\boldsymbol{G}$, porém esta restrição é de $L_m(\boldsymbol{G})$-fechamento. Logo a violação da condição de $L_m(\boldsymbol{G})$-fechamento implica uma violação de controlabilidade de μ e vice-versa.

3.3 Supervisores marcadores

No caso de supervisores marcadores, tem-se que a marcação do sistema é imposta pelo supervisor. Conforme exposto na Seção 2.3, a única condição necessária para a existência de um supervisor $\mathbb{S}$ com ação de marcação para uma especificação $K \subseteq L_m(\boldsymbol{G})$ é a de controlabilidade. Logo, a abordagem para que se calcule um supervisor por meio de $\boldsymbol{G}^{\mu}$ e K^{μ} requer que o evento $\mu \notin \Sigma$ seja controlável, pois não há necessidade de verificar a condição de $L_m(\boldsymbol{G})$-fechamento e, ainda, é preciso fazer com que o supervisor *habilite* a marcação em determinados estados de $\boldsymbol{G}$.

Como neste caso o supervisor decide sozinho a marcação da planta, é necessário que a planta $\boldsymbol{G}$ possua todos os seus estados marcados, fazendo com que $Q_m = Q$, para que a planta não restrinja a marcação durante o cálculo do supervisor. Em posse de uma planta com $Q_m = Q$, constrói-se $\boldsymbol{G}^{\mu}$, K^{μ}, com $\mu \in \Sigma_c^{\mu}$ e calcula-se um supervisor $\mathbb{S}^{\mu}$ ignorando a marcação e do qual é possível obter $\mathbb{S}$.

Proposição 3.3.1 *Dados uma planta $\boldsymbol{G}$, uma especificação $K \subseteq L_m(\boldsymbol{G})$, então K é controlável com relação a $\boldsymbol{G}$ se e somente se K^{μ} é controlável em relação a $\boldsymbol{G}^{\mu}$, sendo μ um evento controlável.*

Prova:

A demonstração desta proposição é elementar, pois a condição de controlabilidade de K com relação a $\boldsymbol{G}$ é transmitida para K^{μ} e $\boldsymbol{G}^{\mu}$ para todo $\sigma \in \Sigma$, restando apenas o evento μ que é controlável, logo não causa conflitos de controlabilidade. Por outro lado se K^{μ} é controlável com relação a $\boldsymbol{G}^{\mu}$ então K também é controlável com relação a $\boldsymbol{G}$, pois $\Sigma_u = \Sigma_u^{\mu}$.

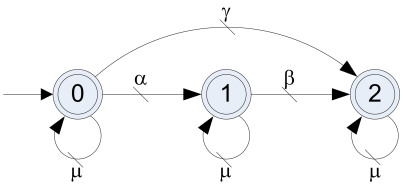

Figura 3.3: Planta $\boldsymbol{G}^{\mu}$

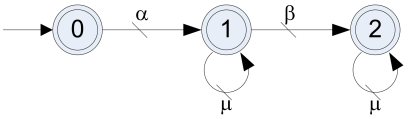

Figura 3.4: Gerador da especificação $\boldsymbol{E}^{\mu}$

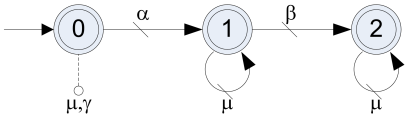

Figura 3.5: Supervisor $\mathbb{S}^{\mu}$

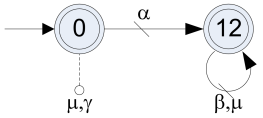

Figura 3.6: Supervisor reduzido $\hat{\mathbb{S}}^{\mu}$

□

Para efeitos de redução, é possível também ignorar a marcação e considerar simplesmente a condição de ação de controle do evento μ em $\mathbb{S}^{\mu}$ para a agregação de estados. Estados que habilitam o evento μ equivalem a estados marcados, estados que desabilitam o evento μ equivalem a estados não-marcados.

Exemplo 3.3.1 (Síntese com supervisor marcador) Considerando o Exemplo 2.2.3 da aceleração de um carro, tem-se que, para o caso de supervisor marcador, deve-se marcar todos os estados da planta $\boldsymbol{G}$ e, em seguida, realizar a transformação para obter $\boldsymbol{G}^{\mu}$, conforme mostrado na Figura 3.3. O gerador $\boldsymbol{E}$, por sua vez, sofre a transformação sem alterações prévias em seus estados marcados, resultando no gerador $\boldsymbol{E}^{\mu}$ representado na Figura 3.4. É importante perceber que, para este caso, o evento $\mu \in \Sigma_c^{\mu}$ é controlável. Após o cálculo do supervisor $\mathbb{S}^{\mu}$, obtém-se o supervisor ilustrado na Figura 3.5, que *desmarca* o estado 0 e *marca* os estados 1 e 2, enquanto que a planta não influencia na marcação. A partir do supervisor $\mathbb{S}$, é possível realizar um redução sensível à marcação levando em conta somente a ação de controle do supervisor. A Figura 3.6 mostra um supervisor reduzido $\hat{\mathbb{S}}$ obtido a partir da cobertura de controle $C = \{\{0\}, \{1, 2\}\}$. É possível perceber que, ao invés da marcação tradicional, as habilitações e desabilitações do evento μ possibilitam a conservação da marcação original.

□

3.4 Supervisores desmarcadores

A marcação dos supervisores desmarcadores é definida conjuntamente pela planta $\boldsymbol{G}$ e pelo supervisor $\mathbb{S}$. Neste caso, a abordagem é bastante similar à apresentada na Seção 3.3, sendo válida a Proposição 3.3.1. Porém não se deve forçar a marcação da planta $\boldsymbol{G}$ fazendo com que $Q_m = Q$, pois a marcação da planta é importante para a marcação do sistema em malha-fechada. Logo, deve-se fazer $\boldsymbol{G}^{\mu}$ mantendo o conjunto $Q_m \subseteq Q$ conforme o significado de marcação do modelo da planta.

Com relação à redução do supervisor $\mathbb{S}^{\mu}$, deve-se ignorar a marcação e considerar a condição de ação de controle do evento $\mu \in \Sigma^{\mu}$ para a construção de elementos da cobertura. Estados que habilitam o evento μ equivalem a estados marcados, estados que desabilitam o evento μ equivalem a estados não-marcados e, ainda, estado nos quais μ possui ação de controle igual a *dc* (*don't care*) equivalem a estados indiferentes à marcação, seguindo a definição de Sivolella [48].

Exemplo 3.4.1 (Síntese com supervisor desmarcador) Quando o supervisor é desmarcador, a planta $\boldsymbol{G}$ deve sofrer a transformação para $\boldsymbol{G}^{\mu}$ sem alterações prévias em sua marcação. Para o Exemplo 2.2.3 da aceleração de um carro, a planta obtida é conforme a Figura 3.7 e o gerador da especificação é o mesmo da Figura 3.4, sendo ambos com $\mu \in \Sigma_c^{\mu}$ controlável. Após a síntese do supervisor $\mathbb{S}^{\mu}$, obtém-se o gerador representado na Figura 3.8. O supervisor *desmarca* o estado 0, a planta *desmarca* o estado 1 e ambos *marcam* o estado 2. Para realizar uma redução sensível à marcação, basta considerar as habilitações e desabilitação de controle. Neste caso, o estado 1 do supervisor é tal que $\Psi(1, \mu) = dc$, representando um estado indiferente à marcação (analogamente à definição de Sivolella [48]), podendo ser agregado tanto ao estado 0 quanto ao estado 2. A Figura 3.9 ilustra o supervisor reduzido $\hat{\mathbb{S}}_1^{\mu}$ para a cobertura de controle $C_1 = \{\{0\}, \{1, 2\}\}$. A Figura 3.10 mostra o supervisor reduzido $\hat{\mathbb{S}}_2^{\mu}$ para a cobertura de controle $C_2 = \{\{0, 1\}, \{2\}\}$.

□

3.5 Conclusão

Este capítulo apresentou uma metodologia original para servir como alternativa ao uso da marcação tradicional de geradores, tanto para a síntese de supervisores quanto para a

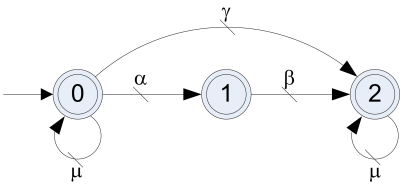

Figura 3.7: Planta $\boldsymbol{G}^{\mu}$

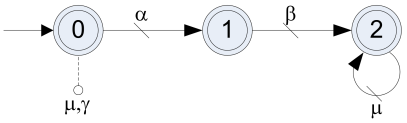

Figura 3.8: Supervisor $\mathbb{S}^{\mu}$

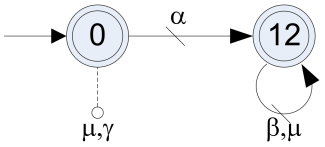

Figura 3.9: Supervisor reduzido $\hat{\mathbb{S}}_1^{\mu}$

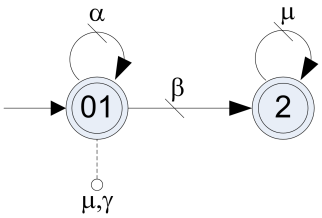

Figura 3.10: Supervisor reduzido $\hat{\mathbb{S}}_2^{\mu}$

redução sensível à marcação. Demonstraram-se matematicamente proposições que explicitam a equivalência entre a abordagem proposta e as tradicionais.

O resultados obtidos neste capítulo servem como base para a elaboração de algoritmos de redução de supervisores que desconsiderem a marcação, levando em conta somente as habilitações e desabilitações de eventos, mesmo que a redução seja sensível à marcação. Os capítulos seguintes apresentam duas metodologias para a redução de supervisores, sendo no Capítulo 4 por meio de Programação Linear Inteira Mista e no Capítulo 5 por meio de Algoritmos Genéticos.

Capítulo 4

Redução por Programação Linear

A Programação Linear é uma área da Otimização [54], que se refere ao estudo de problemas nos quais se deseja encontrar valores ótimos para variáveis de decisão de um modelo matemático, de forma a induzir desempenho ótimo e, ao mesmo tempo, respeitar um conjunto de restrições.

O Problema 2.4.1 do supervisor mínimo se adequa à Otimização no sentido que, para solucioná-lo, busca-se encontrar um supervisor com número mínimo de estados (valor ótimo) e que, ao mesmo tempo, seja controle-equivalente ao supervisor supremo com relação à planta (respeite as restrições).

Para solucionar o problema do supervisor mínimo por Programação Linear é necessário modelá-lo neste contexto. Este capítulo descreve aspectos fundamentais gerais de Otimização e Programação Linear Inteira Mista (PLIM) na Seção 4.1. A Seção 4.2 descreve a metodologia utilizada para aplicar a teoria de PLIM ao problema do supervisor mínimo, ignorando-se a marcação, conforme o Capítulo 3. Na Seção 4.3, apresentam-se resultados obtidos em instâncias reais do problema. Finalmente, na Seção 4.4, conclui-se sobre a metodologia e os resultados obtidos.

4.1 Programação Linear Inteira Mista

Um problema de otimização é composto basicamente por três elementos: função objetivo, variáveis de decisão e restrições. A *função objetivo* é uma função das *variáveis de decisão*. Sendo que as variáveis de decisão devem ser escolhidas de modo a respeitar *restrições* para então se calcular o valor da função objetivo, que deve ser otimizada. Uma configuração de variáveis de decisão que respeita todas as restrições é denominada *solução factível* e o conjunto de todas as soluções factíveis é denominado *espaço de soluções*, enquanto que o conjunto de todas as configurações de variáveis de decisão, factíveis ou não, é denominado *espaço de busca*. Para solucionar um problema de otimização, é necessário modelá-lo matematicamente (tal modelo é conhecido como programação matemática) e sua formulação generalizada é:

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{Sujeito a:} \quad & g(x) \geq 0 \\ & h(x) = 0 \\ & x \in \mathbb{R}^n \end{aligned}$$

onde $f : \mathbb{R}^n \to \mathbb{R}$ é a função objetivo, $g : \mathbb{R}^n \to \mathbb{R}^p$ e $h : \mathbb{R}^n \to \mathbb{R}^q$, com $n, p, q \in \mathbb{N}$ são as restrições que limitam o espaço de soluções factíveis e $x \in \mathbb{R}^n$ é o vetor de variáveis de decisão. No caso generalizado, as funções f, g e h são funções quaisquer, sem restrições quanto à sua linearidade.

No caso específico em que as restrições g e h que delimitam o espaço de soluções $A \subseteq \mathbb{R}^n$ são expressas por meio de equações e inequações lineares e que a função objetivo f é linear, tem-se um problema de *Programação Linear*, que geralmente pode ser resolvido eficientemente mesmo no pior caso [54].

Existem problemas de programação linear nos quais algumas variáveis são restritas a assumirem somente valores inteiros. Tais problemas são denominados problemas de *Programação Linear Inteira Mista* e neste caso, diferentemente de sua versão mais genérica que faz uso de variáveis reais quaisquer, não necessariamente há um método que os solucione eficientemente, pois normalmente expressam problemas NP-difíceis de caráter combinatório.

Um problema de programação linear inteira pode ser expresso em programação matemática

da seguinte forma:

$$\begin{aligned} \text{Minimize} \quad & c^T x \\ \text{Sujeito a:} \quad & Bx \geq b \\ & Cx = d \\ & x \in \mathbb{Z}_+^n \end{aligned}$$

onde $n \in \mathbb{Z}_+$ é o número de variáveis de decisão do problema, B é uma matriz $m \times n$, sendo $m \in \mathbb{Z}_+$ o número de inequações do problema, C é uma matriz $p \times n$, sendo $p \in \mathbb{Z}_+$ o número de equações do problema e b,c e d são vetores de ordem m,n e p, respectivamente.

Em um problema de programação linear inteira mista, não necessariamente $x \in \mathbb{Z}_+^n$, pois algumas componentes de x podem pertencer ao conjunto dos reais, porém pelo menos uma deve pertencer ao conjunto dos inteiros.

Problemas de programação linear são estudados há bastante tempo e considera-se que há métodos eficientes para solucioná-los desde os anos 1930 [54]. Atualmente existem diversas ferramentas [Apêndice B.2] que solucionam eficientemente [13, 30] problemas desta classe.

4.2 Metodologia

O Problema 2.4.1 do supervisor mínimo pode ser encarado como um problema combinatório quando se busca a cobertura com menor número de elementos, que respeitam as restrições de controlabilidade e determinismo. Por ser comprovadamente um problema NP-Difícil [49], optou-se por tentar solucioná-lo por meio de PLIM, que apresenta bons resultados para problemas desta mesma característica, como o problema da mochila [11] e o problema do caixeiro viajante [11].

Este capítulo apresenta a metodologia utilizada para modelar o problema do supervisor mínimo em programação matemática, bem como provas matemáticas que comprovam a equivalência entre o modelo e o problema. Tal metodologia foi publicada em congresso nacional [15].

Para que não seja necessário se preocupar com questões de marcação durante a minimização, considera-se que a marcação de estados por eventos é utilizada, conforme enunciada no Capítulo 3.

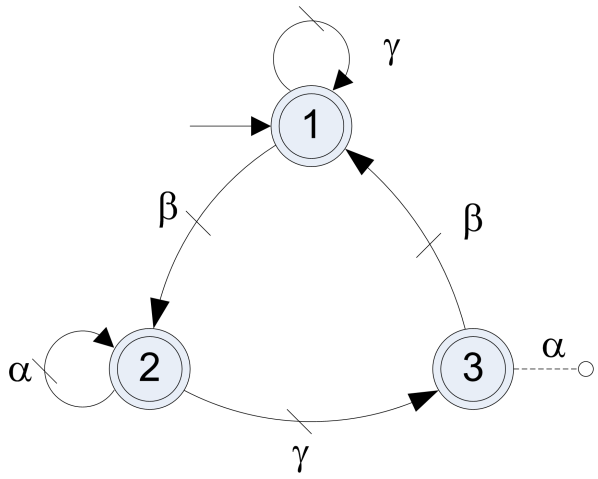

Figura 4.1: Supervisor $\mathbb{S}$.

4.2.1 Dados do problema

Considere o supervisor supremo $\mathbb{S}_{sup} = (\boldsymbol{S}_{sup}, \Psi)$, sendo $\boldsymbol{S}_{sup} = (X, \Sigma, \xi, x_0, X_m)$ um gerador e $\Psi_{sup} : X \times \Sigma \rightarrow \{0, 1, dc\}$ uma lei de controle, que se deseja minimizar, e a cobertura de controle $C = \{X_i, i \in I\}$, tal que $X_i \subseteq X$ e I é um conjunto de índices. Sem perda de generalidade, assume-se que $X = \{1, 2, \ldots, |X|\}$, $\Sigma = \{1, 2, \ldots, |\Sigma|\}$ e $I = \{1, 2, \ldots, |I|\}$. Desta forma, definem-se os seguintes termos (considerados dados para o problema de minimização):

$$T_{j,k,l} := \begin{cases} 1 & \text{se } \xi(j,k)! \wedge \xi(j,k) = l \\ 0 & \text{caso contrário} \end{cases}$$

o termo T mapeia cada tripla $(j \in X, k \in \Sigma, l \in X)$ a um valor binário e representa a função de transição ξ do supervisor supremo no contexto da programação matemática. Ele assume valor igual a 1 quando há uma transição partindo do estado j por meio da ocorrência do evento k com destino ao estado l, caso contrário seu valor é zero.

Exemplo 4.2.1 Considere o supervisor $\mathbb{S}$ ilustrado na Figura 4.1. Sem perda de generalidade, rotula-se os eventos de Σ da seguinte forma: $\alpha = 1$, $\beta = 2$ e $\gamma = 3$. Para o caso do parâmetro T, tem-se os valores mostrados na Tabela 4.1.

□

H é um termo que representa as ações de habilitação de eventos de um supervisor, mapeando cada par $(j \in X, k \in \Sigma)$ a 1 caso k esteja explicitamente habilitado em j, ou a 0

$T_{j,k,1}$				$T_{j,k,2}$				$T_{j,k,3}$			
	α	β	γ		α	β	γ		α	β	γ
1	0	0	1	1	0	1	0	1	0	0	0
2	0	0	0	2	1	0	0	2	0	0	1
3	0	1	0	3	0	0	0	3	0	0	0

Tabela 4.1: Exemplo de $T_{j,k,l}$

caso contrário. Formalmente:

$$H_{j,k} := \begin{cases} 1 & \text{se } \Psi(j,k) = 1 \\ 0 & \text{caso contrário} \end{cases}$$

D é um termo que representa as ações de desabilitação de eventos de um supervisor, mapeando cada par $(j \in X, k \in \Sigma)$ a 1 caso k esteja explicitamente desabilitado em j, ou a 0 caso contrário. Formalmente:

$$D_{j,k} := \begin{cases} 1 & \text{se } \Psi(j,k) = 0 \\ 0 & \text{caso contrário} \end{cases}$$

Exemplo 4.2.2 Considerando os mesmos dados do Exemplo 4.2.1, tem-se que os termos H e D são conforme os mostrados na Tabela 4.2.

□

$H_{j,k}$				$D_{j,k}$			
	α	β	γ		α	β	γ
1	0	1	1	1	0	0	0
2	1	0	1	2	0	0	0
3	0	1	0	3	1	0	0

Tabela 4.2: Exemplo de $H_{j,k}$ e $D_{j,k}$

Além dos termos já definidos, faz-se necessário definir também as variáveis do problema, que são aquelas que devem ter seus valores ajustados, a cada iteração do algoritmo utilizado para solucionar o problema, de forma a encontrar a solução ótima. Tais variáveis são definidas a seguir:

1. $c_{i,j} \in \mathbb{Z}_+$ representa uma cobertura. Sendo que se um elemento $i \in I$ da cobertura

contém o estado $j \in X$ então $c_{i,j} = 1$ e 0 caso contrário.

2. $t_{i,k,l} \in \mathbb{Z}_+$ representa as transições dos elementos $i \in I$ da cobertura. Assumindo valor 1 quando existe pelo menos um estado $j \in X$ pertencente ao elemento $i \in I$ da cobertura que, através do símbolo $k \in \Sigma$, atinge o estado $l \in X$.

3. $z_{i,k,v} \in \mathbb{Z}_+$ é a variável que auxilia na análise de determinismo de uma cobertura de controle, assumindo valor 1 caso todo $t \in X$ pertencente a um elemento $i \in I$ da cobertura, por meio do evento $k \in \Sigma$, leve a um subconjunto de estados pertencentes ao elemento $v \in I$ da cobertura. Caso contrário, assume valor zero.

4. $r_i \in \mathbb{Z}_+$ deve assumir valor 1 quando o elemento $i \in I$ da cobertura contém um ou mais estados e valor 0 caso contrário.

5. $h_{i,k} \in \mathbb{Z}_+$ deve assumir valor 1 quando o elemento $i \in I$ da cobertura contém um ou mais estados que habilitam o evento $k \in \Sigma$. Assume o valor 0 caso contrário.

6. $d_{i,k} \in \mathbb{Z}_+$ deve assumir valor 1 quando o elemento $i \in I$ da cobertura contém um ou mais estados que desabilitam o evento $k \in \Sigma$. Assume o valor 0 caso contrário.

4.2.2 Problema de minimização

Dados T, H, D e sendo $i, v \in I$, $j, l \in X$, $k \in \Sigma$, o problema de minimização de supervisores em programação linear inteira mista é definido como:

$$\begin{aligned}
&\text{Minimize:} && \sum_{i \in I} r_i \\
&\text{Sujeito a:} && \\
&\forall j && \sum_{i \in I} c_{i,j} \geq 1 && (4.1)\\
&\forall i && r_i \leq 1 && (4.2)\\
&\forall i && r_i \leq \sum_{j \in X} c_{i,j} && (4.3)\\
&\forall i, j && r_i \geq c_{i,j} && (4.4)\\
&\forall i, k, l && t_{i,k,l} \leq 1 && (4.5)\\
&\forall i, k, l && t_{i,k,l} \leq \sum_{j \in X} (c_{i,j} \times T_{j,k,l}) && (4.6)\\
&\forall i, k, l, j : T_{j,k,l} = 1 && t_{i,k,l} \geq c_{i,j} && (4.7)\\
&\forall i, k && h_{i,k} \leq 1 && (4.8)\\
&\forall i, k && h_{i,k} \leq \sum_{j \in X} (c_{i,j} \times H_{j,k}) && (4.9)\\
&\forall i, k, j : H_{j,k} = 1 && h_{i,k} \geq c_{i,j} && (4.10)\\
&\forall i, k && d_{i,k} \leq 1 && (4.11)\\
&\forall i, k && d_{i,k} \leq \sum_{j \in X} (c_{i,j} \times D_{j,k}) && (4.12)\\
&\forall i, k, j : D_{j,k} = 1 && d_{i,k} \geq c_{i,j} && (4.13)\\
&\forall i, k && \sum_{v \in I} z_{i,k,v} = 1 && (4.14)\\
&\forall i, k, v, l && z_{i,k,v} \leq c_{v,l} - t_{i,k,l} + 1 && (4.15)\\
&\forall i, k && h_{i,k} + d_{i,k} \leq 1 && (4.16)
\end{aligned}$$

A Equação 4.1 garante que cada estado pertença a pelo menos um elemento da cobertura $c_{i,j}$. Dado que $r_i \in \mathbb{Z}_+$, a Equação 4.2 garante que $r_i \in \{0, 1\}$, a Equação 4.3 força $r_i = 0$ caso X_i seja um elemento vazio da cobertura e a Equação 4.4 garante $r_i \geq 1$ quando o elemento da cobertura X_i possui pelo menos um elemento. Ou seja, o conjunto de equações lineares 4.2, 4.3 e 4.4 fazem com que $r_i = 0$ se $\sum_{j \in X} c_{ij} = 0$ ou $r_i = 1$ se $\sum_{j \in X} c_{ij} \geq 1$. Os grupos de equações {4.5, 4.6, 4.7}, {4.8, 4.9, 4.10} e {4.11, 4.12, 4.13} realizam o mesmo

trabalho para as variáveis t, h e d respectivamente.

As equações 4.14 e 4.15 garantem que as condições de determinismo sejam satisfeitas, pois obrigam a existência de um elemento da cobertura v que seja destino de todas as transições originadas a partir dos estados l de um elemento da cobertura i, por meio de um evento k. Já a condição de consistência da ação de controle é garantida pela Equação 4.16, que limita cada elemento da cobertura a possuir somente uma ação explicita de controle, ou habilitação ou desabilitação para um evento k.

A correspondência do problema modelado pelos termos, variáveis e equações apresentados ao problema da minimização de supervisores é estabelecida pela proposição a seguir:

Proposição 4.2.1 *Se as equações de restrições do problema de programação linear proposto na Seção 4.2.2 são respeitadas, então a variável c representa uma cobertura de controle.*

Para demonstrar matematicamente que a variável c do espaço de busca do problema de programação linear inteira proposto nesta seção representa uma cobertura de controle, deve-se provar que, ao se respeitar todas as equações de restrições, também se está respeitando todas as propriedades que definem uma cobertura de controle, conforme a Seção 2.4. A seguir, destacam-se as quatro propriedades necessárias a uma cobertura de controle e, para cada uma delas, apresentam-se as equivalências entre PLIM e Redução.

Propriedade 1, elementos não-vazios

Ao construir uma cobertura C a partir da variável c, ignoram-se as linhas de c que por ventura contenham somente zeros e, consequentemente, representem elementos da cobertura vazios. Pode-se perceber com facilidade que esta propriedade é respeitada ao se efetuar tal procedimento na construção de C a partir de c.

Propriedade 2, cobertura completa

A equação 4.1 é responsável por garantir que c seja sempre uma cobertura completa, logo:

Proposição 4.2.2 $\forall j \sum_{i \in I} c_{i,j} \geq 1 \Longrightarrow \bigcup_{i \in I} X_i = X$

Prova:

Para se provar por contradição, assume-se:

1. $\forall j \sum_{i \in I} c_{i,j} \geq 1$
2. $\bigcup_{i \in I} X_i \neq X$

Desenvolvendo-se a segunda suposição:
$\bigcup_{i \in I} X_i \neq X$
$\exists (x \in X) x \notin \bigcup_{i \in I} X_i$
$\exists (x \in X) \forall (i \in I) x \notin X_i$
$\exists (x \in X) \forall (i \in I) c_{i,x} = 0$
$\exists (x \in X) \sum_{i \in I} c_{i,x} = 0$
que contradiz a primeira suposição.

□

Propriedade 3, determinismo

As equações 4.14 e 4.15 são responsáveis por garantir a propriedade de determinismo. Conforme explicitado anteriormente, se $\big(t_{i,k,l} \in \mathbb{Z}_+\big) \wedge \big(\forall (i,k,l) t_{i,k,l} \leq 1\big) \wedge \big(\forall (i,k,l) t_{i,k,l} \leq \sum_{j \in X} (c_{i,j} \times T_{j,k,l})\big) \wedge \big(\forall (i,k,l,j) t_{i,k,l} \geq c_{i,j} \times T_{j,k,l}\big)$, então pode-se definir $t_{i,k,l} \in \{0,1\}$ da seguinte forma:

$$t_{i \in I, k \in \Sigma, l \in X} = \begin{cases} 1 & \exists (p \in X) : c_{i,p} = 1 \wedge T_{p,k,l} = 1 \\ 0 & \text{caso contrário} \end{cases}$$

Proposição 4.2.3 $\forall (i_1 \in I, k \in \Sigma) \exists (i_2 \in I) \forall (p \in X) t_{i_1,k,p} \leq c_{i_2,p}$
$\Longrightarrow$
$\forall (i \in I, \sigma \in \Sigma) \big[\exists (y \in X_i) \xi(\sigma, y)! \Rightarrow \exists (j \in I) \forall (x \in X_i) [\xi(\sigma, x)! \rightarrow \xi(\sigma, x) \in X_j]\big]$

Prova:

Para se provar por contradição, assume-se:

1. $\forall (i_1 \in I, k \in \Sigma) \exists (i_2 \in I) \forall (p \in X) t_{i_1,k,p} \leq c_{i_2,p}$
2. $\neg \Big[\forall (i \in I, \sigma \in \Sigma) \big[\exists (y \in X_i) \xi(\sigma, y)! \Rightarrow \exists (j \in I) \forall (x \in X_i) [\xi(\sigma, x)! \rightarrow \xi(\sigma, x) \in X_j]\big]\Big]$

Desenvolvendo-se a segunda suposição:

$\exists(i \in I, \sigma \in \Sigma)\neg\big[\exists(y \in X_i)\xi(\sigma, y)! \Rightarrow \exists(j \in I)\forall(x \in X_i)[\xi(\sigma, x)! \rightarrow \xi(\sigma, x) \in X_j]\big]$

$\exists(i \in I, \sigma \in \Sigma, y \in X_i)\Big[\xi(\sigma, y)! \wedge \neg\big[\exists(j \in I)\forall(x \in X_i)[\xi(\sigma, x)! \rightarrow \xi(\sigma, x) \in X_j]\big]\Big]$

$\exists(i \in I, \sigma \in \Sigma, y \in X_i)\big[\xi(\sigma, y)! \wedge \forall(j \in I)\exists(x \in X_i)\neg[\xi(\sigma, x)! \rightarrow \xi(\sigma, x) \in X_j]\big]$

$\exists(i \in I, \sigma \in \Sigma, y \in X_i)\big[\xi(\sigma, y)! \wedge \forall(j \in I)\exists(x \in X_i)(\xi(\sigma, x)! \wedge \xi(\sigma, x) \notin X_j)\big]$

$\exists(i \in I, \sigma \in \Sigma)\forall(j \in I)\exists(x \in X_i, z \in X)(\xi(\sigma, x)! \wedge \xi(\sigma, x) = z \notin X_j)$

$\exists(i \in I, \sigma \in \Sigma)\forall(j \in I)\exists(x \in X_i, z \in X)(c_{i,x} = 1 \wedge T_{x,\sigma,z} = 1 \wedge c_{j,z} = 0)$

$\exists(i \in I, \sigma \in \Sigma)\forall(j \in I)\exists(z \in X)(t_{i,\sigma,z} = 1 \wedge c_{j,z} = 0)$

$\exists(i \in I, \sigma \in \Sigma)\forall(j \in I)\exists(z \in X)(t_{i,\sigma,z} = 1 > c_{j,z} = 0)$

$\neg\big(\forall(i \in I, \sigma \in \Sigma)\exists(j \in I)\forall(z \in X)t_{i,\sigma,z} \leq c_{j,z}\big)$

Que contradiz a primeira suposição.

□

Propriedade 4, consistência da ação de controle

A equação 4.16 é a responsável por restringir o espaço de busca a coberturas que respeitem a propriedade de controlabilidade. Dado que a variável $h_{i,k} \in \mathbb{Z}_+$ e dadas as equações 4.8, 4.9 e 4.10, pode-se definir h como:

$$h(i \in I, k \in \Sigma) = \begin{cases} 1 & \exists(j \in X) : c_{i,j} = 1 \wedge H_{j,k} = 1 \\ 0 & \text{caso contrário} \end{cases}$$

Da mesma forma, porém para a variável $d_{i,k} \in \mathbb{Z}_+$ e para as equações 4.11, 4.12 e 4.13, pode-se definir d como:

$$d_{i \in I, k \in \Sigma} = \begin{cases} 1 & \exists(j \in X) : c_{i,j} = 1 \wedge D_{j,k} = 1 \\ 0 & \text{caso contrário} \end{cases}$$

Proposição 4.2.4 $\forall(i_1 \in I, k \in \Sigma)h_{i,k} + d_{i,k} \leq 1$

$\Longrightarrow$

$\forall(i \in I, \sigma \in \Sigma, x \in X_i, y \in X_i)\Psi(\sigma, x) \neq dc \neq \Psi(\sigma, y) \Rightarrow \Psi(\sigma, x) = \Psi(\sigma, y)$

Prova:

Para provar por contradição, assume-se:

1. $\forall(i_1 \in I, k \in \Sigma) h_{i,k} + d_{i,k} \leq 1$
2. $\neg\Big[\forall(i \in I, \sigma \in \Sigma, x \in X_i, y \in X_i)\Psi(\sigma, x) \neq dc \neq \Psi(\sigma, y) \Rightarrow \Psi(\sigma, x) = \Psi(\sigma, y)\Big]$

Desenvolvendo-se a segunda suposição:
$\exists(i \in I, \sigma \in \Sigma, x \in X_i, y \in X_i)\neg\big(\Psi(\sigma, x) \neq dc \neq \Psi(\sigma, y) \Rightarrow \Psi(\sigma, x) = \Psi(\sigma, y)\big)$
$\exists(i \in I, \sigma \in \Sigma, x \in X_i, y \in X_i)\neg\big(\neg(\Psi(\sigma, x) \neq dc \neq \Psi(\sigma, y)) \vee (\Psi(\sigma, x) = \Psi(\sigma, y))\big)$
$\exists(i \in I, \sigma \in \Sigma, x \in X_i, y \in X_i)\big((\Psi(\sigma, x) \neq dc \neq \Psi(\sigma, y)) \wedge (\Psi(\sigma, x) \neq \Psi(\sigma, y))\big)$
Sem perda de generalidade:
$\exists(i \in I, \sigma \in \Sigma, x \in X_i, y \in X_i)\Psi(\sigma, x) = 1 \wedge \Psi(\sigma, y) = 0$
$\exists(i \in I, \sigma \in \Sigma, x \in X_i, y \in X_i)(C_{i,x} = 1 \wedge H_{x,\sigma} = 1) \wedge (C_{i,y} = 1 \wedge D_{y,\sigma} = 1)$
$\exists(i \in I, \sigma \in \Sigma) h_{i,\sigma} = 1 \wedge d_{i,\sigma} = 1$
$\exists(i \in I, \sigma \in \Sigma) h_{i,\sigma} + d_{i,\sigma} > 1$
$\neg\big(\forall(i \in I, \sigma \in \Sigma) h_{i,\sigma} + d_{i,\sigma} \leq 1\big)$
Que contradiz a primeira suposição.

□

É importante perceber que as demonstrações matemáticas aqui apresentadas provam somente a condição de suficiência, e não de necessidade e suficiência. Logo, é possível que esta representação possa ser refinada, uma vez que pode ser demasiadamente restritiva.

4.3 Resultados

Por se tratar de um problema NP-difícil, o espaço de busca se torna imenso à medida em que o tamanho de uma instância do problema aumenta. Um alternativa para contornar este problema é a de diminuir o espaço de busca, reduzindo-se $|I|$, da seguinte forma:

1. Estima-se a cota inferior $l_b \in \mathbb{Z}_+$ do número de estados do supervisor mínimo [48, 49].
2. Executa-se o algoritmo de solução limitando-se a dimensão máxima da variável c a l_b.
3. Caso não haja solução factível e $l_b < |X|$, faz-se $l_b = \lceil (l_b + l_b p) \rceil$ (limitando-se l_b a $|X| - 1$) e retorna-se ao passo anterior, onde $p \in (0, 1] \in \mathbb{R}$ é um passo que incrementa a cota inferior a cada execução do algoritmo, aumentando seu espaço de busca.

No pior caso (quando não é possível reduzir uma dada instância do problema), a execução dos passos sugeridos anteriormente causa processamento desnecessário, uma vez que seria possível concluir isso executando-se o algoritmo uma única vez com $|I| = |X|$. Porém, para casos em que a taxa de redução do supervisor mínimo em relação ao supervisor supremo é grande, os passos acima resultam em grandes ganhos de eficiência.

A representação do problema de minimização de supervisores em programação linear inteira mista foi implementada utilizando-se a linguagem de programação matemática AMPL [Apêndice B.2.1]. A ferramenta utilizada para solucionar instâncias de problemas de minimização foi o aplicativo CPLEX [Apêndice B.2.2] executando em um Intel Pentium 4 2.66 GHz com 1 Gb de memória RAM e GNU/Linux Mandrake 9.2.

Exemplos de diferentes tamanhos foram minimizados para que se pudesse avaliar a eficiência da solução do problema de minimização de supervisores por métodos de otimização e, também, compará-lo a métodos tradicionais. A Tabela 4.3 mostra comparações entre o método para se solucionar o problema de programação linear inteira mista (PLIM) e o utilizado pelo aplicativo CTCT [Apêndice B.1.1] (no mesmo computador), quanto ao número de estados obtidos pela redução e o tempo de execução em segundos. A instância testada na primeira linha da tabela é um gerador de 3 estados que é redutível a 2 estados somente por coberturas, não sendo possível reduzí-lo por partições, logo o algoritmo implementado pelo CTCT, que utiliza somente partições, não consegue reduzí-lo. A instância de 6 estados da segunda linha foi obtida a partir do produto síncrono entre o gerador da linha anterior e um outro gerador assíncrono (que não possui eventos em comum) de 2 estados. As instâncias das linhas posteriores foram obtidas da mesma forma. Optou-se por tais exemplos por ser possível encontrar a solução ótima somente por meio de coberturas e por as instâncias maiores apresentarem alta taxa de redução, fato este que é comum [14, 51] em supervisores de sistemas reais de manufatura. Em todas as soluções por PLIM apresentadas nesta tabela, limitou-se inicialmente o tamanho da cobertura de controle à cota inferior do tamanho do supervisor mínimo ($l_b = 2$). A última linha da tabela, indica que o problema não foi solucionado por PLIM em virtude de falta de memória (f.m.), que é uma questão crítica na minimização de supervisores por PLIM, pois o número de variáveis dinâmicas é da ordem $\Theta(|X| \times |\Sigma| \times |I|)$, enquanto que as equações e inequações de restrições é da ordem $\Theta(|X|^2 \times |\Sigma| \times |I|)$.

Conforme demonstrado na Tabela 4.3, a solução por PLIM mostrou-se eficiente para instâncias com até de 384 estados e que podem ser bastantes reduzidas. A Tabela 4.4 mostra o tempo de execução em segundos do algoritmo de solução por PLIM para cada instância da

N^o estados			T. exec. (s)	
$SupC$	CTCT	PLIM	CTCT	PLIM
3	3	2	< 1	$0,010$
6	3	2	< 1	$0,020$
12	3	2	< 1	$0,060$
24	3	2	< 1	$0,140$
48	3	2	< 1	$0,530$
96	3	2	< 1	$2,030$
192	3	2	≈ 19	$9,040$
384	3	2	≈ 1230	$51,570$
768	-	-	> 1500	f.m.

Tabela 4.3: Resultados comparativos

$SupC$	Tamanho limite da cobertura de controle				
	2	3	5	6	10
3	$0,01$	$0,02$	-	-	-
6	$0,02$	$0,11$	$0,31$	$0,72$	-
12	$0,06$	$0,50$	$3,44$	$10,15$	$100,95$
24	$0,14$	$4,66$	$62,47$	$93,39$	> 600
48	$0,53$	$28,98$	$161,67$	> 600	-
96	$2,03$	$164,93$	> 600	-	-
192	$9,04$	> 600	-	-	-
384	$51,57$	> 600	-	-	-

Tabela 4.4: Resultados variando-se $|I|$

Tabela 4.3, porém variando-se a limitação inicial da cobertura de controle. Os resultados da Tabela 4.4 mostram que, caso se inicie a solução por PLIM limitando-se menos o tamanho da cobertura de controle, fato que seria inevitável em instâncias que possuem baixa taxa de minimização (e consequentemente uma alta cota inferior l_b), piora-se de maneira crítica a eficiência do algoritmo.

4.4 Conclusão e perspectivas

Este capítulo mostra que é possível representar o problema de minimização de supervisores utilizando-se programação linear inteira mista. Embora o algoritmo convirja sempre para uma solução ótima, o custo para tal convergência pode ser impraticável, dado que é um problema NP-difícil. Para estes casos, pode-se interromper o algoritmo durante sua execução

e verificar se a atual solução subótima é aceitável com relação à cota inferior.

Apesar de ser inviável a redução de supervisores grandes (com mais de 1000 estados) nesta implementação, abre-se perspectivas para um estudo mais detalhado deste problema relacionado à área de programação linear inteira mista. Acredita-se que é possível melhorar a eficiência do algoritmo buscando cortes específicos que façam uso da estrutura do problema quando solucionado por algoritmos *branch and bound*, bem como do significado de relaxações lineares do problema.

Capítulo 5

Redução por Algoritmos Genéticos

Algoritmos genéticos (AG) são métodos adaptativos, baseados no processo genético de organismos biológicos, que são utilizados para resolver problemas de busca e otimização. O uso de AG não garante que a solução ótima de um dado problema será encontrada. Comumente são utilizados para solucionar problemas NP-Difíceis e problemas que se desconhece grandes detalhes de sua estrutura.

Justifica-se a busca de soluções para o problema da minimização de supervisores (Problema 2.4.1), por meio desta metodologia, por se tratar de um problema NP-Difícil. Além disso, soluções subótimas podem ser úteis, tanto na implementação de supervisores, quanto na interpretação de supervisores, pois grandes reduções no espaço de estados podem representar grandes ganhos. Por exemplo, mesmo se a solução ótima para um problema com 10000 estados seja um supervisor reduzido com 5 estados, o fato de se encontrar uma solução com 15 estados seria de grande valor.

A Seção 5.1 apresenta conceitos preliminares sobre a teoria de Algoritmos Genéticos. Na Seção 5.2, descreve-se a metodologia utilizada para adequar o problema da minimização de supervisores aos AG. Os resultados obtidos com esta metodologia são apresentados na Seção 5.3 e a Seção 5.4 apresenta as conclusões e perspectivas sobre a aplicação de AG ao problema da minimização de supervisores.

5.1 Algoritmos Genéticos

No meio natural, populações evoluem ao longo de gerações de acordo com os princípios de seleção natural e "sobrevivência do mais adaptado", conforme primeiramente explicitado por Charles Darwin na obra *A Origem da Espécies* [16]. De forma análoga, os algoritmos genéticos são capazes de evoluir soluções de problemas reais.

Os princípios básicos dos AG foram rigorosamente explicitados pela primeira vez por Holland [25] e também são bem descritos em outros trabalhos como [3, 4, 5, 6, 40, 53]. A idéia por trás dos algoritmos genéticos é a de simular, para fins específicos, os processos biológicos que são essenciais à evolução natural. Embora não se saiba exatamente quais processos biológicos são essenciais à evolução natural, os fundamentos de tal evolução são claros. Isto é, os algoritmos genéticos utilizam simulações que obedecem aos fundamentos, mas não necessariamente ao processo natural.

Na natureza, indivíduos de uma população competem entre si por recursos como comida, água e abrigo. Além disso, indivíduos de uma mesma espécie (sexuada) competem para atrair parceiros para a reprodução. Os indivíduos mais bem sucedidos, tanto na sobrevivência quanto na reprodução, possuirão relativamente números maiores de descendentes. Já os mal sucedidos tendem a possuir um menor número de descendentes. Isto é, o material genético dos mais bem adaptados será propagado com alta freqüência nas futuras gerações, enquanto que o dos mal adaptados tende a desaparecer, ou a aparecer menos freqüentemente. Na reprodução, a combinação de boas características de diferentes indivíduos pode gerar descendentes ainda mais bem adaptados que seus pais, tornando possível a evolução da espécie.

Os algoritmos genéticos utilizam uma analogia direta da evolução natural. Eles trabalham com uma *população* de *indivíduos*, cada um representando uma possível solução de um dado problema. A cada indivíduo está associado um valor de *aptidão*, que representa o quão boa é esta solução para o problema. Quanto maior for a aptidão de um indivíduo, maior será a sua chance de se reproduzir e de produzir indivíduos ainda melhores que *dominarão* a população. Indivíduos com baixa aptidão tentem a não se reproduzir e, conseqüentemente, a sua carga genética na população tende a diminuir.

O poder dos Algoritmos Genéticos é justificado principalmente pelo fato de que eles lidam com sucesso com problemas de diversas áreas, fornecendo soluções robustas quando não existem métodos alternativos. Embora não se possa garantir que os AG sempre encontrem

```
INÍCIO /* ALGORITMO GENÉTICO */
     gerar população inicial;
     avaliar a aptidão de cada indivíduo;

     ENQUANTO NÃO finalizado FAZER
          PARA (tamanho_população / 2) FAZER
               selecionar dois indivíduos da geração anterior para a reprodução
                    /*tendendo a selecionar os com maior aptidão*/
               recombinar os dois indivíduos para gerar dois novos indivíduos
               calcular a aptidão dos dois novos indivíduos
               inserir os novos descendentes na nova geração
          FIM PARA

          SE a população convergiu ENTÃO
               finalizado := VERDADE
          FIM SE
    FIM ENQUANTO
FIM INÍCIO
```

Figura 5.1: Algoritmo genético tradicional.

uma solução ótima para um dado problema, obtém-se normalmente, quando bem elaborados, soluções *aceitavelmente boas* em um tempo *aceitavelmente curto* [40]. Os AG normalmente perdem em velocidade e precisão quando comparados a métodos tradicionais para problemas específicos, logo devem ser usados para problemas onde não existem tais métodos, ou seus desempenhos não são satisfatórios.

Antes que um Algoritmo Genético possa ser executado, é necessário que o problema em questão seja *representado*. É preciso também que exista uma *função de avaliação*, que atribui valores de aptidão a cada indivíduo representado. E, finalmente, pais devem ser *selecionados* para a reprodução e *recombinados* para gerar descendentes. A seguir descrevem-se os princípios básicos dos itens citados. Um algoritmo genético tradicional está representado na Figura 5.1.

5.1.1 Codificação

Em 1866, Gregor Johann Mendel percebeu, após seus estudos com reprodução sexuada de ervilhas, que havia fatores que passavam de pai para filho por meio de seu material genético, formado por pares de alelos. Tal informação genética determina as propriedades, aparência e forma de um indivíduo. Mais tarde, foi descoberto que tal informação genética é formada por uma cadeia dupla de quatro nucleotídeos, conhecida como DNA. Mendel percebeu que existe, na natureza, uma distinção entre a informação genética de um indivíduo

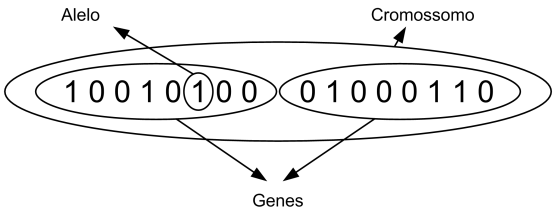

Figura 5.2: Cromossomos, Genes e Alelos.

(conhecida como *genótipo*) e sua aparência observável (chamada de *fenótipo*). O genótipo representa todas as informações guardadas nos cromossomos de um indivíduo, enquanto que o fenótipo representa a interpretação do genótipo pela natureza, resultando na aparência externa do indivíduo. Ou seja, a natureza faz um mapeamento genótipo-fenótipo que utiliza as informações contidas no genótipo para construir o fenótipo. Sendo que, para que seja possível representar um grande número de indivíduos com apenas quatro nucleotídeos, a informação do genótipo não se encontra nos alelos propriamente ditos, mas sim na forma como eles são ordenados. Interpretando-se a sequência de alelos, é possível representar uma variedade enorme de indivíduos com um pequeno número deles.

Um *cromossomo* é uma cadeia de tamanho finito na qual toda a informação genética de um indivíduo está contida. Cada cromossomo é formado por vários *alelos*, que são a menor unidade de informação em um cromossomo. Por exemplo, em uma representação binária, um alelo poderia assumir valor zero ou um. Se uma propriedade do fenótipo de um indivíduo é representada por um ou mais alelos (como por exemplo a cor dos olhos ou da pele), então estes alelos juntos formam um *gene*. Um gene é um região do cromossomo que deve ser interpretada em conjunto e que é responsável por uma característica específica do fenótipo. A Figura 5.2 ilustra a diferenciação entre cromossomo, gene e alelo para um cromossomo de 2 *bytes*, sendo cada *byte* um gene e cada *bit* um alelo.

É muito importante que se distingua bem o genótipo do fenótipo, pois é o fenótipo de um indivíduo que vai determinar seu sucesso perante a natureza. Desta forma, ao se comparar indivíduos, deve-se julgá-los de acordo com seu fenótipo. Enquanto que na reprodução de indivíduos, deve-se analisá-los do ponto de vista do genótipo. Durante a reprodução sexuada, os descendente não herdam características do fenótipo, mas somente do genótipo. Ou seja, os operadores genéticos (como reprodução e mutação) atuam no genótipo, enquanto que a avaliação dos indivíduos atua no fenótipo.

Uma solução em potencial para um problema deve ser representada como um con-

junto de parâmetros. Tais parâmetros, denominados *genes*, são arranjados para formarem *cromossomos*. Idealmente, cada gene deve corresponder independentemente a um característica do fenótipo e pequenas variações no genótipo devem representar pequenas variações no fenótipo [4, 5], embora tais características não sejam tão facilmente atingidas. Tradicionalmente, os genes são formados por dígitos binários que são concatenados e formam uma cadeia de bits que representa um cromossomo, embora existam outras formas de se codificar um cromossomo [46]. Por exemplo, ao se tentar maximizar uma função $f(x, y) : \mathbb{R}^2 \rightarrow \mathbb{R}$, pode-se representar cada variável por um número binário de 8 bits. Desta forma, um cromossomo para esta representação possuiria 2 genes e seria formado por uma cadeia de 16 dígitos binários.

5.1.2 Função de Avaliação

A função de avaliação é responsável por avaliar a aptidão de uma possível solução (indivíduo da população). Basicamente, a função de avaliação avalia o *fenótipo* de um indivíduo e o atribui um valor numérico, que é utilizado pelo AG para comparar indivíduos entre si e julgar quais são os mais adaptados e aptos a sobreviver e reproduzir-se.

Em muitas aplicações, o modo como um fenótipo deve ser avaliado é muito claro e simples. Por exemplo, se o objetivo de um AG é maximizar a função $f(x, y) = -x^4 + 3y^2 - x + 4y + 15$, os indivíduos poderiam ser pares (x, y) e a função de avaliação simplesmente computaria $f(x, y)$, que seria uma forma fácil, clara e pouco custosa computacionalmente. Porém, muitos problemas reais lidam com a infactibilidade de soluções, ou seja, nem todas as soluções candidatas são válidas [38, 39], pois violam restrições. Segundo Coello [9], os AG são feitos originalmente para problemas sem restrições e é um grande desafio adaptá-los a problemas com restrições, que são muitos no mundo real. O meio mais comum de adaptá-los é adicionando penalidades na função de avaliação sempre que uma restrição não é obedecida, porém existem problemas com essa metodologia e vários trabalhos propõem soluções alternativas [9, 52, 53].

Nem sempre é claro como um indivíduo deve ser comparado a outro, isto é, mesmo em posse de duas soluções candidatas, é difícil julgar qual delas é a melhor. Às vezes, é possível que soluções infactíveis sejam consideradas melhores que algumas factíveis, pois *podem conter informações mais importantes para encontrar a solução ótima* [52].

A função de avaliação é de fundamental importância para o correto funcionamento do

AG, podendo levá-lo a problemas de divergência, convergência prematura e convergência desacelerada. Ela em conjunto com a codificação são considerados os aspectos mais importantes do AG [5]. É importante frisar que a função de avaliação deve idealmente ser computacionalmente pouco custosa.

5.1.3 Operadores genéticos

Os operadores genéticos, diferentemente da função de avaliação, agem sobre o *genótipo* dos indivíduos e os alteram de diferentes maneiras. Os operadores mais comuns são os de reprodução sexuada, mutação e correção.

Operadores que realizam reproduções sexuadas são responsáveis pela troca de material genético entre indivíduos, para que novos descendentes sejam gerados. Essas trocas de material genético podem gerar indivíduos mais bem adaptados que seus pais, permitindo a evolução das soluções na direção da solução ótima. Considera-se que a reprodução sexuada auxilia na velocidade de varredura do espaço de busca, pois proporciona o *paralelismo implícito* [4, 25, 40]. Um operador de reprodução pressupõe que existe uma certa independência entre a influência dos genes sobre características do fenótipo [21], isto é, baixa *epistasis* [5]. Além disso, estes operadores são considerados responsáveis pela componente de busca direcionada dos AG. Em alguns problemas, como os de partição de grafos [1] ou evolução de máquinas de estado [18], existe o problema da falta de contexto dos operadores de reprodução [46], que devem respeitar a forma como o genótipo foi codificado e não quebrá-lo e reagrupá-lo sem critério algum.

Os operadores de mutação são responsáveis pela busca aleatória em um AG. Eles realizam um teste probabilístico e alteram um gene de um cromossomo, auxiliando a exploração de regiões do espaço de busca que estejam distantes da maioria das soluções candidatas, evitando a convergência para ótimos locais.

Para problemas com restrições, existe a possibilidade de se construirem operadores de correção para tornar, ou ao menos buscar tornar, soluções infactíveis em factíveis. Segundo Michalewicz [38], a utilização deste tipo de operadores deve ser cuidadosa, pois quando indivíduos infactíveis são substituídos por suas correções factíveis, tem-se a chamada evolução de Lamarck [53], que assume que um indivíduo evolui durante a sua existência e que tal evolução é codificada de volta ao seu genótipo. De acordo com Whitley [53], baseado em resultado empíricos e analíticos, a evolução de Lamarck acarreta em um grande ganho de

velocidade na convergência, porém, para alguns problemas, há convergência prematura, enquanto que a versão sem a estratégia de Lamarck converge para o ótimo global.

5.1.4 Seleção

A seleção é a etapa do AG que decide quais indivíduos de uma população devem permanecer e, ainda, reproduzir-se. Tal seleção é baseada em comparações feitas sobre os valores de aptidão, gerados pela função de avaliação, de cada indivíduo. Logo, analogamente à natureza, a seleção atua sobre o *fenótipo*. Tal fato causa efeitos interessantes, pois de fato, não necessariamente a combinação de indivíduos com maior aptidão gera descendentes melhores ainda e nem a combinação de indivíduos com menor aptidão gera indivíduos piores, pois os operadores genéticos agem sobre o genótipo, que não é diretamente avaliado.

Os operadores de seleção influenciam diretamente na convergência do algoritmo. Caso se *pressione* demasiadamente a seleção, isto é, priorizando demais os melhores indivíduos, a convergência é mais rápida, porém corre-se o risco de uma convergência prematura a um ótimo local. No caso contrário, a convergência pode ser demasiadamente lenta, mas é menos provável que o algoritmo convirja para um ótimo local.

As estratégias mais comumente utilizadas são a seleção por torneio e a seleção por roleta probabilística. Na primeira, escolhe-se aleatoriamente n indivíduos da população (comumente $n = 2$) e, de acordo com uma probabilidade $0,5 < p < 1$, escolhe-se o com maior aptidão. Na segunda, ponderam-se as aptidões de cada indivíduo sobre a soma de todas as aptidões. Em seguida, realizam-se sorteios para decidir qual indivíduo será selecionado, sendo que cada indivíduo pode ser selecionado de acordo com o seu valor ponderado de aptidão.

5.1.5 Funcionamento dos AG

À primeira vista, o porquê de os AG funcionarem pode parecer não tão claro, mas Holland [25] e Goldberg [21] o explicam utilizando *Schemata*, o *Teorema de Schema* e a *Hipótese dos Building-Blocks*. Outros trabalhos [4, 6, 53] apresentam estas explicações de forma clara e resumida.

Infelizmente, os AG não se adequam a qualquer tipo de problema e algumas considerações devem ser feitas para que obtenham sucesso. Primeiramente, com relação à codificação, deve ser possível se chegar a todos os possíveis fenótipos a partir dos genótipos representados por ela, além de que as propriedades de *redundância*, *scaling* e *localidade* devem ser

levadas em conta [46] e, idealmente, cada gene deve ser responsável por uma característica do fenótipo. Com relação à função de avaliação, ela deve poder ser computada rapidamente e o valor por ela retornado deve efetivamente representar o quão bom ou promissor é um fenótipo.

5.2 Metodologia

Esta seção apresenta a metodologia utilizada para buscar solucionar o problema da minimização de supervisores por meio de Algoritmos Genéticos, isto é, explicita quais diferentes codificações, operadores genéticos e técnicas de evolução foram utilizadas, bem como as devidas justificativas.

Foram consideradas basicamente duas formas de genótipo para evoluir soluções para o problema da minimização de supervisores:

- Evoluir coberturas do espaço de estados do supervisor supremo até que se encontre a menor cobertura de controle, conforme a Seção 2.4.
- Evoluir máquinas de estado até que se encontre o menor gerador que seja controle-equivalente à ação do supervisor supremo sobre a planta.

Embora o objetivo de ambas as abordagens seja encontrar o mesmo ótimo global, elas devem ser tratadas como problemas diferentes, uma vez que a primeira caracteriza um problema combinatório, analálogo a problemas como Partição de Grafos [1, 19, 24, 36], Partição de Conjuntos [8, 12, 17, 22, 29, 31] e *Clustering* [10, 27, 28, 32, 37]. Enquanto que a segunda se assemelha a problemas de busca por máquinas de estados preditivas [2, 18, 23, 33, 34, 35, 41, 42, 47].

Embora a solução por meio da evolução de máquinas de estados seja promissora, este trabalho não chega a contemplá-la, limitando-se somente à abordagem que evolui coberturas do espaço de estados do supervisor supremo. Inicialmente optou-se por evoluir coberturas para utilizar conhecimentos específicos do problema da minimização de supervisores em proveito do algoritmo genético a ser desenvolvido. Posteriormente, por falta de tempo, não foi possível elaborar uma metodologia para a evolução de máquinas de estados, que fica como perspectiva para trabalhos futuros.

No caso da evolução de coberturas, o *genótipo* de uma solução deve representar os subconjuntos $X_i \in X, i \in I = \{1, 2, 3, ..., |X|\}$ tais que $\bigcup_{i \in I} X_i = X$, enquanto que o *fenótipo* é um gerador [1] obtido a partir da cobertura representada pelo genótipo.

5.2.1 Codificação

Escolheram-se dois tipos básicos de genótipo para representar coberturas e partições [Apêndice A]. As coberturas são representadas por matrizes binárias, enquanto que as partições por vetores de inteiros.

R1: Representação com genes de elementos da cobertura

Assim como na metodologia de solução por Programação Linear Inteira Mista (Capítulo 4), uma cobertura $C = \{X_i, i \in I\}$, tal que $X_i \subseteq X$ e I é um conjunto de índices, é representada por uma matriz $C_{|I| \times |X|}$ de números binários da seguinte forma:

- Cada linha i representa um elemento da cobertura X_i.
- Cada coluna x representa um estado do supervisor supremo, tal que $x \in X$.
- Se $x_i \in X_i$ então $C_{i,x} = 1$, caso contrário $C_{i,x} = 0$.

Por exemplo, se $X = \{0, 1, 2\}$ e $C = \{\{0, 1\}, \{0, 2\}\}$ então tem-se a seguinte representação matricial:

$$C = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

É possível perceber que qualquer permutação entre as linhas da matriz resulta na mesma cobertura. Ou seja, existe no mínimo uma redundância de $i!$ para esta representação. Outra possibilidade de redundância está nos casos em que há linhas repetidas, pois elas representam o mesmo elemento da cobertura. Além disso, esta representação permite que existam elementos da cobertura X_i que estejam contidos em outro elemento X_j.

Como cada gene representa um elemento da cobertura, é possível se reproduzir de forma sexuada os indivíduos por meio da troca de genes. Porém um elemento X_i de um

[1] Note que conforme exposto na seção 2.4 o gerador obtido a partir de uma cobertura de controle não necessariamente é único, porém seu número de estados sim.

indivíduo pode *destruir* um outro, no sentido de que pode torná-lo não determinístico e não controlável. Ou seja, há um desafio em se encontrar operadores genéticos que considerem a contextualidade do problema de forma a não realizarem reproduções destrutivas, nas quais dois pais válidos gerem dois descendentes inválidos.

É possível limitar o menor número de linhas i para a solução a ser procurada (baseando-se no número mínimo de estados estimado) e diminuir a quantidade de memória necessária para se armazenar a população e a complexidade computacional tanto do cálculo da função de avaliação quanto da transformação fenótipo-genótipo.

R2: Representação de coberturas com matriz diagonal

Assim como a representação anterior, esta representação considera que cada linha i da matriz representa um gene, porém com um diferencial, cada elemento da cobertura i deve conter o estado x tal que $i = x$, isto é, a diagonal da matriz C será sempre formada por 1.

A idéia por trás desta representação é a de que cada linha da matriz seja responsável por identificar de forma modular a qual elemento da cobertura um estado x pertence. Ou seja, no final do processo evolutivo, espera-se obter vários genes iguais que tenham convergido para o mesmo valor, identificando o agrupamento ótimo de um conjunto de estados.

A desvantagem mais clara deste tipo de representação é a necessidade de se ter uma matriz com $|X|$ genes de tamanho $|X|$, que a torna computacionalmente inviável para grandes instâncias.

R3: Representação de partições com genes inteiros

Neste caso, utiliza-se um vetor de $|X|$ números inteiros, que variam de 1 a $|I|$, como sendo o cromossomo, enquanto que cada inteiro é um gene. Tal representação permite somente que partições sejam obtidas, uma vez que não permite que um estado esteja contido em mais de um elemento da cobertura. Por exemplo, para $|X| = 4$ e $|I| = 2$, uma possível partição $C = \{\{1, 4\}, \{2, 3\}\}$ poderia ser representada da seguinte forma:

$$C = \begin{bmatrix} 1 & 2 & 2 & 1 \end{bmatrix}$$

Esta representação também possui alta redundância, possuindo $|X|!$ representações para cada partição representável. Quando comparada à representação anterior, esta representação ap-

resenta um espaço de busca menor, uma vez que as partições são casos particulares de coberturas. Para este caso, há também a preocupação em se construir operadores genéticos que não sejam *destrutivos*.

5.2.2 Mapeamento Genótipo-Fenótipo

O mapeamento entre genótipo e fenótipo deve ser comentado, pois todas as codificações apresentadas permitem a existência de elementos da cobertura (ou partição) que são subconjuntos de outros da mesma cobertura (ou partição). Quanto a este aspecto, as seguintes estratégias são adotadas.

Agregação com subelementos

Cada elemento da cobertura é considerado independentemente de estar contido em outro. Neste caso, quando se obtém o supervisor reduzido a partir de uma cobertura, pode ser que haja estados inválidos, uma vez que haverá redundância de transições.

Agregação sem subelementos

Se um elemento da cobertura está contido em outro, então resta somente o maior elemento da cobertura. Neste caso, os elementos totalmente contidos em outro são ignorados. Tal atitude aumenta bastante a redundância da representação, e restringe muito o seu poder de representatividade, pois grandes elementos da cobertura tendem a prevalecer sobre os menores.

5.2.3 Função de Avaliação

A avaliação da aptidão das soluções candidatas é uma etapa muito delicada dos AG. Conforme enunciado na Seção 5.1, ela independe do tipo de representação utilizada, pois age sobre o fenótipo. Porém é necessário levar em conta um aspecto da representação utilizada: "é possível que indivíduos inválidos sejam gerados?". Nas codificações apresentadas, sim. Logo, deve-se adotar uma estratégia para lidar com a infactibilidade dos indivíduos.

O mapeamento genótipo-fenótipo determina características importantes sobre os possíveis fenótipos gerados. Considera-se que sempre são gerados supervisores reduzidos *trim*, isto é,

ignora-se eventuais estados não acessíveis e não co-acessíveis (conforme a Seção 2.1). Logo, os tipos de violação de restrição que podem ocorrer são de *controlabilidade* e *determinismo* e a função de avaliação conta quantas violações de cada tipo ocorrem e aplica penalidades à aptidão do indivíduo. Nas funções de avaliação utilizadas, quanto maior é a aptidão de um indivíduo, menor é o seu valor numérico.

Estabeleceram-se duas funções de avaliação f_1 e f_2, representadas nas equações 5.1 e 5.2 respectivamente. Ambas recebem como argumento um supervisor reduzido $\hat{\mathbb{S}}$ e retornam um valor numérico, onde $\hat{X}$ é o conjunto de estados do gerador do supervisor reduzido, X_{sup} é o conjunto de estados do gerador do supervisor supremo, $N_c \in \mathbb{N}$ é número de violações de controlabilidade identificados em $\hat{\Psi}$ e $N_d \in \mathbb{N}$ é o número de violações de determinismo identificadas no gerador $\hat{\boldsymbol{S}}$. Os parâmetros $P_c, P_d \in \mathbb{R}$ são pesos a serem multiplicados ao número de violações de controlabilidade e determinismo respectivamente.

$$f_1(\hat{\mathbb{S}}) := |\hat{X}| + |X_{sup}|(P_c N_C + P_d N_D) \tag{5.1}$$

$$f_2(\hat{\mathbb{S}}) := \begin{cases} |\hat{X}| + |X_{sup}| P_c N_c & \text{se } N_d = 0 \\ |\hat{X}| + |X_{sup}| P_c N_c + |X_{sup}|(2 - 2^{-N_d}) & \text{se } N_d > 0 \end{cases} \tag{5.2}$$

Em ambas as funções, indivíduos válidos recebem um valor de aptidão igual ao número de estados do supervisor reduzido. Quando há violações, penalidades são aplicadas proporcionalmente ao número de estados do supervisor supremo.

Vale citar que o cálculo do número de violações de determinismo é uma operação bastante custosa computacionalmente, o que é um empecilho para o cálculo da função de avaliação.

5.2.4 Operadores genéticos

Esta seção descreve os operadores que atuam diretamente sobre o genótipo dos indivíduos. Dentre os operadores citados, há operadores tradicionais de reprodução e mutação e outros específicos para o problema.

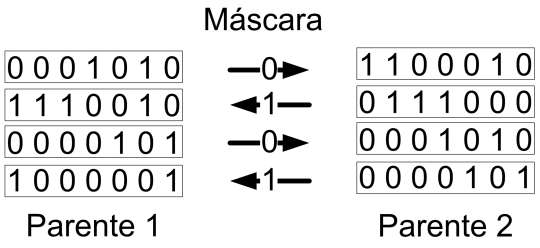

Figura 5.3: Crossover por máscara.

Crossover uniforme

Este é o primeiro operador aplicado durante a evolução. Ele monta, a partir de uma distribuição normal, uma máscara que indica quais genes do parente 1 devem ser passados ao 2 e vice-versa. Porém, tal reprodução pode gerar indivíduo infactíveis, pois não necessariamente um elemento da cobertura (ou partição) que se adequava a uma cobertura, o fará em outra, logo este operador não se adequa muito bem ao contexto do problema. A Figura 5.3 ilustra uma operação de crossover para coberturas. Quando a codificação representa partições o comportamento é análogo, porém os genes, ao invés de linhas da matriz de cobertura, são números inteiros do vetor que representa a partição.

Mutação

O operador de mutação é aplicado a cada indivíduo gerado após a reprodução. Ele altera cada gene seguindo uma pequena probabilidade $p = 0,001$, proporcionando uma pequena parcela de busca aleatória ao AG.

Corretor Lamarckiano de controlabilidade

Este operador corrige genes que podem resultar em eventuais violações de controlabilidade do fenótipo. Trata-se de um tipo de evolução de Lamarck por alterar o material genético de um indivíduo após uma evolução de seu fenótipo. Esta correção é feita quebrando-se elementos da cobertura que possuem estados com ações de controle conflitantes em dois outros elementos, um contendo os estados que habilitam o evento em questão e outro contendo os que desabilitam. Estados que possuam ação de controle igual a *don't care* permanecem em ambos os elementos.

Corretor Lamarckiano de determinismo

Analogamente ao corretor de controlabilidade, este operador corrige algumas das eventuais violações de determinismo, pois o custo computacional para transformar um gerador não-determinístico em determinístico é bastante elevado. A estratégia utilizada por este operador é a seguinte, verifica-se para cada evento $\sigma \in \Sigma$ e elemento X_j da cobertura quais os estados $x \in X$ tais que $\delta(y, \sigma) = x$, com $y \in X_j$. Em seguida verifica-se se existe algum elemento X_i que contenha todos os x, caso não exista, adiciona-se os estados faltantes ao elemento X_i que contenha o maior número de estados destino x. Este operador efetua somente uma iteração para cada evento, logo pode não corrigir o determinismo como um todo, pois os novos elementos X_i gerados podem conter novas violações de determinismo. Além disso, ao se agregar estados a um elemento da cobertura pode-se gerar inconsistências de controle que antes não existiam. Da mesma forma, o operador anterior de correção de controlabilidade, ao quebrar elementos da cobertura, pode causar violações de determinismo que antes não existiam.

Corretor Lamarckiano de redundância de partições

A codificação de partições possui alta redundância, possuindo $|X|!$ representações para cada partição representável. Devido a tal problema, utilizou-se o algoritmo de eliminação de redundância proposto em [27], que renumera os valores dos genes de acordo com a ordem em que eles aparecem, tornando a representação única para cada genótipo.

Reordenação Lamarckiana de linhas da matriz de cobertura

Este operador busca diminuir a redundância da representação de coberturas. Ele ordena os genes em ordem crescente de acordo com o número binário gerado por cada um deles.

5.2.5 Seleção

A estratégia de seleção utilizada é de seleção por torneio, pois ela ajuda a evitar problemas de convergência prematura. Segundo Beasley [5], a seleção por torneio apresenta resultados equivalentes à seleção por roleta probabilística e ambas apresentam desempenhos satisfatórios para uma grande variedade de AG.

5.2.6 Elitismo

Comumente em problemas com restrições utiliza-se elitismo [9], isto é, o melhor indivíduo de uma geração é sempre mantido para a próxima. Adotando-se tal procedimento, evita-se que o AG divirja. A divergência pode ser causada por operadores genéticos destrutivos.

5.2.7 População inicial

Outro aspecto que deve ser levado em conta é como a população inicial é gerada. Neste trabalho, a população inicial é gerada aleatoriamente sempre que se limita $|I| < |X_{sup}|$. Caso $|I| = |X_{sup}|$, sorteiam-se todos os indivíduos com exceção de um, que é a partição que representa o próprio supervisor supremo.

5.3 Resultados

A metodologia apresentada na Seção 5.2 oferece um grande número de possibilidades para se parametrizar um AG para a redução de supervisores, pois possibilita combinações de diferentes operadores genéticos, funções de avaliação e codificações. Apresentar tabelas de resultados com todas as possíveis combinações é desnecessário, pois cada operador e representação influencia de uma forma diferenciada o funcionamento do AG. Esta seção apresenta as principais influências de cada parâmetro sobre o AG e também uma tabela comparativa com os melhores resultados obtidos.

A implementação dos AG utilizou a biblioteca *Java Genetic Algorithm Package* (Apêndice B.3.1), que é uma biblioteca livre em Java e que está em constante desenvolvimento. Para a representação computacional de geradores e mapeamento genótipo-fenótipo, desenvolveu-se uma pequena biblioteca em Java. Para lidar mais eficientemente com testes de determinismo e controlabilidade utilizou-se a biblioteca COLT (Apêndice B.3.2) que implementa operações com matrizes esparsas. Todos os testes foram executados em um Intel Pentium 4 2.66 *GHz* com 1 *Gb* de memória RAM e GNU/Linux Mandrake 9.2.

O mapeamento genótipo-fenótipo por meio da agregação sem subelementos induz a evolução de modo a formar indivíduos com elementos de cobertura com um grande número de estados. Este mapeamento tende a estagnar a população, pois o operador de reprodução

não quebra genes, mas simplesmente os troca. Logo, é necessário utilizá-lo em conjunto com o operador Lamarckiano de controlabilidade, que pode quebrar elementos da cobertura.

As violações de controlabilidade são consideradas mais graves que as de determinismo, pois representam total inconsistência com a controle-equivalência entre o supervisor reduzido e o supervisor supremo. Por outro lado, violações de determinismo podem ser corrigidas sob pena do aumento do número de estados. Baseando-se neste ponto de vista, optou-se por penalizar violações de controlabilidade em uma ordem de grandeza maior que as de determinismo. Ambas as funções de avaliação penalizam eventuais violações de restrições, sendo que a função f_2 (Equação 5.2) possibilita uma penalização mais *suave* com relação ao determinismo e apresentou resultados mais satisfatório quando comparada à função f_1 (Equação 5.1).

Quando utilizados, os operadores Lamarckianos de correção de redundância demostraram proporcionar ganhos satisfatórios no desempenho dos AG. Quando não utilizados, as operações de reprodução tendem a ser mais destrutivas.

A Tabela 5.1 apresenta os tempos médios em minutos necessários para a convergência do AG para a solução ótima em 10 execuções do AG, utilizando as diferentes representações propostas. Assim como na Seção 4.3, a instância exemplo de 3 estados é um supervisor redutível a 2 estados somente por coberturas, não sendo possível reduzí-lo por partições. A instância de 6 estados foi obtida a partir do produto síncrono entre o gerador da coluna anterior e um outro gerador assíncrono (que não possui eventos em comum) de 2 estados. As instâncias das colunas posteriores foram obtidas da mesma forma. Nesta tabela, L indica que os operadores Lamarckianos de reordenação de genes, controlabilidade e determinismo foram utilizados e que a agregação de estados foi sem subelementos. Por outro lado, as linhas com *NL* indicam que os operadores Lamarckianos de controlabilidade e determinismo não foram utilizados e, além disso, que a agregação de estados foi com subelementos. Considerou-se populações de 50 indivíduos, função de avaliação f_2 com $P_c = 100$ e $P_d = 10$. A cada execução permitiu-se um tempo máximo de 5 minutos.

A Figura 5.4 ilustra a evolução interna do AG para um supervisor de 24 estados, demostrando o número de elementos da cobertura, violações de controlabilidade e violações de determinismo do melhor indivíduo em cada iteração. O número de genes dos cromossomos é limitado a 10. Escolheu-se a representação R1, na qual cada gene representa um elemento da cobertura, com operadores Lamarckianos, pois foi a que demonstrou melhores resultados. É possível perceber que as violações de controlabilidade são solucionadas primeiramente,

		Tamanhos					
Rep.	Op.	6	12	24	48	96	192
R1	NL	$0,15$	$2,13$	> 5			
R2	NL	$3,3$	> 5				
R3	NL	$2,8$	$4,92$	> 5			
R1	L	$0,08$	$0,12$	$0,89$	$1,81$	$2,2$	> 5
R2	L	$1,63$	$4,12$	> 5			
R3	L	$0,03$	$0,09$	$0,59$	$2,1$	> 5	

Tabela 5.1: Tempo de convergência para a solução ótima em minutos.

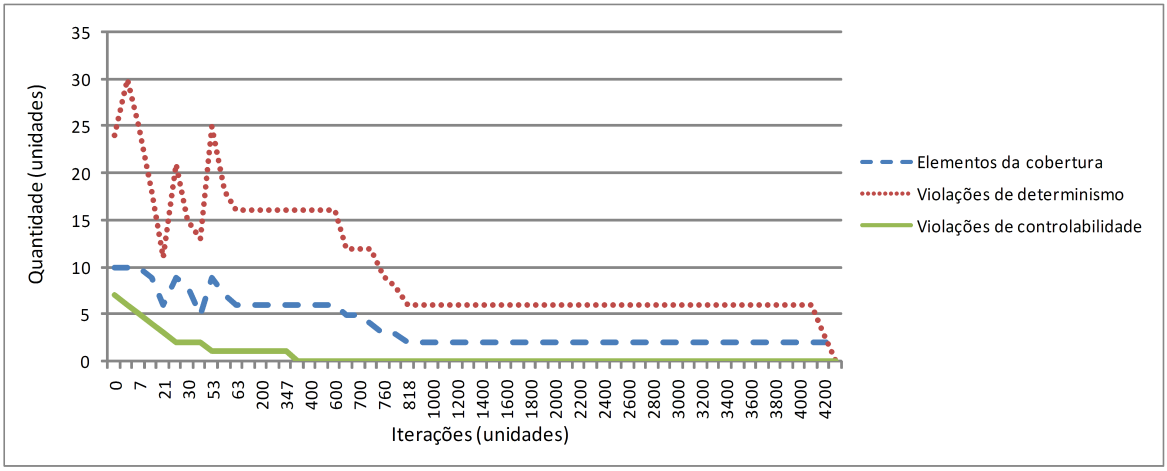

Figura 5.4: Evolução do AG para um supervisor de 24 estados utilizando R1,L.

pois possuem um peso maior na função de avaliação, enquanto que as de determinismo permanecem por mais tempo.

5.4 Conclusão e perspectivas

Esta aplicação da teoria de AG ao problema da redução de supervisores conseguiu efetivamente representar o problema e reduzir algumas instâncias de supervisores. Porém, no que diz respeito ao desempenho do AG, não foi possível superar os métodos tradicionais. Houve problemas de operadores genéticos com pouca sensibilidade de contexto e que se tornaram destrutivos. A função de avaliação se tornou excessivamente custosa para instâncias maiores do problema. Problemas de redundância, localidade e *scaling* das representações propostas prejudicaram o AG. Além disso, ao se permitir um maior número de genes nas representações R1 e R2, é necessário se restringir o tamanho da população por problemas de memória computacional. A metodologia proposta pode ser ainda mais bem explorada implementando-se

algoritmos com melhor desempenho, analisando-se mais detalhadamente o comportamento do AG e melhorando os operadores Lamarckianos de controlabilidade e determinismo. O presente trabalho se preocupou mais em obter resultados que solucionassem algumas instâncias do problema de minimização do que em solucioná-las eficientemente.

Como perspectiva na área dos AG, sugere-se a tentativa de se evoluir geradores para se encontrar o supervisor mínimo, pois a metodologia apresentada, que representa coberturas e partições, incorre em problemas de memória computacional. Existem aplicações bem sucedidas na evolução de máquinas de estados, porém nenhuma especificamente para a redução de supervisores. Fica também a possibilidade de se conseguir uma representação melhor para as coberturas do espaço de estados do supervisor supremo.

Capítulo 6

Conclusão e Perspectivas

Este trabalho apresentou uma breve revisão sobre a teoria de linguagens, geradores e de Controle Supervisório e, organizadamente, explicitou as diferenças entre supervisor não-marcador, supervisor marcador e supervisor desmarcador bem como suas relações com a marcação da planta e do sistema em malha-fechada. Além disso, revisou os conceitos da redução de supervisores, as diferentes metodologias já propostas para tal propósito e as relações entre marcação, redução, tipo de implementação e tipo de supervisor.

Contribuiu-se com uma metodologia original para a marcação de estados por meio de um evento de marcação μ. Esta metodologia possibilita a redução de supervisores sem considerar restrições de marcação, pois as trata como um problema de controlabilidade de eventos. Nesta metodologia, cada estado que era marcado recebe um auto-laço do evento de marcação. Dependendo da aplicação, o evento μ pode ser não-controlável (para a síntese de supervisores não-marcadores) ou controlável (para a redução de supervisores marcadores ou desmarcadores). Demonstrou-se matematicamente a equivalência entre a utilização do evento μ e a utilização da marcação tradicional.

Apresentou-se uma metodologia para a solução do problema da minimização de supervisores por meio de Programação Linear Inteira Mista. Neste contexto, o problema foi modelado, demonstrou-se matematicamente a validade do modelo, alguns supervisores foram minimizados com sucesso e comparou-se o desempenho da metodologia proposta com o algoritmo de redução do CTCT [Apêndice B.1.1]. A solução do problema por PLIM supera o CTCT no que diz respeito a encontrar a solução ótima do problema, porém perde computacionalmente, pois enfrenta problemas de memória para grandes instâncias do problema. Por meio de PLIM não foi possível reduzir supervisores com mais de 384 estados, enquanto que

o CTCT consegue reduzir instâncias em torno de 1000 estados.

Alternativamente tentou-se solucionar o problema utilizando-se algoritmos genéticos, por meio da evolução de coberturas do espaço de estados. Foram apresentados diferentes codificações para coberturas de controle, operadores genéticos e funções de avaliação. Conseguiu-se efetivamente reduzir pequenas instâncias do problema. Os resultados obtidos com os AG apresentaram baixo desempenho computacional, pois as dificuldades de se avaliar soluções, problemas de redundância, localidade e *scaling* das representações propostas e problemas de memória computacional prejudicaram o AG.

Como perspectiva para trabalhos futuros, fica a adequação da metodologia de marcação por eventos para o caso da marcação multitarefa [44] e a realização de uma comparação entre a complexidade computacional de se utilizar a marcação de estados tradicional e a marcação de estados por eventos. Para melhorar a metodologia de PLIM, deve-se buscar uma forma menos custosa de formulação das equações de restrições, para que problemas de falta de memória não ocorram. Com relação à metodologia de AG, sugere-se buscar uma codificação que contorne os problemas de redundância, localidade e *scaling* ou uma que evolua máquinas de estados, e não coberturas de controle. As metodologias de redução por Programação Linear Inteira Mista e Algoritmos Genéticos, embora não possam ser utilizadas para substituir os algoritmos tradicionais, representam um passo na busca por métodos alternativos para solucionar o problema da redução.

Apêndice A

Coberturas e partições de conjuntos

Uma cobertura sobre um conjunto A qualquer é um conjunto C de $n \in \mathbb{N}$ subconjuntos A_i de A tais que sua união resulte em A e que nenhum deles seja vazio. Matematicamente define-se uma cobertura C sobre A como:

$$C = \{A_i, A_i \subseteq A\}$$

tal que $i \in I = \{1, 2, 3, \ldots, n-1, n\}$, $A_i \neq \emptyset$ e com $\bigcup A_{ii \in I} = A$.

Por exemplo, a Figura A.1 representa uma cobertura sobre o conjunto $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, com $A_1 = \{1, 2\}$, $A_2 = \{2, 3, 4\}$, $A_3 = \{4, 5, 6, 8, 10\}$, $A_4 = \{7\}$ e $A_5 = \{8, 9\}$. E é tal que $A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 = A$.

Uma partição é um caso especial de cobertura na qual os subconjuntos A_i possuem interseção vazia, isto é $\bigcap A_{ii \in I} = \emptyset$. A Figura A.2 exemplifica uma partição.

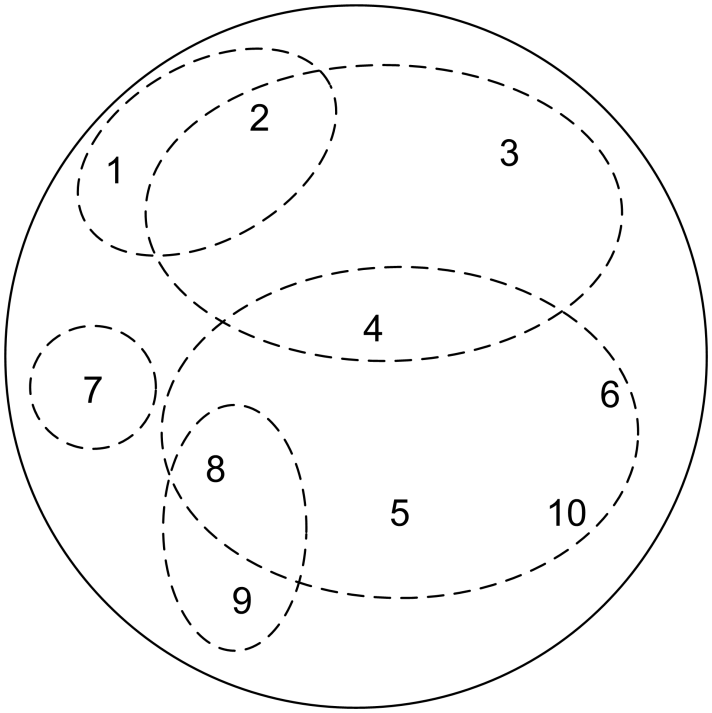

Figura A.1: Cobertura sobre o conjunto A.

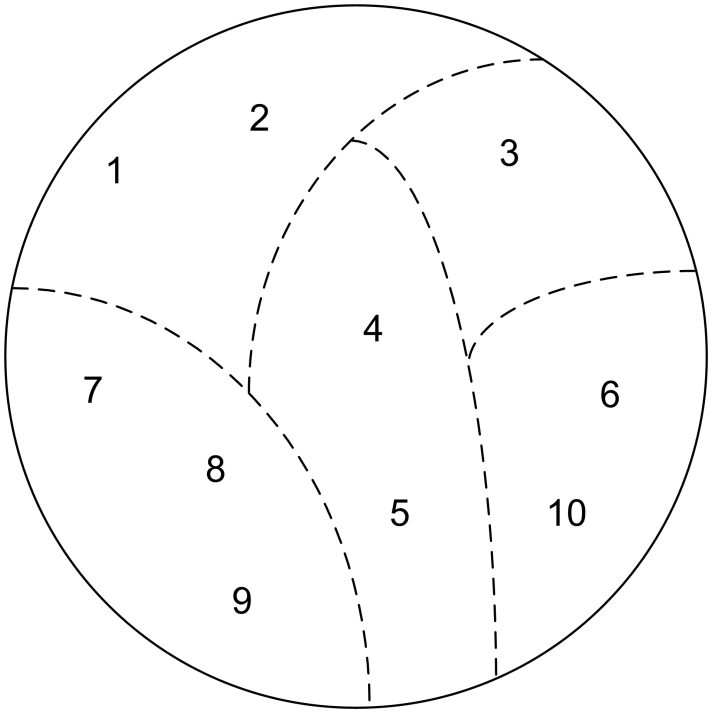

Figura A.2: Partição sobre o conjunto A.

Apêndice B

Ferramentas computacionais

Este apêndice apresenta algumas ferramentas computacionais que auxiliam, ou até mesmo tornam viável, a solução de problemas da Teoria de Controle Supervisório, Programação Linear Inteira Mista e Algoritmos Genéticos. Tais problemas, muitas vezes, tornam-se computacionalmente complexos com elevada rapidez, mesmo que para sistemas reais simples.

B.1 Redução de Supervisores

O problema da redução de supervisores de sistemas a eventos discretos, que são modelados por geradores e solucionados pela TCS, tendem a aumentar seu tamanho rapidamente com relação ao número de sistemas modelados e sua complexidade. As ferramentas apresentadas a seguir auxiliam a solução de problemas desta classe.

B.1.1 CTCT

O CTCT é um aplicativo computacional desenvolvido na Universidade de Toronto (Canadá) utilizado para a análise, síntese de controladores e redução de supervisores para sistemas a eventos discretos.

O CTCT disponibiliza operações como trim, produto síncrono, *supC*, minimização de estados de geradores e redução de supervisores utilizando o algoritmo proposto em [49], que reduz supervisores utilizando somente partições do espaço de estados do supervisor.

Informações mais detalhadas podem ser encontradas em [55] ou no sítio
`http://www.toronto.control.edu/DES`.

Na sua representação de geradores, estados e eventos são representados por números. Para o caso dos eventos, números ímpares representam eventos não-controláveis, enquanto que número pares representam eventos controláveis.

O CTCT possui uma funcionalidade muito útil, que avisa o usuário com antecedência quando há falta de memória e não é possível solucionar um problema devido a limitações de *Hardware*.

B.1.2 GRAIL

O GRAIL [45] é composto por uma biblioteca de controle supervisório, desenvolvida pelo Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina, e por um software desenvolvido na Universidade de Waterloo. Ele fornece praticamente as mesmas funcionalidades que o CTCT, mas com uma interface mais amigável, pois seus arquivos são legíveis e editáveis em editores de texto quaisquer e é possível fazer *scripts* para execução de operações repetitivas. Porém esta ferramenta apresenta um desempenho computacional muito inferior ao CTCT e não alerta ao usuário eventuais limitações de *Hardware*.

B.2 Programação Linear Inteira Mista

Nesta seção, apresentam-se as ferramentas computacionais utilizadas para auxiliar a solução de problemas de Programação Linear Inteira Mista.

B.2.1 Linguagem de modelagem: AMPL

Além de modelar um programa matematicamente e separá-lo em função objetivo, variáveis de decisão e restrições, é necessário um esforco a mais para solucioná-lo na prática. Problemas considerados grandes necessitam de aplicativos computacionais para serem resolvidos. Uma etapa indispensável para a solução dos mesmos é a tradução dos modelos matemáticos para uma linguagem de modelagem que possa ser *entendida* pelo aplicativo a ser utilizado.

A linguagem de modelagem escolhida neste trabalho foi a *A Mathematical Programming Language* (AMPL) [20], por se tratar de uma linguagem simples, amplamente utilizada e compatível com diversas implementações de algoritmos de otimização. A linguagem AMPL é bastante intuitiva e separa o problema em três arquivos: modelo, dados e comandos. O arquivo do *modelo* contém o equivalente à formulação do problema em linguagem matemática e o mesmo arquivo é sempre utilizado para solucionar diversas instâncias do problema. O arquivo de *dados* contém os parâmetros particulares a uma instância do problema, como dimensões das matrizes e eventuais dados de inicialização. Já o arquivo de *comandos* especifica comandos ao aplicativo, indicando por exemplo, método de solução a ser utilizado, número de iterações, tempo máximo de execução, variáveis a serem observadas, etc. A separação nestes três arquivos torna a linguagem bastante versátil e modular, simplificando testes de várias instâncias completamente diferentes bem como eventuais correções ao modelo.

B.2.2 CPLEX

Ferramenta computacional [30] proprietária utilizada para solucionar problemas de programação linear inteira mista e que aceita arquivos AMPL como entrada. Ela possui algoritmos robustos e de alto desempenho e, além disso, oferece interfaces para a utilização de seus algoritmos por meio de programas feitos em Java, C, C++, Fortran e Visual Basic. O CPLEX é considerado a melhor ferramenta profissional para programação linear, pogramação linear inteira, programação linear inteira mista e programação quadrática inteira mista. Maiores detalhes podem ser obtidos no sítio `http://www.ilog.com/products/cplex/`.

B.3 Algoritmos Genéticos

A solução de problemas por Algoritmos Genéticos requer, muitas vezes, que haja grande esforço de criação, sendo muito difícil que se construa uma ferramenta genérica o suficiente para atender a todas as possibilidades ou, até mesmo, à maioria delas. Desta forma, opta-se normalmente por bibliotecas de programação ao invés de aplicativos prontos.

B.3.1 JGAP

Java Genetic Algorithm Package é uma biblioteca livre que dá suporte à solução de problemas de Algoritmos Genéticos. Ela é toda escrita em Java, é muito bem documentada e

possui diversos exemplos. Por meio dela, é possível criar facilmente novas codificações, operadores genéticos e funções de avaliação. Informações mais detalhadas podem ser encontradas no sítio `http://jgap.sourceforge.net/`.

B.3.2 COLT

O projeto COLT oferece um conjunto de bibliotecas livres para computação técnica e científica de alto desempenho, utilizando a linguagem Java. Dentre as bibliotecas oferecidas, utilizou-se especificamente a biblioteca que dá suporte a matrizes esparsas. Maiores detalhes podem ser encontrados no sítio `http://acs.lbl.gov/~hoschek/colt/`.

Referências Bibliográficas

[1] Adriana Apetrei, Ovidiu Gheoghies, Henri Luchian, and Rolf Drechsler. An evolutionary approach to graph partitioning. *Evolutionary Methods for Design Optimisation and Control*, 2002.

[2] S. G. Araujo, A. C. P. Pedroza, and A. C. Mesquita. Evolutionary synthesis of communication protocols. In *10th International Conference on Telecommunications*, pages 986–993, Tahiti, 2003.

[3] Daniel Ashlock. *Evolutionary Computation for Modeling and Optimization.* Springer Science+Business Media, Inc., 2005.

[4] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *Inter-University Commitee on Computing.*, 1993.

[5] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 2, research topics. *Inter-University Commitee on Computing.*, 1993.

[6] G. Bittencourt. *Inteligência Artificial Ferramentas e Teoria.* Editora da UFSC, 2006.

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

[8] P. Chu and J. Beasley. A genetic algorithm for the set partitioning problem. Technical report, Department of Electrical and Computer Engineering, University of Toronto, Imperial College, The Management School, London, England, 1995.

[9] C. Coello. Theoretical and numerical constraint handling techniques used with evolutionary algorithms: A survey of the state of the art. *Art. Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, 2002.

[10] Rowena Marie Cole. Clustering with genetic algorithms. Master's thesis, Department of Computer Science, University of Western Australia, 1998.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[12] L. Coslovich, R. Pesenti, and W. Ukovich. Large-scale set partitioning problems: Some real-world instances hide a beneficial structure. *Technological and Economic Development of Economy*, 2006.

[13] J. Czyzyk, M. Mesnier, and J. Moré. The neos server. *IEEE Journal on Computational Science and Engineering, 5*, 1998.

[14] F.A.C. da Silva Neto. Controllability analysis for composed discrete event systems and realization of a local modular supervisory control for a filling shop using iec 1131-3 languages. Technical report, Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina, Brasil, 2005.

[15] F.A.C. da Silva Neto, M. H. de Queiroz, and E. Camponogara. Representação em programação linear inteira mista do problema de minimização de supervisores. *Simpósio Brasileiro de Automática Inteligente (SBAI)*, 2007.

[16] Charles Darwin. *A Origem das Espécies - Col. A Obra Prima de Cada Autor - Série Ouro*. Martin Claret., 2004.

[17] Teresa Galvão Dias, Jorge Pinho de Sousa, and João Falcão e Cunha. Genetic algorithms for the bus driver scheduling problem: a case study. *Journal of the Operational Research Society 53*, 2002.

[18] Vit Fabera, Vlastimil Janes, and Maria Janesova. Automata construct with genetic algorithm. In *DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pages 460–463, 2006.

[19] Per-Olof Fjällström. Algorithms for graph partitioning: A survey. *Linköping University Electronic Press*, 1998.

[20] Robert Fourer, David M. Gay, and Brian W. Kernighan. *A Modeling Language for Mathematical Programming*, 1990.

[21] David E. Goldberg, Kelsey Milman, and Christina Tidd. Genetic algorithms: A bibliography. Technical Report IlliGAL Report No 97001, Department of General Engineering, 117 Transportation Building, 104 South Mathews Avenue, Urbana, IL 61801-2996, 1997.

[22] William A. Greene. Genetic algorithms for partitioning sets. *International Journal on Artificial Intelligence Tools*, 2001.

[23] Petr Hoffmann. Learning restarting automata by genetic algorithms. In *SOFSEM 2002: Student Research Forum*, pages 15–20, Milovy, Czech Republic, 2002. Mária Bieliková.

[24] C. Hohn and C. Reeves. Graph partitioning using genetic algorithms. *IEEE Computer Society Press*, 1996.

[25] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[26] John E. Hopcroft and Jefferey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Adison-Wesley Publishing Company, Reading, Massachusets, USA, 1979.

[27] Eduardo R. Hruschka. Applying a clustering genetic algorithm for extracting rules from a supervised neural network. In *IJCNN '00: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)-Volume 3*, page 3407, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0619-4.

[28] Eduardo R. Hruschka and Nelson F. F. Ebecken. A genetic algorithm for cluster analysis. *Intell. Data Anal.*, 7(1):15–25, 2003. ISSN 1088-467X.

[29] William H. Hsu, Michael Welge, Jie Wu, and Ting-Hao Yang. Genetic algorithms for selection and partitioning of attributes in large-scale data mining problems. *In Proceedings of the AAAI-99 and GECCO-99 Workshop on Data Mining with Evolutionary Algorithms*, 1999.

[30] ILOG. *ILOG CPLEX 9.0 User's Manual*, 2003.

[31] Bastian Kneer, Martin Holzer, and Markus Rupp. Novel genome coding of genetic algorithms for the system partitioning problem. *International Symposium on Industrial Embedded Systems*, pages 134–141, 2007.

[32] Petra Kudova. Clustering genetic algorithm. In *DEXA '07: Proceedings of the 18th International Conference on Database and Expert Systems Applications*, pages 138–142, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2932-1.

[33] Benoit Leblanc, Evelyne Lutton, and Jean-Paul Allouche. Inverse problems for finite automata: A solution based on genetic algorithms. *Lecture Notes in Computer Science*, 1363:157, 1998.

[34] P. G. Lobanov and A. A. Shalyto. Application of genetic algorithms for automatic construction of finite-state automata in the problem of flibs. In *Journal of Computer and Systems Sciences International*, pages 792–801. Pleiades Publishing, Ltd., 2007.

[35] Simon M. Lucas and T. Jeff Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27 (7):1063–1074, 2005. ISSN 0162-8828. doi: http://dx.doi.org/10.1109/TPAMI.2005.143.

[36] Harpal Maini, Kishan Mehrotra, Chilukuri Mohan, and Sanjay Ranka. Genetic algorithms for graph partitioning and incremental graph partitioning. *in Proceedings of the IEEE Supercomputing Conference*, 1994.

[37] U. Maulik and S. Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern Recognition*, 33:1455–1465, 2000.

[38] Z. Michalewicz, D. Dasgupta, R. G. Le Riche, and M. Schoenauer. Evolutionary algorithms for constrained engineering problems. *Computers and Industrial Engineering Journal*, 1996.

[39] Zbigniew Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. *Proceedings of the 4th Annual Conference on Evolutionary Programming*, 1995.

[40] Melanie Mitchell. *An Introduction to Genetic Algorithms.* The MIT Press, 1999.

[41] Nattee Niparnan and Prabhas Chongstitvatana. An improved genetic algorithm for the inference of finite state machine. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, page 189, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-878-8.

[42] H. Pistori, P. S. Martins, and A. A. Castro Jr. Adaptive finite state automata and genetic algorithms: Merging individual adaptation and population evolution. In *Proceed. Int. Conf. on Adaptive and Natural Computing Algorithms - ICANNGA 2005*, Coimbra, Portugal, 2005.

[43] M. H. Queiroz and J. E. R. Cury. Synthesis and implementation of local modular supervisory control for a manufacturing cell. In *Proc. 6th Internat. Workshop on Discrete Event Systems*, pages 377–382, Zaragoza, Spain, out. 2002.

[44] M. H. Queiroz and J. E. R. Cury. Modular multitasking supervisory control of composite discrete-event systems. In *Proc. 16th IFAC WORLD CONGRESS*, volume 1, pages 1–6, Prague, Czech Republic, 2005.

[45] C. Reiser, A. E. C. da Cunha, and J. E. R. Cury. Environment grail for supervisory control of discrete event systems. In *Proc. 8th Workshop on Discrete Event Systems*, volume 1, pages 390–391, Ann Arbor, USA, 2006.

[46] Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms.* Springer Science+Business Media, Inc., 2006.

[47] Hooman Shayani and Peter J. Bentley. A more bio-plausible approach to the evolutionary inference of finite state machines. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2937–2944, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-698-1. doi: http://doi.acm.org/10.1145/1274000.1274039.

[48] L. F. Sivollela, A. E. C. da Cunha, and R. Ades. Redução de supervisores como ferramenta para a implementação de supervisores em controladores discretos. *XVI Congresso Brasileiro de Automática*, 2006.

[49] R. Su and W. M. Wonham. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems*, 14(1):31–53, jan. 2004.

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

[51] A.D. Vieira. Implementação de estrutura de controle de sistema a eventos discretos em controlador lógico programável utilizando a teoria controle supervisório modular local. Technical report, Universidade Federal de Santa Catarina and Pontifícia Universidade Católica do Paraná, Brazil, 2003.

[52] Yong Wang, Zixing Cai, Yuren Zhou, and Wei Zeng. An adaptive tradeoff model for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 2007.

[53] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 2004.

[54] L. A. Wolsey. *Integer Programming.* John Wiley, 1998.

[55] W. M. Wonham. *Supervisory Control of Discrete-Event Systems.* Systems Control Group, University of Toronto, 2006.