DISSERTAÇÃO DE MESTRADO Nº 1163

ABSTRAÇÃO DO SUPERVISOR PARA SISTEMAS COM RETRABALHO VISANDO A SOLUÇÃO DE UM PROBLEMA DE PLANEJAMENTO

FRANKLIN MEER GARCIA ACEVEDO

DATA DA DEFESA: 18/12/2019

Universidade Federal de Minas Gerais

Escola de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica

ABSTRAÇÃO DO SUPERVISOR PARA SISTEMAS COM RETRABALHO VISANDO A SOLUÇÃO DE UM PROBLEMA DE PLANEJAMENTO

FRANKLIN MEER GARCIA ACEVEDO

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do Título de Mestre em Engenharia Elétrica.

Orientadora: Profa. Patrícia Nascimento Pena

Belo Horizonte - MG

Dezembro de 2019

Garcia Acevedo, Franklin Meer.
G216a Abstração do supervisor para sistemas com retrabalho visando a solução de um problema de planejamento [recurso eletrônico] / Franklin Meer Garcia Acevedo . - 2019.
1 recurso online (xxx, 112 f. : il., color.) : pdf.

Orientador: Patrícia Nascimento Pena.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f. 75-82.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia elétrica - Teses. 2. Abstração - Teses. 3. Planejamento - Teses. I. Pena, Patrícia Nascimento. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

Ficha catalográfica: Biblioteca Profº Mário Werneck, Escola de Engenharia da UFMG

"Abstração do Supervisor para Sistemas com Retrabalho Visando a Solução de um Problema de Planejamento"

FRANKLIN MEER GARCIA ACEVEDO

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 18 de dezembro de 2019.

Por:

Prof. Dr. Patrícia Nascimento Pena
DELT (UFMG)

Prof. Dr. Carlos Andrey Maia
DEE (UFMG)

Prof. Dr. Ricardo Hiroshi Caldeira Takahashi
DMAT (UFMG)

Dedico esse trabalho a meus pais
Pedro Garcia e Maria Acevedo

Agradecimentos

Eu gostaria de começar agradecendo primeiramente a meu A-do-nai dizendo, Baruch Atá A-do-nai Heloheinu Melech Aholam (Bendito és Tu, A-do-nai, nosso D-us, Rei do Universo) por ter me acompanhado nesta conquista. Quero agradecer a meu pai, Pedro Garcia, por me ensinar a caminhar nos caminhos cristãos e por me educar com tanto amor (a suas correções fizeram de mim a pessoa que hoje sou). À minha mãe, Maria Acevedo, por ser essa mulher na minha vida que sempre me encorajou a lutar pelos meus sonhos, você me fez acreditar que nada é impossível. Á meu irmão, Jefferson Garcia, não há palavras para descrever o quanto eu te amo, obrigado por ter sido tão paciente comigo todos estos anos, você irmãozinho sempre foi e será minha inspiração.

Por último, e não menos importante, quero agradecer a minha orientadora, Professora Patrícia Nascimento Pena, pela confiança, amor e carinho. Obrigado por sempre levar salgadinhos no LACSED em cada reunião da semana (sempre chegava dizendo " a titia trouxe bolo"). Não há palavras para expressar quão grato me sinto de ter tido você como minha orientadora. Eu também Gostaria de agradecer todos os meus amigos do LACSED, pela paciência e amor que sempre tiveram comigo. Além disso, gostaria de agradecer as agências de financiamento à pesquisa FAPEMIG, CAPES e CNPQ, por investirem na minha educação.

“La liberté commence où l’ignorance finit.”

(Victor Hugo)

Resumo

Autômatos de Estados Finitos e a Teoria de Controle Supervisório (TCS) têm sido usados para modelar e solucionar problemas de planejamento de tarefas em sistemas de manufatura. Ao modelar o sistema usando a TCS, garante-se segurança e não bloqueio do sistema sob controle, além de e garantir que as soluções geradas poderão ser executadas no sistema.

Trabalhar com PO-Abstrações do comportamento em malha fechada permite reduzir ainda mais o universo de busca do problema de planejamento sem perder as qualidades advindas da TCS. No presente trabalho, faz-se uma extensão de resultado anterior, para o caso em que a abstração do comportamento em malha fechada não é PO-Abstração. Um procedimento para construir, a partir de uma sequência selecionada por seu desempenho sob algum critério, uma linguagem que seja executável até o final na planta. Esta extensão torna possível o uso de abstrações do comportamento em malha fechada de sistemas que por natureza não são PO-Abstrações.

Abstract

Finite state automata and Supervisory Control Theory (TCS) have been used to model and solve task planning problems in manufacturing systems. By modeling the system using TCS, it ensures security and non-blocking of the system under control, and ensures that the generated solutions can be executed on the system.

Working with PO-Abstractions closed-loop behavior allow you to further reduce the universe of planning problem search without losing the qualities that come from TCS. In the present work, an extension of the previous result is made, for the case where the closed-loop behavior abstraction is not a PO-Abstraction. A procedure for building, from a sequence selected by its performance under some criteria, a language that is executable to the end of the plant. This extension makes it possible to use closed-loop behavior abstractions of systems that are not, by nature, PO-Abstractions.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

TCS	Teoria de Controle Supervisório
PO	Propriedade do Observador
BDD	Binary Decision Diagrams
DES	Discrete Event Systems
DESFM	Discrete Event Systems with Flexible Marking
VNS	Variable Neighborhood Search
MFE	Most Frequent Event
CSA	Clonal Selection Algorithm

Lista de símbolos

N	número de tarefas.
M	conjunto de máquinas.
J_n	tarefa.
M_m	máquina.
O_{nm}	sequência de operação.
$\in$	pertence à.
!	fatorial.
$(n!)^m$	máximo de soluções.
Σ^*	conjunto de todas as cadeias possíveis em Σ.
$\geq$	maior ou igual (os operadores são números reais).
Σ	conjunto de eventos ou alfabeto.
ϵ	cadeia vazia.
$\|s\|$	comprimento da cadeia s.
$=$	igual.
$\subseteq$	contido em ou igual a.
L	linguagem.

$\emptyset$	conjunto vazio.
st	concatenação.
$t \leqslant s$	t é prefixo de s.
$\overline{L}$	prefixo fechamento da linguagem L.
L^*	fechamento Kleene da linguagem L.
$\cup$	união.
G	autômato.
Q	conjunto de estados do autômato.
δ	função de transição do autômato.
q_0	estado inicial do autômato.
Q_m	conjunto de estados marcados do autômato.
Γ	conjunto de eventos factíveis.
$\mathcal{L}(G)$	linguagem gerada pelo autômato G.
$\mathcal{L}_m(G)$	linguagem marcada pelo autômato G.
$Ac(G)$	parte acessível do autômato G.
$CoAc(G)$	parte coacessível do autômato G.
$Trim(G)$	parte acessível e coacessível do autômato G.
$\times$	produto de autômatos (os operadores são autômatos).
$\neq$	diferente.
$\wedge$	operador lógico E.
$\cap$	interseção.
$P : \Sigma^* \rightarrow \Sigma_i^*$	projeção natural.
$\notin$	não pertence.

$P^{-1}_{\Sigma \to \Sigma_i}$	projeção inversa.
$\exists$	existe.
$\Longrightarrow$	implica.
$\forall$	implica.
$\|\|$	operador de composição paralela.
$\overline{V}$	autômato verificador.
M	autômato auxiliar.
Σ_r	conjunto de eventos relevantes.
Σ_c	conjunto de eventos controláveis.
Σ_u	conjunto de eventos não controláveis.
E	especificação.
K	linguagem desejada.
S	autômato que implementa o supervisor.
s_{ot}	sequência de produção.
A	linguagem recuperada.
M_i	autômato para a máquina i.
$\Pi(t)$	operador de permutação dos eventos de uma cadeia t.
$SupC(K, G)$	operação que resulta na máxima sublinguagem controlável de K em relação à G.
E_q	especificação de quantidade.
$r_A(q)$	rótulo de uma composição.
$d'(\delta_p(q_0^P, t))$	rótulo de uma projeção.

$d(d'(q))$	função que converte um conjunto de tuplas em uma tupla de conjuntos da função que calcula o rótulo de uma projeção.
q_a	estado ativo.
w_{i_k}	ciclo de trabalho da máquina i.
$\sigma^{\uparrow}_{i_k}$	evento (ligar) controlável que dispara o início de w_{i_k}.
$t^{\downarrow}_{i_k}$	evento (desligar) não controlável que causa o final de w_{i_k}.
$\Sigma_{\sigma\uparrow}$	conjunto de eventos de liga.
$\Sigma_{t\downarrow}$	conjunto de eventos de desliga.
Σ^{C}_{fix}	conjunto de eventos controláveis que não pertencem ao conjunto $\Sigma_{\sigma\uparrow}$ (liga).
Σ_{Ru}	conjunto de eventos não controláveis renomeados.
M^{Ru}	conjunto de máquinas que contém eventos do conjunto Σ_{Ru} (eventos renomeados).
E_p	especificação de quantidade.
$\mathcal{F}^{\uparrow}$	linguagem construída de um procedimento (algoritmo proposto).
T_{Eq}	autômato produto de uma composição paralela.
$\setminus$	diferença de conjuntos.

1

Introdução

A indústria de manufatura vem desempenhando um papel muito importante na evolução da sociedade moderna com o processo evolutivo de fabricação, desde a produção artesanal, produção em massa até personalização em massa de produtos (Tao et al., 2017). Essas industrias de manufatura de hoje são desafiadas a atender às crescentes necessidades dos clientes, com prazo de entrega menor e menos desperdício na produção (Jovane et al., 2008), (Smith & Ball, 2012).

Com o objetivo de garantir um menor prazo de entrega de forma a atender a necessidade dos clientes, o problema de planejamento surge. Os problemas de planejamento contêm um conjunto de tarefas a serem realizadas e um conjunto de recursos disponíveis para executar essas tarefas. Dadas tarefas e recursos, juntamente com algumas informações sobre incertezas, o problema geral é determinar o tempo das tarefas enquanto reconhece a capacidade dos recursos. Esse problema geralmente surge dentro de uma hierarquia

de tomada de decisão na qual o agendamento segue algumas decisões anteriores mais básicas (Baker & Trietsch, 2013). O agendamento como processo de tomada de decisão, desempenha um papel importante na maioria dos sistemas de fabricação e produção (Pinedo, 2008).

Para minimizar o tempo de execução de todas as tarefas em um sistema de manufatura é importante encontrar uma sequência de operações para cada subsistema (máquina) (Pinson, 1995). O Problema de planejamento é considerado um problema de otimização e diversas técnicas já foram desenvolvidas para a sua solução (Arisha et al., 2001).

Dentre as técnicas, utilizam-se a modelagem dos sistemas como Sistemas a Eventos Discretos associada à utilização da Teoria de Controle Supervisório (TCS) proposta por Ramadge & Wonham (1989). A TCS permite a obtenção de uma estrutura, chamada supervisor, que representa o comportamento livre de falhas e bloqueios do sistema, respeitando todas as restrições impostas. A TCS permite uma redução significativa do espaço de busca pela solução ótima, porém, ela não tem como finalidade realizar o escalonamento ótimo.

Uma redução ainda maior do universo de busca pelas sequências ótimas é obtida através do uso de abstrações da abordagem monolítica do supervisor. Vilela & Pena (2016) propõe uma abstração do modelo monolítico que consistiu em retirar do modelo as informações que não são utilizadas diretamente no processo de formação de soluções candidatas. Porém a abstração realizada no supervisor deve possuir a propriedade do observador que foi apresentada por Wong (1998). Portanto, se essa propriedade é garantida na abstração, então qualquer plano de produção escolhido poderá ser executado de inicio a fim no sistema a controlar.

O resultado principal de Vilela & Pena (2016) só é aplicável se a abstração do comportamento em malha fechada possui a propriedade do observador. Apoiando-se na ideia de poder trabalhar com abstrações que visam reduzir o espaço de busca pelas soluções ótimas para o problema de planejamento, este trabalho estende o resultado de Vilela & Pena (2016) para o caso em que a abstração do comportamento em malha fechada não possui a propriedade do observador e apresenta um procedimento para construir, a partir de uma sequência selecionada sob algum critério, uma linguagem que seja executável até o

final na planta. Esta extensão torna possível o uso de abstrações do comportamento em malha fechada de sistemas cujas abstrações não são PO-Abstrações.

Este trabalho é organizado da seguinte maneira: o Capítulo 2 apresenta uma revisão bibliográfica sobre o problema de planejamento e de algumas aplicações da Teoria de Controle Supervisório em problemas de planejamento, além disso, são apresentados alguns trabalhos que utilizam abstrações em diferentes contextos. O Capítulo 3 apresenta os conceitos que fundamentam a modelagem por linguagens e autômatos, além de apresentar resumidamente os conceitos essenciais de trabalhos que verificam e buscam a propriedade do observador em uma abstração. O Capítulo 4 apresenta o trabalho que foi usado como base para a construção deste trabalho. O Capítulo 5 apresenta um exemplo de aplicação do resultado principal deste trabalho. Finalmente, o Capítulo 6 traz as conclusões do trabalho proposto.

1.1 Publicação

Franklin Garcia; Patrícia Pena. "Abstração do Supervisor para Sistemas com Retrabalho visando a Solução de um Problema de Planejamento". In: ANAIS DO 14º SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 2019, Ouro Preto. Anais eletrônicos... Campinas, GALOÁ, 2020. Disponível em: <https://proceedings.science/sbai-2019/papers/abstracao-do-supervisor-para-sistemas-com-retrabalho-visando-a-solucao-de-um—problema-de-planejamento>.

2

Revisão da Literatura

Este capítulo apresenta uma revisão da literatura acerca dos temas abordados neste trabalho, tais como o problema de planejamento e a aplicação da Teoria de Controle Supervisório na solução de problemas de planejamento.

2.1 Problema de Planejamento

O planejamento de tarefas ou problema de escalonamento é uma das atividades industriais mais importantes, especialmente no planejamento de sistemas de manufatura (Arisha et al., 2001). O planejamento de tarefas, em outros termos, é um problema de otimização em que várias tarefas são atribuídas a máquinas em momentos específicos ao tentar minimizar uma variável de interesse (makespan, potência elétrica).

O planejamento de tarefas tem um impacto direto sobre a eficiência e custo de produção em um sistema de manufatura, assim, tem atraído a atenção de pesquisadores desde 1956 (Zhang et al., 2019) e é visto como um problema difícil em otimização combinatória (Liu et al., 2018).

Habitualmente, aplica-se o processo de planejamento em sistemas de manufatura compostos de M máquinas diferentes, onde N tarefas devem ser realizadas. A transformação de um insumo bruto em um insumo acabado é considerado como uma tarefa acabada e é representado como J_n com $n \in \{1, \ldots, N\}$ tarefas que serão processadas em diferentes máquinas M_m, com $m \in \{1, \ldots, M\}$, numa ordem específica. Cada tarefa J_n é composta de uma sequência de operação O_{nm}, cada operação O_{nm} sendo executada em uma máquina selecionada a partir do conjunto disponível M_m. O elemento t_{nm} indica a unidade de tempo de processamento da operação O_{nm} na máquina M_m. O tempo decorrido para a realização de todas as tarefas é chamado de *makespan* (Cavalca & Fernandes, 2018).

Com a formulação do problema definido, é possível obter o escalonamento das operações sobre as máquinas de tal forma que o *makespan* seja minimizado. Isto consiste em determinar a sequência das operações O_{nm} de forma a minimizar a função de tempos de finalização (*makespan*). A Figura 2.1 mostra um Diagrama de Gantt que é um dos modelos mais simples e mais utilizados na representação analógica do planejamento de tarefas. Em sua forma básica, o gráfico de Gantt exibe a alocação de recursos ao longo do tempo, com recursos específicos mostrados ao longo do eixo vertical. O gráfico de Gantt assume que os tempos de processamento são conhecidos com certeza (Baker & Trietsch, 2013).

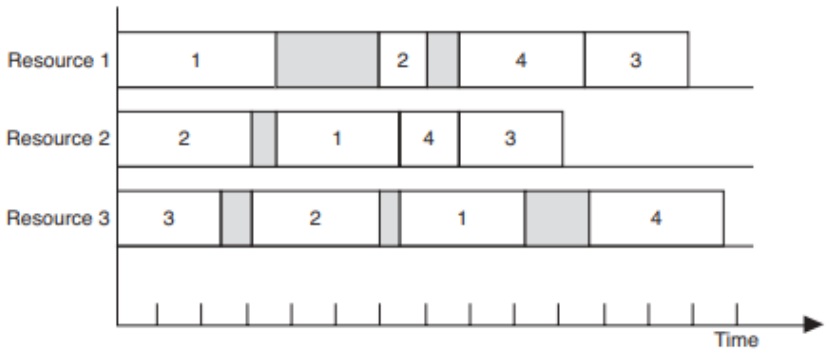


Figura 2.1: Diagrama de Gantt. Adaptado de Baker & Trietsch (2013).

Ghallab et al. (2004) propõe um modelo conceitual para o processo de planejamento, exposto na Figura 2.2.

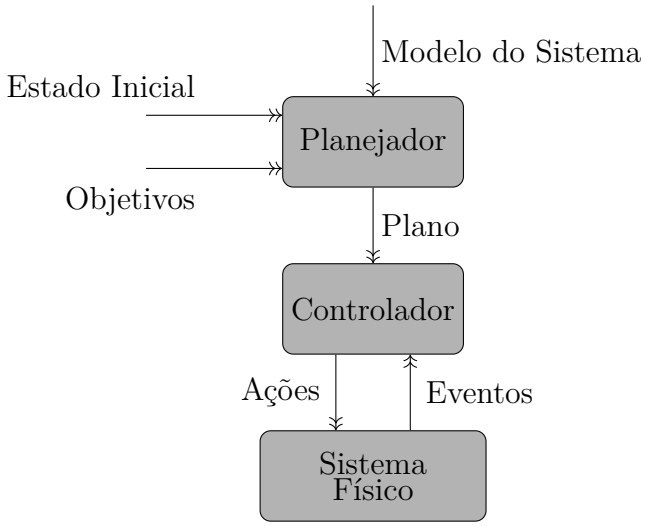


Figura 2.2: Modelo conceitual para o processo de planejamento. Adaptado por Vilela & Pena (2016)

O modelo é composto por três camadas: Planejador, Controlador e Sistema Físico. O planejador recebe informação do modelo do sistema, o estado inicial e os objetivos a serem alcançados. O Planejador é o responsável por encontrar um plano ótimo de modo que os objetivos desejados sejam atingidos, ou seja, é encontrado o plano a ser executado. O plano encontrado pelo Planejador é então enviado ao Controlador, que tem como função fazer com que esse plano seja executado no Sistema Físico. O Controlador, por sua vez, acompanha o Sistema Físico observando os eventos e atuando sobre o mesmo de acordo com o especificado pelo plano.

Encontrar o melhor plano pelo planejador é um problema de otimização e diversas técnicas já foram desenvolvidas para a sua solução. Arisha et al. (2001) apresentam essas técnicas em dois grupos: técnicas tradicionais e técnicas avançadas.

As técnicas tradicionais podem ser classificadas em duas categorias: técnicas analíticas e heurísticas. Entre as técnicas analíticas encontra-se Enumeração Explícita, *Branch and Bound*, Numeração Parcial, Programação Linear, Programação Inteira entre outras.

A técnica de Enumeração Explícita gera uma árvore de enumeração completa. As folhas da árvore representam todas as soluções viáveis. O caminho desde a raiz a uma folha representa uma solução. A principal limitação deste método é o tamanho de pesquisa gerada pois há um máximo de $(n!)^m$ soluções a considerar (onde n é o número de tarefas e m o número de máquinas).

Outra técnica utilizada é a conhecida como *Branch and Bound* que consiste em cortar ramificações de árvores enumeradas e, portanto, reduzir substancialmente o número dos nós gerados. Uma solução pode ser encontrada sistematicamente examinando os subconjuntos de uma solução viável (Arisha et al., 2001). Em outras palavras, o método consiste em particionar o problema maior em dois ou mais subproblemas e seguidamente calcular o limite inferior para a solução ótima de cada subproblema (Baker & Trietsch, 2013).

Ignall & Schrage (1965) introduziram a técnica *Branch and Bound* para minimizar o tempo total de fluxo de tarefas em n trabalhos e duas máquinas. Autores como (McMahon & Burton, 1967), (Ashour, 1970), (Gupta, 1970), (Gonzalez & Sahni, 1978), (Bansal, 1979), entre outros, desenvolveram diferentes métodos da técnica *Branch and Bound* para várias medidas de desempenho tais como *makespan*, tempo médio de fluxo, atraso médio e atraso máximo.

As técnicas heurísticas, são capazes de obter boas soluções com esforço computacional limitado. Em contraste com o método *Branch and Bound*, essas técnicas não garantem que o ótimo seja encontrado, mas elas são relativamente simples e efetivas. Dentre as técnicas heurísticas, o método de amostragem aleatória é uma estratégia que consiste em usar algum mecanismo aleatório que constrói, avalia e identifica a melhor sequência na amostra. A amostragem aleatória é um procedimento para obter boas soluções para problemas combinatórios com lógica simples e direta e esforço computacional limitado (Baker & Trietsch, 2013).

O método de pesquisa em vizinhança parte de dois elementos básicos que são o conceito de vizinhança de uma solução e um mecanismo para gerar vizinhanças. O mecanismo de geração é um método de tomar uma sequência como uma semente e criar sistematicamente uma coleção de sequências relacionadas. Alguns estudos experimentais indicaram que mesmo a versão fundamental do algoritmo de busca de vizinhança é bastante confiável como um procedimento heurístico de propósito geral. Esse método é aplicado em trabalhos como os de Dauzère-Pérès & Paulli (1997), Mastrolilli & Gambardella (2000) e Pena et al. (2016).

A técnica de busca tabu em sua forma básica, pode ser vista como uma forma modificada de pesquisa de vizinhança. Cada vez que uma vizinhança é gerada e uma nova semente é

selecionada, chamamos a mudança de uma semente para a próxima de movimento. Um movimento é definido pelo mecanismo que gera vizinhanças e pela regra de seleção de uma solução na vizinhança. Na pesquisa tabu, o costume é selecionar o melhor valor da função objetivo na vizinhança (Baker & Trietsch, 2013). Esse método é aplicado em trabalhos como os de Dell'Amico & Trubian (1993), Nowicki & Smutnicki (1996) e Bożejko et al. (2017).

Quanto às técnicas avançadas, existem diversas abordagens (Arisha et al., 2001). Essas técnicas são mais recentes e buscam soluções satisfatórias e não soluções ótimas. Como exemplos, tem-se métodos baseados em redes neurais (Weckman et al., 2008), (Ramanan et al., 2011), lógica *fuzzy* (Canbolat & Gundogar, 2004), algoritmos genéticos (Ruiz et al., 2006), (Della Croce et al., 1995), algoritmos evolucionários (Onwubolu & Davendra, 2006), (Rafael, 2018), Pena et al. (2016), etc.

2.2 Teoria de Controle Supervisório

Nesta seção será feita uma breve introdução à Teoria de Controle Supervisório - TCS - proposta por Ramadge & Wonham (1989).

A Teoria de Controle Supervisório constitui-se como uma importante ferramenta para o controle de Sistemas a Eventos Discretos. Baseado em um modelo que descreve o comportamento livre do sistema a ser controlado (planta) e num conjunto de especificações comportamentais é possível realizar a síntese formal de um supervisor. A ação de controle do supervisor restringe, de forma minimamente restritiva, o comportamento de sistema (planta e especificações) de forma a satisfazer o conjunto de especificações (Vieira et al., 2007).

O conjunto de especificações num sistema consideram aspectos como: segurança e vivacidade do sistema, prioridade de eventos e justiça na alocação de produtos. Conforme Queiroz et al. (2004), uma especificação de segurança visa garantir que "nada de ruim aconteça" ao sistema, ao passo que uma especificação de vivacidade visa garantir que "algo de bom aconteça".

A Teoria de Controle Supervisório propõe que uma estrutura denominada supervisor observe a sequência de eventos gerada espontaneamente no sistema a ser controlado e que, instantaneamente após a observação de um novo evento, determine o subconjunto de eventos que concatenados a esta sequência preservam o comportamento do sistema dentro do desejado pelas especificações (Vieira et al., 2007).

Na Figura 2.3 é apresentado graficamente a arquitetura de controle supervisório conforme definido em (Ramadge & Wonham, 1989).

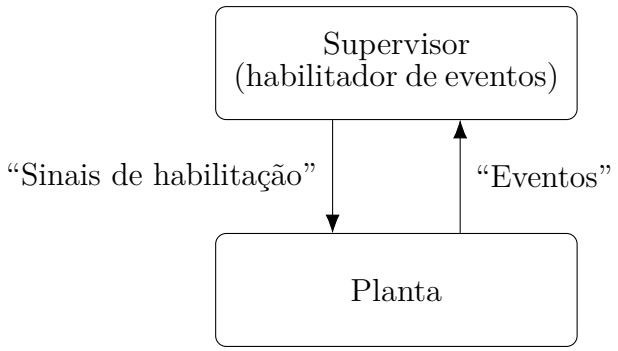


Figura 2.3: Supervisão de um Sistema a Eventos Discretos.

A síntese de um único supervisor para toda a planta é chamada de Controle Monolítico. A dificuldade mais fundamental desta abordagem, como mostrado por Gohari & Wonham (2000), é que a síntese do supervisor Monolítico é NP-difícil, na medida em que o tamanho do espaço de estado cresce exponencialmente com o número de componentes individuais da planta e especificações de controle, logo se torna inviável a obtenção do supervisor por essa abordagem para sistemas de médio e grande porte (Wonham & Ramadge, 1988), (Wong, 1998), (Komenda & van Schuppen, 2005).

Para tratar essa questão, existem extensões da TCS como o Controle Modular, Controle Modular Local e o Controle Hierárquico que visam contornar esse problema. Recentemente o problema da explosão de estados foi também mitigado por um procedimento de síntese que emprega uma decomposição hierárquica para representar o espaço de estados e diagramas de decisão binária (BDD). Assim, o estado nas funções de transição e controle é substituído pela lista de nós do BDD, cujo tamanho é, em casos favoráveis, aproximadamente logarítmico no tamanho de estados (Cai & Wonham, 2013).

O Controle Modular, proposto por Wonham & Ramadge (1988) é uma extensão do Controle Monolítico que tem como objetivo contornar o problema da explosão de estados,

com a síntese de vários supervisores de tamanhos menores. O Controle Modular considera a planta como um todo e para cada especificação desejada obtém-se um supervisor. O problema é decomposto horizontalmente em vários subproblemas (Thistle, 1996).

No entanto, ao decompor o problema em subproblemas, surge o conceito de conflito, que ocorre quando um supervisor interfere na ação de outro, o que pode levar a planta para uma situação não desejada de bloqueio (Flordal & Malik, 2006). Verificar o não-conflito, é computacionalmente caro, em geral. O problema da ocorrência de conflitos é abordado no literatura de diferentes maneiras. Alguns autores desenvolveram abordagens onde os supervisores são não conflitantes por construção (Chen & Lafortune, 1991), (Hill et al., 2008). Outros resolveram o problema do bloqueio usando coordenadores (Wong & Wonham, 1998), (Feng & Wonham, 2006). Malik (2004) propõe um método para detecção de conflitos baseado numa caracterização do conflito e Malik et al. (2004) estudou conflitos de um ponto de vista algébrico.

O Controle Modular Local proposto por Queiroz & Cury (2000), visa contornar o problema da explosão de estados, com a síntese de vários supervisores ainda menores. Essa abordagem, considera como planta apenas os subsistemas que são diretamente afetados pelas especificações.

A complexidade computacional para síntese do conjunto de supervisores locais é significativamente reduzida em relação à complexidade computacional para síntese do Supervisor Monolítico. A abordagem do Controle Modular Local é um exemplo de decomposição horizontal (Thistle, 1996). Assim como no Controle Modular, a atuação de vários supervisores sobre o mesmo sistema pode gerar situações de bloqueio. Portanto, é preciso que os supervisores obtidos por esta abordagem sejam não-conflitantes. Pena et al. (2009) propõem um método eficiente para a verificação do conflito de supervisores Modulares Locais.

O Controle Hierárquico (Wong & Wonham, 1996) é uma forma de decomposição vertical. O Controle Hierárquico é uma estrutura de dois níveis, no qual o nível superior é um modelo do nível inferior. O nível superior é construído por meio de uma abstração de nível inferior. A comunicação entre o sistema e a abstração é feita por um canal de informação. O conceito de consistência hierárquica é introduzido para garantir que as

ações impostas pelo controlador do nível superior sejam realizáveis pelo nível inferior. A garantia de que o não-bloqueio seja preservado entre o nível superior e o nível inferior é resumida no conceito de consistência hierárquica. Isto é garantido se o canal de informação possui o conceito de "observador", conceito introduzido por Wong & Wonham (1996) e utilizado neste trabalho.

A abordagem de Controle Hierárquico se dá na forma de abstrações. Cunha & Cury (2007) propõem uma supervisão hierárquica de dois níveis, esquema de controle construído em uma abordagem de baixo para cima. O modelo de baixo nível é assumido como tendo a estrutura padrão de controle (Ramadge & Wonham, 1989), e o modelo de alto nível é assumido como um DES com Marcação Flexível (siglas em inglês DESFM) (Cury et al., 2004). Fazendo o DES de alto nível um DESFM, é possível obter uma representação compacta dos possíveis comportamentos marcados em malha fechada de alto nível como abstração dos possíveis comportamentos marcados em malha fechada de baixo nível. Portanto, nessa abordagem, é possível desconsiderar condições estruturais habituais relacionadas com a consistência do comportamento marcado, como a consistência de marcação. A vantagem desta abordagem é a redução da complexidade para modelo DES de alto nível, em termos de número de eventos e estados.

Abstrações são utilizadas em diversos trabalhos. Pena et al. (2009) propõem um método eficiente para a verificação de não-conflito na abordagem modular local. A abordagem consistiu em realizar projeções naturais dos supervisores modulares locais mantendo a propriedade do observador e apresentando condições sobre o alfabeto que é mantido na projeção, denominado alfabeto de eventos relevantes.

O uso de abstrações no Controle Supervisório, visa reduzir a complexidade da síntese de supervisores em grande escala, ao mesmo tempo que exploram a estrutura da planta (Schmidt & Cury, 2012). Para este fim, é desejado calcular uma abstração menor da planta para sintetizar um supervisor não bloqueante e, se possível, maximamente permissivo. Na literatura, os métodos em (Feng & Wonham, 2008), (Hill et al., 2010), (Leduc et al., 2005), (Schmidt & Breindl, 2010) e (Schmidt et al., 2008) calculam abstrações usando projeções naturais.

2.3 Aplicações da TCS em Problemas de Planejamento

Atualmente, diversos trabalhos utilizam sistemas a eventos discretos e a TCS para a solução de problemas de escalonamento e que posteriormente, aplicam alguma técnica já conhecida para obter uma solução.

Abdeddaim & Maler (2001) modelam o problema clássico de planejamento de *job-shop* através de um autômato temporizado. A solução do problema de planejamento foi proposta no contexto da metodologia de verificação em duas direções ortogonais: da verificação à síntese. A verificação consistiu em verificar a existência de certos caminhos de um autômato, enquanto na síntese o autômato foi restrito para um ou mais caminhos ótimos. Em outras palavras, a solução foi obtida pela busca do menor caminho com relação ao tempo. Abdeddaim & Maler (2001) apresentaram alguns algoritmos de otimização e heurísticas. O trabalho não apresentou ganhos significativos em relação ao tempo de computação na busca da solução, se comparado com outros abordagens. A principal contribuição foi a utilização de autômatos temporizados à modelagem do problema de planejamento. A ideia de aplicar a síntese aos autômatos temporizados foi explorada pela primeira vez em (Wong-Toi & Hoffmann, 1991).

Panek et al. (2004) também abordam o problema de escalonamento usando autômatos temporizados. A solução foi obtida aplicando técnicas de programação linear inteira e propondo um algoritmo que trabalha nos limites inferiores na árvore de alcançabilidade do autômato temporizado. Os autores concluíram que a especificação de um determinado problema de j*ob-shop* usando ferramentas de programação inteira mista pode eficientemente resolvê-lo, mas é um processo incômodo para a modelagem algébrica do problema e tarefa demorada.

Ainda sobre autômatos temporizados, Nishi & Wakatake (2014) propõem um algoritmo de decomposição para o problema de escalonamento. O modelo proposto compreende a composição paralela de submodelos. O procedimento da metodologia proposta foi dividido em duas etapas. O primeiro passo foi decompor o modelo do autômato temporizado em vários submodelos usando condições decomponíveis. O segundo passo foi combinar

a solução individual de subproblemas para os submodelos decompostos pelo método da função de penalidade. Outro método proposto na literatura para solucionar o problema de planejamento foi o *sleep set* combinado com a técnica de redução do espaço de estados para reduzir o espaço de busca, eliminando execuções redundantes de busca (Panek et al., 2008), (Subbiah et al., 2009).

A Teoria de Controle Supervisório juntamente com programação linear inteira mista é usada por Kobetski & Fabian (2006) para obter as sequências ótimas para a operação de robôs. O trabalho propõe um método de conversão automática entre autômatos de estados finitos e programação linear inteira mista. A característica mais significativa da técnica é a possibilidade de explorar a Teoria de Controle Supervisório juntamente com um método de otimização bem estudado e difundido, facilitando a geração de planejamentos ótimos, garantindo ao mesmo tempo que o sistema se comporte conforme especificado.

Atualmente, os sistemas têm uma tendência a serem maiores e cada vez mais complexos. Fabre & Jezequel (2009) trataram esse problema modelando uma grande rede de autômatos como um sistema distribuído. O planejamento nesse sistema consiste em selecionar e organizar ações para atingir um estado de meta de maneira ideal, assumindo que as ações têm um custo. Para lidar com a complexidade do sistema, propuseram uma abordagem de planejamento distribuída (modular). A solução para cada problema de planejamento relacionado a cada "agente" é obtida fazendo o uso de algoritmos clássicos de alcançabilidade.

Su (2012) propôs um método de abstração autônomo determinístico que permite que a abstração do modelo seja realizada de forma distribuída e, portanto, com potencial de reduzir significativamente o tamanho de uma planta, de modo que a síntese subsequente do supervisor de tempo ótimo possa ser feita de uma maneira computacionalmente eficiente. Depois de mostrar como derivar um supervisor ótimo de tempo com base em um modelo abstraído, explica como usar um supervisor para roteamento e planejamento de tarefas.

Oliveira et al. (2013) propuseram uma abordagem para o problema de planejamento de tarefas baseada no uso de um algoritmo de seleção clonal combinada com a Teoria de Controle Supervisório. Duas metodologias foram propostas. A primeira, o Algoritmo de Seleção Clonal (CSA pelas suas siglas em inglês) realiza a busca da solução ideal, utilizando

buscas aleatórias sobre permutações de sequências de operações. A segunda é semelhante, mas o CSA usa uma pesquisa local para melhorar o melhor indivíduo de cada geração. Os resultados obtidos dependem muito dos parâmetros do algoritmo de seleção clonal. Portanto, testes mais exaustivos e sistemáticos, com uma ampla gama de problemas e com diferentes parâmetros do algoritmo devem ser realizados para estabelecer resultados mais conclusivos. A metodologia proposta é limitada pelo fato de que só pode ser aplicada para lidar com pequenos lotes de produtos.

Pena et al. (2016) apresenta uma nova abordagem para o sequenciamento ideal de tarefas em sistemas de manufatura, com base na combinação de algoritmos meta-heurísticos que tentam otimizar a composição da solução, usando a Teoria do Controle Supervisório de Sistemas de Eventos Discretos para modelar as restrições. Essa abordagem foi denominada Controle Supervisório e Otimização (SCO pelas suas siglas em inglês). Em relação à pesquisa meta-heurística, a codificação de embaralhamento de palavras induz um espaço de variável de decisão no qual pesquisas eficientes podem ser executadas. Com essa codificação, uma versão do *variable neighborhood search* (VNS) foi desenvolvida. Embora tenha-se mostrado uma alternativa para lidar com essa classe de problemas, a metodologia da SCO é limitada pelo fato de que só pode ser aplicada para lidar com pequenos lotes de produtos.

Costa et al. (2018) propõem uma abordagem complementar à SCO (Supervisory Control and Optimization), proposta em Pena et al. (2016), denominada SCO-Concat, desenvolvida para realizar o planejamento em lotes maiores de produção. A metodologia proposta foi testada em um sistema flexível de manufatura, retirado da literatura, e os resultados obtidos mostram que é possível executar o agendamento de lotes de produção tão grandes quanto necessário, a um custo computacional bastante viável. De acordo com esses testes, o SCO-Concat provou ser eficiente para gerar planos de produção para grandes lotes.

Malik & Pena (2018) propuseram uma abordagem que demonstra o uso da verificação de modelo para resolver o problema do agendamento ideal de tarefas em um sistema de fabricação flexível usando a TCS. O estudo de caso considerado no artigo foi o Sistema de Manufatura Flexível apresentado em de Queiroz et al. (2005). O método proposto produziu com sucesso agendamentos ótimos para fabricar até 30 produtos de dois tipos diferentes. Além disso, o método é usado para encontrar uma ciclo ideal resolvendo o

problema de agendamento do caso de estudo para um número arbitrário de produtos em um tempo ideal ou assintoticamente próximo do tempo ideal.

Alves et al. (2016) aborda o problema de planejamento em sistemas de manufatura, propondo um método para encontrar uma sequência de eventos em um supervisor que maximiza o paralelismo entre equipamentos. O supervisor é obtido usando a Teoria do Controle de Supervisório que implementa a máxima sub-linguagem controlável contida no comportamento desejado do sistema. O objetivo principal é encontrar, dentre todas as execuções permitidas pelo supervisor, aquele que acumula mais equipamentos trabalhando ao mesmo tempo durante todo o processo de produção. A sequência obtida pelo máximo paralelismo não necessariamente é obtida em relação ao *makespan*, mas apresenta boas soluções em tempo.

Considerando o modelo conceitual da Figura 2.2, o controlador só pode atuar sobre os eventos controláveis da TCS. Assim, considera-se válido o argumento de que o plano (sequência) ótimo que o planejador passa ao supervisor seja composto apenas de eventos controláveis. Fundamentando-se nisso, Vilela & Pena (2016) estabelecem condições suficientes que garantem que a busca pelas sequências de solução do problema de planejamento pode ser realizada sobre uma abstração no conjunto de eventos controláveis do comportamento em malha fechada. As condições garantem que qualquer sequência escolhida na versão abstraída poderá ser executada no sistema físico sem nenhuma violação (controlabilidade). A condição suficiente para que qualquer sequência obtida na abstração para o conjunto de eventos controláveis do supervisor para ser usada como plano é que a abstração tenha a propriedade do observador. O resultado principal de Vilela & Pena (2016) por si só não é a solução do problema de planejamento, pois não oferece informação que permita quantificar o quão boa é a solução.

Uma extensão ao trabalho de Vilela & Pena (2016) é proposta por Alves & Pena (2017), que propõem o uso de abstrações de supervisores obtidos pela síntese modular local no lugar da abordagem monolítica. Essa extensão torna viável a aplicação do resultado anterior para os casos em que a verificação da propriedade do observador sobre a projeção do supervisor monolítico ou a própria projeção não possam ser computadas. Do mesmo modo que o trabalho de Vilela & Pena (2016), o resultado principal de Alves & Pena (2017) não é a solução final para o problema de planejamento, mas uma boa ferramenta

que reduz o espaço de busca das sequências.

Em Rafael (2018) foram propostos dois métodos para a aplicação do resultado teórico de Vilela & Pena (2016) que gera soluções para a otimização de problemas de escalonamento. O primeiro método proposto utilizou a TCS para criar uma solução que codifica o comportamento em malha fechada do sistema combinando-o com o Algoritmo de Seleção Clonal. Nesta técnica, não é produzida nenhuma sequência infactível, ou seja, todas as soluções encontradas podem ser executadas no sistema, tornando a abordagem muito eficiente. Rafael (2018) propôs um segundo método para gerar sequências chamado algoritmo do Evento Mais Frequente (MFE pelas suas siglas em inglês), que é um algoritmo heurístico que favorece os eventos que estão mais frequentemente disponíveis durante a operação do sistema. Apesar de não garantir resultados ótimos, ambos os métodos, em geral, foram capaz de encontrar as melhores soluções. O MFE foi o mais rápido em relação ao tempo de execução em todas as instâncias testadas, enquanto CSA + o algoritmo de geração individual aleatório apresentou os valores mais curtos do *makespan*.

3

Preliminares

Neste capítulo, alguns conceitos básicos de Sistemas a Eventos Discretos, Teoria de Linguagens e Autômatos, Teoria de Controle Supervisório serão apresentados.

3.1 Sistemas a Eventos Discretos

Sistemas a Eventos Discretos se referem a uma classe de sistemas dinâmicos que possuem um espaço de estados discretos e evoluem em resposta abrupta à ocorrência de eventos (Partovi & Lin, 2018). Estes eventos podem ser vistos como uma ação específica tomada, como por exemplo: pressionar um botão ou ligar uma máquina. Ou podem ser vistos como uma mudança interna do sistema, tal como o término de uma tarefa ou uma interrupção temporizada. Seja qual for a mudança (interna ou externa), caracterizam-se por serem abruptas e instantâneas (Cassandras & Lafortune, 2009).

Sistemas como automação da manufatura, redes de comunicação, robótica, tráfego de veículos, entre outros, podem ser vistos como Sistemas a Eventos Discretos. Existem diversas formas de modelar e analisar os Sistemas a Eventos Discretos. A abordagem na qual se baseia este trabalho é a de autômatos não temporizados. Nas próximas seções serão apresentadas algumas operações sobre linguagens e autômatos e a Teoria de Controle Supervisório, que serão úteis na compreensão do trabalho.

3.2 Teoria de Linguagens e Autômatos

Seja Σ um conjunto finito de símbolos (eventos) distintos. Σ é também chamado de alfabeto de um Sistema a Eventos Discretos (SED). Seja Σ^* o conjunto de todas as cadeias finitas da forma $\mu_1\mu_2, \ldots \mu_k$, com $k \geq 1$, geradas a partir de eventos do alfabeto Σ, incluindo a cadeia formada de zero eventos chamada de cadeia vazia e denotada por ϵ. O comprimento de uma cadeia é o número de eventos que ela contém, incluindo também as repetições do mesmo evento. Seja s uma cadeia, onde o seu comprimento é denotado por $|s|$. Por convenção, o comprimento da cadeia vazia ϵ é zero ($|\epsilon| = 0$).

Uma linguagem L é um conjunto de cadeias finitas formadas por elementos de Σ, $L \subseteq \Sigma^*$. Em particular $\emptyset, \Sigma$ e Σ^* são linguagens sobre Σ. Na próxima seção serão apresentadas as principais operações que podem ser aplicadas às linguagens.

3.3 Operações Sobre Linguagens

Sejam os eventos t e u pertencentes ao alfabeto Σ, a operação de concatenação de t e u denota-se por tu. O elemento neutro da concatenação é a cadeia vazia ϵ, ou seja, $\epsilon t = t\epsilon = t$. Se os eventos t e u formam a palavra s, pode-se dizer que t é prefixo de s e denota-se $t \leq s$ e u o sufixo de s.

A operação *prefixo-fechamento* de uma linguagem é denotada por $\overline{L}$ e consiste de todos

os prefixos de todas as cadeias em L. O fechamento da linguagem L está definido a seguir:

$$\overline{L} = \{s \in \Sigma^* \mid s \leq t \text{ para algum } t \in L\}.$$

Em geral $L \subseteq \overline{L}$. No entanto, se $L = \overline{L}$, então L é chamada de *prefixo-fechada*. O operador *fechamento de Kleene*, denotado por L^*, também pode ser aplicado sobre linguagens. Um elemento de L^* pode ser formado pela concatenação de elementos de L, incluindo também a cadeia vazia ϵ. L^* está definida como:

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

3.3.1 Autômatos

A forma mais simples de apresentar a noção de um autômato é considerar sua representação gráfica dirigida ou diagrama de transição de estados. Um autômato determinístico de estados finitos é uma quintúpla:

$$G = (Q, \Sigma, \delta, q_0, Q_m).$$

em que:

- Q é o conjunto finito de estados.
- Σ é o conjunto finito de eventos associados a G.
- $\delta : Q \times \Sigma \to Q$ é a função de transição: $\delta(x, \alpha) = y$ significa que existe uma transição rotulada pelo evento α do estado x ao estado y.
- q_0 é o estado inicial, onde $q_0 \in Q$.
- $Q_m \in Q$ é o conjunto de estados marcados.

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

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

Esta função mapeia para qual estado de Q o sistema transita a partir de um estado de Q com a ocorrência de uma sequência $s \in \Sigma^*$. A notação $\Gamma : Q \to 2^{\Sigma}$ representa a função do conjunto de eventos factíveis. Os nós em um autômato representados por círculos, são os estados, e os arcos, representados por setas, são as transições. Os nós representados com um duplo círculo, são os estados marcados e o estado inicial é indicado por uma seta que não possui rótulo. O rótulo de cada arco é o evento associado à transição. A Figura 3.1 mostra um exemplo de autômato determinístico. Um autômato G tem associadas dois

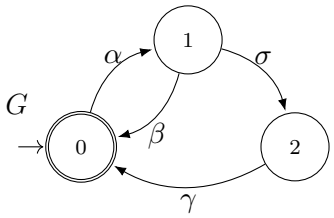


Figura 3.1: Autômato determinístico.

tipos de linguagens, a linguagem gerada

$$\mathcal{L}(G) = \{s \in \Sigma^* : \delta(q_0, s) \text{ está definida}\}$$

e a linguagem marcada

$$\mathcal{L}_m(G) = \{s \in \Sigma^* : \delta(q_0, s) \in Q_m\}.$$

A linguagem gerada $\mathcal{L}(G)$ possui todas as cadeias de eventos que partem do estado inicial até qualquer estado, enquanto a linguagem marcada $\mathcal{L}_m(G)$ representa todos os caminhos a partir do estado inicial até um dos estados marcados. Portanto, o comportamento de um SED pode ser representado por um autômato G, sendo $\mathcal{L}(G)$ a linguagem gerada pelo sistema e $\mathcal{L}_m(G)$ a linguagem marcada do sistema. A linguagem $\mathcal{L}_m(G)$ é subconjunto de $\mathcal{L}(G)$, ou seja $\mathcal{L}_m(G) \subseteq \mathcal{L}(G)$.

Se um estado $q \in Q$ de um autômato G não pode ser alcançado por nenhuma sequência a partir do estado inicial, então esse estado q é chamado de não acessível. É possível separar a componente acessível $Ac(G)$ de um autômato eliminando do autômato os estados que não sejam acessíveis, $G = Ac(G)$.

Se G possui um estado $q \in Q$ do qual não é possível alcançar nenhum estado marcado, então esse estado é dito ser não coacessível. A componente coacessível $CoAc(G)$ de um autômato G é obtida eliminando todos os estados não coacessíveis, $G = CoAc(G)$.

Um autômato G é dito ser não bloqueante se:

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

O bloqueio pode ocorrer de duas formas. A primeira é o *deadlock*, que ocorre quando um estado q não marcado não tem nenhum evento que pode ocorrer, então o autômato fica bloqueado para esse estado. A segunda forma de bloqueio acontece quando existe um conjunto de estados não marcados em G que formam um componente fortemente conectada, mas com nenhuma transição saindo do conjunto. Essa componente fortemente conectada é chamada de *livelock*.

Um autômato que é acessível e coacessível é dito ser *trim* A operação *Trim* é:

$$Trim(G) = CoAc[Ac(G)] = Ac[CoAc(G)].$$

onde a comutatividade de Ac e CoAc é facilmente verificada.

3.3.2 Projeção Natural

A projeção natural $P : \Sigma^* \to \Sigma_i^*$ é uma operação sobre linguagens definida sobre o conjunto de eventos Σ e Σ_i, onde $\Sigma_i \subseteq \Sigma$. A projeção natural mapeia cadeias de Σ^* para cadeias em Σ_i^* apagando todos os eventos que não estão em Σ_i. Esta operação está definida para

cadeias como:

$$P_{\Sigma\to\Sigma_i}(\varepsilon) := \varepsilon$$

$$P_{\Sigma\to\Sigma_i}(s\sigma) := \begin{cases} P_{\Sigma\to\Sigma_i}(s) \ se \ s \in \Sigma^*, \sigma \notin \Sigma_i \\ P_{\Sigma\to\Sigma_i}(s)\sigma \ se \ s \in \Sigma^*, \sigma \in \Sigma_i. \end{cases}$$

Como pode-se ver, a operação da projeção natural toma cadeias formadas com elementos do alfabeto Σ e apaga aqueles eventos que não fazem parte do alfabeto Σ_i. Wong (1998) mostrou que a complexidade de uma projeção natural é no pior caso exponencial. A operação inversa da projeção é a função $P^{-1}_{\Sigma\to\Sigma_i} : \Sigma_i^* \to 2^{\Sigma^*}$ e está definida como:

$$P^{-1}_{\Sigma\to\Sigma_i}(t) = \{s \in \Sigma^* : P_{\Sigma\to\Sigma_i}(s) = t\}.$$

Em palavras, a projeção inversa $P^{-1}_{\Sigma\to\Sigma_i}(t)$ de uma cadeia t retorna todas as cadeias de Σ^*, que são mapeadas para a cadeia projetada t. Essas definições de projeção natural e projeção inversa podem ser estendidas para linguagens aplicando-as a todas as cadeias da linguagem. Para $L \in \Sigma^*$:

$$P_{\Sigma\to\Sigma_i}(L) := \{t \in \Sigma_i^* : (\exists s \in L)[P_{\Sigma\to\Sigma_i}(s) = t]\}.$$

e para $L_i \subseteq \Sigma_i^*$,

$$P^{-1}_{\Sigma\to\Sigma_i}(L_i) := \{s \in \Sigma^* : (\exists t \in L_i)[P_{\Sigma\to\Sigma_i}(s) = t]\}.$$

Para mostrar a ideia de projeção natural, o Exemplo 1 é apresentado.

Exemplo 1. *Considere o alfabeto* $\Sigma = \{\alpha, \beta, \gamma, \sigma\}$ *e o subconjunto* $\Sigma_i = \{\alpha, \beta\}$. *Sejam as linguagens* $L_1 = \{\gamma, \alpha\beta, \alpha\gamma, \beta, \sigma\gamma\alpha\}$ *e* $L_2 = \{\alpha\alpha, \sigma\beta, \gamma\sigma\alpha\}$. *Considere a projeção* $P : \Sigma^* \to \Sigma_i^*$. *Então temos que:*

$$P(L_1) = \{\epsilon, \alpha\beta, \alpha, \beta, \alpha\};$$

$$P(L_2) = \{\alpha\alpha, \beta, \alpha\}.$$

Em Cassandras & Lafortune (2009) definem-se 5 propriedades da projeção natural que são usadas nesse trabalho. Considere que A,B e L são linguagens definidas no alfabeto Σ. Considere também a projeção $P : \Sigma^* \to \Sigma_i^*$, com $\Sigma_i \subseteq \Sigma$. A seguir essas propriedades são apresentadas:

1) $P(P^{-1}(L)) = L$
   $L \subseteq P^{-1}(P(L))$

2) Se $A \subseteq B$, então $P(A) \subseteq P(B)$ e $P^{-1}(A) \subseteq P^{-1}(B)$

3) $P(A \cup B) = P(A) \cup P(B)$
   $P(A \cap B) \subseteq P(A) \cap P(B)$

4) $P^{-1}(A \cup B) = P^{-1}(A) \cup P^{-1}(B)$
   $P^{-1}(A \cap B) = P^{-1}(A) \cap P^{-1}(B)$

5) $P(AB) = P(A)P(B)$
   $P^{-1}(AB) = P^{-1}(A)P^{-1}(B)$

O uso das projeções nesse trabalho têm um papel muito importante. Uma propriedade muito conhecida da projeção natural é a propriedade do observador, apresentada na Definição 1.

Definição 1 (Wong (1998)). *Seja uma linguagem $L \subseteq \Sigma^*$, um alfabeto $\Sigma_i \subseteq \Sigma$ e $P_{\Sigma \to \Sigma_i^*} : \Sigma^* \to \Sigma_i^*$. Se $(\forall a \in \overline{L})(\forall b \in \Sigma_i^*)$, $P_{\Sigma \to \Sigma_i}(a)b \in P_{\Sigma \to \Sigma_i}(L) \Longrightarrow (\exists c \in \Sigma^*)P_{\Sigma \to \Sigma_i}(ac) = P_{\Sigma \to \Sigma_i}(a)b$ e $ac \in L$, então a projeção natural $P_{\Sigma \to \Sigma_i}(L)$ tem a propriedade de observador.* □

Se a projeção natural tem a propriedade do observador, então, garante-se que o cálculo da mesma tenha complexidade polinomial no pior dos casos. Para aclarar a definição da Propriedade do Observador (PO), considere a Figura 3.2.

A elipse em azul (parte A), delimita uma linguagem L gerada a partir de eventos do alfabeto Σ, enquanto a elipse em salmão (parte B), representa a projeção natural dessa linguagem ($P(L)$) para o alfabeto Σ_i. Escolhe-se uma cadeia a que é prefixo de uma cadeia da linguagem L. Logo faz-se a projeção da cadeia a ($P_{\Sigma \to \Sigma_i}(a)$) para o alfabeto Σ_i. Em seguida completa-se a projeção de a, $P_{\Sigma \to \Sigma_i}(a)$, com um sufixo b, formado apenas

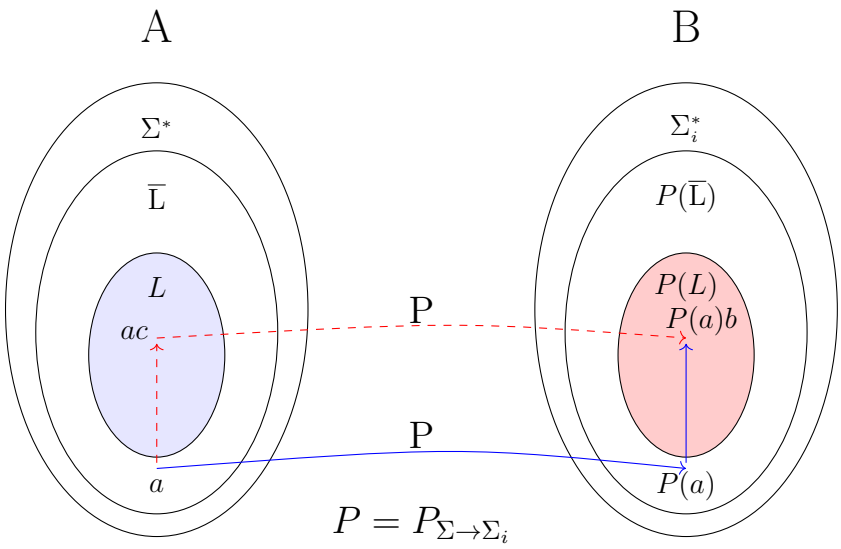


Figura 3.2: Representação da propriedade do observador.

de eventos de Σ_i , levando à obtenção de uma cadeia que está na projeção de L, ou seja $P_{\Sigma \to \Sigma_i}(a)b \in P_{\Sigma \to \Sigma_i}(L)$. Essa operações são indicadas pelas setas em azul.

Se a projeção possui a propriedade do observador, então, para todo a e b tal que $P_{\Sigma \to \Sigma_i}(a)b \in P_{\Sigma \to \Sigma_i}(L)$, existirá c tal que $P_{\Sigma \to \Sigma_i}(ac) = P_{\Sigma \to \Sigma_i}(a)b$, que é o caminho representado pelas setas em vermelho. Portanto, tem-se que $P_{\Sigma \to \Sigma_i}(c) = b$.

A linguagem $P(L)$ pode ser chamada de abstração. Uma abstração que possua a propriedade do observador é chamada de PO-abstração (Pena et al., 2008). A propriedade do observador pode ser verificada usando o algoritmo apresentado em (Pena et al., 2014).

3.3.3 Composição Paralela

A operação de composição paralela, também chamada composição síncrona, tem como resultado um autômato que modela o comportamento conjunto de dois ou mais autômatos. Em geral, a operação de composição paralela é a utilizada para a construção de modelos de sistemas compostos de sistemas individuais (subsistemas). A composição paralela dos autômatos $G_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_{m1})$ e $G_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, Q_{m2})$ é o autômato:

$$G_1 \,||\, G_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$$

onde

$$\delta((x_1,x_2),e) := \begin{cases} (\delta_1(x_1,e),\delta_2(x_2,e)) & \text{se } \delta_1(x_1,e)\neq\emptyset \wedge \delta_2(x_2,e))\neq\emptyset \\ (\delta_1(x_1,e),x_2) & \text{se } \delta_1(x_1,e)\neq\emptyset \wedge e\notin\Sigma_2 \\ ((x_1,\delta_2(x_2,e)) & \text{se } \delta_2(x_2,e)\neq\emptyset \wedge e\notin\Sigma_1 \\ \text{indefinido} & \text{caso contrário} \end{cases}$$

Na composição paralela os eventos em comum ou seja, que pertençam a $\Sigma_1 \cap \Sigma_2$ podem ser executados se os autômatos podem executá-los simultaneamente ou seja, os autômatos são sincronizados nos eventos em comum. Os eventos privados, pertencentes a $(\Sigma_1 \backslash \Sigma_2) \cup (\Sigma_2 \backslash \Sigma_1)$, não sofrem nenhuma restrição e podem ser executados sempre que puderem em seus autômatos . Se não existe evento em comum entre os dois alfabetos, ou seja, $\Sigma_1 \cap \Sigma_2 = \emptyset$, então não há transições sincronizadas e qualquer comportamento possível em G_1 e G_2 também é possível em $G_1||G_2$.

A noção de projeção inversa descrita na seção 3.3.2 pode ser usada para prover a definição formal da operação de composição paralela, denotada por ||, de linguagens $L_i \subseteq \Sigma_i^*$, com $i \in I$, e $\Sigma = \bigcup_{i\in I} \Sigma_i$:

$$\|_{i\in I} L_i = \bigcap_{i\in I} P^{-1}_{\Sigma\to\Sigma_i}(L_i).$$

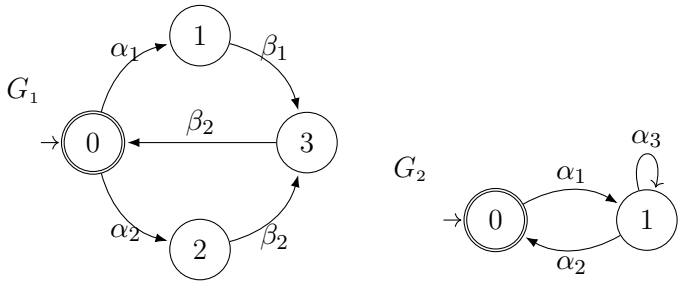
Do mesmo modo, a noção de projeção inversa pode ser usada para representar as linguagens marcada e gerada de uma composição paralela como:

$$\mathcal{L}(G_1||G_2) = P^{-1}_{\Sigma\to\Sigma_1}[\mathcal{L}(G_1)] \cap P^{-1}_{\Sigma\to\Sigma_2}[\mathcal{L}(G_2)]$$
$$\mathcal{L}_m(G_1||G_2) = P^{-1}_{\Sigma\to\Sigma_1}[\mathcal{L}_m(G_1)] \cap P^{-1}_{\Sigma\to\Sigma_2}[\mathcal{L}_m(G_2)]$$

O Exemplo 2 ilustra a composição.

Exemplo 2. *Sejam os subsistemas G_1 e G_2 apresentados na Figura 3.3 (a) e (b) correspondentemente. O autômato G da Figura 3.4 ilustra a composição paralela entre os dois subsistemas.*

Os eventos α_1 e α_2 evoluem sincronicamente, porém, os eventos β_1, β_2 e α_3 evoluem assincronicamente. Cada estado do autômato G (Figura 3.3 (c)) apresenta a informação



(a) Subsistema G_1. (b) Subsistema G_2.

Figura 3.3: Subsistemas G_1 e G_2.

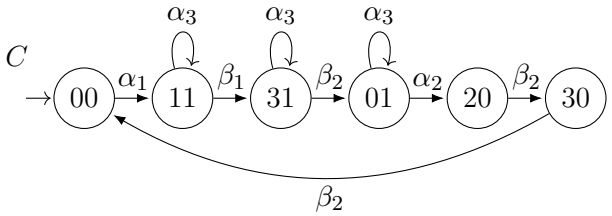


Figura 3.4: Composição paralela.

atual (rótulos) dos estados de cada autômato G_1 e G_2 correspondentemente.

3.4 Verificação da Propriedade do Observador

Pena et al. (2014), propõem um algoritmo, com complexidade quadrática, para a verificação da propriedade do observador para uma projeção natural arbitrária $P_{\Sigma\to\Sigma_r}$ sem a necessidade de se calcular a projeção natural propriamente dita. Esse algoritmo é uma extensão do algoritmo proposto por (Pena et al., 2008) onde a desvantagem desse método era a restrição imposta sobre G, que exigia a não existência de ciclos de eventos não relevantes (que sumiram na projeção).

O algoritmo Pena et al. (2014) constrói um autômato não determinístico $\overline{V}$, denominado verificador, a partir de um autômato também não determinístico $\overline{M}$ e um alfabeto de interesse Σ_r. Esse autômato $\overline{M}$ agrupa os ciclos de eventos não relevantes como macroestados. O autômato $\overline{M}$ é obtido de um autômato determinístico M, chamado autômato auxiliar, gerado do modelo do sistema G. Caso o estado DEAD não seja alcançado no autômato $\overline{V}$ então a abstração $P_{\Sigma\to\Sigma_r}(G)$ verifica a propriedade do observador. Este autômato auxiliar $M = (Q^M, \Sigma, \delta^M, q_0^M)$ é obtido a partir do autômato $G = (Q^G, \Sigma^G, \delta^G, q_0^G, Q_m^G)$ que modela o sistema, substituindo os estados marcados de G por estados não marcados com autolaços

com um evento artificial $\tau \in \Sigma_r$ que fará parte do conjunto de eventos relevantes. A estrutura do autômato verificador $\overline{V} = (\overline{Q}, \overline{\Sigma}, \overline{\delta}, \{A_0^{\overline{M}}\})$, permite classificar, sem restrição nenhuma, se uma projeção tem ou não a propriedade do observador.

No Exemplo 3 é realizada a verificação da propriedade do observador a partir da construção do verificador $\overline{V}$.

Exemplo 3. *Para o autômato G apresentado na Figura 3.5 deseja-se saber se para a projeção natural $P_{\Sigma\to\Sigma_r}$, com $\Sigma_r = \{\alpha\}$, $P(G)$, possui a propriedade do observador.*

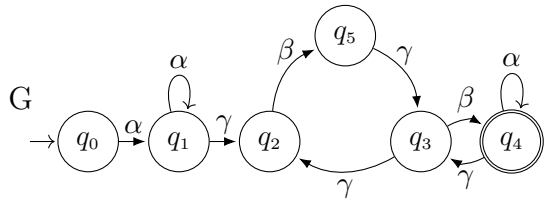
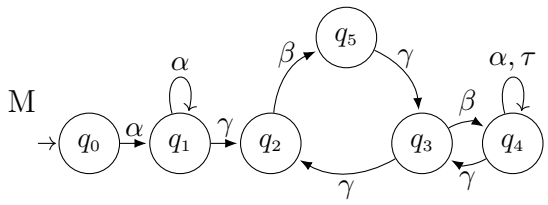
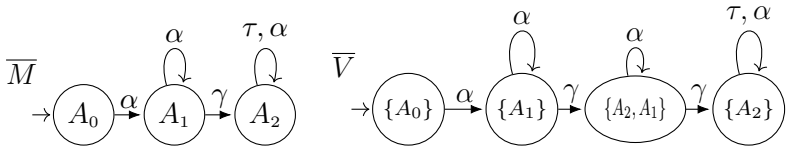


Figura 3.5: Exemplo 3: Autômato para a planta G.



(a) Autômato auxiliar M.



(b) Autômato auxiliar $\overline{M}$. (c) Verificador $\overline{V}$.

Figura 3.6: Exemplo 3: Verificação da propriedade do observador.

O primeiro passo consiste em obter o autômato auxiliar M, que possui a mesma estrutura de G mas os estados marcados são expressos por um autôlaço com o evento $\tau \in \Sigma_r$. Observe-se que o estado DEAD não é alcançado no autômato verificador $\overline{V}$, portanto $P_{\Sigma\to\Sigma_r}(G)$ possui a propriedade do observador.

Se o autômato verificador $\overline{V}$ alcança o estado DEAD, então diz-se que a projeção do sistema no conjunto de eventos relevantes $P_{\Sigma\to\Sigma_r}(G)$ não possui a propriedade do observador. Portanto se não possuir a PO, existem instâncias de eventos que, ao serem mostrados na

projeção, permitem construir uma PO-abstração. A seguinte seção mostra o processo da busca dessas instâncias de eventos.

3.5 Busca da Propriedade do Observador

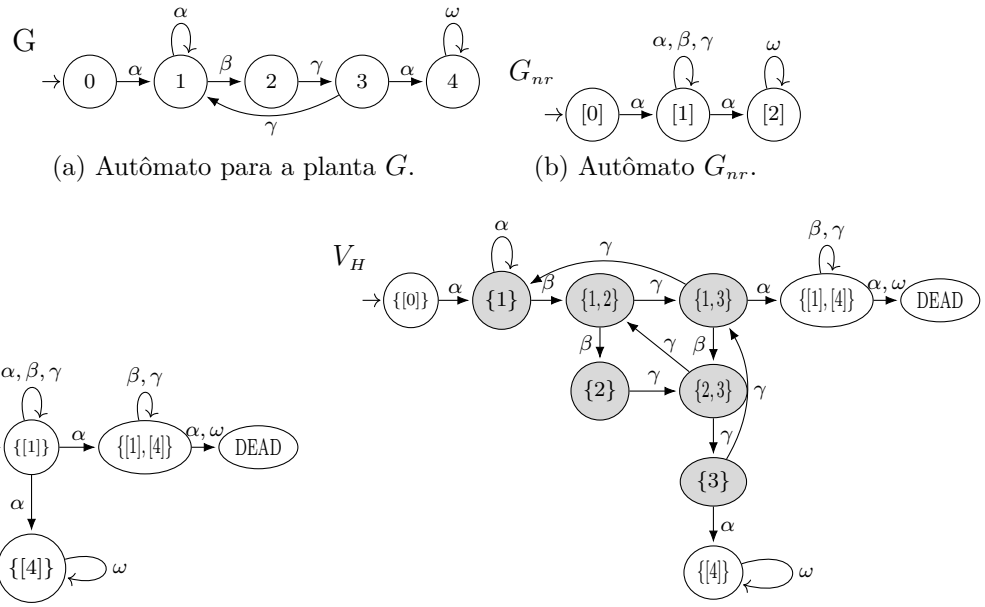
O problema da busca por PO abstrações consiste no renomeamento de eventos de um dado sistema e a sua projeção natural que não satisfazem a PO. O renomeamento pode ser realizado selecionando e transformando um evento não relevante em relevante.

Bravo et al. (2014) propõem um algoritmo com complexidade no pior caso de $O(|Q|^3(|\Sigma|+1)^2)$ (onde Q é o conjunto de estados do autômato e $|\Sigma|$ é a cardinalidade do alfabeto), para a busca da propriedade do observador, onde essa complexidade é a mesma que o algoritmo proposto por Pena et al. (2010) (complexidade polinomial). O algoritmo proposto é uma combinação dos resultados de (Pena et al., 2010) e (Bravo et al., 2012). O método proposto funciona construindo um verificador híbrido e expandindo alguns dos componentes de (Pena et al., 2010), a fim de identificar transições que causam a violação da propriedade do observador. O algoritmo pode ser aplicado a autômatos que apresentam ciclos de eventos não relevantes, limitação do algoritmo de Pena et al. (2010). A continuação apresenta-se o Exemplo 4 para mostrar esta ideia.

Exemplo 4. *Para a planta G apresentada na Figura 3.7(a), deseja-se saber se para a projeção natural $P : \Sigma^* \to \Sigma_r^*$, com $\Sigma_r = \{\alpha, \omega\}$, $P(G)$, possui a propriedade do observador, se não tiver, então buscar a propriedade.*

O algoritmo proposto por Bravo et al. (2014) constrói o autômato G_{nr} apresentado na Figura 3.7(b) a partir do autômato da planta G na Figura 3.7(a), onde G_{nr} não apresenta ciclos de eventos não relevantes (exceto os auto-laços). Os estados $\{1, 2, 3\}$ da Figura 3.7(a) formam um macroestado de eventos não relevantes originando o estado $[1]$ da Figura 3.7(b). O autômato não determinístico V_G da Figura 3.7(c), denominado verificador, é construído a partir do autômato G_{nr}. Como o estado DEAD é alcançado em V_G na Figura 3.7(c) então a abstração $P_{\Sigma\to\Sigma_r}(G)$ não é PO-abstração.

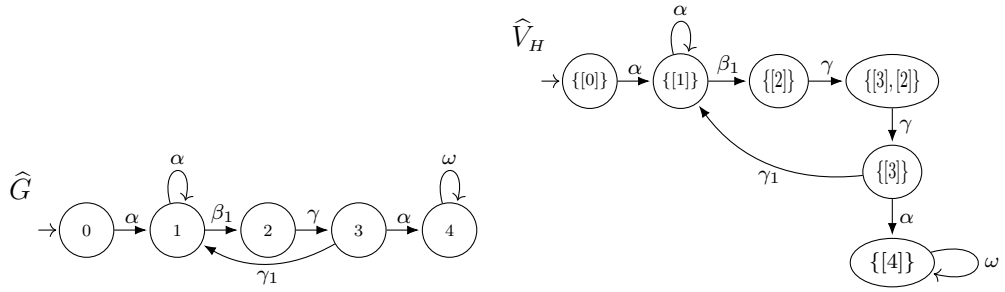
Identifica-se uma cadeia de eventos que leva ao estado DEAD para renomear alguns do seus eventos não relevantes, porém todos os eventos são relevantes (controlá-



(a) Autômato para a planta G. (b) Autômato G_{nr}.

(c) Autômato verificador V_G. (d) Verificador hibrido V_H.

Figura 3.7: Exemplo 4: Verificação e Busca da propriedade do observador.



(a) Autômato $\widehat{G}$ com os eventos renomeados. (b) Autômato $\widehat{V}_H$.

o.

Figura 3.8: Exemplo 4: Verificação e Busca da propriedade do observador.

veis). Portanto, para solucionar esse problema, faz-se a expansão de alguns dos macro estados singleton $(\{1\}, \{2\}$ *ou* $\{3\})$ *que estão no mesmo caminho que leva ao estado DEAD do verificador* V_G*, usando os estados originais (estados da planta) formando uma componente fortemente conexa. Essa expansão origina o autômato verificador hibrido* V_H *da Figura 3.8(d). A expansão do macro estado selecionado* $\{[1]\}$ *será o conjunto* $\{[1]\} = \{\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\}$*, onde cada um deles é um estado (em cinza) que leva ao estado DEAD.*

O próximo passo é escolher um dos estados simples $(\{\{1\}, \{2\}, \{3\}\})$ *do conjunto de estados não seguros e renomear o seu evento não relevante por relevante. Seleciona-se o estado* 3 *do autômato G na Figura 3.7(a) e a transição* γ *criando o novo evento relevante*

γ_1, originado $\widehat{\Sigma}_r = \{\alpha, \omega, \gamma_1\}$. Substitui-se a transição $\delta(3, \gamma) = 1$ por $\widehat{\delta}(3, \gamma_1) = 1$ em G. Seguindo a ideia do algoritmo, constrói-se novamente o verificador para testar se o estado DEAD é acessível. Portanto o estado DEAD ainda é acessível. Na segunda iteração, seleciona-se o estado 1 e o evento β e substitui-se por β_1, criando a transição $\widehat{\delta}(1, \beta_1) = 2$. O resultado dessas duas iterações origina o autômato $\widehat{G}$, apresentado na Figura 3.8(a).

Com o novo conjunto de eventos relevantes, $\widehat{\Sigma} = \{\alpha, \omega, \gamma_1, \beta_1\}$, tem-se o verificador $\widehat{V}_H$ mostrado na Figura 3.8(b). Neste caso, o estado DEAD não é mais acessível em $\widehat{V}_H$, portanto a abstração do autômato $P(\widehat{G})$ possui a propriedade do observador.

Neste trabalho o conjunto de instâncias de eventos não-controláveis que não podem ser apagados na projeção natural, posto que causam a violação da propriedade do observador, será chamado de Σ_{Ru}. Neste exemplo, o conjunto Σ_{Ru} é $\Sigma_{Ru} = \{\gamma_1, \beta_1\}$.

3.6 Teoria de Controle Supervisório

A Teoria de Controle Supervisório (TCS) (Ramadge & Wonham, 1989) propõe a síntese de controladores para *Sistemas a Eventos Discretos* (SED), matematicamente fundamentada na teoria de linguagens e autômatos (Cassandras & Lafortune, 2009). No controle

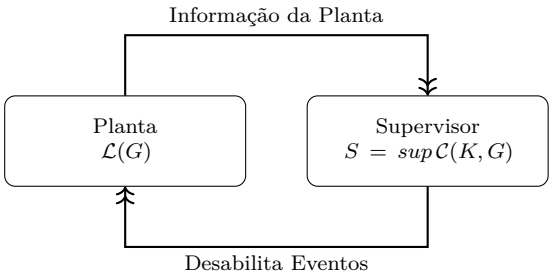


Figura 3.9: Estrutura da Teoria de Controle Supervisório.

Supervisório, considera-se a existência de uma planta G (sistema físico) que modela o comportamento em malha aberta de um SED. No entanto, para limitar o comportamento do sistema G e evitar que alcance certos estados que levariam a um bloqueio ou uma situação insegura, exerce-se alguma ação de controle sobre o sistema, Figura 3.9. Essas restrições, ou especificações, são modeladas como m autômatos E_i, com $i \in I = \{1, \ldots, m\}$, e a especificação global é obtida pelo autômato $E = \|_{i \in I} E_i$. A ação sobre a planta é feita

por meio de uma estrutura (supervisor) que garante o funcionamento do sistema sem que as restrições sejam violadas. Essa estrutura de controle é obtida pela proibição da ocorrência de determinados eventos em certos estados.

O comportamento do sistema G está modelado por um subconjunto de eventos controláveis e um subconjunto de eventos não controláveis. Assim, $\Sigma = \Sigma_c \cup \Sigma_u$, onde Σ_c e Σ_u são, respectivamente, o conjunto de eventos controláveis e não controláveis. Qualquer evento $\sigma \in \Sigma_c$ pode ser desabilitado, enquanto nenhum $\sigma \in \Sigma_u$ pode ser desabilitado pela estrutura de controle.

O comportamento desejado da planta em malha fechada é obtido por meio da operação de composição síncrona entre o autômato $G = (Q, \Sigma, \delta, q_0, Q_m)$ que modela a planta e o autômato E que modela a especificação do sistema, $K = G||E$. K é dito ser controlável com relação a G se e somente se:

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}.$$

Essa propriedade de controlabilidade estabelece que para K ser controlável em relação à $\mathcal{L}(G)$, todo $s \in \overline{K}$ concatenado com um $\sigma \in \Sigma_u$, de modo que $s\sigma \in \mathcal{L}(G)$, deve gerar uma cadeia $s\sigma$ em $\overline{K}$. Quando a especificação E não for verificada, então K é não controlável com relação à $\mathcal{L}(G)$. Se K for não controlável, então a máxima sublinguagem controlável e não bloqueante $S = Sup\mathcal{C}(K, G)$ pode ser sintetizada e implementada por um supervisor (Ramadge & Wonham, 1989). S é dito ser não bloqueante se $\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)}$. A expressão S/G é lida como G sob o controle do supervisor S e representa o sistema em malha fechada.

Por construção, o sistema controlado S/G tem quatro principais propriedades: (a) controlabilidade (S não pode desabilitar nenhum evento não controlável), (b) segurança (S/G gera apenas cadeias legais, ou seja $\mathcal{L}(S/G) \subseteq \mathcal{L}(K)$), (c) não bloqueio ($S/G$ é não bloqueante, ou seja $\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)}$), e (d) máxima permissividade (S/G não pode ser maior sem violar uma das três propriedades anteriores) (Lafortune, 2019).

4

Resultados Anteriores

Neste capítulo apresenta-se o resultado de outro trabalho que serviu como base para a elaboração do resultado principal desse trabalho.

4.1 Abstração de Supervisores

Em Vilela & Pena (2016), as condições sobre as quais a projeção natural do comportamento em malha fechada nos eventos controláveis, $P_{\Sigma \to \Sigma_c}(S)$, mantém as propriedades originais do supervisor são apresentadas. Esse resultado teórico permite encontrar soluções ótimas em um universo de busca muito menor, $P_{\Sigma \to \Sigma_c}(S)$ ao invés de S. A condição suficiente para que este resultado seja aplicável é que a projeção natural do comportamento em malha fechada para o conjunto dos eventos controláveis tem que possuir a propriedade do observador (Definição 1). Esse resultado é apresentado no Teorema 1.

Teorema 1 (Vilela & Pena (2016)). *Seja G uma planta, S a máxima sublinguagem controlável contida na linguagem desejada K e $P_{\Sigma\to\Sigma_c} : \Sigma^* \to \Sigma_c^*$ a projeção natural nos eventos controláveis. Para toda sequência $s_{ot} \in P_{\Sigma\to\Sigma_c}(S)$ e $A = P^{-1}_{\Sigma\to\Sigma_c}(s_{ot}) \cap S$, se $P_{\Sigma\to\Sigma_c}(S)$ possui a propriedade do observador, então A será controlável com relação à $\mathcal{L}(G)$.*

Em outros termos, o Teorema 1 garante que qualquer linguagem $A \subseteq S$ obtida da cadeia $s_{ot} \in P_{\Sigma\to\Sigma_c}(S)$, é capaz de ser executada na planta G sem que seja necessário desabilitar eventos não-controláveis. Isto significa que a linguagem A é controlável com relação à G. A linguagem A garante também as condições de segurança e não bloqueio. ■

Para aplicar o resultado do Teorema 1, é preciso verificar se a projeção do comportamento em malha fecha no conjunto dos eventos controláveis possui a propriedade do observador. No entanto, verificar a propriedade do observador possui complexidade quadrática no tamanho do espaço dos estados do supervisor (Pena et al., 2014).

A necessidade dessa verificação pode ser evitada com a utilização de resultados adicionais apresentados por Vilela & Pena (2016). Inicialmente apresentam-se duas definições que servem de diretrizes para a modelagem do problema:

Definicão 2 (Sistema de Produção). *Um sistema de produção G é definido como um sistema composto por m autômatos $M_i = (\Sigma_i, Q_i, \delta, Q_i^m, q_i^0)$, $i \in I = \{1, \ldots, m\}$, tal que:*

i) $G = \|_{i=1}^{m} M_i$, $\Sigma = \bigcup_{i=1}^{m} \Sigma_i$;

ii) $\Sigma_i \cap \Sigma_j = \emptyset$, $i, j \in I$ *e* $i \neq j$;

iii) M_i *é caracterizada por:*

a) $\forall \sigma \in \Sigma_i$ *tal que* $\delta(q_0, \sigma) \neq \emptyset$, $\sigma \in \Sigma_{ic}$;

b) $P_{\Sigma_i\to\Sigma_{ic}}(M_i)$ *possui a propriedade do observador.* □

Em palavras, um sistema de produção é composto por m autômatos com eventos disjuntos (*ii*), todo autômato deve iniciar com evento controlável *iii*)a) e a projeção natural no conjunto dos eventos controláveis deve possuir a propriedade do observador *iii*)b).

Definicão 3 (Especificação de Coordenação). *A especificação de coordenação $E = (\Sigma_E, Q_E, \delta, Q_E^m, q_E^0)$ é uma especificação de cordenação se:*

i) $\mathcal{L}(E)$ *é controlável com relação à* $\mathcal{L}(G)$*;*

ii) $\forall s \in \Sigma^*, t \in \Sigma_u^*, \sigma \in \Sigma_c, u \in \Pi(t).[st\sigma \in \mathcal{L}(E)]$ *e* $[su \in \mathcal{L}(E)] \Longrightarrow su\sigma \in \mathcal{L}(E)$. □

O operador $\Pi(s)$ é operador de permutação de uma palavra s, uma linguagem formada para todas as palavras obtidas por permutações (com possíveis repetições) de s. Uma especificação de coordenação deve ser controlável com relação à planta G. Além disso, se um evento σ está habilitado após uma cadeia $st \in \mathcal{L}(E)$ que termina com evento não controlável, se existe uma outra cadeia u também composta pelos mesmos eventos não controláveis de t, então σ estará habilitado após a cadeia $su \in \mathcal{L}(E)$.

Com a Definição 2 e a Definição 3 é possível estabelecer a Proposição 4.1:

[(Vilela & Pena, 2016)]

Seja a planta G um sistema de produção segundo a Definição 2. Seja E uma especificação de coordenação segundo a Definição 3 e $P_{\Sigma \to \Sigma_c} : \Sigma^* \to \Sigma_c^*$ uma projeção natural. Se $K = G||E$ é controlável $(S = SupC(K, G) = K)$ então $P_{\Sigma \to \Sigma_c}(S)$ tem a propriedade do observador.

A Proposição 4.1 diz que se a planta G é modelada como um sistema de produção e as especificações (composição síncrona de todas) como especificações de coordenação então $P_{\Sigma \to \Sigma_c}(S)$ possui a propriedade do observador.

Caso a especificação E não seja controlável, então obtém-se o supervisor reduzido e usado como especificação de coordenação ao invés de E, conforme estabelecido em (Pena et al., 2009).

Para encontrar o conjunto de todos os caminhos legais de eventos controláveis que produzem K produtos, calcula-se $E_q||P_{\Sigma \to \Sigma_c}(S)$. Define-se E_q na Definição 4.

Definicão 4 (Rafael (2018))**.** *Um autômato* $E_q = (-, \Sigma_{E_q}, -, -, -)$ *é uma especificação de quantidade, para a produção de um lote de* k *produtos, tem-se as seguintes características:*

1. $\Sigma_{E_q} = \{\sigma\}$*, onde* $\sigma \in \Sigma_c$ *e é o último evento controlável da sequência de produção que produz 1 produto;*

2. $\mathcal{L}_m(E_q) = v \subseteq \Sigma_{E_q}^*$*, onde* $\mid v \mid = k$ *e* k *representa a quantidade de produtos.* □

O exemplo a seguir apresenta o resultado do Teorema 1.

Exemplo 5. *O Sistema de Manufatura Simplificado (SFS) é uma extensão do problema da pequena fábrica (Wonham, 2015). O sistema consiste de 4 máquinas (M_1, M_2, M_3, M_4) e 3 buffers de capacidade unitária (B_1, B_2, B_3), Figura 4.1. Como restrição de segurança, deve-se garantir que não haja a ocorrência de underflow ou overflow nos buffers. O conjunto de eventos controláveis $\Sigma_c = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ marcam o início de um processo e o conjunto de eventos não controláveis $\Sigma_u = \{\beta_1, \beta_2, \beta_3, \beta_4\}$ representam o fim da operação de cada máquina.*

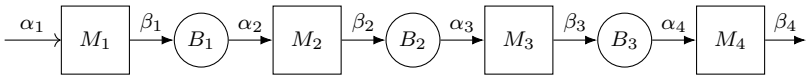
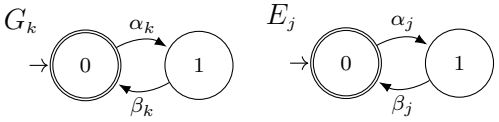


Figura 4.1: Pequena fábrica.

As máquinas são modeladas como autômatos de 2 estados G_k, onde o estado 0 é o estado de inatividade e o estado 1 representa o estado de operação ou de trabalho. As máquinas e os buffers são apresentados na Figura 4.2.



(a) Máquina M_k. (b) Especificação dos buffers E_j.

Figura 4.2: G_k é o modelo das máquinas ($k = 1, \ldots, 4$) e E_j das especificações ($j = 1, 2, 3$).

O problema de controle é evitar underflow e overflow dos buffers unitários, neste caso o buffer E_j é considerado vazio quando está no estado 0 e cheio no estado 1. Uma vez as plantas (G_1, G_2, G_3, G_4) e as especificações (E_1, E_2, E_3) são definidas, elas são usadas para computar o supervisor S (54 estados e 120 transições) e então a projeção natural $P_{\Sigma \to \Sigma_c}(S)$ sobre os eventos controláveis é aplicada. As duas operações foram feitas usando UltraDES (Alves et al., 2017). A Figura 4.3 apresenta a projeção do supervisor (8 estados e 12 transições).

Para gerar todas as cadeias de eventos controláveis que permitem a produção de 2 produtos no SFS, primeiro faz-se a composição paralela entre a projeção natural ($P(S)$) e a especificação de quantidade (E_q). Para um lote de tamanho 2, a especificação de quantidade é mostrada na Figura 4.4.

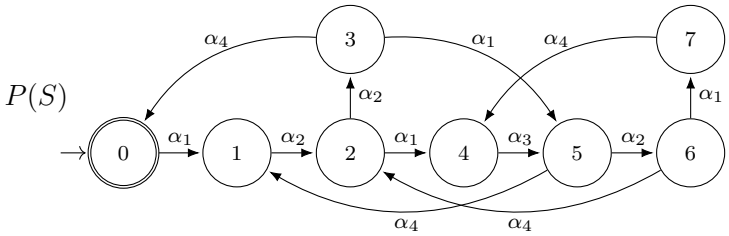


Figura 4.3: Projeção natural do supervisor S sobre o conjunto dos eventos controláveis no SFS.

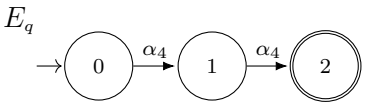


Figura 4.4: Especificação de quantidade (E_q) para um lote de tamanho 2, Definição 4.

A projeção do supervisor da Figura 4.3 possui a propriedade do observador segundo o algoritmo de verificação proposto por Pena et al. (2014). O autômato resultante $P_{Eq} = E_q||P_{\Sigma\to\Sigma_c}(S)$ *(Figura 4.5), representa todas as sequências possíveis de eventos controláveis, que uma vez executados, levam à fabricação de 2 produtos no SFS.*

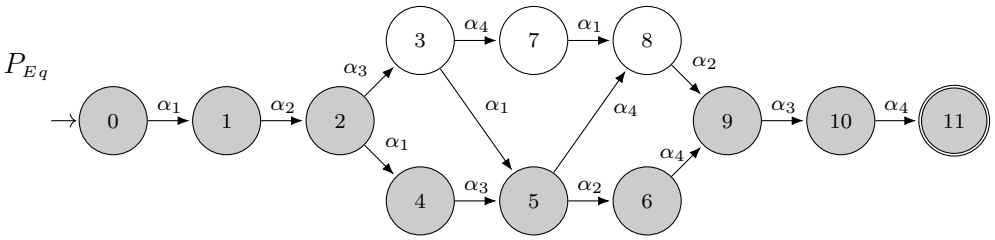


Figura 4.5: Autômato P_{Eq}.

Escolhe-se uma sequência $s_{ot} \in P_{Eq}$*, sendo* $s_{ot} = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\alpha_4\alpha_3\alpha_4$ *(sequência que passa pelos estados em cinza). De acordo com o Teorema 1, a linguagem A é controlável com respeito à* $\mathcal{L}(G)$*. O autômato A pode-se calcular aplicando:* $A = P^{-1}_{\Sigma\to\Sigma_c}(s_{ot}) \cap S$*, Figura 4.6. A adição dos eventos não controláveis (interseção com S) na sequência* s_{ot} *não afeta a controlabilidade da linguagem A.*

As condições apresentadas no Teorema 1 são suficientes para que a linguagem recuperada A para qualquer sequência $s_{ot} \in P_{\Sigma\to\Sigma_c}(S)$ *seja controlável com relação à* $\mathcal{L}(G)$*, mas não são necessárias.*

Essa linguagem A é vista como um novo supervisor, mas que implementa a linguagem composta de todas as cadeias de $\mathcal{L}_m(S/G)$ que projetam em s_{ot}. Como a linguagem A é

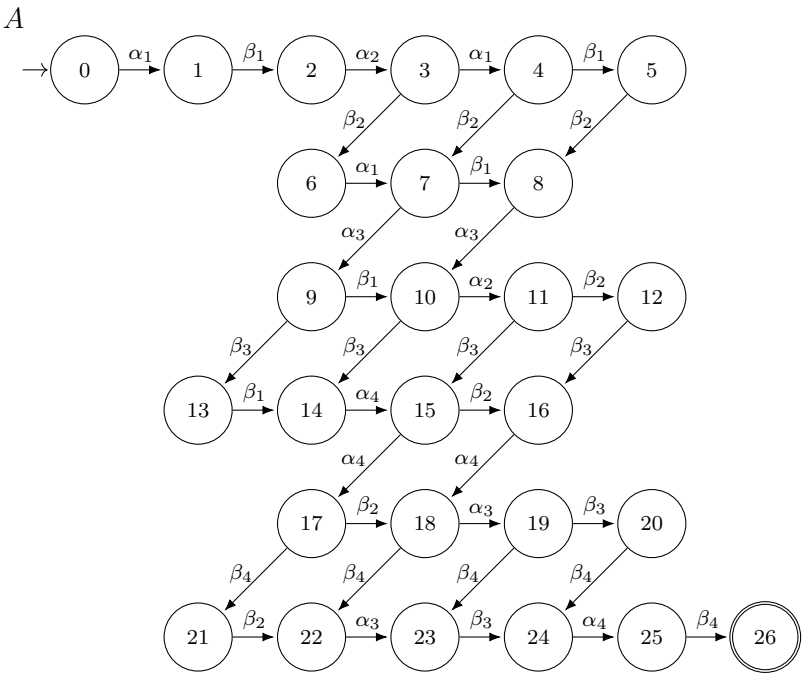


Figura 4.6: Autômato que implementa A.

controlável, pode-se assegurar a implementação da sequência s_{ot}. A condição principal do Teorema 1 é que a $P_{\Sigma\to\Sigma_c}(S)$ seja uma PO-abstração.

A seguir, apresenta-se um exemplo em que as condições para a aplicação do Teorema 1 não estão presentes.

Exemplo 6. *Considere o sistema de manufatura (Zhang et al., 2017) mostrado na Figura 4.7, composto por 2 máquinas (M_1 e M_2) e uma unidade de teste (TU), conectados por um buffer de capacidade 3 (B_1) e um buffer unitário (B_2). A máquina M_1 modelada pelo autômato da Figura 4.8(a) recebe o insumo bruto (evento α_1) e após processá-lo o deposita no buffer B_1 (evento β_1). A máquina M_2 retira as peças (evento α_2) de trabalho do buffer de entrada B_1, processa e deposita as mesmas (evento β_2) no buffer de saída (B_2). A máquina TU, modelada pelo autômato da Figura 4.8(b), testa a qualidade das peças (evento α_3) que foram retiradas de B_2 e aprova (evento β_3) ou rejeita (evento β_4). Desse modo, um insumo bruto deve ser processado pelos 3 subsistemas, em sequência para que torne um produto final.*

A restrição deste problema é evitar underflow ou overflow nos buffers B_1 e B_2 modelados pelos autômatos da Figura 4.9.

Para aplicar o resultado do Teorema 1, é necessário que a projeção natural do com-

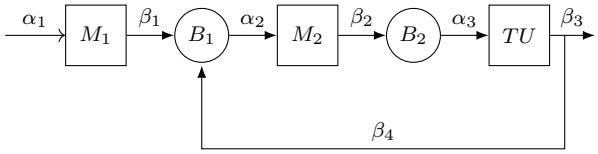
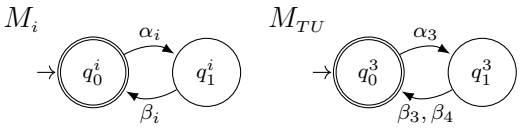
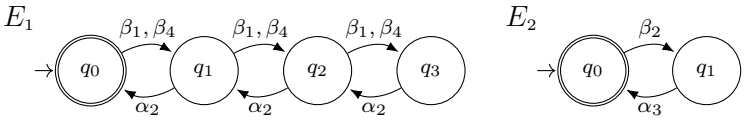


Figura 4.7: Sistema de manufatura com retrabalho.



(a) M_i, $i = \{1, 2\}$. (b) M_{TU}.

Figura 4.8: Modelos das máquinas.



(a) Especificação do *Buffer* B_1. (b) Especificação do *Buffer* B_2.

Figura 4.9: Modelos das especificações.

portamento em malha fechada, projetado no conjunto de eventos controláveis, tenha a propriedade do observador. No caso deste sistema, não acontece, o que é esperado já que a sequência para produção de uma peça depende de eventos não controláveis (β_3 e β_4), ou seja, $\alpha_1\alpha_2\alpha_3$ produz um produto, caso seja seguido de β_3. Alternativamente, quando falha o teste (β_4), a produção de um produto é dada pela sequência $\alpha_1\alpha_2\alpha_3\alpha_2\alpha_3$.

Os conceitos apresentados nesse capítulo e no Capítulo 2 fundamentam a contribuição principal deste trabalho, que será apresentada no próximo capítulo.

5

Resultados

O Teorema 1 apresentado no capítulo anterior refere-se à aplicação da abstração do comportamento em malha fechada pela síntese monolítica. Não obstante, a aplicação dessa abordagem está sujeita à condição principal de que a abstração do comportamento em malha fechada deve possuir a propriedade do observador. Por conseguinte, ao violar essa condição principal, a linguagem recuperada A não será garantidamente controlável com relação à $\mathcal{L}(G)$ porque em alguns do seus estados, eventos não controláveis podem ser desabilitados. Desse modo, a formulação do problema deste trabalho é apresentada na Seção 5.1.

5.1 Formulação do Problema

Seja um sistema físico modelado por um autômato $G = (-, \Sigma, -, -, -)$, com $\mathcal{L}(G) \subseteq \Sigma^*$ e com alfabeto $\Sigma = \Sigma_c \cup \Sigma_u$. Seja o conjunto de restrições modeladas por E, a linguagem desejada para o sistema em malha fechada por $K = E||G$ e o supervisor $S = SupC(K, G)$. Seja $P : \Sigma^* \to \Sigma_c^*$ uma projeção natural. Caso $P_{\Sigma \to \Sigma_c}(S)$ não possua a propriedade do observador que é a condição principal para a aplicação do Teorema 1, propor outra abstração tal que seja possível resolver o problema de planejamento, ou seja, obter a partir desta abstração uma linguagem A que será controlável com relação à $\mathcal{L}(G)$.

Portanto, o objetivo deste trabalho é estender o resultado principal de Vilela & Pena (2016) para lidar com o caso em que a abstração do comportamento em malha fechada não é PO-abstração. A estratégia proposta consiste em estender o conjunto de eventos mantidos na projeção (Bravo et al., 2014), para obter uma PO-abstração e lidar com as modificações necessárias nas definições e teorema, causados pela presença de eventos não-controláveis.

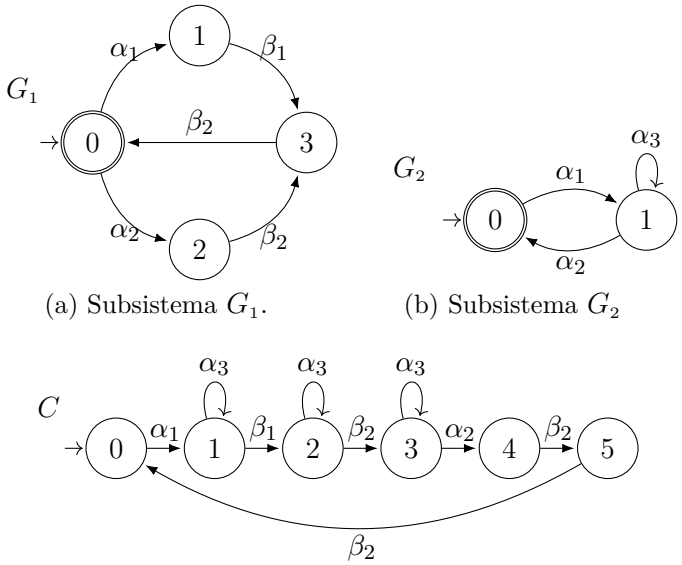
5.2 Definições

Antes de apresentar a principal contribuição deste trabalho, serão apresentadas algumas definições que auxiliarão a demonstração do mesmo. A seguir é apresentada a Definição 5, que calcula o rótulo de uma composição paralela entre dois autômatos.

Definicão 5 (Rótulo de uma Composição)**.** *Seja* $A_i = (-, \Sigma_i, \delta_i, q_i^0, -)$*, com* $i \in I = \{1, ..., m\}$ *e* $A = (Q, \Sigma, \delta, q_0, -)$ *obtido por* $A = \parallel_{i=1}^{m} A_i$*. Seja* $P_i : \Sigma^* \to \Sigma_i^*$*. A função que apresenta o rótulo de cada estado de uma composição síncrona é:* $r_A(q) = ((\delta_1(q_1^0, P_1(s)), \delta_2(q_2^0, P_2(s)), \ldots, \delta_m(q_m^0, P_m(s)))$ *onde* $s \in \Sigma^*$ *e* $q = \delta(q_0, s)$ *para* $q \in Q$*.* □

A função rótulo, segundo a Definição 5, apresenta a informação dos estados de cada autômato A_i que compõem cada estado $q \in Q$ de A. Ela pode ser aplicada a qualquer autômato resultante de uma composição. A fim de ilustrar essa definição, o Exemplo 7 é apresentado.

Exemplo 7. *Sejam os autômatos* $G_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, -), G_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, -)$ *e* $G = (Q_G, -, \delta_G, q_0^G, -)$ *apresentados no Exemplo 2 (são apresentados novamente na Fig. 5.1). Calcular o rótulo do estado* $4 \in Q_G$ *usando a Definição 5.*



(a) Subsistema G_1. (b) Subsistema G_2

(c) Autômato G que implementa $G_1||G_2$

Figura 5.1: Rótulo de uma composição.

O rótulo do estado 4 de G é:

$$\begin{aligned} r_G(4) &= (\delta_1(q_1^0, P_1(\alpha_1\beta_1\beta_2\alpha_2)), \delta_2(q_2^0, P_2(\alpha_1\beta_1\beta_2\alpha_2)) \\ &= (\delta_1(q_1^0, \alpha_1\beta_1\beta_2\alpha_2), \delta_2(q_3^0, \alpha_1\alpha_2)) \\ &= (2, 0) \end{aligned}$$

O rótulo $r_G(4)$ *do autômato G, pela Definição 5, é* $(2, 0)$ *sendo* $s = \alpha_1\beta_1\beta_2\alpha_2$ *tal que* $\delta_G(0, s) = 4$*. Portanto, para todo estado* $q \in Q_G$ *é possível obter* $r_G(q)$*:*

Tabela 5.1: Rótulos do autômato G.

Estado de G	s	$r(q)$
0	ϵ	$(0, 0)$
1	α_1	$(1, 1)$
2	$\alpha_1\beta_1$	$(3, 1)$
3	$\alpha_1\beta_1\beta_2$	$(0, 1)$
4	$\alpha_1\beta_1\beta_2\alpha_2$	$(2, 0)$
5	$\alpha_1\beta_1\beta_2\alpha_2\beta_2$	$(3, 0)$

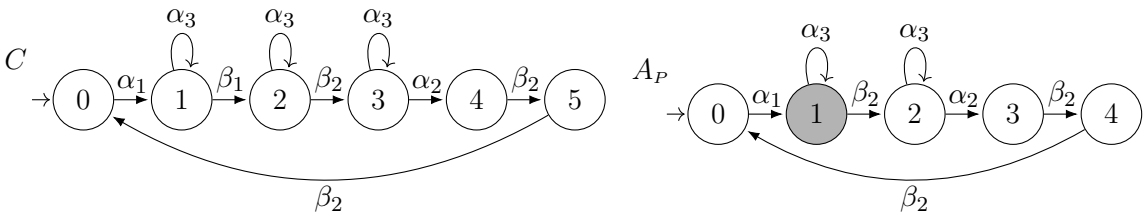
A Definição 5 pode ser usada para calcular o rótulo de cada estado do autômato que implemente $K = \mathcal{L}(G)||\mathcal{L}(E)$ que implementa o comportamento desejado pelo simples fato de K ser resultante de uma composição síncrona. Se K não é controlável, então a máxima sublinguagem controlável e não bloqueante $S = Sup\mathcal{C}(K, G)$ pode ser sintetizada e implementada por um supervisor (Ramadge & Wonham, 1989). Porém, como S é um subautômato de K, então os rótulos calculados para o autômato que implementa K são os mesmos para o autômato que implementa S (depois da eliminação dos maus estados de K).

A seguir apresenta-se a função que calcula o rótulo para cada estado de um autômato resultante de uma projeção natural.

Definição 6 (Rótulo de uma Projeção)**.** *Seja o autômato $A_p = P_{\Sigma_A \to \Sigma_B}(C)$ onde $A_p = (Q_p, -, \delta_p, q_0^P, -)$ e $C = (Q_c, -, \delta_c, q_0^C, -)$. Estabelece-se a função $d' : Q_p \to 2^{Q_C}$ com: $d'(\delta_p(q_0^P, t)) = \{q' \in \{\delta_c(q_0^C, s) \mid s \in P^{-1}(t) \cap \mathcal{L}(C)\}\}$. A função $d(d'(q))$ converte um conjunto de tuplas em uma tupla de conjuntos.* □

A função rótulo segundo a Definição 6, apresenta a informação dos estados do autômato C que compõem cada estado $q \in Q_p$ de A_p. Ela pode ser aplicada a qualquer autômato resultante da operação de uma projeção natural. A fim de aclarar essa definição o Exemplo 8 é apresentado.

Exemplo 8. *Seja o autômato $C = (Q_C, -, \delta_C, q_0^C, -)$ da Figura 5.2(a).*



(a) Autômato C. (b) Projeção natural $A_p = P(G)$.

Figura 5.2: Rótulo do estado de um autômato resultante de uma projeção.

O autômato $A_p = (Q_p, -, \delta_p, q_0^P, -)$ da Fig. 5.2 (b) implementa a linguagem $P_{\Sigma \to \Sigma_1}(C)$ com $\Sigma_1 = \{\alpha_1, \alpha_2, \beta_2\}$. O rótulo do estado 1, $d'(1) = d'(\delta_p(q_o^p, \alpha_1))$ de A_p é obtido identificando, inicialmente, quais estados de C que são alcançados por cadeias que projetam em α_1 (no

caso, $\alpha_1, \alpha_1\beta_1$*, que levam a* $\{1,2\} \subseteq Q_C$*). Portanto, o rótulo do estado* 1 *de* A_p *é calculado como:*

$$
\begin{aligned}
d'(\delta_p(q_0^P, \alpha_1)) &= \{q' \in \{\delta_C(q_0^C, s) \mid s \in P^{-1}(\alpha_1) \cap \mathcal{L}(C)\}\} \\
d'(1) &= \{q' \in \{\delta_C(q_0^C, s) \mid s \in \beta_1^* \alpha_1 \beta_1^* \cap \mathcal{L}(C)\}\} \\
d'(1) &= \{q' \in \{\delta_C(q_0^C, s) \mid s \in \{\alpha_1, \alpha_1\beta_1\}\}\} \\
d'(1) &= \{q' \in \{1,2\}\} \\
d'(1) &= \{1,2\}
\end{aligned}
$$

Por fim, a função $d(d'(q))$ *converte o conjunto de tuplas* $\{1,2\}$ *em uma tupla de conjuntos, assim:* $d(d'(1)) = (\{1\},\{2\})$.

A seguir, apresenta-se a Definição 7, a qual define o sistema de produção adaptado de (Vilela & Pena, 2016). Esta definição diz respeito à forma com que o sistema é modelado.

Definicão 7 (Sistema de Produção)**.** *Seja* G *um sistema composto por* m *subsistemas. Cada subsistema é modelado por um autômato* $M_i = (Q_i, \Sigma_i, \delta_i, Q_i^m, q_i^0)$*,* $i \in I = \{1, ...m\}$ *tal que:*

i) $G = \prod_{i=1}^{m} M_i$*,* $\Sigma = \bigcup_{i=1}^{m} \Sigma_i$ *onde* $\Sigma_i = \Sigma_{i_c} \cup \Sigma_{i_u}$*;*

ii) $\Sigma_i \cap \Sigma_j = \emptyset$*,* $i, j \in I$ *e* $i \neq j$*;*

iii) M_i *é caracterizada por:*

- **a)** $\forall \sigma \in \Sigma_i$ *tal que* $\delta(q_i^0, \sigma) = q_a$*,* $\sigma \in \Sigma_{ic}$*;*
- **b)** $\exists w_{i_k} = \sigma_{i_k}^{\uparrow} t_{i_k}^{\downarrow} \in \mathcal{L}_m(M_i)$ *com* $k \geq 1$*,* $\sigma_{i_k}^{\uparrow} \in \Sigma_{i_c}$ *e* $t_{i_k}^{\downarrow} \in \Sigma_{i_u}$ *tal que* $\delta(q_i^0, \sigma_{i_k}^{\uparrow}) = q_a$*,* $\delta(q_a, t_{i_k}^{\downarrow}) = q_i^0$*, e* q_a *é denominado estado* ***ativo****;*

iv) $\Sigma_{\sigma\uparrow} = \{\sigma_{i_k}^{\uparrow} | \forall w_{i_k} \in \mathcal{L}_m(M_i), i = 1, \ldots, m\}$*,* $\Sigma_{t\downarrow} = \{t_{i_k}^{\downarrow} | \forall w_{i_k} \in \mathcal{L}_m(M_i), i = 1, \ldots, m\}$*;*

v) $\Sigma_{fix}^C = \{\sigma \in \Sigma_c | \forall t \in \Sigma_u \backslash \Sigma_{t\downarrow} \Rightarrow \exists \sigma \in \Sigma_c$*, que retorna à produção*$\}$*;*

vi) $M^{Ru} = \{M_i | \Sigma_i \cap \Sigma_{Ru} \neq \emptyset\}$*;*

vii) $\Sigma_c = \Sigma_{\sigma\uparrow} \cup \Sigma^C_{fix}$, *onde os alfabetos* $\Sigma_{\sigma\uparrow}$ *e* Σ^C_{fix} *não necessariamente são disjuntos.* □

A definição de Sistema de Produção introduzida por (Vilela & Pena, 2016) e apresentada na Seção 4.1 (Definição 2) se diferencia da Definição 7 no que diz respeito às características que deve ter cada subsistema M_i (item *iii*). No item *iii* da Definição 2 cada subsistema deve possuir a propriedade do observador. Essa caraterística não é relevante na Definição 7, pois o objetivo neste trabalho é trabalhar com sistemas que não são PO-abstração.

Em palavras, o Sistema de Produção da Definição 7 é composto por autômatos com conjuntos de eventos disjuntos (item *ii*)). Estes autômatos são sempre iniciados com um evento controlável ($\sigma^{\uparrow}_{i_k}$) e finalizados com um evento não controlável ($t^{\downarrow}_{i_k}$) (item *iii*)b)). w_{i_k} é uma cadeia que representa um ciclo de trabalho de M_i e o índice $k \in \mathbb{Z}^+$ representa o número de tarefas que o subsistema M_i realiza (item *iii*)b)). Em cada subsistema M_i existe pelo menos uma cadeia $w_{i_k} = \sigma^{\uparrow}_{i_k} t^{\downarrow}_{i_k}$. O evento $\sigma^{\uparrow}_{i_k}$ (ligar) dispara o início de w_{i_k} e o evento $t^{\downarrow}_{i_k}$ (desligar) causa o final de w_{i_k}. O item *b*) nesta definição, foi adaptado de (Bravo et al., 2018).

O conjunto $\Sigma_{\sigma\uparrow}$ contém todos os eventos controláveis de liga e $\Sigma_{t\downarrow}$ contém todos os eventos não controláveis que causam o final de uma tarefa (item *iv*)). O conjunto Σ^C_{fix} do item *v*) é um conjunto de eventos controláveis tal que para todo evento não controlável pertencente ao conjunto $\Sigma_u \backslash \Sigma_{t\downarrow}$ existe pelo menos um evento controlável que leva o sistema de volta à condição típica de funcionamento. $\Sigma_u \backslash \Sigma_{t\downarrow}$ são eventos que podem ocorrer no sistema mas não contribuem para a produção de um produto (Seção 5.4). Por último, o conjunto M^{Ru} (item *v*)) contém um conjunto de máquinas M_i que apresentam eventos do conjunto Σ_{Ru}, onde Σ_{Ru} é o conjunto de eventos acrescentados na projeção para obter a PO (Seção 3.5).

A seguir, apresenta-se a Definição 8, a qual apresenta o autômato que implementa uma especificação de quantidade para a produção de um lote de K produtos. A Definição 8 é uma modificação da Definição 4 no que diz respeito ao conjunto de eventos Σ_{E_p}.

Definicão 8. *Um autômato* $E_p = (-, \Sigma_{E_p}, -, -, -)$ *é uma especificação de produção para a produção de um lote de k produtos e tem as seguintes características:*

1. $\Sigma_{E_p} = \{\sigma\}$, *sendo* σ *o último evento da sequência de produção que produz 1 produto,*

seja controlável ou não controlável, segundo o sistema de produção.

2. $\mathcal{L}_m(E_p) = v \subseteq \Sigma^*_{E_p}$, $\mid v \mid = k$, *onde* $k \in \mathbb{Z}^+$ *representa a quantidade de produtos.* □

Como foi dito na Seção 3.5, o algoritmo proposto por (Bravo et al., 2014) busca um conjunto de instâncias de eventos não-controláveis que não podem ser apagados na projeção natural posto que causam a violação da propriedade do observador. Portanto, neste trabalho, esse conjunto de eventos não-controláveis será chamado de Σ_{Ru} com $\Sigma_{Ru} \subseteq \Sigma_u$, que, ao ser mantido na projeção, torna a mesma uma PO-Abstração. Por conseguinte, o novo conjunto de eventos na projeção do comportamento em malha fechada é $\Sigma_c \cup \Sigma_{Ru}$ de modo que $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)$ seja uma PO-abstração.

No resultado anterior (Teorema 1, (Vilela & Pena, 2016)), escolhia-se uma sequência s_{o_t} qualquer da projeção do comportamento em malha fechada e constrói-se a partir dela, uma linguagem $A = P^{-1}_{\Sigma\rightarrow\Sigma_c}(s_{o_t}) \cap S$ que será controlável com relação a $\mathcal{L}(G)$. Nesta extensão, é necessário construir uma linguagem $\mathcal{F}^{\uparrow}$ invés de só escolher uma sequência s_{o_t} da projeção do comportamento em malha fechada para obter uma linguagem $A = P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S$ que será controlável com relação a $\mathcal{L}(G)$. Para mostrar esta ideia da linguagem $\mathcal{F}^{\uparrow}$, o Exemplo 9 é apresentado.

Exemplo 9. *Seja o Sistema de Manufatura Simplificado (SFS) ou pequena fábrica apresentado em (Vilela & Pena, 2016). O sistema consiste de 2 máquinas (M_1 e M_2) e 1 buffer de capacidade unitária (B_1), Figura 5.3. A máquina M_1 modelada pelo autômato da Figura 5.3(a) terá um evento (β_3) que levará o subsistema para um estado de quebra e um evento de conserto (α_3) que retornará o subsistema novamente para o estado de trabalho. A máquina M_2 não terá seus eventos de quebra e conserto representados, Figura 5.3(b).*

Como restrição de segurança, deve-se garantir que não haja a ocorrência de underflow ou overflow no buffer B_1, Figura 5.5. O conjunto de eventos controláveis são $\Sigma_c = \{\alpha_1, \alpha_2, \alpha_3\}$ e o conjunto de eventos não controláveis são $\Sigma_u = \{\beta_1, \beta_2, \beta_3\}$.

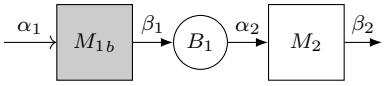
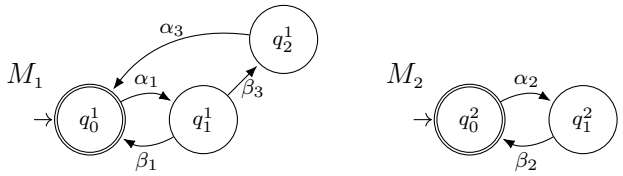


Figura 5.3: Pequena fábrica.



(a) Modelo da Máquina M_1 com quebra.

(b) Modelo da Máquina M_2.

Figura 5.4: Modelo das Máquinas.

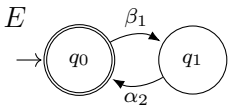


Figura 5.5: Especificação do buffer B_1.

Uma vez que a especificação e as plantas são definidas, elas são usadas para computar o supervisor S e a projeção natural $P_{\Sigma\to\Sigma_c}(S)$ sobre os eventos controláveis é aplicada. A Figura 5.6 apresenta o supervisor e a Figura 5.7 apresenta a projeção do supervisor. Para explicitar as sequências de eventos que levam à produção de um lote de tamanho 2, utiliza-se uma especificação de quantidade conforme a Definição 4, mostrada na Figura 5.8.

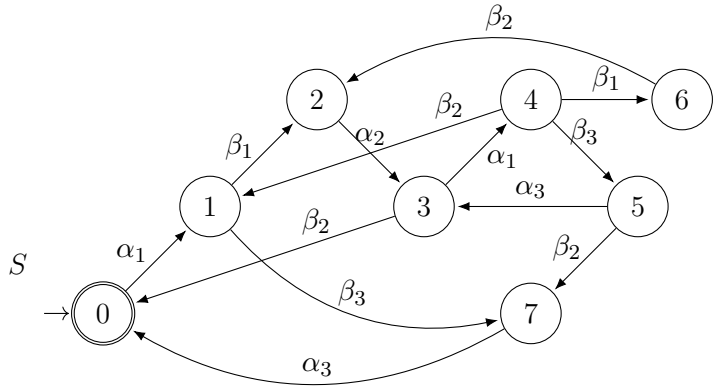


Figura 5.6: Supervisor.

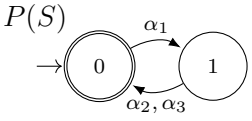


Figura 5.7: Projeção do Supervisor.

A projeção do comportamento em malha fecha mostrada na Figura 5.7 não possui a propriedade do observador. Faz-se então Trim sobre o resultado da composição paralela da abstração do supervisor e da especificação de quantidade. O resultado obtido é mostrado na Figura 5.9.

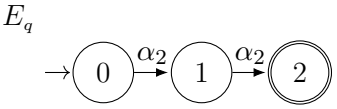


Figura 5.8: Especificação de quantidade para a produção de 2 produtos.

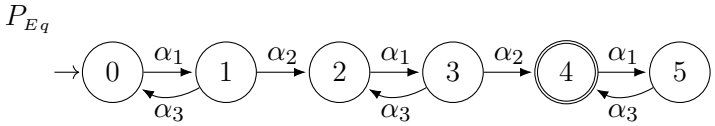


Figura 5.9: Composição paralela entre $P_{\Sigma\to\Sigma_c}(S)$ e a especificação de quantidade para 2 produtos.

O autômato da Figura 5.9 explicita todas as sequências de eventos controláveis que levam à produção de dois produtos. Os ciclos do evento α_3 representam o conserto da quebra na máquina M_1 causada pela ocorrência do evento β_4 mas que foi apagado pela projeção.

Depois de cada ocorrência do evento α_1, existem duas possibilidades de eventos acontecerem: a primeira delas é acontecer o evento α_3 que representa o conserto da falha na máquina M_1 e a segunda possibilidade é acontecer o evento α_2 que representa ligar a máquina M_2. Portanto, seja qual for o evento que vai acontecer (α_3 ou α_2), um deles terá que ser desabilitado. Do autômato da Figura 5.9, tem-se a seguinte sequência de produção:

$$s_1 = \alpha_1\alpha_2\alpha_1\alpha_2,$$

Supondo que a quebra/reparo não faz parte do comportamento de produção. Esse autômato será o modelo do sistema, que é uma das entradas do Planejador da Figura 2.2. O resultado do Teorema 1 será aplicado sobre esse autômato. Suponha que o Planejador tenha retornado como plano ótimo s_{o_t} a sequência s_1. Assim:

$$s_{o_t} = s_1 = \alpha_1\alpha_2\alpha_1\alpha_2$$

A sequência $s_{o_t} = \alpha_1\alpha_2\alpha_1\alpha_2$, não apresenta informação do evento α_3, o qual significa que o evento β_4 que foi apagado na projeção natural está sendo desabilitado. Essa sequência deveria apresentar o evento α_3 para não ter que desabilitar o evento β_4 que representa a ocorrência da quebra da máquina M_1 e o evento β_1 que representa a desliga da máquina

M_1.

O Teorema 1 diz que, caso $P(S)$ seja PO, a linguagem A, que é uma sublinguagem do comportamento em malha fechada que projeta para a sequência s_{o_t}, é controlável. Constrói-se $A = P^{-1}_{\Sigma\to\Sigma_c}(s_{ot}) \cap S$, representada pelo autômato mostrado na Figura 5.10 e percebe-se claramente que A não é controlável, evento β_3 desabilitado nos estados $1, 4$ e 7.

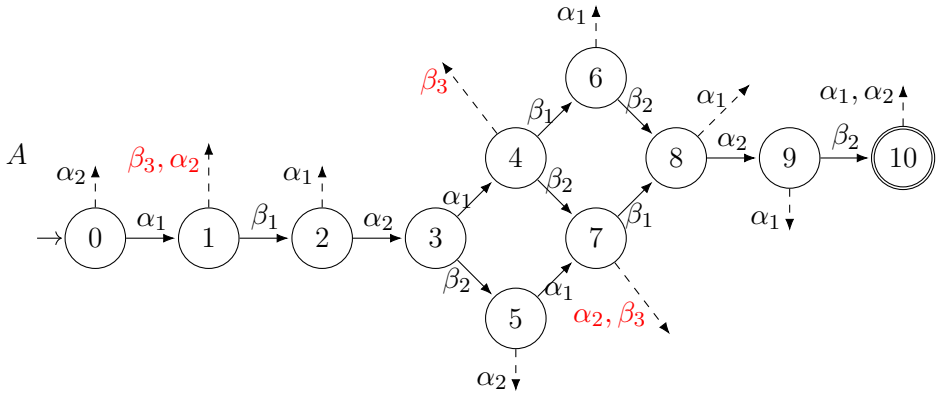


Figura 5.10: Linguagem A.

As setas tracejadas da Figura 5.10 representam as desabilitações de eventos. Como pode ser observado, o evento β_3 está sendo desabilitado nos estados $\{1, 4, 7\}$, mostrando assim a não controlabilidade da linguagem A em relação à planta. Portanto, com este exemplo fica evidenciado que simplesmente uma sequência não é suficiente para obter uma linguagem A controlável com relação à planta em sistemas onde sua $P_{\Sigma\to\Sigma_c}(S)$ não tem a propriedade do observador, portanto, é preciso acrescentar algumas informações na sequência s_{o_t}.

Como pode ser observado no autômato da Figura 5.10, os estados onde o evento β_3 está sendo desabilitado, são aqueles que estão depois da ocorrência do evento α_1 que pertence à máquina M_1 a qual contém a quebra. Em vista disso, a sequência $s_{o_t} = \alpha_1\alpha_2\alpha_1\alpha_2$ deve mostrar depois de cada ocorrência do evento α_1, os futuros dos eventos β_1 e β_3 os quais são diferentes. Quando essa informação é acrescentada na sequência $s_{o_t} = \alpha_1\alpha_2\alpha_1\alpha_2$, deixa de ser uma sequência e começa a tornar-se uma linguagem ($\mathcal{F}^{\uparrow}$).

O primeiro passo é usar o algoritmo OP-search detalhado na Seção 3.5 para obter uma projeção com a propriedade do observador, Figura 5.11. Pode-se observar na Figura 5.11 que os eventos β_1 e β_3 estão agora factíveis depois da ocorrência do evento α_1, os quais não estavam presentes na projeção do supervisor sem a propriedade do observador (Figura

5.7).

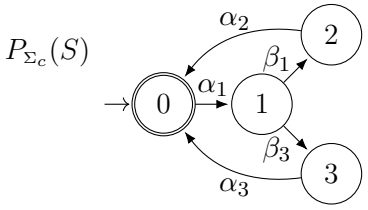


Figura 5.11: Projeção do supervisor com a propriedade do observador.

Depois da aplicação do algoritmo OP-search, os eventos não controláveis β_1 e β_3 não poderão ser apagados na projeção para garantir a propriedade do observador. Esses eventos não controláveis são o conjunto de eventos renomeados $\Sigma_{R_u} = \{\beta_1, \beta_3\}$, portanto, a nova projeção do comportamento em malha fechada com a propriedade do observador é definida com $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)$. Agora com esta projeção é possível escolher uma sequência s e construir uma linguagem $\mathcal{F}^{\uparrow}$. Este processo é descrito na próxima seção.

5.3 Algoritmo que Calcula a Linguagem $\mathcal{F}^{\uparrow}$

O seguinte algoritmo constrói uma linguagem $\mathcal{F}^{\uparrow}$ a partir de uma sequência $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ onde $T_{E_q} = Trim(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$ que será usada no Teorema 2 proposto neste trabalho. O algoritmo consiste de uma série de passos e procedimentos para gerar a linguagem $\mathcal{F}^{\uparrow}$, análoga à cadeia s_{o_t}, usada no Teorema 1. A seguir, os passos do algoritmo são apresentados.

Algorithm 1 Linguagem $\mathcal{F}^{\uparrow}$

Entrada: $s, \Sigma_{\sigma\uparrow}, \Sigma_{t\downarrow}, \Sigma^{C}_{fix}, \Sigma_{Ru}, M^{Ru}$.

1) ***Sequência*** s_{o_t}***:***

   (i) $s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s)$;

   (ii) $\mathcal{L}_m(G_{s_{o_t1}}) \leftarrow \{s_{o_t}\}$.

2) ***Auto-laços de*** Σ_{Ru} ***e*** Σ^{C}_{fix} ***em*** $G_{s_{o_t1}}$:

   (i) $\mathcal{L}_m(G_{s_{o_t2}}) = P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})\to\Sigma_{\sigma\uparrow}}(\mathcal{L}_m(G_{s_{o_t1}}))$;

   (ii.1) se $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$, então:
   $\mathcal{L}_m(G_{s_{o_t3}}) = P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\to(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})}(\mathcal{L}_m(G_{s_{o_t2}}))$.

   (ii.2) se $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$, então:
   Redefine-se δ de $G_{s_{o_t2}}$ tal que: $(\forall q_i \in Q_{s_{o_t2}})(\forall b \in \Sigma^{C}_{fix})$, se $b \notin \Gamma(q_i) \Longrightarrow \delta(q_i, b) = q_i$.
   Assim, obtém-se o autômato $G_{s_{o_t3}}$.

3) ***Auto-laços do evento*** $\sigma^{\uparrow}_i$ ***em cada estado*** $q \in Q$ ***de*** $G_{s_{o_t3}}$ ***onde o subsistema*** $M_i \in M^{Ru}$ ***está no estado ativo*** q_a:

   (i) Redefine-se δ de $G_{s_{o_t3}}$, para incluir $\delta(q, \sigma^{\uparrow}_i) = q$, se o i-ésimo elemento da tupla de conjuntos obtida fazendo $d(d'(q))$ inclui o estado $q_a \in Q_i$ e $M_i \in M^{Ru}$. Assim, obtém-se o autômato $G_{s_{o_t4}}$.

   (ii) $\mathcal{F}^{\uparrow} \leftarrow \mathcal{L}_m(G_{s_{o_t4}})$.

O autômato $T_{E_q} = Trim(P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$, é resultado da composição paralela entre o autômato que implementa a projeção natural do comportamento em malha fechada $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ com a propriedade do observador e a especificação de produção E_p (Definição 8). A linguagem $\mathcal{F}^{\uparrow}$ é construída em três procedimentos. O passo (1) constrói o autômato $G_{s_{o_t1}}$ a partir da sequência $s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s)$ onde $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ (s é uma entrada do algoritmo). A sequência s está composta apenas por eventos de liga $\Sigma_{\sigma\uparrow}$ e desliga $\Sigma_{t\downarrow}$ (lembrando que os eventos de $\Sigma_{t\downarrow}$ são só os que fazem interseção com Σ_{Ru}).

O passo (2) é subdividido em dois subpassos. No subpasso 2(i) faz-se uma projeção inversa gerando a linguagem $\mathcal{L}_m(G_{s_{o_t2}}) = P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})\to\Sigma_{\sigma\uparrow}}(\mathcal{L}_m(G_{s_{o_t1}}))$. O subpasso 2($ii$) que acrescenta autolaços de Σ^{C}_{fix}, é subdividido em dois casos:

O primeiro caso é se $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$, então faz-se, $P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\to(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})}(\mathcal{L}_m(G_{s_{o_t2}})) = \mathcal{L}_m(G_{s_{o_t3}})$.

O segundo caso é se $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$, então redefine-se δ de $G_{s_{o_t2}}$ tal que: $(\forall q_i \in Q_{s_{o_t2}})(\forall b \in$

Σ^{C}_{fix}), se $b \notin \Gamma(q_i) \Longrightarrow \delta(q_i, b) = q_i$. Assim, obtém-se o autômato $G_{s_{o_t3}}$.

Seja o caso $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, o caso $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$, significa que há eventos de Σ^{C}_{fix} na cadeia s. O caso $2(ii.1, ii.2)$ do algoritmo acrescenta autolaços de Σ^{C}_{fix}, sem tornar o autômato não determinístico. O autômato $G_{s_{o_t3}}$ é obtido ao acrescentar autolaços de Σ^{C}_{fix} seja pelo caso $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$ no autômato $G_{s_{o_t2}}$, o qual significa que a linguagem do autômato $G_{s_{o_t2}}$ continua contida na linguagem do autômato $G_{s_{o_t3}}$.

Por último, o passo (3) acrescenta auto-laços do evento $\sigma^{\uparrow}_{i}$ em cada estado $q \in Q$ do autômato $G_{s_{o_t3}}$ ($G_{s_{o_t3}}$ gerado no passo anterior) onde o subsistema $M_i \in M^{Ru}$ está no estado ativo q_a. Para saber se o subsistema $M_i \in M^{Ru}$ está no estado ativo basta olhar para a tupla de conjuntos do rótulo $d(d'(q))$ em cada estado $q \in Q$. Este último passo, redefine a função δ de $G_{s_{o_t3}}$ para incluir $\delta(q, \sigma^{\uparrow}_{i}) = q$. A linguagem obtida no final do processo é chamada de linguagem $\mathcal{F}^{\uparrow}$.

Com o propósito de mostrar este procedimento do Algoritmo, retoma-se o Exemplo 9. A projeção do comportamento em malha fechada do Sistema de Manufatura Simplificado (Exemplo 9) com a propriedade do observador depois de aplicar o algoritmo OP-search é apresentada na Figura 5.12.

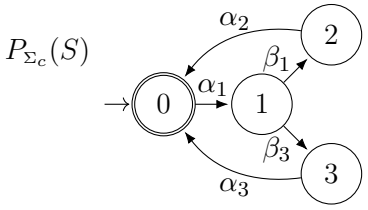


Figura 5.12: Projeção do supervisor com a propriedade do observador.

Observe que foi necessário para obter a propriedade do observador, incluir os eventos β_1 e β_3 na projeção. Estes eventos definem futuros diferentes e portanto devem ser mostrados. O autômato da projeção é composto com E_p e obtém-se o autômato que explicita todas as sequências de eventos que levam à produção de dois produtos, Figura 5.13.

O autômato T_{E_q} apresenta todas as sequências que contribuem com a produção de dois produtos (unicamente com eventos de $(\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})$) e sequências que não contribuem para a produção de dois produtos (sequências onde o evento β_3 de quebra da máquina M_1 está presente). Neste exemplo só existe uma sequência $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ que contribui

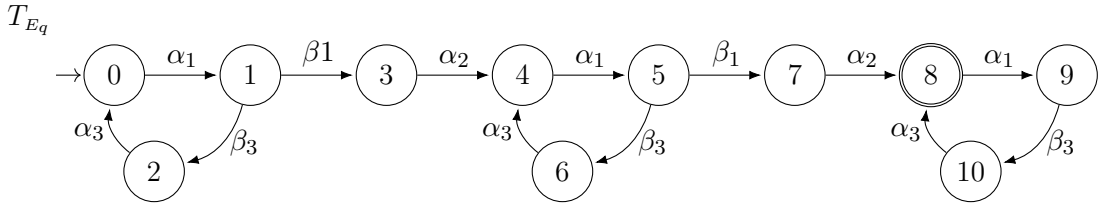


Figura 5.13: Autômato T_{E_q} que implementa $Trim(P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$.

com a produção de dois produtos:

$$s = \alpha_1\beta_1\alpha_2\alpha_1\beta_1\alpha_2.$$

A seguir a aplicação do algoritmo 1 para construir a linguagem $\mathcal{F}^{\uparrow}$ é realizada. Faz-se a projeção da sequência s sobre o alfabeto $\Sigma_{\sigma\uparrow}$ e assim obtém-se a sequência $s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s)$ (passo 1(i)). Obtém-se o autômato $G_{s_{o_t1}}$ e a linguagem $\mathcal{L}_m(G_{s_{o_t1}}) \leftarrow \{s_{o_t}\}$ da sequência s_{o_t} (passo 1(ii)), assim,

$$s = \alpha_1\beta_1\alpha_2\alpha_1\beta_1\alpha_2$$
$$s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s) = \alpha_1\alpha_2\alpha_1\alpha_2.$$

O autômato que implementa a sequência s_{o_t} é mostrado na Figura 5.14.

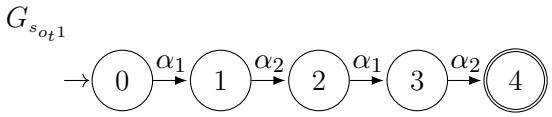


Figura 5.14: Autômato $G_{s_{o_t1}}$ que implementa a sequência s_{o_t} (passo 1(ii)).

O próximo passo (passo 2 do algoritmo) é acrescentar auto-laços do conjunto Σ^C_{fix} e Σ_{Ru} no autômato $G_{s_{o_t1}}$. O conjunto $\Sigma^C_{fix} = \{\sigma \in \Sigma_c | \forall t \in \Sigma_u \backslash \Sigma_{t\downarrow} \Rightarrow \exists \sigma \in \Sigma_c, \text{que retorna à produção}\}$ está composto de eventos controláveis tal que para todo evento t pertencente a $\Sigma_u \backslash \Sigma_{t\downarrow}$ existe pelo menos um evento σ pertencente ao conjunto de eventos controláveis que retorna um produto à linha de produção (Def.7 v). No exemplo, $\Sigma^C_{fix} = \{\alpha_3\}$ pois depois do evento β_3 o evento controlável que retorna à produção é α_3, como pode ser observado na Figura 5.13. No exemplo tem-se que $\Sigma_{Ru} = \{\beta_1, \beta_3\}$.

O passo (2) está subdividido em dois subpassos. O subpasso 2(i) tá definido como, $P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})\to\Sigma_{\sigma\uparrow}}(\mathcal{L}_m(G_{s_{o_t1}}))$ onde autolaços do conjunto Σ_{Ru} são acrescentados originado a

linguagem $\mathcal{L}_m(G_{s_{o_t2}})$. o autômato $G_{s_{o_t2}}$ é apresentado na Figura 5.15.

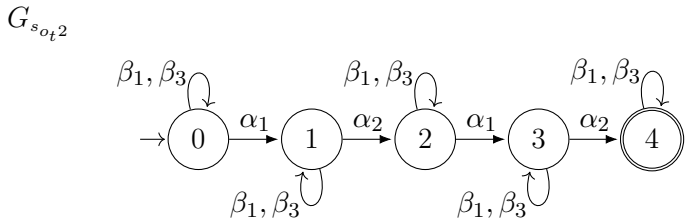


Figura 5.15: Autômato $G_{s_{o_t2}}$ que apresenta Autolaços de Σ_{Ru}.

O subpasso 2(ii) é subdividido em dois casos mas este exemplo executa o caso (ii.1), onde $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$, então faz-se, $P^{-1}_{(\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru} \cup \Sigma^C_{fix}) \to (\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru})}(\mathcal{L}_m(G_{s_{o_t2}})) = \mathcal{L}_m(G_{s_{o_t3}})$. Sendo assim, tem-se que o autômato $G_{s_{o_t3}}$ é:

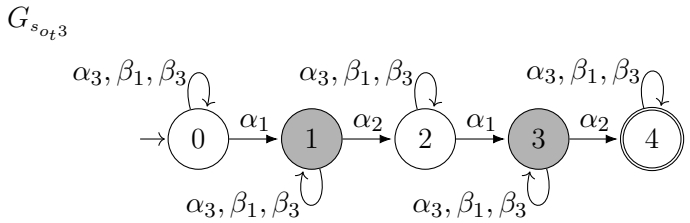


Figura 5.16: Autômato $G_{s_{o_t3}}$ que apresenta Autolaços de Σ^C_{fix} e Σ_{Ru}.

Por último, o passo (3) do algoritmo e o mais importante, pois consiste em identificar, em quais estados $q \in Q$ de $G_{s_{o_t3}}$ as máquinas $M_i \in M_{Ru}$ (M_{Ru} segundo Def.7) estão em estados ativos q_a (q_a segundo Def.7) pela aplicação da função rótulo $d(d'(q))$ e acrescentar auto-laços com os eventos $\sigma^{\uparrow}_{i_k}$. É importante determinar primeiramente neste passo o conjunto M^{Ru} e o rótulo $d(d'(q))$.

O conjunto $M^{Ru} = \{M_i | \Sigma_i \cap \Sigma_{Ru} \neq \emptyset\}$ no passo (3) está composto pelos subsistemas M_i onde os eventos de Σ_{Ru} estão definidos. Portanto, tem-se que $M^{Ru} = \{M_1\}$. Em seguida, determina-se o rótulo $d(d'(q))$ para saber em quais estados o subsistema M_1 está em estado ativo q_a, ou seja, onde o terceiro elemento da tupla de conjuntos possui o estado q^1_1, Tabela 5.2 terceira coluna. Porém, tem-se que os auto-laços acrescentados no autômato $G_{s_{o_t3}}$ serão do evento $\sigma^{\uparrow}_{1_1} = \alpha_1$.

Percebe-se no autômato da Figura 5.16 que os auto-laços do evento $\sigma^{\uparrow}_{1_1} = \alpha_1$ serão acrescentados nos estados $q = 1$ e $q = 3$ pois o primeiro elemento da tupla de conjuntos está

Tabela 5.2: Aplicação de $d'(q)$ e $d(d'(q))$.

q	$d'(q)$	$d(d'(q))$
0	$\{(q_0^1, q_0^2)\}$	$(\{q_0^1\}, \{q_0^2\})$
1	$\{(q_1^1, q_0^2)\}$,$\{(q_0^1, q_0^2)\}$	$(\{q_1^1, q_0^1\}, \{q_0^2\})$
2	$\{(q_0^1, q_1^2)\}$, $\{(q_0^1, q_0^2)\}$	$(\{q_0^1\}, \{q_1^2, q_0^2\})$
3	$\{(q_1^1, q_0^2)\}$, $\{(q_0^1, q_0^2)\}$	$(\{q_1^1, q_0^1\}, \{q_0^2\})$
4	$\{(q_0^1, q_1^2)\}$, $\{(q_0^1, q_0^2)\}$	$(\{q_0^1\}, \{q_1^2, q_0^2\})$

em estado ativo q_1^1 (estados em cinza da Tabela 5.2). Finalmente, o autômato resultante $G_{s_{o_t 4}}$ da execução do passo (3) do algoritmo é apresentado na Figura 5.17, onde a linguagem marcada do autômato $G_{s_{o_t 4}}$ é a linguagem $\mathcal{F}^{\uparrow}$, assim.

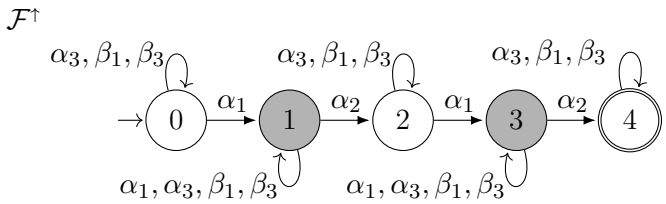


Figura 5.17: Autômato que implementa $\mathcal{F}^{\uparrow} \leftarrow \mathcal{L}_m(G_{s_{o_t 4}})$.

A seguir são apresentados um lema e um corolário que ajudam na demostração do teorema proposto.

Lema 1. *Seja uma cadeia $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$. Seja a linguagem $\mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ que é obtida a partir do Algoritmo 1, onde $\mathcal{F}^{\uparrow}$ é construída a partir de s. $\forall a \in S \subseteq \Sigma^*$ tal que $P_{\Sigma \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))}(a) = s$, então $P_{\Sigma \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))}(a) \in \mathcal{F}^{\uparrow}$.* △

Demostração. Como $s \in T_{E_q}$, pode-se dizer que $s \in \mathcal{L}_m(T_{E_q}) \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$. Por outro lado $s \in (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$. Assim:

$$s \in [(\Sigma_c \cap \Sigma_{\sigma\uparrow}) \cup (\Sigma_{Ru} \cap \Sigma_{t\downarrow})]^* = [\Sigma_{\sigma\uparrow} \cup (\Sigma_{Ru} \cap \Sigma_{t\downarrow})]^* \tag{5.1}$$

$\mathcal{F}^{\uparrow}$ é construído a partir de s em 3 passos. No primeiro passo:

$$P_{(\Sigma_{\sigma\uparrow} \cup (\Sigma_{Ru} \cap \Sigma_{t\downarrow})) \to \Sigma_{\sigma\uparrow}}(s) = s_{o_t}, \quad \text{passo } 1(i) \text{ do algoritmo.} \tag{5.2}$$

O segundo passo é subdividido em dois subpassos.

Subpasso i). $P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(s_{o_t}) = \mathcal{L}_m(G_{s_{o_t2}})$, passo 2($i$) do algoritmo. Substituindo s_{o_t} pela equação 5.2, então temos que, $P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(P_{(\Sigma_{\sigma^{\uparrow}}\cup(\Sigma_{Ru}\cap\Sigma_{t\downarrow}))\rightarrow\Sigma_{\sigma^{\uparrow}}}(s)) = \mathcal{L}_m(G_{s_{o_t2}})$. Portanto, tem-se que:

$$s \in P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(P_{(\Sigma_{\sigma^{\uparrow}}\cup(\Sigma_{Ru}\cap\Sigma_{t\downarrow}))\rightarrow\Sigma_{\sigma^{\uparrow}}}(s)) = \mathcal{L}_m(G_{s_{o_t2}}). \tag{5.3}$$

Já que $\Sigma_{Ru}\cap\Sigma_{t\downarrow}\subseteq\Sigma_{Ru}$. Assim:

$$s \in \mathcal{L}_m(G_{s_{o_t2}}). \tag{5.4}$$

O subpasso ii), é subdividido em dois casos:

O primeiro caso é se $\Sigma^{C}_{fix}\cap\Sigma_{\sigma^{\uparrow}} = \emptyset$, então faz-se:

$$P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\rightarrow(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})}(\mathcal{L}_m(G_{s_{o_t2}})) = \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.5}$$

Substituindo $P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(s_{o_t}) = \mathcal{L}_m(G_{s_{o_t2}})$ em (5.5), então temos que:

$$P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\rightarrow(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})}(P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(s_{o_t})) = \mathcal{L}_m(G_{s_{o_t3}}) \tag{5.6}$$

Usando a propriedade (1) das projeções extraída de Cassandras & Lafortune (2009), onde $t\in L\rightarrow t\in P^{-1}(L)$ em (5.6), então, é sempre verdade que:

$$s_{o_t} \in P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\rightarrow(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})}(P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(s_{o_t})) = \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.7}$$

$$s_{o_t} \in \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.8}$$

Substituindo (5.3) em (5.5), tem-se que:

$$s \in P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\rightarrow(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})}(P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(P_{(\Sigma_{\sigma^{\uparrow}}\cup(\Sigma_{Ru}\cap\Sigma_{t\downarrow}))\rightarrow\Sigma_{\sigma^{\uparrow}}}(s))) = \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.9}$$

O segundo caso é se $\Sigma^{C}_{fix}\cap\Sigma_{\sigma^{\uparrow}}\neq\emptyset$.

Seja o caso $\Sigma^{C}_{fix}\cap\Sigma_{\sigma^{\uparrow}} = \emptyset$ ou, o caso $\Sigma^{C}_{fix}\cap\Sigma_{\sigma^{\uparrow}}\neq\emptyset$, significa que há eventos de Σ^{C}_{fix} na cadeia s, passo 2(ii) do algoritmo. O algoritmo acrescenta autolaços de Σ^{C}_{fix}, sem tornar o

autômato não determinístico. O autômato $G_{s_{o_t3}}$ é obtido ao acrescentar autolaços de Σ_{fix}^C seja pelo caso $\Sigma_{fix}^C \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, $\Sigma_{fix}^C \cap \Sigma_{\sigma\uparrow} \neq \emptyset$ no autômato $G_{s_{o_t2}}$, o qual significa que a linguagem do autômato $G_{s_{o_t2}}$ continua contida na linguagem do autômato $G_{s_{o_t3}}$. Sendo verdade o anteriormente dito, então temos que:

$$\mathcal{L}_m(G_{s_{o_t2}}) \subseteq \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.10}$$

Como $s \in G_{s_{o_t2}}$ segundo a equação 5.4 e com a informação da equação 5.10, então tem-se que:

$$s \in \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.11}$$

Sendo assim (equação 5.11), a cadeia original s continua pertencendo à linguagem implementada pelo autômato resultante. No passo 3 do algoritmo, obtém-se $G_{s_{o_t4}}$ pela inclusão de autolaços de $\sigma_i^{\uparrow} \in \Sigma_c$ em alguns dos estados de $G_{s_{o_t3}}$. As modificações feitas em $G_{s_{o_t3}}$ não removem cadeias existentes. Portanto, a linguagem $\mathcal{L}_m(G_{s_{o_t4}})$, contém todas as cadeias da linguagem $\mathcal{L}_m(G_{s_{o_t3}})$. Sendo isso verdade, Tem-se que:

$$s \in \mathcal{L}_m(G_{s_{o_t3}}) \subseteq \mathcal{L}_m(G_{s_{o_t4}}) = \mathcal{F}^{\uparrow} \subseteq (\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru} \cup \Sigma_{fix}^C)^*. \tag{5.12}$$

Segundo o enunciado deste lema, sabemos que, $P_{\Sigma\to(\Sigma_{\sigma\uparrow}\cup(\Sigma_{t\downarrow}\cap\Sigma_{Ru}))}(a) = s$, e que ao substituir essa informação em 5.12, tem-se que:

$$P_{\Sigma\to(\Sigma_{\sigma\uparrow}\cup(\Sigma_{t\downarrow}\cap\Sigma_{Ru}))}(a) \in \mathcal{F}^{\uparrow} \subseteq (\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru} \cup \Sigma_{fix}^C)^* = (\Sigma_c \cup \Sigma_{Ru})^*. \tag{5.13}$$

Pois $\Sigma_c = \Sigma_{\sigma\uparrow} \cup \Sigma_{fix}^C$, o que demostra o lema. □

Com base no resultado do Lema 1, propõe-se o Corolário 1.

Corolário 1. *Seja uma cadeia $s \in \mathcal{L}_m(T_{Eq}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$. Seja a linguagem $\mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ obtida a partir de s por meio do Algoritmo 1, onde $\mathcal{F}^{\uparrow}$ é construída a partir de s. $\forall a \in S \subseteq \Sigma^*$ tal que $P_{\Sigma\to(\Sigma_{\sigma\uparrow}\cup(\Sigma_{t\downarrow}\cap\Sigma_{Ru}))}(a) = s$, então $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(a) \in \mathcal{F}^{\uparrow}$.* ▲

Demonstração. O lema 1 mostrou que $P_{\Sigma\to(\Sigma_{\sigma\uparrow}\cup(\Sigma_{t\downarrow}\cap\Sigma_{Ru}))}(a) \in \mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ (equação

5.13) e como $(\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})) \subseteq (\Sigma_c \cup \Sigma_{Ru})$, então, rescreve-se (5.13), assim:

$$P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))}(P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a)) = s \in \mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*. \tag{5.14}$$

A partir de (5.14), conclui-se que $s \in (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})^*$. Também de (5.14), tem-se que $s \in \mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$. Assim, pode-se concluir que:

$$s \in P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \subseteq (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))^*. \tag{5.15}$$

Usando (5.14) e (5.15) obtém-se:

$$P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))}(P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a)) \in P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow}). \tag{5.16}$$

Aplica-se projeção inversa em (5.16):

$$\begin{aligned} P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a))] \subseteq \\ P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow})]. \end{aligned} \tag{5.17}$$

Da equação (5.17) e usando a propriedade (1) das projeções extraída de Cassandras & Lafortune (2009), onde $L \in P^{-1}(P(L))$, então, é verdade que:

$$\begin{aligned} P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a) &\in P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a))], \\ P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a) &\in P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow})]. \end{aligned} \tag{5.18}$$

A operação realizada entre (5.14) e (5.15) consistiu na eliminação de autolaços na linguagem $\mathcal{F}^{\uparrow}$. Os autolaços removidos são realocados com a operação de projeção inversa e, neste caso:

$$P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow})] = \mathcal{F}^{\uparrow}. \tag{5.19}$$

Então, de (5.18) e (5.19), tem-se que:

$$P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a) \in \mathcal{F}^{\uparrow}. \tag{5.20}$$

□

A equação 5.20 demostra o corolário. Com base no resultado do Corolário 1, propõe-se o Corolário 1.

Proposição 1. *Seja S o comportamento em malha fechada. Seja uma cadeia $s \in \mathcal{L}_m(T_{Eq}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$. Seja a linguagem $\mathcal{F}^\uparrow \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ obtida a partir de s por meio do Algoritmo 1, então, $\overline{\mathrm{S}} \cap \overline{P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow)} = \overline{\mathrm{P}^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap \mathrm{S}}$.*

♦

Demonstração. É sempre verdade que $\overline{\mathrm{S}} \cap \overline{P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow)} \subseteq \overline{\mathrm{P}^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap \mathrm{S}}$. Para mostrar a igualdade basta demostrar que:

$$P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\overline{\mathcal{F}^\uparrow}) \cap \overline{\mathrm{S}} \subseteq \overline{\mathrm{P}^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap \mathrm{S}} \tag{5.21}$$

Seja $a_1 \in \overline{\mathrm{S}} \cap \overline{P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow)}$, tem-se que $a_1 \in \overline{\mathrm{S}}$. Portanto, $\exists u \in \Sigma^*$ tal que $a_1 u \in S$. Seja $a = a_1 u \in S$, usando o Corolário 1:

$$P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(a) \in \mathcal{F}^\uparrow \tag{5.22}$$

Assim, $a \in \overline{P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow)}$, portanto:

$$a \in S \cap P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \tag{5.23}$$

Então, demostra-se que::

$$a_1 \in \overline{\mathrm{P}^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap \mathrm{S}}. \tag{5.24}$$

□

A equação 5.24 demonstra a Proposição. Após apresentar o procedimento para obtenção da linguagem $\mathcal{F}^\uparrow$, o Lema 1, o Corolário 1 e a Proposição 1, enuncia-se o Teorema 2, que estende o teorema original de Vilela & Pena (2016).

Teorema 2. *Seja G um sistema, S a máxima sublinguagem controlável contida na linguagem desejada de K e $P : \Sigma^* \to (\Sigma_c \cup \Sigma_{Ru})^*$ a projeção natural com a propriedade do observador. Para toda linguagem $\mathcal{F}^\uparrow \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ com as características do Algoritmo 1, e $A = P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap S$, como $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ tem a propriedade do observador, então A será controlável com relação a $\mathcal{L}(G)$.* ■

Demostração. Para mostrar que A é controlável em relação a $\mathcal{L}(G)$, $\forall a \in \overline{\mathrm{A}}$ e $\forall \sigma \in \Sigma_u$ com $a\sigma \in \mathcal{L}(G)$, é preciso mostrar que $a\sigma \in \overline{\mathrm{A}}$.

Seja o autômato $T_{E_q} = Trim(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$, onde E_p é segundo a Definição 8. Uma cadeia $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ é escolhida para construir a linguagem $\mathcal{F}^{\uparrow}$, onde $s \in P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)$. Seja uma sequência tal que $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a') = s$. Seja $a \leq a'$, então:

$$a \in \overline{\mathrm{S}}. \tag{5.25}$$

Usando o Corolário 1, $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a') = s$, então $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a) \in \overline{\mathcal{F}^{\uparrow}}$. Portanto, ao aplicar a operação de projeção inversa nos dois lados, tem-se que:

$$\begin{aligned} P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a) &\in \overline{\mathcal{F}^{\uparrow}} \\ P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a)) &\subseteq \overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})} \\ a \in P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a)) &\subseteq \overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})}. \end{aligned} \tag{5.26}$$

Portanto, de (5.26) pode-se concluir que:

$$a \in \overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})}. \tag{5.27}$$

De (5.25) e (5.27), conclui-se que:

$$a \in \overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})} \cap \overline{\mathrm{S}}. \tag{5.28}$$

Como neste caso $\overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})} \cap \overline{\mathrm{S}} = \overline{\mathrm{P}^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{\mathrm{Ru}})}(\mathcal{F}^{\uparrow}) \cap \mathrm{S}} = \overline{\mathrm{A}}$ segundo a Proposição 1, então:

$$a \in \overline{\mathrm{A}} \subseteq \Sigma^*. \tag{5.29}$$

O próximo passo é mostrar que $a\sigma \in \overline{\mathrm{A}}$. Como S é controlável, tem-se que:

$$\forall a \in \overline{\mathrm{S}}, \forall \sigma \in \Sigma_u, a\sigma \in \mathcal{L}(G) \Rightarrow a\sigma \in \overline{\mathrm{S}}. \tag{5.30}$$

De (5.30), conclui-se que $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma) \in P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\overline{S})$. Como $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)$ possui a

propriedade do observador segundo a Definição 1, então $\exists c \in \Sigma^*$ tal que:

$$P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma c) = P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma)b \text{ e}$$

$$a\sigma c \in S. \tag{5.31}$$

Pelo Corolário 1 e (5.31), tem-se que:

$$P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma c) \in \mathcal{F}^{\uparrow}. \tag{5.32}$$

Aplica-se a operação de projeção inversa em (5.32), assim:

$$P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma c)) \quad \subseteq \quad P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \text{ onde,}$$

$$a\sigma c \in P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma c)) \subseteq P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}). \tag{5.33}$$

Portanto, de (5.33) pode-se concluir que:

$$a\sigma c \in P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}). \tag{5.34}$$

De (5.31) e (5.34) conclui-se que:

$$a\sigma c \in P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S. \tag{5.35}$$

A partir de $A = P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S$ e de (5.35), conclui-se que $a\sigma c \in A$, logo:

$$a\sigma \in \overline{\mathrm{A}}. \tag{5.36}$$

□

Com (5.36), prova-se que a linguagem $A = P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S$ é controlável com relação à $\mathcal{L}(G)$, então, conclui-se que:

$$\forall a \in \overline{\mathrm{A}}, \forall \sigma \in \Sigma_u, a\sigma \in \mathcal{L}(G) \Rightarrow a\sigma \in \overline{\mathrm{A}}. \tag{5.37}$$

O resultado apresentado no Teorema 2 permite obter uma linguagem A controlável com

relação à $\mathcal{L}(G)$ a partir de uma linguagem $\mathcal{F}^{\uparrow}$. Assim, escolhe-se uma cadeia s e obtém-se, a partir dela, A controlável:

$$s \in \mathcal{L}_m(T_{Eq}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^* \subseteq P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S) \implies A = P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S \text{ é controlável.}$$

Esta ideia é ilustrada na Figura 5.18. A figura ilustra simplificadamente o processo de obtenção da linguagem recuperada A.

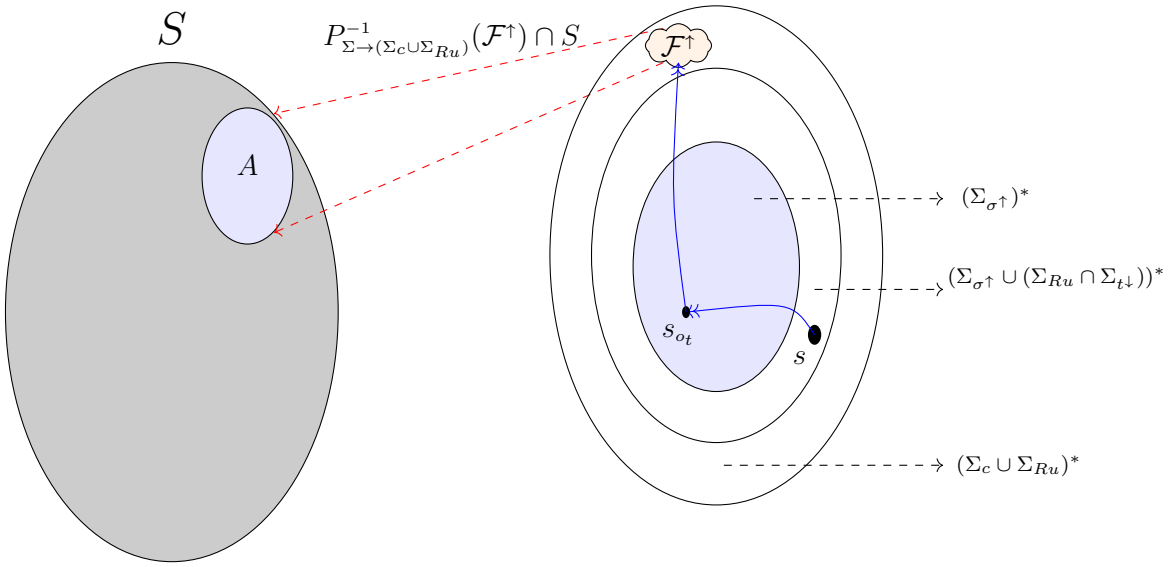


Figura 5.18: Ideia conceitual para construir a linguagem A.

A Figura 5.18 mostra o espaço de busca do supervisor (S) e o espaço de busca da projeção $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p$ que possui a propriedade do observador. Uma sequência s é escolhida na projeção (ponto preto s, elipse externa da direita) e depois projetada no alfabeto $\Sigma_{\sigma\uparrow}$, assim $s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s)$ (ponto preto s_{o_t} elipse interna da direita). A linguagem $\mathcal{F}^{\uparrow}$ é obtida a partir da sequência s_{o_t} aplicando o Algoritmo 1. Portanto, com a linguagem $\mathcal{F}^{\uparrow}$, é possível recuperar uma linguagem A que está contida na linguagem que implementa o comportamento em malha fechada (elipse da esquerda).

A próxima seção mostra um estudo de caso da aplicação do resultado principal deste trabalho.

5.4 Estudo de Caso

Seja o sistema apresentado na Seção 4.1 (Exemplo 6), que possui 3 equipamentos, M_1, M_2, M_{TU} e $G = M_1||M_2||M_{TU}$ (Fig 5.19, Fig 5.20 e Fig 5.21). Cada um dos sistemas M_i atende às restrições da Definição 7, ou seja, alfabetos assíncronos (Def.7 (i,ii)), todos os eventos nos estados iniciais são controláveis (Def.7 (iii.a)), há os seguintes ciclos de trabalho: $w_{1_1} = \alpha_1\beta_1$, $w_{2_1} = \alpha_2\beta_2$ e $w_{TU_1} = \alpha_3\beta_3$, um em cada máquina (Def.7 (iii.b)). Pode-se ainda definir os conjuntos de eventos $\Sigma_{\sigma\uparrow} = \{\sigma_1, \sigma_2, \sigma_3\}$, $\Sigma_{t\downarrow} = \{\beta_1, \beta_2, \beta_3\}$ e $\Sigma_C^{fix} = \{\alpha_2\}$ (Def.7 (iv,v)).

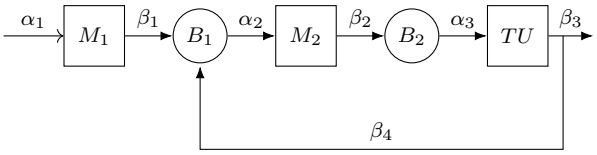
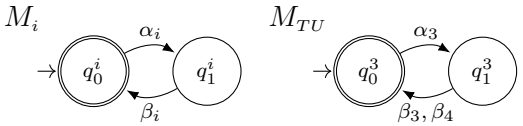
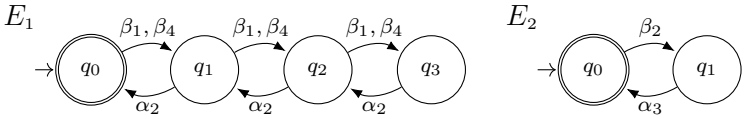


Figura 5.19: Sistema de manufatura com retrabalho.



(a) M_i, $i = \{1, 2\}$. (b) M_{TU}.

Figura 5.20: Modelos das máquinas.



(a) Especificação do *Buffer* B_1. (b) Especificação do *Buffer* B_2.

Figura 5.21: Modelos das especificações.

Suponha que deseja-se realizar a produção de um lote de tamanho 2, no menor tempo possível. Na solução deste problema será utilizada a abstração proposta neste trabalho.

O primeiro passo é obter o comportamento em malha fechada do sistema $S = SupC(K, G)$ (28 estados e 65 transições) e a abstração do mesmo com a propriedade do observador. Como foi dito anteriormente no Exemplo 6, a abstração do comportamento em malha fechada desse sistema sobre os eventos controláveis ($P_{\Sigma\to\Sigma_c}(S)$) não possui a propriedade do observador. Porém, faz-se necessária a aplicação do algoritmo de busca da propriedade do

observador, o qual retorna que todas as instâncias dos eventos β_3 e β_4 devem ser incluídas na projeção natural, ou seja, o conjunto $\Sigma_{Ru} = \{\beta_3, \beta_4\}$ fará parte da projeção, de tal forma que $P : \Sigma^* \to (\Sigma_c \cup \Sigma_{Ru})^*$ é PO-abstração.

O autômato que implementa $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ é apresentado na Figura 5.22 e os estados em cinza são aqueles que possuem os dois eventos $\Sigma_{Ru} = \{\beta_3, \beta_4\}$ possíveis. Para explicitar

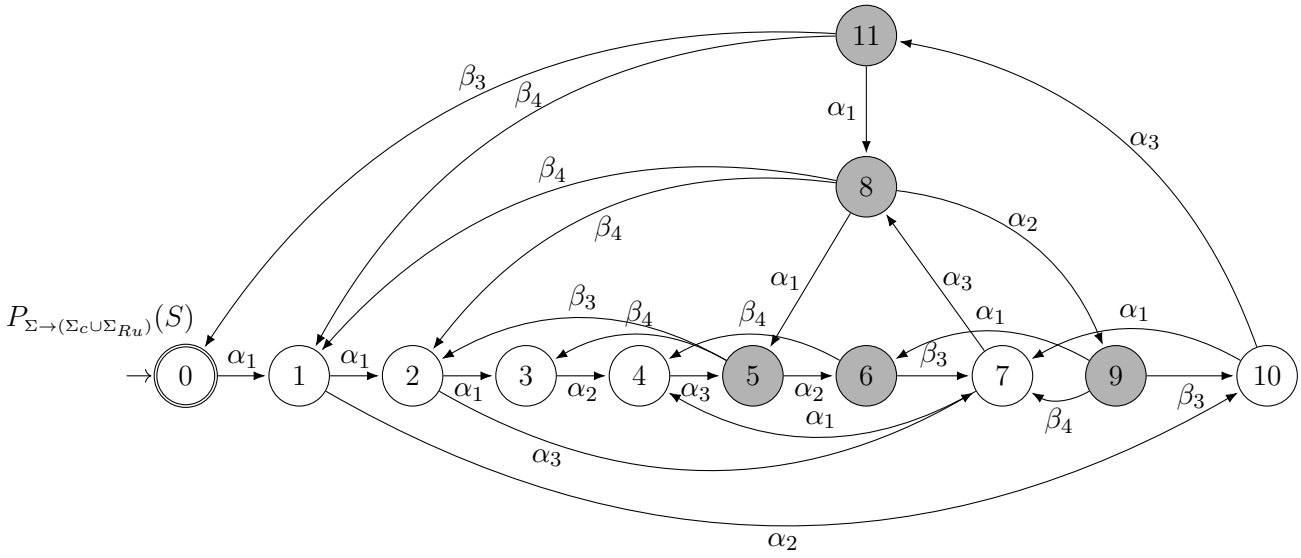


Figura 5.22: Abstração do supervisor com PO.

as sequências de eventos que levam à produção de um lote de tamanho 2, utiliza-se uma especificação de produção conforme a Definição 8, mostrada na Figura 5.23.

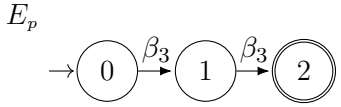


Figura 5.23: Especificação de produção.

A inclusão dos eventos não controláveis faz com que ciclos possam se formar (no caso, quando ocorre o evento β_4, que retorna o produto para a máquina M_2). Como estes eventos, β_3 e β_4, são não-controláveis, não é possível prevenir que eles aconteçam e deve-se lidar com as possibilidades. Considerando o equipamento M_{TU}, há um ciclo $w_{TU} = \alpha_3\beta_3$, $\sigma^{\uparrow}_{TU} = \alpha_3$ e $t^{\downarrow}_{TU} = \beta_3$.

Com os autômatos $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ e E_p, o próximo passo é aplicar o Algoritmo 1 que calcula a linguagem $\mathcal{F}^{\uparrow}$ que será usada no resultado principal deste trabalho (Lema **??**).

Primeiro realiza-se a composição paralela entre $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ e a especificação de produção E_p e faz *Trim* sobre este resultado. Assim, obtém-se o autômato $T_{E_q} = Trim(P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$ mostrado na Figura 5.24.

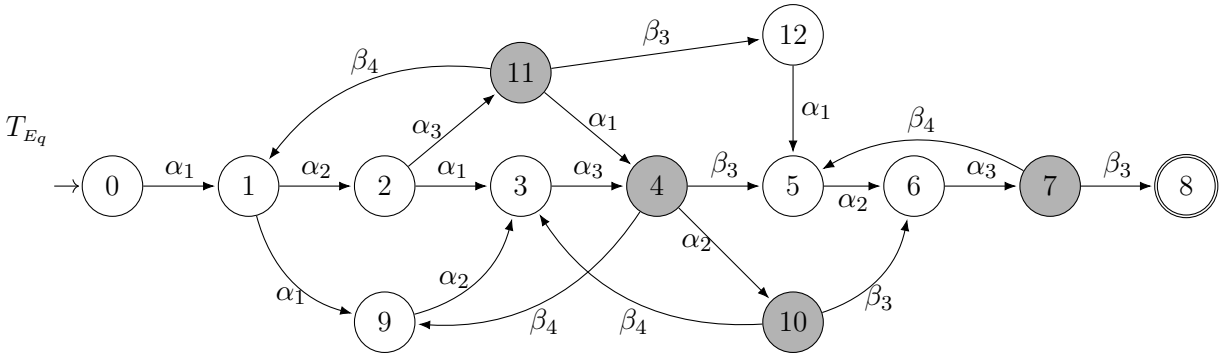


Figura 5.24: Autômato que implementa $T_{E_q} = Trim(P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$.

O autômato da Figura 5.24 explicita todas as sequências $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ que levam à produção de dois produtos. Neste caso tem-se as sequências:

$$s_1 = \alpha_1\alpha_2\alpha_3\alpha_1\beta_3\alpha_2\alpha_3\beta_3$$

$$s_2 = \alpha_1\alpha_2\alpha_3\beta_3\alpha_1\alpha_2\alpha_3\beta_3$$

$$s_3 = \alpha_1\alpha_2\alpha_1\alpha_3\beta_3\alpha_2\alpha_3\beta_3$$

$$s_4 = \alpha_1\alpha_1\alpha_2\alpha_3\beta_3\alpha_2\alpha_3\beta_3$$

$$s_5 = \alpha_1\alpha_1\alpha_2\alpha_3\alpha_2\beta_3\alpha_3\beta_3$$

$$s_6 = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\beta_3\alpha_3\beta_3$$

$$s_7 = \alpha_1\alpha_2\alpha_3\alpha_1\alpha_2\beta_3\alpha_3\beta_3$$

Escolhe-se uma das sequências anteriores $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$, por exemplo a sequência s_{ot6}. Assim,

$$s_6 = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\beta_3\alpha_3\beta_3$$

$$s_{ot6} = P_{\Sigma_s\to\Sigma_c}(s_6) = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\alpha_3.$$

O autômato que implementa a sequência s_{ot6} é mostrado na Figura 5.25. O próximo passo

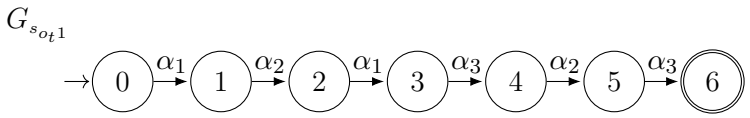


Figura 5.25: Autômato $G_{s_{ot}1}$ que implementa a sequência s_{ot6} (passo 1(ii)).

(passo 2(i, ii) do algoritmo) é acrescentar auto-laços do conjunto Σ^C_{fix} e Σ_{Ru} no autômato $G_{s_{ot}1}$. O conjunto $\Sigma^C_{fix} = \{\sigma \in \Sigma_c | \forall t \in \Sigma_u \backslash \Sigma_{t\downarrow} \Rightarrow \exists \sigma \in \Sigma_c$,que retorna à produção$\}$ está composto de eventos controláveis tal que para todo evento t pertencente a $\Sigma_u \backslash \Sigma_{t\downarrow}$ existe pelo menos um evento σ pertencente ao conjunto de eventos controláveis que retorna um produto à linha de produção (Def.7 v). No exemplo, $\Sigma^C_{fix} = \{\alpha_2\}$ pois depois do evento β_4 o evento controlável que retorna à produção é α_3, como pode ser observado na Figura 5.19. No exemplo $\Sigma_{Ru} = \{\beta_3, \beta_4\}$.

O passo 2(i) acrescenta autolaços de Σ_{Ru} em $G_{s_{ot}1}$ e obtém-se o autômato $G_{s_{ot}2}$ assim, $\mathcal{L}_m(G_{s_{ot}2}) = P^{-1}_{(\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru}) \to \Sigma_{\sigma\uparrow}}(\mathcal{L}_m(G_{s_{ot}1}))$. O passo 2($ii$) está dividido em dois casos. Seja o caso $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, o caso $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$, significa que há eventos de Σ^C_{fix} na cadeia s. Este exemplo é do tipo caso 2(ii.2), quando $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$ pois existe uma interseção que é $\Sigma^C_{fix} = \{\alpha_2\}$. O algoritmo acrescenta autolaços de Σ^C_{fix}, sem tornar o autômato não determinístico. O autômato $G_{s_{ot}3}$ é obtido ao acrescentar autolaços de $\Sigma^C_{fix} = \{\alpha_2\}$ seja pelo caso $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$ no autômato $G_{s_{ot}2}$, o qual significa que a linguagem do autômato $G_{s_{ot}2}$ continua contida na linguagem do autômato $G_{s_{ot}3}$. Sendo assim, tem-se que o autômato $G_{s_{ot}3}$ é:

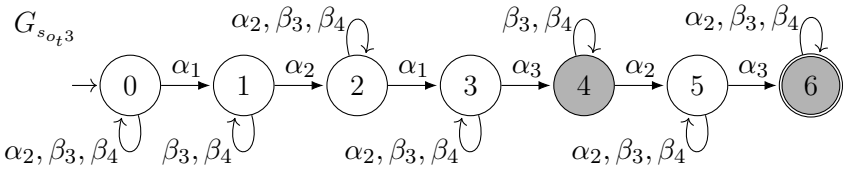


Figura 5.26: Auto-laços de Σ^C_{fix} e Σ_{Ru}, passo (2) do algoritmo.

Pode-se observar que $\alpha_2 \in \Sigma^C_{fix}$ não foi posto como auto-laço nos estados 1 e 4 para não gerar não determinismo (condição segundo o passo 2(ii.2) do algoritmo).

Por último, o passo (3) do algoritmo e o mais importante, consiste em identificar, quais estados $q \in Q$ de $Q_{s_{ot}3}$ as máquinas $M_i \in M_{Ru}$ (M_{Ru} segundo Def.7(vi)) estão em estados ativos q_a (q_a segundo Def.7) segundo pela aplicação da função rótulo $d(d'(q))$ e acrescentar auto-laços com os eventos $\sigma^{\uparrow}_{i_k}$. É importante determinar primeiramente neste passo o

conjunto M^{Ru} e o rótulo $d(d'(q))$.

O conjunto $M^{Ru} = \{M_i | \Sigma_i \cap \Sigma_{Ru} \neq \emptyset\}$ está composto pelos subsistemas M_i onde os eventos de Σ_{Ru} estão definidos. Portanto, tem-se que $M^{Ru} = \{M_{TU}\}$. Em seguida, determina-se o rótulo $d(d'(q))$ para saber aonde o subsistema M_{TU} está em estado ativo q_a, ou seja, onde o terceiro elemento da tupla de conjuntos possui o estado q_1^3. Porém, tem-se que os auto-laços acrescentados no autômato $Q_{s_{o_t 3}}$ serão do evento $\sigma_{3_1}^{\uparrow} = \alpha_3$. A aplicação de $d(d'(q))$ é detalhada na Tabela 5.3.

Tabela 5.3: Aplicação de $d'(q)$ e $d(d'(q))$.

q	$d'(q)$	$d(d'(q))$
0	$\{(q_0^1, q_0^2, q_0^3)\}$	$(\{q_0^1\}, \{q_0^2\}, \{q_0^3\})$
1	$\{(q_1^1, q_0^2, q_0^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_1^1, q_0^1\}, \{q_0^2\}, \{q_0^3\})$
2	$\{(q_0^1, q_1^2, q_0^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_0^1\}, \{q_1^2, q_0^2\}, \{q_0^3\})$
3	$\{(q_1^1, q_0^2, q_0^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_1^1, q_0^1\}, \{q_0^2\}, \{q_0^3\})$
4	$\{(q_0^1, q_0^2, q_1^3)\}$	$(\{q_0^1\}, \{q_0^2\}, \{q_1^3\})$
5	$\{(q_0^1, q_1^2, q_1^3), (q_0^1, q_1^2, q_0^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_0^1\}, \{q_1^2, q_0^2\}, \{q_1^3, q_0^3\})$
6	$\{(q_0^1, q_0^2, q_1^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_0^1\}, \{q_0^2\}, \{q_1^3, q_0^3\})$

Percebe-se pela Tabela 5.3, que os estados $4, 5$ e 6 possuem na tupla o estado $\{q_1^3\}$. Como no estado $q = 5$ o evento α_3 já está factível originalmente, então coloca-se auto-laços apenas nos estados 4 e 6 (em cinza). Finalmente, o autômato $G_{s_{o_t 4}}$ resultante do passo (3) do algoritmo é apresentado na Figura 5.27 onde a linguagem marcada de $G_{s_{o_t 4}}$ é a linguagem $\mathcal{F}^{\uparrow}$.

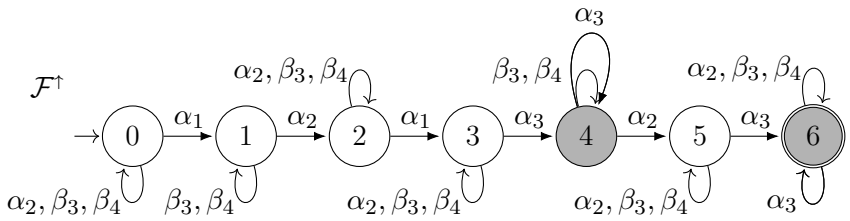


Figura 5.27: Autômato que implementa $\mathcal{F}^{\uparrow} \leftarrow \mathcal{L}_m(G_{s_{o_t 4}})$.

O Lema **??** diz que a linguagem A, que é uma sublinguagem do comportamento em malha fechada que projeta para a linguagem $\mathcal{F}^{\uparrow}$, é controlável. Isso é verdade para qualquer

linguagem $\mathcal{F}^{\uparrow}$ que projeta para uma sequência $s \in \mathcal{L}_m(T_{Eq}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ que o planejador da Figura 2.2 possa escolher.

Constrói-se o autômato $A = P^{-1}_{\Sigma\rightarrow(\Sigma_c \cup \Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S$ que contém 39 estados e 62 transições. A linguagem A obtida é uma sublinguagem do comportamento em malha fechada que respeita as restrições e leva à produção de um lote de tamanho 2. Como o controlador só pode atuar sobre os eventos controláveis, então é essencial que todos os eventos que ele deve desabilitar sejam controláveis. Isto equivale a dizer que a linguagem A é controlável com relação à planta G.

A linguagem $\mathcal{F}^{\uparrow}$ deve ser implementada no sistema visando executar a cadeia otimizada $s_6 = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\beta_3\alpha_3\beta_3$. Não se pode implementar somente s_6, pois após de cada ocorrência do evento α_3, existe a possibilidade de ocorrer β_4 no lugar de β_3, ambos não-controláveis. Caso ocorra β_3, segue-se a produção com o restante da cadeia s_6, mas se ocorrer β_4, é preciso que o sistema seja capaz de lidar com a situação. A introdução de $\Sigma_C^{fix} = \{\alpha_2\}$ e dos auto-laços com $\sigma^{\uparrow}_{3_1} = \alpha_3$, garantem o retorno do sistema para a situação de produção e, portanto, garantem a controlabilidade da linguagem resultante A.

6

Conclusões

Este trabalho estabelece as condições necessárias para aplicação de abstrações sobre supervisores pela síntese monolítica para a busca das sequências de solução no problema de planejamento. O propósito principal dessa abstração sobre o comportamento em malha fechada é reduzir o espaço de busca da solução de problemas de planejamento.

Apresenta-se neste trabalho uma extensão do Teorema de Vilela & Pena (2016) para o caso em que a linguagem que implementa a projeção para o conjunto de eventos controláveis do comportamento em malha fechada não possui a propriedade do observador. A inexistência da PO demonstra a importância para a produção de alguns eventos não controláveis. Desta forma, estende-se a abstração para incluir tais eventos. Esta inclusão transforma a cadeia otimizada em uma linguagem $\mathcal{F}^{\uparrow}$ a ser implementada no sistema.

Para obtenção de tal linguagem, $\mathcal{F}^{\uparrow}$, parte-se de uma cadeia otimizada, escolhida entre aquelas que produzem um número de produtos, pressupondo que eventos não controláveis

específicos vão ocorrer. Em seguida, constrói-se a linguagem que garante que mesmo que os não-controláveis previstos não ocorram, é possível finalizar a produção.

A extensão proposta neste trabalho se justifica pelo fato de que existem sistemas de manufatura que por natureza a projeção natural do seu comportamento em malha fechada não apresentam a propriedade do observador, e onde o seu espaço de estados do comportamento em malha fechada é muito grande (explosão de estados). Assim, é legítimo propor a utilização de abstrações estendendo a projeção natural no conjunto de eventos controláveis e um subconjunto de eventos não controláveis sobre o supervisor pela síntese monolítica.

A extensão apresentada neste trabalho, no entanto, não é a solução final para o problema de planejamento, mas torna o problema de otimização mais fácil de ser resolvido.

Uma proposta de continuidade consiste na implementação de um algoritmo que busca a cadeia s que será usada para obter $\mathcal{F}^{\uparrow}$ e a integração desta solução com um protótipo de sistema de manufatura disponível no LACSED/UFMG. Adicionalmente, deseja-se unir resultados obtidos sobre distinção de eventos em sistemas com retrabalho (Cury et al., 2015) com o presente trabalho.

Referências Bibliográficas

Abdeddaim, Y. & Maler, O. (2001). Job-shop scheduling using timed automata? In *International Conference on Computer Aided Verification* (pp. 478–492).: Springer.

Alves, L. V., Bravo, H. J., Pena, P. N., & Takahashi, R. H. (2016). Planning on discrete events systems: A logical approach. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)* (pp. 1055–1060).: IEEE.

Alves, L. V., Martins, L. R., & Pena, P. N. (2017). Ultrades-a library for modeling, analysis and control of discrete event systems. *IFAC-PapersOnLine*, 50(1), 5831–5836.

Alves, M. R. d. C. & Pena, P. N. (2017). Abstrações de supervisores localmente modulares para aplicação na solução de problemas de planejamento. *XIII Simpósio Brasileiro de Automação Inteligente, Porto Alegre-RS*, (pp. 699–704).

Arisha, A., Young, P., & El Baradie, M. (2001). Job shop scheduling problem: an overview.

Ashour, S. (1970). A branch-and-bound algorithm for flow shop scheduling problems. *AIIE Transactions*, 2(2), 172–176.

Baker, K. R. & Trietsch, D. (2013). *Principles of sequencing and scheduling*. John Wiley & Sons.

Bansal, S. (1979). On lower bounds in permutation flow shop problems. *International Journal of Production Research*, 17(4), 411–418.

Bożejko, W., Gnatowski, A., Pempera, J., & Wodecki, M. (2017). Parallel tabu search

for the cyclic job shop scheduling problem. *Computers & Industrial Engineering*, 113, 512–524.

Bravo, H. J., da Cunha, A. E. C., Pena, P. N., Malik, R., & Cury, J. E. (2012). Generalised verification of the observer property in discrete event systems. *IFAC Proceedings Volumes*, 45(29), 337–342.

Bravo, H. J., Pena, P. N., Alves, L. V., & Takahashi, R. H. (2018). Factorization-based approach for computing a minimum makespan controllable sublanguage. *IFAC-PapersOnLine*, 51(7), 19–24.

Bravo, H. J., Pena, P. N., da Cunha, A. E. C., Malik, R., & Cury, J. E. (2014). Generalised search for the observer property in discrete event systems1. *IFAC Proceedings Volumes*, 47(2), 350–355.

Cai, K. & Wonham, W. M. (2013). Supervisor localization of discrete-event systems based on state tree structures. *IEEE Transactions on Automatic Control*, 59(5), 1329–1335.

Canbolat, Y. B. & Gundogar, E. (2004). Fuzzy priority rule for job shop scheduling. *Journal of intelligent manufacturing*, 15(4), 527–533.

Cassandras, C. G. & Lafortune, S. (2009). *Introduction to discrete event systems*. Springer Science & Business Media.

Cavalca, D. L. & Fernandes, R. A. (2018). Hybrid particle swarm algorithm applied to flexible job-shop problem. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–6).: IEEE.

Chen, E. & Lafortune, S. (1991). On nonconflicting languages that arise in supervisory control of discrete event systems. *Systems & Control Letters*, 17(2), 105–113.

Costa, T. A., Pena, P. N., & Takahashi, R. H. (2018). Sco-concat: a solution to a planning problem in flexible manufacturing systems using supervisory control theory and optimization techniques. *Journal of Control, Automation and Electrical Systems*, 29(4), 500–511.

Cunha, A. E. C. & Cury, J. E. R. (2007). Hierarchical supervisory control based on discrete event systems with flexible marking. *IEEE Transactions on Automatic Control*, 52(12), 2242–2253.

Cury, J. E., de Queiroz, M. H., Bouzon, G., & Teixeira, M. (2015). Supervisory control of discrete event systems with distinguishers. *Automatica*, 56, 93–104.

Cury, J. E., Torrico, C. R., & da Cunha, A. E. (2004). Supervisory control of discrete event systems with flexible marking. *European Journal of Control*, 10(1), 47–60.

Dauzère-Pérès, S. & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70, 281–306.

de Queiroz, M. H., Cury, J. E., & Wonham, W. M. (2005). Multitasking supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 15(4), 375–395.

Della Croce, F., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1), 15–24.

Dell'Amico, M. & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations research*, 41(3), 231–252.

Fabre, E. & Jezequel, L. (2009). Distributed optimal planning: an approach by weighted automata calculus. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference* (pp. 211–216).: IEEE.

Feng, L. & Wonham, W. M. (2006). Computationally efficient supervisor design: Abstraction and modularity. In *2006 8th International Workshop on Discrete Event Systems* (pp. 3–8).: IEEE.

Feng, L. & Wonham, W. M. (2008). Supervisory control architecture for discrete-event systems. *IEEE Transactions on Automatic Control*, 53(6), 1449–1461.

Flordal, H. & Malik, R. (2006). Modular nonblocking verification using conflict equivalence. In *2006 8th International Workshop on Discrete Event Systems* (pp. 100–106).: IEEE.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: theory and practice*. Elsevier.

Gohari, P. & Wonham, W. M. (2000). On the complexity of supervisory control design in the rw framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 30(5), 643–652.

Gonzalez, T. & Sahni, S. (1978). Flowshop and jobshop schedules: complexity and approximation. *Operations research*, 26(1), 36–52.

Gupta, J. (1970). M-stage flowshop scheduling by branch and bound. *Opsearch*, 7(1), 37–43.

Hill, R., Cury, J., De Queiroz, M., & Tilbury, D. (2008). Modular requirements for hierarchical interface-based supervisory control with multiple levels. In *2008 American Control Conference* (pp. 483–490).: IEEE.

Hill, R. C., Cury, J. E. R., de Queiroz, M. H., Tilbury, D. M., & Lafortune, S. (2010). Multi-level hierarchical interface-based supervisory control. *Automatica*, 46(7), 1152–1164.

Ignall, E. & Schrage, L. (1965). Application of the branch and bound technique to some flow-shop scheduling problems. *Operations research*, 13(3), 400–412.

Jovane, F., Yoshikawa, H., Alting, L., Boer, C. R., Westkamper, E., Williams, D., Tseng, M., Seliger, G., & Paci, A. (2008). The incoming global technological and industrial revolution towards competitive sustainable manufacturing. *CIRP annals*, 57(2), 641–659.

Kobetski, A. & Fabian, M. (2006). Scheduling of discrete event systems using mixed integer linear programming. In *2006 8th International Workshop on Discrete Event Systems* (pp. 76–81).: IEEE.

Komenda, J. & van Schuppen, J. H. (2005). Modular antipermissive control of discrete-event systems. *IFAC Proceedings Volumes*, 38(1), 97–102.

Lafortune, S. (2019). Discrete event systems: Modeling, observation, and control. *Annual Review of Control, Robotics, and Autonomous Systems*.

Leduc, R. J., Lawford, M., & Wonham, W. M. (2005). Hierarchical interface-based supervisory control-part ii: parallel case. *IEEE Transactions on Automatic Control*, 50(9), 1336–1348.

Liu, S. Q., Kozan, E., Masoud, M., Zhang, Y., & Chan, F. T. (2018). Job shop scheduling with a combination of four buffering constraints. *International Journal of Production Research*, 56(9), 3274–3293.

Malik, R. (2004). On the set of certain conflicts of a given language. *Proceedings of 7th Workshop on Discrete Event Systems (WODES'04)*, (pp. 277–282).

Malik, R. & Pena, P. N. (2018). Optimal task scheduling in a flexible manufacturing system using model checking. *IFAC-PapersOnLine*, 51(7), 230–235.

Malik, R., Streader, D., & Reeves, S. (2004). Fair testing revisited: A process-algebraic characterisation of conflicts. In *International Symposium on Automated Technology for Verification and Analysis* (pp. 120–134).: Springer.

Mastrolilli, M. & Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of scheduling*, 3(1), 3–20.

McMahon, G. & Burton, P. (1967). Flow-shop scheduling with the branch-and-bound method. *Operations Research*, 15(3), 473–481.

Nishi, T. & Wakatake, M. (2014). Decomposition of timed automata for solving scheduling problems. *International Journal of Systems Science*, 45(3), 472–486.

Nowicki, E. & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management science*, 42(6), 797–813.

Oliveira, A. C., Costa, T. A., Pena, P. N., & Takahashi, R. H. (2013). Clonal selection algorithms for task scheduling in a flexible manufacturing cell with supervisory control. In *2013 IEEE Congress on Evolutionary Computation* (pp. 982–988).: IEEE.

Onwubolu, G. & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, 171(2), 674–692.

Panek, S., Engell, S., Subbiah, S., & Stursberg, O. (2008). Scheduling of multi-product batch plants based upon timed automata models. *Computers & Chemical Engineering*, 32(1-2), 275–291.

Panek, S., Stursberg, O., & Engell, S. (2004). Job-shop scheduling by combining reachability analysis with linear programming. *IFAC Proceedings Volumes*, 37(18), 195–200.

Partovi, A. & Lin, H. (2018). Reactive supervisory control of open discrete-event systems. *arXiv preprint arXiv:1804.00037*.

Pena, P. N., Bravo, H. J., da Cunha, A. E., Malik, R., Lafortune, S., & Cury, J. E. (2014). Verification of the observer property in discrete event systems. *IEEE Transactions on Automatic Control*, 59(8), 2176–2181.

Pena, P. N., Costa, T. A., Silva, R. S., & Takahashi, R. H. (2016). Control of flexible manufacturing systems under model uncertainty using supervisory control theory and evolutionary computation schedule synthesis. *Information Sciences*, 329, 491–502.

Pena, P. N., Cury, J. E., & Lafortune, S. (2008). Polynomial-time verification of the observer property in abstractions. In *2008 American Control Conference* (pp. 465–470).: IEEE.

Pena, P. N., Cury, J. E., & Lafortune, S. (2009). Verification of nonconflict of supervisors using abstractions. *IEEE Transactions on Automatic Control*, 54(12), 2803–2815.

Pena, P. N., Cury, J. E., Malik, R., & Lafortune, S. (2010). Efficient computation of observer projections using op-verifiers. In *WODES* (pp. 406–411).

Pinedo, M. (2008). *Scheduling - Theory, Algorithms and Systems. 3a. ed. [S.l.].* Springer.

Pinson, E. (1995). The job shop scheduling problem: A concise survey and some recent developments. *Scheduling Theory and its Applications*, (pp. 277–294).

Queiroz, M. H. & Cury, J. E. (2000). Modular supervisory control of large scale discrete event systems. In *Discrete Event Systems* (pp. 103–110). Springer.

Queiroz, M. H. d. et al. (2004). Controle supervisório modular e multitarefa de sistemas compostos.

Rafael, G. C. (2018). Solution of a scheduling problem using an abstraction of the closed loop behaviour of a discrete event system.

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

Ramanan, T. R., Sridharan, R., Shashikant, K. S., & Haq, A. N. (2011). An artificial neural network based heuristic for flow shop scheduling problems. *Journal of Intelligent Manufacturing*, 22(2), 279–288.

Ruiz, R., Maroto, C., & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34(5), 461–476.

Schmidt, K. & Breindl, C. (2010). Maximally permissive hierarchical control of decentralized discrete event systems. *IEEE Transactions on Automatic Control*, 56(4), 723–737.

Schmidt, K., Moor, T., & Perk, S. (2008). Nonblocking hierarchical control of decentralized discrete event systems. *IEEE Transactions on Automatic Control*, 53(10), 2252–2265.

Schmidt, K. W. & Cury, J. E. R. (2012). Efficient abstractions for the supervisory control of modular discrete event systems. *IEEE Transactions on Automatic Control*, 57(12), 3224–3229.

Smith, L. & Ball, P. (2012). Steps towards sustainable manufacturing through modelling material, energy and waste flows. *International Journal of Production Economics*, 140(1), 227–238.

Su, R. (2012). Abstraction-based synthesis of timed supervisors for time-weighted systems. *IFAC Proceedings Volumes*, 45(29), 128–134.

Subbiah, S., Tometzki, T., Panek, S., & Engell, S. (2009). Multi-product batch scheduling with intermediate due dates using priced timed automata models. *Computers & Chemical Engineering*, 33(10), 1661–1676.

Tao, F., Cheng, Y., Zhang, L., & Nee, A. Y. (2017). Advanced manufacturing systems: socialization characteristics and trends. *Journal of Intelligent Manufacturing*, 28(5), 1079–1094.

Thistle, J. G. (1996). Supervisory control of discrete event systems. *Mathematical and Computer Modelling*, 23(11-12), 25–53.

Vieira, A. D. et al. (2007). Método de implementaçao do controle de sistemas e eventos discretos com aplicaçao da teoria de controle supervisório. Tese submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Doutor em Engenharia Elétrica.

Vilela, J. N. & Pena, P. N. (2016). Supervisor abstraction to deal with planning problems in manufacturing systems. In *Discrete Event Systems (WODES), 2016 13th International Workshop on* (pp. 117–122).: IEEE.

Weckman, G. R., Ganduri, C. V., & Koonce, D. A. (2008). A neural network job-shop scheduler. *Journal of Intelligent Manufacturing*, 19(2), 191–201.

Wong, K. (1998). On the complexity of projections of discrete-event systems. In *Proc. of WODES* (pp. 201–206).

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

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

Wong-Toi, H. & Hoffmann, G. (1991). The control of dense real-time discrete event systems. In *[1991] Proceedings of the 30th IEEE Conference on Decision and Control* (pp. 1527–1528).: IEEE.

Wonham, W. M. (2015). *Supervisory control of discrete-event systems*. Springer.

Wonham, W. M. & Ramadge, P. J. (1988). Modular supervisory control of discrete-event systems. *Mathematics of control, signals and systems*, 1(1), 13–30.

Zhang, J., Ding, G., Zou, Y., Qin, S., & Fu, J. (2019). Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing*, 30(4), 1809–1830.

Zhang, R., Cai, K., & Wonham, W. M. (2017). Supervisor localization of discrete-event systems under partial observation. *Automatica*, 81, 142–147.

Franklin Meer Garcia Acevedo

ABSTRAÇÃO DO SUPERVISOR PARA SISTEMAS COM RETRABALHO VISANDO A SOLUÇÃO DE UM PROBLEMA DE PLANEJAMENTO

Dissertação de Mestrado.

Orientadora: Profa. Patrícia Nascimento Pena

Belo Horizonte, Brasil

2020

Dedico esse trabalho a meus pais
Pedro Garcia e Maria Acevedo

Agradecimentos

Eu gostaria de começar agradecendo primeiramente a meu A-do-nai dizendo, Baruch Atá A-do-nai Heloheinu Melech Aholam (Bendito és Tu, A-do-nai, nosso D-us, Rei do Universo) por ter me acompanhado nesta conquista. Quero agradecer a meu pai, Pedro Garcia, por me ensinar a caminhar nos caminhos cristãos e por me educar com tanto amor (a suas correções fizeram de mim a pessoa que hoje sou). À minha mãe, Maria Acevedo, por ser essa mulher na minha vida que sempre me encorajou a lutar pelos meus sonhos, você me fez acreditar que nada é impossível. Á meu irmão, Jefferson Garcia, não há palavras para descrever o quanto eu te amo, obrigado por ter sido tão paciente comigo todos estos anos, você irmãozinho sempre foi e será minha inspiração.

Por último, e não menos importante, quero agradecer a minha orientadora, Professora Patrícia Nascimento Pena, pela confiança, amor e carinho. Obrigado por sempre levar salgadinhos no LACSED em cada reunião da semana (sempre chegava dizendo " a titia trouxe bolo"). Não há palavras para expressar quão grato me sinto de ter tido você como minha orientadora. Eu também Gostaria de agradecer todos os meus amigos do LACSED, pela paciência e amor que sempre tiveram comigo. Além disso, gostaria de agradecer as agências de financiamento à pesquisa FAPEMIG, CAPES e CNPQ, por investirem na minha educação.

“La liberté commence où l’ignorance finit.”

(Victor Hugo)

Resumo

Autômatos de Estados Finitos e a Teoria de Controle Supervisório (TCS) têm sido usados para modelar e solucionar problemas de planejamento de tarefas em sistemas de manufatura. Ao modelar o sistema usando a TCS, garante-se segurança e não bloqueio do sistema sob controle, além de e garantir que as soluções geradas poderão ser executadas no sistema.

Trabalhar com PO-Abstrações do comportamento em malha fechada permite reduzir ainda mais o universo de busca do problema de planejamento sem perder as qualidades advindas da TCS. No presente trabalho, faz-se uma extensão de resultado anterior, para o caso em que a abstração do comportamento em malha fechada não é PO-Abstração. Um procedimento para construir, a partir de uma sequência selecionada por seu desempenho sob algum critério, uma linguagem que seja executável até o final na planta. Esta extensão torna possível o uso de abstrações do comportamento em malha fechada de sistemas que por natureza não são PO-Abstrações.

Abstract

Finite state automata and Supervisory Control Theory (TCS) have been used to model and solve task planning problems in manufacturing systems. By modeling the system using TCS, it ensures security and non-blocking of the system under control, and ensures that the generated solutions can be executed on the system.

Working with PO-Abstractions closed-loop behavior allow you to further reduce the universe of planning problem search without losing the qualities that come from TCS. In the present work, an extension of the previous result is made, for the case where the closed-loop behavior abstraction is not a PO-Abstraction. A procedure for building, from a sequence selected by its performance under some criteria, a language that is executable to the end of the plant. This extension makes it possible to use closed-loop behavior abstractions of systems that are not, by nature, PO-Abstractions.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

TCS	Teoria de Controle Supervisório
PO	Propriedade do Observador
BDD	Binary Decision Diagrams
DES	Discrete Event Systems
DESFM	Discrete Event Systems with Flexible Marking
VNS	Variable Neighborhood Search
MFE	Most Frequent Event
CSA	Clonal Selection Algorithm

Lista de símbolos

N	número de tarefas.
M	conjunto de máquinas.
J_n	tarefa.
M_m	máquina.
O_{nm}	sequência de operação.
$\in$	pertence à.
!	fatorial.
$(n!)^m$	máximo de soluções.
Σ^*	conjunto de todas as cadeias possíveis em Σ.
$\geq$	maior ou igual (os operadores são números reais).
Σ	conjunto de eventos ou alfabeto.
ϵ	cadeia vazia.
$\|s\|$	comprimento da cadeia s.
$=$	igual.
$\subseteq$	contido em ou igual a.
L	linguagem.

$\emptyset$	conjunto vazio.
st	concatenação.
$t \leqslant s$	t é prefixo de s.
$\overline{L}$	prefixo fechamento da linguagem L.
L^*	fechamento Kleene da linguagem L.
$\cup$	união.
G	autômato.
Q	conjunto de estados do autômato.
δ	função de transição do autômato.
q_0	estado inicial do autômato.
Q_m	conjunto de estados marcados do autômato.
Γ	conjunto de eventos factíveis.
$\mathcal{L}(G)$	linguagem gerada pelo autômato G.
$\mathcal{L}_m(G)$	linguagem marcada pelo autômato G.
$Ac(G)$	parte acessível do autômato G.
$CoAc(G)$	parte coacessível do autômato G.
$Trim(G)$	parte acessível e coacessível do autômato G.
$\times$	produto de autômatos (os operadores são autômatos).
$\neq$	diferente.
$\wedge$	operador lógico E.
$\cap$	interseção.
$P : \Sigma^* \rightarrow \Sigma_i^*$	projeção natural.
$\notin$	não pertence.

$P^{-1}_{\Sigma \to \Sigma_i}$	projeção inversa.
$\exists$	existe.
$\Longrightarrow$	implica.
$\forall$	implica.
$\|\|$	operador de composição paralela.
$\overline{V}$	autômato verificador.
M	autômato auxiliar.
Σ_r	conjunto de eventos relevantes.
Σ_c	conjunto de eventos controláveis.
Σ_u	conjunto de eventos não controláveis.
E	especificação.
K	linguagem desejada.
S	autômato que implementa o supervisor.
s_{ot}	sequência de produção.
A	linguagem recuperada.
M_i	autômato para a máquina i.
$\Pi(t)$	operador de permutação dos eventos de uma cadeia t.
$SupC(K, G)$	operação que resulta na máxima sublinguagem controlável de K em relação à G.
E_q	especificação de quantidade.
$r_A(q)$	rótulo de uma composição.
$d'(\delta_p(q_0^P, t))$	rótulo de uma projeção.

$d(d'(q))$	função que converte um conjunto de tuplas em uma tupla de conjuntos da função que calcula o rótulo de uma projeção.
q_a	estado ativo.
w_{i_k}	ciclo de trabalho da máquina i.
$\sigma^{\uparrow}_{i_k}$	evento (ligar) controlável que dispara o início de w_{i_k}.
$t^{\downarrow}_{i_k}$	evento (desligar) não controlável que causa o final de w_{i_k}.
$\Sigma_{\sigma\uparrow}$	conjunto de eventos de liga.
$\Sigma_{t\downarrow}$	conjunto de eventos de desliga.
Σ^{C}_{fix}	conjunto de eventos controláveis que não pertencem ao conjunto $\Sigma_{\sigma\uparrow}$ (liga).
Σ_{Ru}	conjunto de eventos não controláveis renomeados.
M^{Ru}	conjunto de máquinas que contém eventos do conjunto Σ_{Ru} (eventos renomeados).
E_p	especificação de quantidade.
$\mathcal{F}^{\uparrow}$	linguagem construída de um procedimento (algoritmo proposto).
T_{Eq}	autômato produto de uma composição paralela.
$\setminus$	diferença de conjuntos.

1

Introdução

A indústria de manufatura vem desempenhando um papel muito importante na evolução da sociedade moderna com o processo evolutivo de fabricação, desde a produção artesanal, produção em massa até personalização em massa de produtos (Tao et al., 2017). Essas industrias de manufatura de hoje são desafiadas a atender às crescentes necessidades dos clientes, com prazo de entrega menor e menos desperdício na produção (Jovane et al., 2008), (Smith & Ball, 2012).

Com o objetivo de garantir um menor prazo de entrega de forma a atender a necessidade dos clientes, o problema de planejamento surge. Os problemas de planejamento contêm um conjunto de tarefas a serem realizadas e um conjunto de recursos disponíveis para executar essas tarefas. Dadas tarefas e recursos, juntamente com algumas informações sobre incertezas, o problema geral é determinar o tempo das tarefas enquanto reconhece a capacidade dos recursos. Esse problema geralmente surge dentro de uma hierarquia

de tomada de decisão na qual o agendamento segue algumas decisões anteriores mais básicas (Baker & Trietsch, 2013). O agendamento como processo de tomada de decisão, desempenha um papel importante na maioria dos sistemas de fabricação e produção (Pinedo, 2008).

Para minimizar o tempo de execução de todas as tarefas em um sistema de manufatura é importante encontrar uma sequência de operações para cada subsistema (máquina) (Pinson, 1995). O Problema de planejamento é considerado um problema de otimização e diversas técnicas já foram desenvolvidas para a sua solução (Arisha et al., 2001).

Dentre as técnicas, utilizam-se a modelagem dos sistemas como Sistemas a Eventos Discretos associada à utilização da Teoria de Controle Supervisório (TCS) proposta por Ramadge & Wonham (1989). A TCS permite a obtenção de uma estrutura, chamada supervisor, que representa o comportamento livre de falhas e bloqueios do sistema, respeitando todas as restrições impostas. A TCS permite uma redução significativa do espaço de busca pela solução ótima, porém, ela não tem como finalidade realizar o escalonamento ótimo.

Uma redução ainda maior do universo de busca pelas sequências ótimas é obtida através do uso de abstrações da abordagem monolítica do supervisor. Vilela & Pena (2016) propõe uma abstração do modelo monolítico que consistiu em retirar do modelo as informações que não são utilizadas diretamente no processo de formação de soluções candidatas. Porém a abstração realizada no supervisor deve possuir a propriedade do observador que foi apresentada por Wong (1998). Portanto, se essa propriedade é garantida na abstração, então qualquer plano de produção escolhido poderá ser executado de inicio a fim no sistema a controlar.

O resultado principal de Vilela & Pena (2016) só é aplicável se a abstração do comportamento em malha fechada possui a propriedade do observador. Apoiando-se na ideia de poder trabalhar com abstrações que visam reduzir o espaço de busca pelas soluções ótimas para o problema de planejamento, este trabalho estende o resultado de Vilela & Pena (2016) para o caso em que a abstração do comportamento em malha fechada não possui a propriedade do observador e apresenta um procedimento para construir, a partir de uma sequência selecionada sob algum critério, uma linguagem que seja executável até o

final na planta. Esta extensão torna possível o uso de abstrações do comportamento em malha fechada de sistemas cujas abstrações não são PO-Abstrações.

Este trabalho é organizado da seguinte maneira: o Capítulo 2 apresenta uma revisão bibliográfica sobre o problema de planejamento e de algumas aplicações da Teoria de Controle Supervisório em problemas de planejamento, além disso, são apresentados alguns trabalhos que utilizam abstrações em diferentes contextos. O Capítulo 3 apresenta os conceitos que fundamentam a modelagem por linguagens e autômatos, além de apresentar resumidamente os conceitos essenciais de trabalhos que verificam e buscam a propriedade do observador em uma abstração. O Capítulo 4 apresenta o trabalho que foi usado como base para a construção deste trabalho. O Capítulo 5 apresenta um exemplo de aplicação do resultado principal deste trabalho. Finalmente, o Capítulo 6 traz as conclusões do trabalho proposto.

1.1 Publicação

Franklin Garcia; Patrícia Pena. "Abstração do Supervisor para Sistemas com Retrabalho visando a Solução de um Problema de Planejamento". In: ANAIS DO 14º SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 2019, Ouro Preto. Anais eletrônicos... Campinas, GALOÁ, 2020. Disponível em: <https://proceedings.science/sbai-2019/papers/abstracao-do-supervisor-para-sistemas-com-retrabalho-visando-a-solucao-de-um—problema-de-planejamento>.

2

Revisão da Literatura

Este capítulo apresenta uma revisão da literatura acerca dos temas abordados neste trabalho, tais como o problema de planejamento e a aplicação da Teoria de Controle Supervisório na solução de problemas de planejamento.

2.1 Problema de Planejamento

O planejamento de tarefas ou problema de escalonamento é uma das atividades industriais mais importantes, especialmente no planejamento de sistemas de manufatura (Arisha et al., 2001). O planejamento de tarefas, em outros termos, é um problema de otimização em que várias tarefas são atribuídas a máquinas em momentos específicos ao tentar minimizar uma variável de interesse (makespan, potência elétrica).

O planejamento de tarefas tem um impacto direto sobre a eficiência e custo de produção em um sistema de manufatura, assim, tem atraído a atenção de pesquisadores desde 1956 (Zhang et al., 2019) e é visto como um problema difícil em otimização combinatória (Liu et al., 2018).

Habitualmente, aplica-se o processo de planejamento em sistemas de manufatura compostos de M máquinas diferentes, onde N tarefas devem ser realizadas. A transformação de um insumo bruto em um insumo acabado é considerado como uma tarefa acabada e é representado como J_n com $n \in \{1, \ldots, N\}$ tarefas que serão processadas em diferentes máquinas M_m, com $m \in \{1, \ldots, M\}$, numa ordem específica. Cada tarefa J_n é composta de uma sequência de operação O_{nm}, cada operação O_{nm} sendo executada em uma máquina selecionada a partir do conjunto disponível M_m. O elemento t_{nm} indica a unidade de tempo de processamento da operação O_{nm} na máquina M_m. O tempo decorrido para a realização de todas as tarefas é chamado de *makespan* (Cavalca & Fernandes, 2018).

Com a formulação do problema definido, é possível obter o escalonamento das operações sobre as máquinas de tal forma que o *makespan* seja minimizado. Isto consiste em determinar a sequência das operações O_{nm} de forma a minimizar a função de tempos de finalização (*makespan*). A Figura 2.1 mostra um Diagrama de Gantt que é um dos modelos mais simples e mais utilizados na representação analógica do planejamento de tarefas. Em sua forma básica, o gráfico de Gantt exibe a alocação de recursos ao longo do tempo, com recursos específicos mostrados ao longo do eixo vertical. O gráfico de Gantt assume que os tempos de processamento são conhecidos com certeza (Baker & Trietsch, 2013).

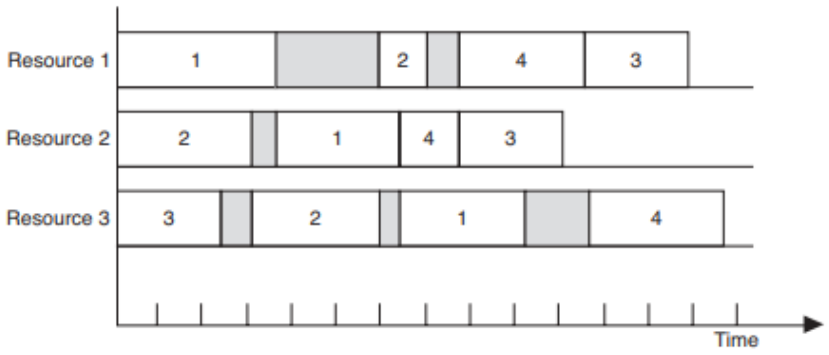


Figura 2.1: Diagrama de Gantt. Adaptado de Baker & Trietsch (2013).

Ghallab et al. (2004) propõe um modelo conceitual para o processo de planejamento, exposto na Figura 2.2.

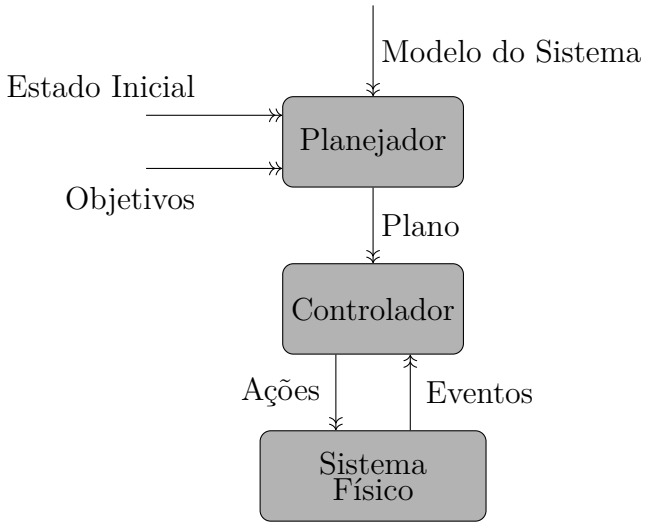


Figura 2.2: Modelo conceitual para o processo de planejamento. Adaptado por Vilela & Pena (2016)

O modelo é composto por três camadas: Planejador, Controlador e Sistema Físico. O planejador recebe informação do modelo do sistema, o estado inicial e os objetivos a serem alcançados. O Planejador é o responsável por encontrar um plano ótimo de modo que os objetivos desejados sejam atingidos, ou seja, é encontrado o plano a ser executado. O plano encontrado pelo Planejador é então enviado ao Controlador, que tem como função fazer com que esse plano seja executado no Sistema Físico. O Controlador, por sua vez, acompanha o Sistema Físico observando os eventos e atuando sobre o mesmo de acordo com o especificado pelo plano.

Encontrar o melhor plano pelo planejador é um problema de otimização e diversas técnicas já foram desenvolvidas para a sua solução. Arisha et al. (2001) apresentam essas técnicas em dois grupos: técnicas tradicionais e técnicas avançadas.

As técnicas tradicionais podem ser classificadas em duas categorias: técnicas analíticas e heurísticas. Entre as técnicas analíticas encontra-se Enumeração Explícita, *Branch and Bound*, Numeração Parcial, Programação Linear, Programação Inteira entre outras.

A técnica de Enumeração Explícita gera uma árvore de enumeração completa. As folhas da árvore representam todas as soluções viáveis. O caminho desde a raiz a uma folha representa uma solução. A principal limitação deste método é o tamanho de pesquisa gerada pois há um máximo de $(n!)^m$ soluções a considerar (onde n é o número de tarefas e m o número de máquinas).

Outra técnica utilizada é a conhecida como *Branch and Bound* que consiste em cortar ramificações de árvores enumeradas e, portanto, reduzir substancialmente o número dos nós gerados. Uma solução pode ser encontrada sistematicamente examinando os subconjuntos de uma solução viável (Arisha et al., 2001). Em outras palavras, o método consiste em particionar o problema maior em dois ou mais subproblemas e seguidamente calcular o limite inferior para a solução ótima de cada subproblema (Baker & Trietsch, 2013).

Ignall & Schrage (1965) introduziram a técnica *Branch and Bound* para minimizar o tempo total de fluxo de tarefas em n trabalhos e duas máquinas. Autores como (McMahon & Burton, 1967), (Ashour, 1970), (Gupta, 1970), (Gonzalez & Sahni, 1978), (Bansal, 1979), entre outros, desenvolveram diferentes métodos da técnica *Branch and Bound* para várias medidas de desempenho tais como *makespan*, tempo médio de fluxo, atraso médio e atraso máximo.

As técnicas heurísticas, são capazes de obter boas soluções com esforço computacional limitado. Em contraste com o método *Branch and Bound*, essas técnicas não garantem que o ótimo seja encontrado, mas elas são relativamente simples e efetivas. Dentre as técnicas heurísticas, o método de amostragem aleatória é uma estratégia que consiste em usar algum mecanismo aleatório que constrói, avalia e identifica a melhor sequência na amostra. A amostragem aleatória é um procedimento para obter boas soluções para problemas combinatórios com lógica simples e direta e esforço computacional limitado (Baker & Trietsch, 2013).

O método de pesquisa em vizinhança parte de dois elementos básicos que são o conceito de vizinhança de uma solução e um mecanismo para gerar vizinhanças. O mecanismo de geração é um método de tomar uma sequência como uma semente e criar sistematicamente uma coleção de sequências relacionadas. Alguns estudos experimentais indicaram que mesmo a versão fundamental do algoritmo de busca de vizinhança é bastante confiável como um procedimento heurístico de propósito geral. Esse método é aplicado em trabalhos como os de Dauzère-Pérès & Paulli (1997), Mastrolilli & Gambardella (2000) e Pena et al. (2016).

A técnica de busca tabu em sua forma básica, pode ser vista como uma forma modificada de pesquisa de vizinhança. Cada vez que uma vizinhança é gerada e uma nova semente é

selecionada, chamamos a mudança de uma semente para a próxima de movimento. Um movimento é definido pelo mecanismo que gera vizinhanças e pela regra de seleção de uma solução na vizinhança. Na pesquisa tabu, o costume é selecionar o melhor valor da função objetivo na vizinhança (Baker & Trietsch, 2013). Esse método é aplicado em trabalhos como os de Dell'Amico & Trubian (1993), Nowicki & Smutnicki (1996) e Bożejko et al. (2017).

Quanto às técnicas avançadas, existem diversas abordagens (Arisha et al., 2001). Essas técnicas são mais recentes e buscam soluções satisfatórias e não soluções ótimas. Como exemplos, tem-se métodos baseados em redes neurais (Weckman et al., 2008), (Ramanan et al., 2011), lógica *fuzzy* (Canbolat & Gundogar, 2004), algoritmos genéticos (Ruiz et al., 2006), (Della Croce et al., 1995), algoritmos evolucionários (Onwubolu & Davendra, 2006), (Rafael, 2018), Pena et al. (2016), etc.

2.2 Teoria de Controle Supervisório

Nesta seção será feita uma breve introdução à Teoria de Controle Supervisório - TCS - proposta por Ramadge & Wonham (1989).

A Teoria de Controle Supervisório constitui-se como uma importante ferramenta para o controle de Sistemas a Eventos Discretos. Baseado em um modelo que descreve o comportamento livre do sistema a ser controlado (planta) e num conjunto de especificações comportamentais é possível realizar a síntese formal de um supervisor. A ação de controle do supervisor restringe, de forma minimamente restritiva, o comportamento de sistema (planta e especificações) de forma a satisfazer o conjunto de especificações (Vieira et al., 2007).

O conjunto de especificações num sistema consideram aspectos como: segurança e vivacidade do sistema, prioridade de eventos e justiça na alocação de produtos. Conforme Queiroz et al. (2004), uma especificação de segurança visa garantir que "nada de ruim aconteça" ao sistema, ao passo que uma especificação de vivacidade visa garantir que "algo de bom aconteça".

A Teoria de Controle Supervisório propõe que uma estrutura denominada supervisor observe a sequência de eventos gerada espontaneamente no sistema a ser controlado e que, instantaneamente após a observação de um novo evento, determine o subconjunto de eventos que concatenados a esta sequência preservam o comportamento do sistema dentro do desejado pelas especificações (Vieira et al., 2007).

Na Figura 2.3 é apresentado graficamente a arquitetura de controle supervisório conforme definido em (Ramadge & Wonham, 1989).

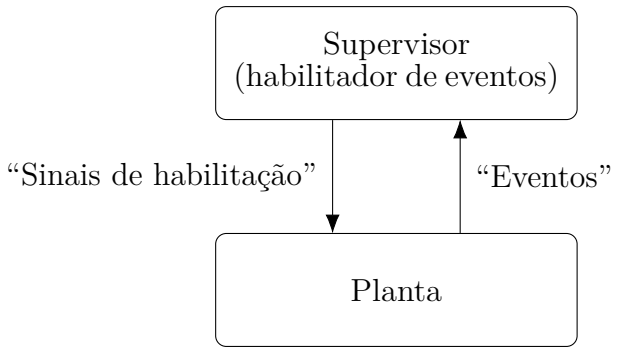


Figura 2.3: Supervisão de um Sistema a Eventos Discretos.

A síntese de um único supervisor para toda a planta é chamada de Controle Monolítico. A dificuldade mais fundamental desta abordagem, como mostrado por Gohari & Wonham (2000), é que a síntese do supervisor Monolítico é NP-difícil, na medida em que o tamanho do espaço de estado cresce exponencialmente com o número de componentes individuais da planta e especificações de controle, logo se torna inviável a obtenção do supervisor por essa abordagem para sistemas de médio e grande porte (Wonham & Ramadge, 1988), (Wong, 1998), (Komenda & van Schuppen, 2005).

Para tratar essa questão, existem extensões da TCS como o Controle Modular, Controle Modular Local e o Controle Hierárquico que visam contornar esse problema. Recentemente o problema da explosão de estados foi também mitigado por um procedimento de síntese que emprega uma decomposição hierárquica para representar o espaço de estados e diagramas de decisão binária (BDD). Assim, o estado nas funções de transição e controle é substituído pela lista de nós do BDD, cujo tamanho é, em casos favoráveis, aproximadamente logarítmico no tamanho de estados (Cai & Wonham, 2013).

O Controle Modular, proposto por Wonham & Ramadge (1988) é uma extensão do Controle Monolítico que tem como objetivo contornar o problema da explosão de estados,

com a síntese de vários supervisores de tamanhos menores. O Controle Modular considera a planta como um todo e para cada especificação desejada obtém-se um supervisor. O problema é decomposto horizontalmente em vários subproblemas (Thistle, 1996).

No entanto, ao decompor o problema em subproblemas, surge o conceito de conflito, que ocorre quando um supervisor interfere na ação de outro, o que pode levar a planta para uma situação não desejada de bloqueio (Flordal & Malik, 2006). Verificar o não-conflito, é computacionalmente caro, em geral. O problema da ocorrência de conflitos é abordado no literatura de diferentes maneiras. Alguns autores desenvolveram abordagens onde os supervisores são não conflitantes por construção (Chen & Lafortune, 1991), (Hill et al., 2008). Outros resolveram o problema do bloqueio usando coordenadores (Wong & Wonham, 1998), (Feng & Wonham, 2006). Malik (2004) propõe um método para detecção de conflitos baseado numa caracterização do conflito e Malik et al. (2004) estudou conflitos de um ponto de vista algébrico.

O Controle Modular Local proposto por Queiroz & Cury (2000), visa contornar o problema da explosão de estados, com a síntese de vários supervisores ainda menores. Essa abordagem, considera como planta apenas os subsistemas que são diretamente afetados pelas especificações.

A complexidade computacional para síntese do conjunto de supervisores locais é significativamente reduzida em relação à complexidade computacional para síntese do Supervisor Monolítico. A abordagem do Controle Modular Local é um exemplo de decomposição horizontal (Thistle, 1996). Assim como no Controle Modular, a atuação de vários supervisores sobre o mesmo sistema pode gerar situações de bloqueio. Portanto, é preciso que os supervisores obtidos por esta abordagem sejam não-conflitantes. Pena et al. (2009) propõem um método eficiente para a verificação do conflito de supervisores Modulares Locais.

O Controle Hierárquico (Wong & Wonham, 1996) é uma forma de decomposição vertical. O Controle Hierárquico é uma estrutura de dois níveis, no qual o nível superior é um modelo do nível inferior. O nível superior é construído por meio de uma abstração de nível inferior. A comunicação entre o sistema e a abstração é feita por um canal de informação. O conceito de consistência hierárquica é introduzido para garantir que as

ações impostas pelo controlador do nível superior sejam realizáveis pelo nível inferior. A garantia de que o não-bloqueio seja preservado entre o nível superior e o nível inferior é resumida no conceito de consistência hierárquica. Isto é garantido se o canal de informação possui o conceito de "observador", conceito introduzido por Wong & Wonham (1996) e utilizado neste trabalho.

A abordagem de Controle Hierárquico se dá na forma de abstrações. Cunha & Cury (2007) propõem uma supervisão hierárquica de dois níveis, esquema de controle construído em uma abordagem de baixo para cima. O modelo de baixo nível é assumido como tendo a estrutura padrão de controle (Ramadge & Wonham, 1989), e o modelo de alto nível é assumido como um DES com Marcação Flexível (siglas em inglês DESFM) (Cury et al., 2004). Fazendo o DES de alto nível um DESFM, é possível obter uma representação compacta dos possíveis comportamentos marcados em malha fechada de alto nível como abstração dos possíveis comportamentos marcados em malha fechada de baixo nível. Portanto, nessa abordagem, é possível desconsiderar condições estruturais habituais relacionadas com a consistência do comportamento marcado, como a consistência de marcação. A vantagem desta abordagem é a redução da complexidade para modelo DES de alto nível, em termos de número de eventos e estados.

Abstrações são utilizadas em diversos trabalhos. Pena et al. (2009) propõem um método eficiente para a verificação de não-conflito na abordagem modular local. A abordagem consistiu em realizar projeções naturais dos supervisores modulares locais mantendo a propriedade do observador e apresentando condições sobre o alfabeto que é mantido na projeção, denominado alfabeto de eventos relevantes.

O uso de abstrações no Controle Supervisório, visa reduzir a complexidade da síntese de supervisores em grande escala, ao mesmo tempo que exploram a estrutura da planta (Schmidt & Cury, 2012). Para este fim, é desejado calcular uma abstração menor da planta para sintetizar um supervisor não bloqueante e, se possível, maximamente permissivo. Na literatura, os métodos em (Feng & Wonham, 2008), (Hill et al., 2010), (Leduc et al., 2005), (Schmidt & Breindl, 2010) e (Schmidt et al., 2008) calculam abstrações usando projeções naturais.

2.3 Aplicações da TCS em Problemas de Planejamento

Atualmente, diversos trabalhos utilizam sistemas a eventos discretos e a TCS para a solução de problemas de escalonamento e que posteriormente, aplicam alguma técnica já conhecida para obter uma solução.

Abdeddaim & Maler (2001) modelam o problema clássico de planejamento de *job-shop* através de um autômato temporizado. A solução do problema de planejamento foi proposta no contexto da metodologia de verificação em duas direções ortogonais: da verificação à síntese. A verificação consistiu em verificar a existência de certos caminhos de um autômato, enquanto na síntese o autômato foi restrito para um ou mais caminhos ótimos. Em outras palavras, a solução foi obtida pela busca do menor caminho com relação ao tempo. Abdeddaim & Maler (2001) apresentaram alguns algoritmos de otimização e heurísticas. O trabalho não apresentou ganhos significativos em relação ao tempo de computação na busca da solução, se comparado com outros abordagens. A principal contribuição foi a utilização de autômatos temporizados à modelagem do problema de planejamento. A ideia de aplicar a síntese aos autômatos temporizados foi explorada pela primeira vez em (Wong-Toi & Hoffmann, 1991).

Panek et al. (2004) também abordam o problema de escalonamento usando autômatos temporizados. A solução foi obtida aplicando técnicas de programação linear inteira e propondo um algoritmo que trabalha nos limites inferiores na árvore de alcançabilidade do autômato temporizado. Os autores concluíram que a especificação de um determinado problema de j*ob-shop* usando ferramentas de programação inteira mista pode eficientemente resolvê-lo, mas é um processo incômodo para a modelagem algébrica do problema e tarefa demorada.

Ainda sobre autômatos temporizados, Nishi & Wakatake (2014) propõem um algoritmo de decomposição para o problema de escalonamento. O modelo proposto compreende a composição paralela de submodelos. O procedimento da metodologia proposta foi dividido em duas etapas. O primeiro passo foi decompor o modelo do autômato temporizado em vários submodelos usando condições decomponíveis. O segundo passo foi combinar

a solução individual de subproblemas para os submodelos decompostos pelo método da função de penalidade. Outro método proposto na literatura para solucionar o problema de planejamento foi o *sleep set* combinado com a técnica de redução do espaço de estados para reduzir o espaço de busca, eliminando execuções redundantes de busca (Panek et al., 2008), (Subbiah et al., 2009).

A Teoria de Controle Supervisório juntamente com programação linear inteira mista é usada por Kobetski & Fabian (2006) para obter as sequências ótimas para a operação de robôs. O trabalho propõe um método de conversão automática entre autômatos de estados finitos e programação linear inteira mista. A característica mais significativa da técnica é a possibilidade de explorar a Teoria de Controle Supervisório juntamente com um método de otimização bem estudado e difundido, facilitando a geração de planejamentos ótimos, garantindo ao mesmo tempo que o sistema se comporte conforme especificado.

Atualmente, os sistemas têm uma tendência a serem maiores e cada vez mais complexos. Fabre & Jezequel (2009) trataram esse problema modelando uma grande rede de autômatos como um sistema distribuído. O planejamento nesse sistema consiste em selecionar e organizar ações para atingir um estado de meta de maneira ideal, assumindo que as ações têm um custo. Para lidar com a complexidade do sistema, propuseram uma abordagem de planejamento distribuída (modular). A solução para cada problema de planejamento relacionado a cada "agente" é obtida fazendo o uso de algoritmos clássicos de alcançabilidade.

Su (2012) propôs um método de abstração autônomo determinístico que permite que a abstração do modelo seja realizada de forma distribuída e, portanto, com potencial de reduzir significativamente o tamanho de uma planta, de modo que a síntese subsequente do supervisor de tempo ótimo possa ser feita de uma maneira computacionalmente eficiente. Depois de mostrar como derivar um supervisor ótimo de tempo com base em um modelo abstraído, explica como usar um supervisor para roteamento e planejamento de tarefas.

Oliveira et al. (2013) propuseram uma abordagem para o problema de planejamento de tarefas baseada no uso de um algoritmo de seleção clonal combinada com a Teoria de Controle Supervisório. Duas metodologias foram propostas. A primeira, o Algoritmo de Seleção Clonal (CSA pelas suas siglas em inglês) realiza a busca da solução ideal, utilizando

buscas aleatórias sobre permutações de sequências de operações. A segunda é semelhante, mas o CSA usa uma pesquisa local para melhorar o melhor indivíduo de cada geração. Os resultados obtidos dependem muito dos parâmetros do algoritmo de seleção clonal. Portanto, testes mais exaustivos e sistemáticos, com uma ampla gama de problemas e com diferentes parâmetros do algoritmo devem ser realizados para estabelecer resultados mais conclusivos. A metodologia proposta é limitada pelo fato de que só pode ser aplicada para lidar com pequenos lotes de produtos.

Pena et al. (2016) apresenta uma nova abordagem para o sequenciamento ideal de tarefas em sistemas de manufatura, com base na combinação de algoritmos meta-heurísticos que tentam otimizar a composição da solução, usando a Teoria do Controle Supervisório de Sistemas de Eventos Discretos para modelar as restrições. Essa abordagem foi denominada Controle Supervisório e Otimização (SCO pelas suas siglas em inglês). Em relação à pesquisa meta-heurística, a codificação de embaralhamento de palavras induz um espaço de variável de decisão no qual pesquisas eficientes podem ser executadas. Com essa codificação, uma versão do *variable neighborhood search* (VNS) foi desenvolvida. Embora tenha-se mostrado uma alternativa para lidar com essa classe de problemas, a metodologia da SCO é limitada pelo fato de que só pode ser aplicada para lidar com pequenos lotes de produtos.

Costa et al. (2018) propõem uma abordagem complementar à SCO (Supervisory Control and Optimization), proposta em Pena et al. (2016), denominada SCO-Concat, desenvolvida para realizar o planejamento em lotes maiores de produção. A metodologia proposta foi testada em um sistema flexível de manufatura, retirado da literatura, e os resultados obtidos mostram que é possível executar o agendamento de lotes de produção tão grandes quanto necessário, a um custo computacional bastante viável. De acordo com esses testes, o SCO-Concat provou ser eficiente para gerar planos de produção para grandes lotes.

Malik & Pena (2018) propuseram uma abordagem que demonstra o uso da verificação de modelo para resolver o problema do agendamento ideal de tarefas em um sistema de fabricação flexível usando a TCS. O estudo de caso considerado no artigo foi o Sistema de Manufatura Flexível apresentado em de Queiroz et al. (2005). O método proposto produziu com sucesso agendamentos ótimos para fabricar até 30 produtos de dois tipos diferentes. Além disso, o método é usado para encontrar uma ciclo ideal resolvendo o

problema de agendamento do caso de estudo para um número arbitrário de produtos em um tempo ideal ou assintoticamente próximo do tempo ideal.

Alves et al. (2016) aborda o problema de planejamento em sistemas de manufatura, propondo um método para encontrar uma sequência de eventos em um supervisor que maximiza o paralelismo entre equipamentos. O supervisor é obtido usando a Teoria do Controle de Supervisório que implementa a máxima sub-linguagem controlável contida no comportamento desejado do sistema. O objetivo principal é encontrar, dentre todas as execuções permitidas pelo supervisor, aquele que acumula mais equipamentos trabalhando ao mesmo tempo durante todo o processo de produção. A sequência obtida pelo máximo paralelismo não necessariamente é obtida em relação ao *makespan*, mas apresenta boas soluções em tempo.

Considerando o modelo conceitual da Figura 2.2, o controlador só pode atuar sobre os eventos controláveis da TCS. Assim, considera-se válido o argumento de que o plano (sequência) ótimo que o planejador passa ao supervisor seja composto apenas de eventos controláveis. Fundamentando-se nisso, Vilela & Pena (2016) estabelecem condições suficientes que garantem que a busca pelas sequências de solução do problema de planejamento pode ser realizada sobre uma abstração no conjunto de eventos controláveis do comportamento em malha fechada. As condições garantem que qualquer sequência escolhida na versão abstraída poderá ser executada no sistema físico sem nenhuma violação (controlabilidade). A condição suficiente para que qualquer sequência obtida na abstração para o conjunto de eventos controláveis do supervisor para ser usada como plano é que a abstração tenha a propriedade do observador. O resultado principal de Vilela & Pena (2016) por si só não é a solução do problema de planejamento, pois não oferece informação que permita quantificar o quão boa é a solução.

Uma extensão ao trabalho de Vilela & Pena (2016) é proposta por Alves & Pena (2017), que propõem o uso de abstrações de supervisores obtidos pela síntese modular local no lugar da abordagem monolítica. Essa extensão torna viável a aplicação do resultado anterior para os casos em que a verificação da propriedade do observador sobre a projeção do supervisor monolítico ou a própria projeção não possam ser computadas. Do mesmo modo que o trabalho de Vilela & Pena (2016), o resultado principal de Alves & Pena (2017) não é a solução final para o problema de planejamento, mas uma boa ferramenta

que reduz o espaço de busca das sequências.

Em Rafael (2018) foram propostos dois métodos para a aplicação do resultado teórico de Vilela & Pena (2016) que gera soluções para a otimização de problemas de escalonamento. O primeiro método proposto utilizou a TCS para criar uma solução que codifica o comportamento em malha fechada do sistema combinando-o com o Algoritmo de Seleção Clonal. Nesta técnica, não é produzida nenhuma sequência infactível, ou seja, todas as soluções encontradas podem ser executadas no sistema, tornando a abordagem muito eficiente. Rafael (2018) propôs um segundo método para gerar sequências chamado algoritmo do Evento Mais Frequente (MFE pelas suas siglas em inglês), que é um algoritmo heurístico que favorece os eventos que estão mais frequentemente disponíveis durante a operação do sistema. Apesar de não garantir resultados ótimos, ambos os métodos, em geral, foram capaz de encontrar as melhores soluções. O MFE foi o mais rápido em relação ao tempo de execução em todas as instâncias testadas, enquanto CSA + o algoritmo de geração individual aleatório apresentou os valores mais curtos do *makespan*.

3

Preliminares

Neste capítulo, alguns conceitos básicos de Sistemas a Eventos Discretos, Teoria de Linguagens e Autômatos, Teoria de Controle Supervisório serão apresentados.

3.1 Sistemas a Eventos Discretos

Sistemas a Eventos Discretos se referem a uma classe de sistemas dinâmicos que possuem um espaço de estados discretos e evoluem em resposta abrupta à ocorrência de eventos (Partovi & Lin, 2018). Estes eventos podem ser vistos como uma ação específica tomada, como por exemplo: pressionar um botão ou ligar uma máquina. Ou podem ser vistos como uma mudança interna do sistema, tal como o término de uma tarefa ou uma interrupção temporizada. Seja qual for a mudança (interna ou externa), caracterizam-se por serem abruptas e instantâneas (Cassandras & Lafortune, 2009).

Sistemas como automação da manufatura, redes de comunicação, robótica, tráfego de veículos, entre outros, podem ser vistos como Sistemas a Eventos Discretos. Existem diversas formas de modelar e analisar os Sistemas a Eventos Discretos. A abordagem na qual se baseia este trabalho é a de autômatos não temporizados. Nas próximas seções serão apresentadas algumas operações sobre linguagens e autômatos e a Teoria de Controle Supervisório, que serão úteis na compreensão do trabalho.

3.2 Teoria de Linguagens e Autômatos

Seja Σ um conjunto finito de símbolos (eventos) distintos. Σ é também chamado de alfabeto de um Sistema a Eventos Discretos (SED). Seja Σ^* o conjunto de todas as cadeias finitas da forma $\mu_1\mu_2, \ldots \mu_k$, com $k \geq 1$, geradas a partir de eventos do alfabeto Σ, incluindo a cadeia formada de zero eventos chamada de cadeia vazia e denotada por ϵ. O comprimento de uma cadeia é o número de eventos que ela contém, incluindo também as repetições do mesmo evento. Seja s uma cadeia, onde o seu comprimento é denotado por $|s|$. Por convenção, o comprimento da cadeia vazia ϵ é zero ($|\epsilon| = 0$).

Uma linguagem L é um conjunto de cadeias finitas formadas por elementos de Σ, $L \subseteq \Sigma^*$. Em particular $\emptyset, \Sigma$ e Σ^* são linguagens sobre Σ. Na próxima seção serão apresentadas as principais operações que podem ser aplicadas às linguagens.

3.3 Operações Sobre Linguagens

Sejam os eventos t e u pertencentes ao alfabeto Σ, a operação de concatenação de t e u denota-se por tu. O elemento neutro da concatenação é a cadeia vazia ϵ, ou seja, $\epsilon t = t\epsilon = t$. Se os eventos t e u formam a palavra s, pode-se dizer que t é prefixo de s e denota-se $t \leq s$ e u o sufixo de s.

A operação *prefixo-fechamento* de uma linguagem é denotada por $\overline{L}$ e consiste de todos

os prefixos de todas as cadeias em L. O fechamento da linguagem L está definido a seguir:

$$\overline{L} = \{s \in \Sigma^* \mid s \leq t \text{ para algum } t \in L\}.$$

Em geral $L \subseteq \overline{L}$. No entanto, se $L = \overline{L}$, então L é chamada de *prefixo-fechada*. O operador *fechamento de Kleene*, denotado por L^*, também pode ser aplicado sobre linguagens. Um elemento de L^* pode ser formado pela concatenação de elementos de L, incluindo também a cadeia vazia ϵ. L^* está definida como:

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

3.3.1 Autômatos

A forma mais simples de apresentar a noção de um autômato é considerar sua representação gráfica dirigida ou diagrama de transição de estados. Um autômato determinístico de estados finitos é uma quintúpla:

$$G = (Q, \Sigma, \delta, q_0, Q_m).$$

em que:

- Q é o conjunto finito de estados.
- Σ é o conjunto finito de eventos associados a G.
- $\delta : Q \times \Sigma \to Q$ é a função de transição: $\delta(x, \alpha) = y$ significa que existe uma transição rotulada pelo evento α do estado x ao estado y.
- q_0 é o estado inicial, onde $q_0 \in Q$.
- $Q_m \in Q$ é o conjunto de estados marcados.

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

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

Esta função mapeia para qual estado de Q o sistema transita a partir de um estado de Q com a ocorrência de uma sequência $s \in \Sigma^*$. A notação $\Gamma : Q \to 2^{\Sigma}$ representa a função do conjunto de eventos factíveis. Os nós em um autômato representados por círculos, são os estados, e os arcos, representados por setas, são as transições. Os nós representados com um duplo círculo, são os estados marcados e o estado inicial é indicado por uma seta que não possui rótulo. O rótulo de cada arco é o evento associado à transição. A Figura 3.1 mostra um exemplo de autômato determinístico. Um autômato G tem associadas dois

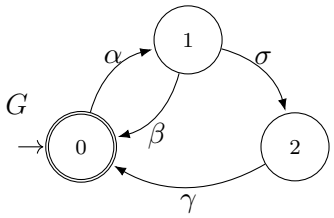


Figura 3.1: Autômato determinístico.

tipos de linguagens, a linguagem gerada

$$\mathcal{L}(G) = \{s \in \Sigma^* : \delta(q_0, s) \text{ está definida}\}$$

e a linguagem marcada

$$\mathcal{L}_m(G) = \{s \in \Sigma^* : \delta(q_0, s) \in Q_m\}.$$

A linguagem gerada $\mathcal{L}(G)$ possui todas as cadeias de eventos que partem do estado inicial até qualquer estado, enquanto a linguagem marcada $\mathcal{L}_m(G)$ representa todos os caminhos a partir do estado inicial até um dos estados marcados. Portanto, o comportamento de um SED pode ser representado por um autômato G, sendo $\mathcal{L}(G)$ a linguagem gerada pelo sistema e $\mathcal{L}_m(G)$ a linguagem marcada do sistema. A linguagem $\mathcal{L}_m(G)$ é subconjunto de $\mathcal{L}(G)$, ou seja $\mathcal{L}_m(G) \subseteq \mathcal{L}(G)$.

Se um estado $q \in Q$ de um autômato G não pode ser alcançado por nenhuma sequência a partir do estado inicial, então esse estado q é chamado de não acessível. É possível separar a componente acessível $Ac(G)$ de um autômato eliminando do autômato os estados que não sejam acessíveis, $G = Ac(G)$.

Se G possui um estado $q \in Q$ do qual não é possível alcançar nenhum estado marcado, então esse estado é dito ser não coacessível. A componente coacessível $CoAc(G)$ de um autômato G é obtida eliminando todos os estados não coacessíveis, $G = CoAc(G)$.

Um autômato G é dito ser não bloqueante se:

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

O bloqueio pode ocorrer de duas formas. A primeira é o *deadlock*, que ocorre quando um estado q não marcado não tem nenhum evento que pode ocorrer, então o autômato fica bloqueado para esse estado. A segunda forma de bloqueio acontece quando existe um conjunto de estados não marcados em G que formam um componente fortemente conectada, mas com nenhuma transição saindo do conjunto. Essa componente fortemente conectada é chamada de *livelock*.

Um autômato que é acessível e coacessível é dito ser *trim* A operação *Trim* é:

$$Trim(G) = CoAc[Ac(G)] = Ac[CoAc(G)].$$

onde a comutatividade de Ac e CoAc é facilmente verificada.

3.3.2 Projeção Natural

A projeção natural $P : \Sigma^* \to \Sigma_i^*$ é uma operação sobre linguagens definida sobre o conjunto de eventos Σ e Σ_i, onde $\Sigma_i \subseteq \Sigma$. A projeção natural mapeia cadeias de Σ^* para cadeias em Σ_i^* apagando todos os eventos que não estão em Σ_i. Esta operação está definida para

cadeias como:

$$P_{\Sigma\to\Sigma_i}(\varepsilon) := \varepsilon$$

$$P_{\Sigma\to\Sigma_i}(s\sigma) := \begin{cases} P_{\Sigma\to\Sigma_i}(s) \ se \ s \in \Sigma^*, \sigma \notin \Sigma_i \\ P_{\Sigma\to\Sigma_i}(s)\sigma \ se \ s \in \Sigma^*, \sigma \in \Sigma_i. \end{cases}$$

Como pode-se ver, a operação da projeção natural toma cadeias formadas com elementos do alfabeto Σ e apaga aqueles eventos que não fazem parte do alfabeto Σ_i. Wong (1998) mostrou que a complexidade de uma projeção natural é no pior caso exponencial. A operação inversa da projeção é a função $P^{-1}_{\Sigma\to\Sigma_i} : \Sigma_i^* \to 2^{\Sigma^*}$ e está definida como:

$$P^{-1}_{\Sigma\to\Sigma_i}(t) = \{s \in \Sigma^* : P_{\Sigma\to\Sigma_i}(s) = t\}.$$

Em palavras, a projeção inversa $P^{-1}_{\Sigma\to\Sigma_i}(t)$ de uma cadeia t retorna todas as cadeias de Σ^*, que são mapeadas para a cadeia projetada t. Essas definições de projeção natural e projeção inversa podem ser estendidas para linguagens aplicando-as a todas as cadeias da linguagem. Para $L \in \Sigma^*$:

$$P_{\Sigma\to\Sigma_i}(L) := \{t \in \Sigma_i^* : (\exists s \in L)[P_{\Sigma\to\Sigma_i}(s) = t]\}.$$

e para $L_i \subseteq \Sigma_i^*$,

$$P^{-1}_{\Sigma\to\Sigma_i}(L_i) := \{s \in \Sigma^* : (\exists t \in L_i)[P_{\Sigma\to\Sigma_i}(s) = t]\}.$$

Para mostrar a ideia de projeção natural, o Exemplo 1 é apresentado.

Exemplo 1. *Considere o alfabeto* $\Sigma = \{\alpha, \beta, \gamma, \sigma\}$ *e o subconjunto* $\Sigma_i = \{\alpha, \beta\}$. *Sejam as linguagens* $L_1 = \{\gamma, \alpha\beta, \alpha\gamma, \beta, \sigma\gamma\alpha\}$ *e* $L_2 = \{\alpha\alpha, \sigma\beta, \gamma\sigma\alpha\}$. *Considere a projeção* $P : \Sigma^* \to \Sigma_i^*$. *Então temos que:*

$$P(L_1) = \{\epsilon, \alpha\beta, \alpha, \beta, \alpha\};$$

$$P(L_2) = \{\alpha\alpha, \beta, \alpha\}.$$

Em Cassandras & Lafortune (2009) definem-se 5 propriedades da projeção natural que são usadas nesse trabalho. Considere que A,B e L são linguagens definidas no alfabeto Σ. Considere também a projeção $P : \Sigma^* \to \Sigma_i^*$, com $\Sigma_i \subseteq \Sigma$. A seguir essas propriedades são apresentadas:

1) $P(P^{-1}(L)) = L$
   $L \subseteq P^{-1}(P(L))$

2) Se $A \subseteq B$, então $P(A) \subseteq P(B)$ e $P^{-1}(A) \subseteq P^{-1}(B)$

3) $P(A \cup B) = P(A) \cup P(B)$
   $P(A \cap B) \subseteq P(A) \cap P(B)$

4) $P^{-1}(A \cup B) = P^{-1}(A) \cup P^{-1}(B)$
   $P^{-1}(A \cap B) = P^{-1}(A) \cap P^{-1}(B)$

5) $P(AB) = P(A)P(B)$
   $P^{-1}(AB) = P^{-1}(A)P^{-1}(B)$

O uso das projeções nesse trabalho têm um papel muito importante. Uma propriedade muito conhecida da projeção natural é a propriedade do observador, apresentada na Definição 1.

Definição 1 (Wong (1998)). *Seja uma linguagem $L \subseteq \Sigma^*$, um alfabeto $\Sigma_i \subseteq \Sigma$ e $P_{\Sigma \to \Sigma_i^*} : \Sigma^* \to \Sigma_i^*$. Se $(\forall a \in \overline{L})(\forall b \in \Sigma_i^*)$, $P_{\Sigma \to \Sigma_i}(a)b \in P_{\Sigma \to \Sigma_i}(L) \Longrightarrow (\exists c \in \Sigma^*)P_{\Sigma \to \Sigma_i}(ac) = P_{\Sigma \to \Sigma_i}(a)b$ e $ac \in L$, então a projeção natural $P_{\Sigma \to \Sigma_i}(L)$ tem a propriedade de observador.* □

Se a projeção natural tem a propriedade do observador, então, garante-se que o cálculo da mesma tenha complexidade polinomial no pior dos casos. Para aclarar a definição da Propriedade do Observador (PO), considere a Figura 3.2.

A elipse em azul (parte A), delimita uma linguagem L gerada a partir de eventos do alfabeto Σ, enquanto a elipse em salmão (parte B), representa a projeção natural dessa linguagem ($P(L)$) para o alfabeto Σ_i. Escolhe-se uma cadeia a que é prefixo de uma cadeia da linguagem L. Logo faz-se a projeção da cadeia a ($P_{\Sigma \to \Sigma_i}(a)$) para o alfabeto Σ_i. Em seguida completa-se a projeção de a, $P_{\Sigma \to \Sigma_i}(a)$, com um sufixo b, formado apenas

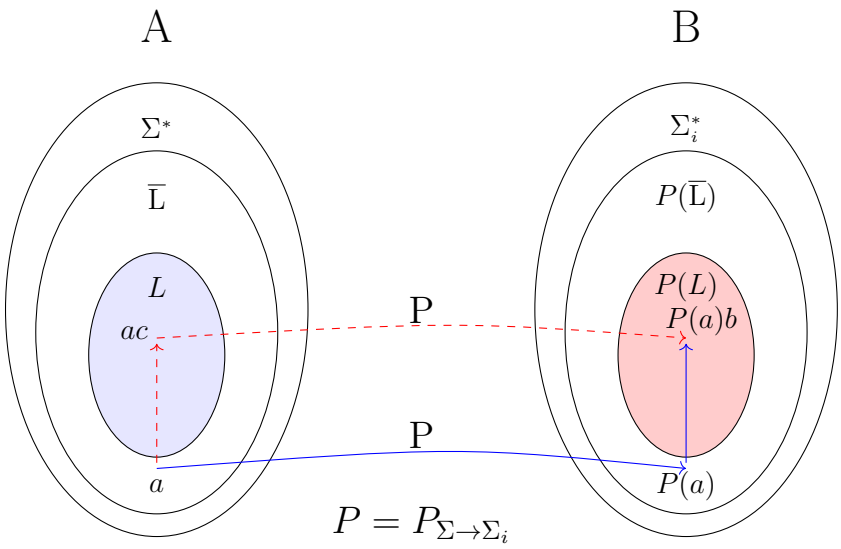


Figura 3.2: Representação da propriedade do observador.

de eventos de Σ_i , levando à obtenção de uma cadeia que está na projeção de L, ou seja $P_{\Sigma \to \Sigma_i}(a)b \in P_{\Sigma \to \Sigma_i}(L)$. Essa operações são indicadas pelas setas em azul.

Se a projeção possui a propriedade do observador, então, para todo a e b tal que $P_{\Sigma \to \Sigma_i}(a)b \in P_{\Sigma \to \Sigma_i}(L)$, existirá c tal que $P_{\Sigma \to \Sigma_i}(ac) = P_{\Sigma \to \Sigma_i}(a)b$, que é o caminho representado pelas setas em vermelho. Portanto, tem-se que $P_{\Sigma \to \Sigma_i}(c) = b$.

A linguagem $P(L)$ pode ser chamada de abstração. Uma abstração que possua a propriedade do observador é chamada de PO-abstração (Pena et al., 2008). A propriedade do observador pode ser verificada usando o algoritmo apresentado em (Pena et al., 2014).

3.3.3 Composição Paralela

A operação de composição paralela, também chamada composição síncrona, tem como resultado um autômato que modela o comportamento conjunto de dois ou mais autômatos. Em geral, a operação de composição paralela é a utilizada para a construção de modelos de sistemas compostos de sistemas individuais (subsistemas). A composição paralela dos autômatos $G_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_{m1})$ e $G_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, Q_{m2})$ é o autômato:

$$G_1 \,||\, G_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$$

onde

$$\delta((x_1,x_2),e) := \begin{cases} (\delta_1(x_1,e),\delta_2(x_2,e)) & \text{se } \delta_1(x_1,e)\neq\emptyset \wedge \delta_2(x_2,e))\neq\emptyset \\ (\delta_1(x_1,e),x_2) & \text{se } \delta_1(x_1,e)\neq\emptyset \wedge e\notin\Sigma_2 \\ ((x_1,\delta_2(x_2,e)) & \text{se } \delta_2(x_2,e)\neq\emptyset \wedge e\notin\Sigma_1 \\ \text{indefinido} & \text{caso contrário} \end{cases}$$

Na composição paralela os eventos em comum ou seja, que pertençam a $\Sigma_1 \cap \Sigma_2$ podem ser executados se os autômatos podem executá-los simultaneamente ou seja, os autômatos são sincronizados nos eventos em comum. Os eventos privados, pertencentes a $(\Sigma_1 \backslash \Sigma_2) \cup (\Sigma_2 \backslash \Sigma_1)$, não sofrem nenhuma restrição e podem ser executados sempre que puderem em seus autômatos . Se não existe evento em comum entre os dois alfabetos, ou seja, $\Sigma_1 \cap \Sigma_2 = \emptyset$, então não há transições sincronizadas e qualquer comportamento possível em G_1 e G_2 também é possível em $G_1||G_2$.

A noção de projeção inversa descrita na seção 3.3.2 pode ser usada para prover a definição formal da operação de composição paralela, denotada por ||, de linguagens $L_i \subseteq \Sigma_i^*$, com $i \in I$, e $\Sigma = \bigcup_{i\in I} \Sigma_i$:

$$\|_{i\in I} L_i = \bigcap_{i\in I} P^{-1}_{\Sigma\to\Sigma_i}(L_i).$$

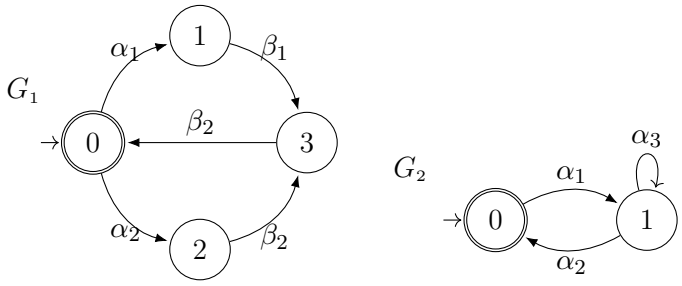
Do mesmo modo, a noção de projeção inversa pode ser usada para representar as linguagens marcada e gerada de uma composição paralela como:

$$\mathcal{L}(G_1||G_2) = P^{-1}_{\Sigma\to\Sigma_1}[\mathcal{L}(G_1)] \cap P^{-1}_{\Sigma\to\Sigma_2}[\mathcal{L}(G_2)]$$
$$\mathcal{L}_m(G_1||G_2) = P^{-1}_{\Sigma\to\Sigma_1}[\mathcal{L}_m(G_1)] \cap P^{-1}_{\Sigma\to\Sigma_2}[\mathcal{L}_m(G_2)]$$

O Exemplo 2 ilustra a composição.

Exemplo 2. *Sejam os subsistemas G_1 e G_2 apresentados na Figura 3.3 (a) e (b) correspondentemente. O autômato G da Figura 3.4 ilustra a composição paralela entre os dois subsistemas.*

Os eventos α_1 e α_2 evoluem sincronicamente, porém, os eventos β_1, β_2 e α_3 evoluem assincronicamente. Cada estado do autômato G (Figura 3.3 (c)) apresenta a informação



(a) Subsistema G_1. (b) Subsistema G_2.

Figura 3.3: Subsistemas G_1 e G_2.

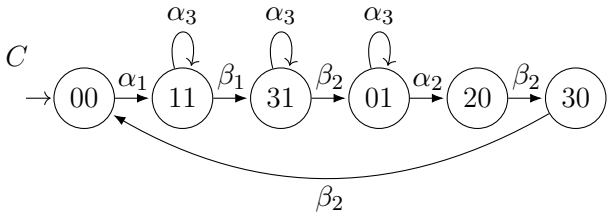


Figura 3.4: Composição paralela.

atual (rótulos) dos estados de cada autômato G_1 e G_2 correspondentemente.

3.4 Verificação da Propriedade do Observador

Pena et al. (2014), propõem um algoritmo, com complexidade quadrática, para a verificação da propriedade do observador para uma projeção natural arbitrária $P_{\Sigma\to\Sigma_r}$ sem a necessidade de se calcular a projeção natural propriamente dita. Esse algoritmo é uma extensão do algoritmo proposto por (Pena et al., 2008) onde a desvantagem desse método era a restrição imposta sobre G, que exigia a não existência de ciclos de eventos não relevantes (que sumiram na projeção).

O algoritmo Pena et al. (2014) constrói um autômato não determinístico $\overline{V}$, denominado verificador, a partir de um autômato também não determinístico $\overline{M}$ e um alfabeto de interesse Σ_r. Esse autômato $\overline{M}$ agrupa os ciclos de eventos não relevantes como macroestados. O autômato $\overline{M}$ é obtido de um autômato determinístico M, chamado autômato auxiliar, gerado do modelo do sistema G. Caso o estado DEAD não seja alcançado no autômato $\overline{V}$ então a abstração $P_{\Sigma\to\Sigma_r}(G)$ verifica a propriedade do observador. Este autômato auxiliar $M = (Q^M, \Sigma, \delta^M, q_0^M)$ é obtido a partir do autômato $G = (Q^G, \Sigma^G, \delta^G, q_0^G, Q_m^G)$ que modela o sistema, substituindo os estados marcados de G por estados não marcados com autolaços

com um evento artificial $\tau \in \Sigma_r$ que fará parte do conjunto de eventos relevantes. A estrutura do autômato verificador $\overline{V} = (\overline{Q}, \overline{\Sigma}, \overline{\delta}, \{A_0^{\overline{M}}\})$, permite classificar, sem restrição nenhuma, se uma projeção tem ou não a propriedade do observador.

No Exemplo 3 é realizada a verificação da propriedade do observador a partir da construção do verificador $\overline{V}$.

Exemplo 3. *Para o autômato G apresentado na Figura 3.5 deseja-se saber se para a projeção natural $P_{\Sigma\to\Sigma_r}$, com $\Sigma_r = \{\alpha\}$, $P(G)$, possui a propriedade do observador.*

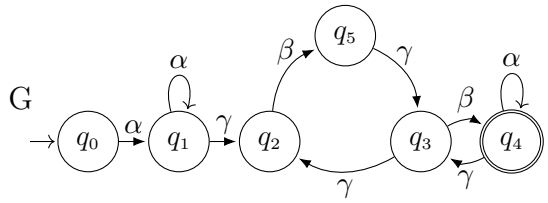
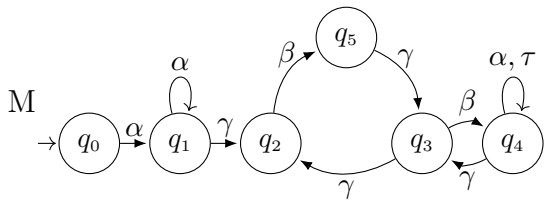
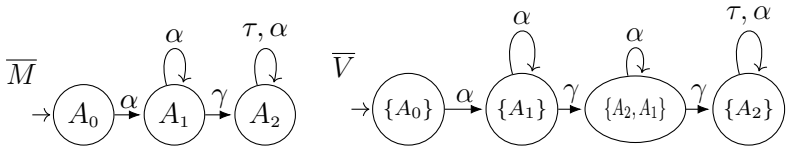


Figura 3.5: Exemplo 3: Autômato para a planta G.



(a) Autômato auxiliar M.



(b) Autômato auxiliar $\overline{M}$. (c) Verificador $\overline{V}$.

Figura 3.6: Exemplo 3: Verificação da propriedade do observador.

O primeiro passo consiste em obter o autômato auxiliar M, que possui a mesma estrutura de G mas os estados marcados são expressos por um autôlaço com o evento $\tau \in \Sigma_r$. Observe-se que o estado DEAD não é alcançado no autômato verificador $\overline{V}$, portanto $P_{\Sigma\to\Sigma_r}(G)$ possui a propriedade do observador.

Se o autômato verificador $\overline{V}$ alcança o estado DEAD, então diz-se que a projeção do sistema no conjunto de eventos relevantes $P_{\Sigma\to\Sigma_r}(G)$ não possui a propriedade do observador. Portanto se não possuir a PO, existem instâncias de eventos que, ao serem mostrados na

projeção, permitem construir uma PO-abstração. A seguinte seção mostra o processo da busca dessas instâncias de eventos.

3.5 Busca da Propriedade do Observador

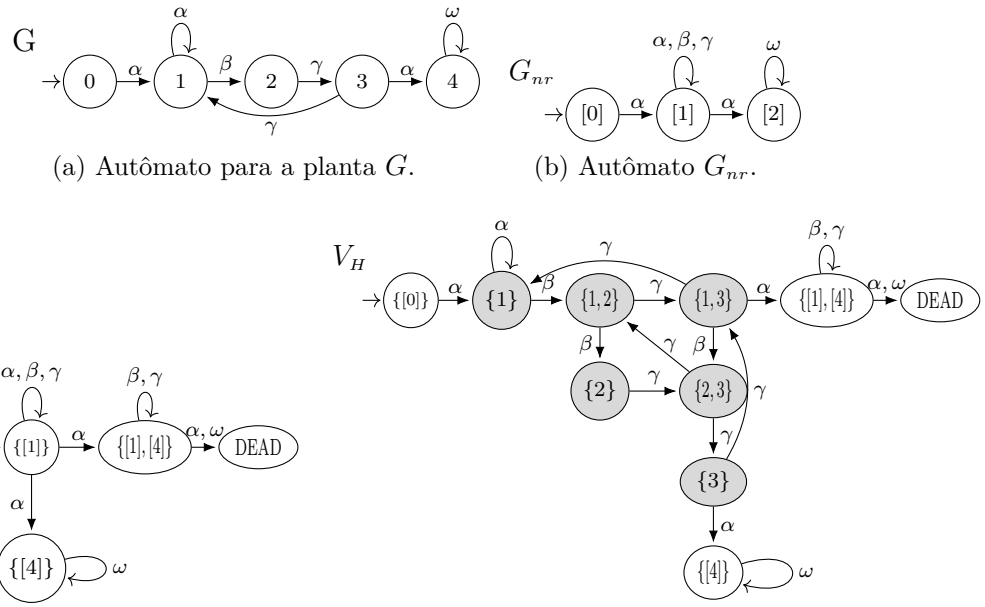
O problema da busca por PO abstrações consiste no renomeamento de eventos de um dado sistema e a sua projeção natural que não satisfazem a PO. O renomeamento pode ser realizado selecionando e transformando um evento não relevante em relevante.

Bravo et al. (2014) propõem um algoritmo com complexidade no pior caso de $O(|Q|^3(|\Sigma|+1)^2)$ (onde Q é o conjunto de estados do autômato e $|\Sigma|$ é a cardinalidade do alfabeto), para a busca da propriedade do observador, onde essa complexidade é a mesma que o algoritmo proposto por Pena et al. (2010) (complexidade polinomial). O algoritmo proposto é uma combinação dos resultados de (Pena et al., 2010) e (Bravo et al., 2012). O método proposto funciona construindo um verificador híbrido e expandindo alguns dos componentes de (Pena et al., 2010), a fim de identificar transições que causam a violação da propriedade do observador. O algoritmo pode ser aplicado a autômatos que apresentam ciclos de eventos não relevantes, limitação do algoritmo de Pena et al. (2010). A continuação apresenta-se o Exemplo 4 para mostrar esta ideia.

Exemplo 4. *Para a planta G apresentada na Figura 3.7(a), deseja-se saber se para a projeção natural $P : \Sigma^* \to \Sigma_r^*$, com $\Sigma_r = \{\alpha, \omega\}$, $P(G)$, possui a propriedade do observador, se não tiver, então buscar a propriedade.*

O algoritmo proposto por Bravo et al. (2014) constrói o autômato G_{nr} apresentado na Figura 3.7(b) a partir do autômato da planta G na Figura 3.7(a), onde G_{nr} não apresenta ciclos de eventos não relevantes (exceto os auto-laços). Os estados $\{1, 2, 3\}$ da Figura 3.7(a) formam um macroestado de eventos não relevantes originando o estado $[1]$ da Figura 3.7(b). O autômato não determinístico V_G da Figura 3.7(c), denominado verificador, é construído a partir do autômato G_{nr}. Como o estado DEAD é alcançado em V_G na Figura 3.7(c) então a abstração $P_{\Sigma\to\Sigma_r}(G)$ não é PO-abstração.

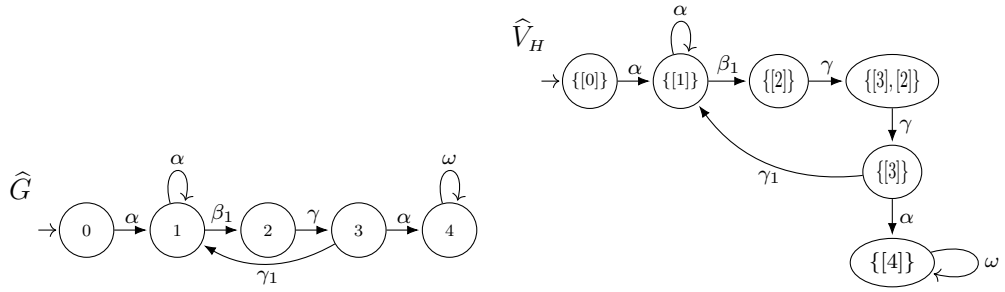
Identifica-se uma cadeia de eventos que leva ao estado DEAD para renomear alguns do seus eventos não relevantes, porém todos os eventos são relevantes (controlá-



(a) Autômato para a planta G. (b) Autômato G_{nr}.

(c) Autômato verificador V_G. (d) Verificador hibrido V_H.

Figura 3.7: Exemplo 4: Verificação e Busca da propriedade do observador.



(a) Autômato $\widehat{G}$ com os eventos renomeados. (b) Autômato $\widehat{V}_H$.

o.

Figura 3.8: Exemplo 4: Verificação e Busca da propriedade do observador.

veis). Portanto, para solucionar esse problema, faz-se a expansão de alguns dos macro estados singleton $(\{1\}, \{2\}$ *ou* $\{3\})$ *que estão no mesmo caminho que leva ao estado DEAD do verificador* V_G*, usando os estados originais (estados da planta) formando uma componente fortemente conexa. Essa expansão origina o autômato verificador hibrido* V_H *da Figura 3.8(d). A expansão do macro estado selecionado* $\{[1]\}$ *será o conjunto* $\{[1]\} = \{\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\}$*, onde cada um deles é um estado (em cinza) que leva ao estado DEAD.*

O próximo passo é escolher um dos estados simples $(\{\{1\}, \{2\}, \{3\}\})$ *do conjunto de estados não seguros e renomear o seu evento não relevante por relevante. Seleciona-se o estado* 3 *do autômato G na Figura 3.7(a) e a transição* γ *criando o novo evento relevante*

γ_1, originado $\widehat{\Sigma}_r = \{\alpha, \omega, \gamma_1\}$. Substitui-se a transição $\delta(3, \gamma) = 1$ por $\widehat{\delta}(3, \gamma_1) = 1$ em G. Seguindo a ideia do algoritmo, constrói-se novamente o verificador para testar se o estado DEAD é acessível. Portanto o estado DEAD ainda é acessível. Na segunda iteração, seleciona-se o estado 1 e o evento β e substitui-se por β_1, criando a transição $\widehat{\delta}(1, \beta_1) = 2$. O resultado dessas duas iterações origina o autômato $\widehat{G}$, apresentado na Figura 3.8(a).

Com o novo conjunto de eventos relevantes, $\widehat{\Sigma} = \{\alpha, \omega, \gamma_1, \beta_1\}$, tem-se o verificador $\widehat{V}_H$ mostrado na Figura 3.8(b). Neste caso, o estado DEAD não é mais acessível em $\widehat{V}_H$, portanto a abstração do autômato $P(\widehat{G})$ possui a propriedade do observador.

Neste trabalho o conjunto de instâncias de eventos não-controláveis que não podem ser apagados na projeção natural, posto que causam a violação da propriedade do observador, será chamado de Σ_{Ru}. Neste exemplo, o conjunto Σ_{Ru} é $\Sigma_{Ru} = \{\gamma_1, \beta_1\}$.

3.6 Teoria de Controle Supervisório

A Teoria de Controle Supervisório (TCS) (Ramadge & Wonham, 1989) propõe a síntese de controladores para *Sistemas a Eventos Discretos* (SED), matematicamente fundamentada na teoria de linguagens e autômatos (Cassandras & Lafortune, 2009). No controle

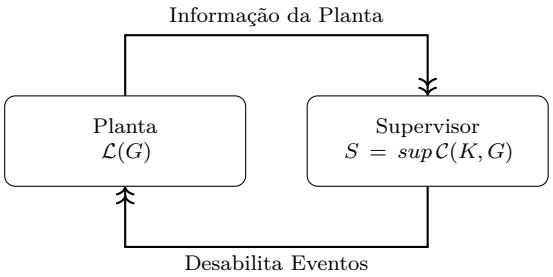


Figura 3.9: Estrutura da Teoria de Controle Supervisório.

Supervisório, considera-se a existência de uma planta G (sistema físico) que modela o comportamento em malha aberta de um SED. No entanto, para limitar o comportamento do sistema G e evitar que alcance certos estados que levariam a um bloqueio ou uma situação insegura, exerce-se alguma ação de controle sobre o sistema, Figura 3.9. Essas restrições, ou especificações, são modeladas como m autômatos E_i, com $i \in I = \{1, \ldots, m\}$, e a especificação global é obtida pelo autômato $E = \|_{i \in I} E_i$. A ação sobre a planta é feita

por meio de uma estrutura (supervisor) que garante o funcionamento do sistema sem que as restrições sejam violadas. Essa estrutura de controle é obtida pela proibição da ocorrência de determinados eventos em certos estados.

O comportamento do sistema G está modelado por um subconjunto de eventos controláveis e um subconjunto de eventos não controláveis. Assim, $\Sigma = \Sigma_c \cup \Sigma_u$, onde Σ_c e Σ_u são, respectivamente, o conjunto de eventos controláveis e não controláveis. Qualquer evento $\sigma \in \Sigma_c$ pode ser desabilitado, enquanto nenhum $\sigma \in \Sigma_u$ pode ser desabilitado pela estrutura de controle.

O comportamento desejado da planta em malha fechada é obtido por meio da operação de composição síncrona entre o autômato $G = (Q, \Sigma, \delta, q_0, Q_m)$ que modela a planta e o autômato E que modela a especificação do sistema, $K = G||E$. K é dito ser controlável com relação a G se e somente se:

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}.$$

Essa propriedade de controlabilidade estabelece que para K ser controlável em relação à $\mathcal{L}(G)$, todo $s \in \overline{K}$ concatenado com um $\sigma \in \Sigma_u$, de modo que $s\sigma \in \mathcal{L}(G)$, deve gerar uma cadeia $s\sigma$ em $\overline{K}$. Quando a especificação E não for verificada, então K é não controlável com relação à $\mathcal{L}(G)$. Se K for não controlável, então a máxima sublinguagem controlável e não bloqueante $S = Sup\mathcal{C}(K, G)$ pode ser sintetizada e implementada por um supervisor (Ramadge & Wonham, 1989). S é dito ser não bloqueante se $\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)}$. A expressão S/G é lida como G sob o controle do supervisor S e representa o sistema em malha fechada.

Por construção, o sistema controlado S/G tem quatro principais propriedades: (a) controlabilidade (S não pode desabilitar nenhum evento não controlável), (b) segurança (S/G gera apenas cadeias legais, ou seja $\mathcal{L}(S/G) \subseteq \mathcal{L}(K)$), (c) não bloqueio ($S/G$ é não bloqueante, ou seja $\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)}$), e (d) máxima permissividade (S/G não pode ser maior sem violar uma das três propriedades anteriores) (Lafortune, 2019).

4

Resultados Anteriores

Neste capítulo apresenta-se o resultado de outro trabalho que serviu como base para a elaboração do resultado principal desse trabalho.

4.1 Abstração de Supervisores

Em Vilela & Pena (2016), as condições sobre as quais a projeção natural do comportamento em malha fechada nos eventos controláveis, $P_{\Sigma \to \Sigma_c}(S)$, mantém as propriedades originais do supervisor são apresentadas. Esse resultado teórico permite encontrar soluções ótimas em um universo de busca muito menor, $P_{\Sigma \to \Sigma_c}(S)$ ao invés de S. A condição suficiente para que este resultado seja aplicável é que a projeção natural do comportamento em malha fechada para o conjunto dos eventos controláveis tem que possuir a propriedade do observador (Definição 1). Esse resultado é apresentado no Teorema 1.

Teorema 1 (Vilela & Pena (2016)). *Seja G uma planta, S a máxima sublinguagem controlável contida na linguagem desejada K e $P_{\Sigma\to\Sigma_c} : \Sigma^* \to \Sigma_c^*$ a projeção natural nos eventos controláveis. Para toda sequência $s_{ot} \in P_{\Sigma\to\Sigma_c}(S)$ e $A = P^{-1}_{\Sigma\to\Sigma_c}(s_{ot}) \cap S$, se $P_{\Sigma\to\Sigma_c}(S)$ possui a propriedade do observador, então A será controlável com relação à $\mathcal{L}(G)$.*

Em outros termos, o Teorema 1 garante que qualquer linguagem $A \subseteq S$ obtida da cadeia $s_{ot} \in P_{\Sigma\to\Sigma_c}(S)$, é capaz de ser executada na planta G sem que seja necessário desabilitar eventos não-controláveis. Isto significa que a linguagem A é controlável com relação à G. A linguagem A garante também as condições de segurança e não bloqueio. ■

Para aplicar o resultado do Teorema 1, é preciso verificar se a projeção do comportamento em malha fecha no conjunto dos eventos controláveis possui a propriedade do observador. No entanto, verificar a propriedade do observador possui complexidade quadrática no tamanho do espaço dos estados do supervisor (Pena et al., 2014).

A necessidade dessa verificação pode ser evitada com a utilização de resultados adicionais apresentados por Vilela & Pena (2016). Inicialmente apresentam-se duas definições que servem de diretrizes para a modelagem do problema:

Definicão 2 (Sistema de Produção). *Um sistema de produção G é definido como um sistema composto por m autômatos $M_i = (\Sigma_i, Q_i, \delta, Q_i^m, q_i^0)$, $i \in I = \{1, \ldots, m\}$, tal que:*

i) $G = \|_{i=1}^{m} M_i$, $\Sigma = \bigcup_{i=1}^{m} \Sigma_i$;

ii) $\Sigma_i \cap \Sigma_j = \emptyset$, $i, j \in I$ *e* $i \neq j$;

iii) M_i *é caracterizada por:*

a) $\forall \sigma \in \Sigma_i$ *tal que* $\delta(q_0, \sigma) \neq \emptyset$, $\sigma \in \Sigma_{ic}$;

b) $P_{\Sigma_i\to\Sigma_{ic}}(M_i)$ *possui a propriedade do observador.* □

Em palavras, um sistema de produção é composto por m autômatos com eventos disjuntos (*ii*), todo autômato deve iniciar com evento controlável *iii*)a) e a projeção natural no conjunto dos eventos controláveis deve possuir a propriedade do observador *iii*)b).

Definicão 3 (Especificação de Coordenação). *A especificação de coordenação $E = (\Sigma_E, Q_E, \delta, Q_E^m, q_E^0)$ é uma especificação de cordenação se:*

i) $\mathcal{L}(E)$ *é controlável com relação à* $\mathcal{L}(G)$*;*

ii) $\forall s \in \Sigma^*, t \in \Sigma_u^*, \sigma \in \Sigma_c, u \in \Pi(t).[st\sigma \in \mathcal{L}(E)]$ *e* $[su \in \mathcal{L}(E)] \Longrightarrow su\sigma \in \mathcal{L}(E)$. □

O operador $\Pi(s)$ é operador de permutação de uma palavra s, uma linguagem formada para todas as palavras obtidas por permutações (com possíveis repetições) de s. Uma especificação de coordenação deve ser controlável com relação à planta G. Além disso, se um evento σ está habilitado após uma cadeia $st \in \mathcal{L}(E)$ que termina com evento não controlável, se existe uma outra cadeia u também composta pelos mesmos eventos não controláveis de t, então σ estará habilitado após a cadeia $su \in \mathcal{L}(E)$.

Com a Definição 2 e a Definição 3 é possível estabelecer a Proposição 4.1:

[(Vilela & Pena, 2016)]

Seja a planta G um sistema de produção segundo a Definição 2. Seja E uma especificação de coordenação segundo a Definição 3 e $P_{\Sigma \to \Sigma_c} : \Sigma^* \to \Sigma_c^*$ uma projeção natural. Se $K = G||E$ é controlável $(S = SupC(K, G) = K)$ então $P_{\Sigma \to \Sigma_c}(S)$ tem a propriedade do observador.

A Proposição 4.1 diz que se a planta G é modelada como um sistema de produção e as especificações (composição síncrona de todas) como especificações de coordenação então $P_{\Sigma \to \Sigma_c}(S)$ possui a propriedade do observador.

Caso a especificação E não seja controlável, então obtém-se o supervisor reduzido e usado como especificação de coordenação ao invés de E, conforme estabelecido em (Pena et al., 2009).

Para encontrar o conjunto de todos os caminhos legais de eventos controláveis que produzem K produtos, calcula-se $E_q||P_{\Sigma \to \Sigma_c}(S)$. Define-se E_q na Definição 4.

Definicão 4 (Rafael (2018))**.** *Um autômato* $E_q = (-, \Sigma_{E_q}, -, -, -)$ *é uma especificação de quantidade, para a produção de um lote de* k *produtos, tem-se as seguintes características:*

1. $\Sigma_{E_q} = \{\sigma\}$*, onde* $\sigma \in \Sigma_c$ *e é o último evento controlável da sequência de produção que produz 1 produto;*

2. $\mathcal{L}_m(E_q) = v \subseteq \Sigma_{E_q}^*$*, onde* $\mid v \mid = k$ *e* k *representa a quantidade de produtos.* □

O exemplo a seguir apresenta o resultado do Teorema 1.

Exemplo 5. *O Sistema de Manufatura Simplificado (SFS) é uma extensão do problema da pequena fábrica (Wonham, 2015). O sistema consiste de 4 máquinas (M_1, M_2, M_3, M_4) e 3 buffers de capacidade unitária (B_1, B_2, B_3), Figura 4.1. Como restrição de segurança, deve-se garantir que não haja a ocorrência de underflow ou overflow nos buffers. O conjunto de eventos controláveis $\Sigma_c = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ marcam o início de um processo e o conjunto de eventos não controláveis $\Sigma_u = \{\beta_1, \beta_2, \beta_3, \beta_4\}$ representam o fim da operação de cada máquina.*

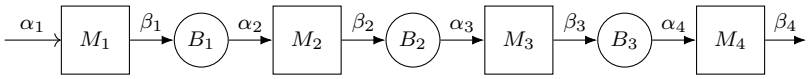
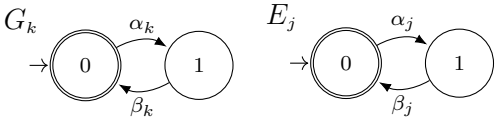


Figura 4.1: Pequena fábrica.

As máquinas são modeladas como autômatos de 2 estados G_k, onde o estado 0 é o estado de inatividade e o estado 1 representa o estado de operação ou de trabalho. As máquinas e os buffers são apresentados na Figura 4.2.



(a) Máquina M_k. (b) Especificação dos buffers E_j.

Figura 4.2: G_k é o modelo das máquinas ($k = 1, \ldots, 4$) e E_j das especificações ($j = 1, 2, 3$).

O problema de controle é evitar underflow e overflow dos buffers unitários, neste caso o buffer E_j é considerado vazio quando está no estado 0 e cheio no estado 1. Uma vez as plantas (G_1, G_2, G_3, G_4) e as especificações (E_1, E_2, E_3) são definidas, elas são usadas para computar o supervisor S (54 estados e 120 transições) e então a projeção natural $P_{\Sigma \to \Sigma_c}(S)$ sobre os eventos controláveis é aplicada. As duas operações foram feitas usando UltraDES (Alves et al., 2017). A Figura 4.3 apresenta a projeção do supervisor (8 estados e 12 transições).

Para gerar todas as cadeias de eventos controláveis que permitem a produção de 2 produtos no SFS, primeiro faz-se a composição paralela entre a projeção natural ($P(S)$) e a especificação de quantidade (E_q). Para um lote de tamanho 2, a especificação de quantidade é mostrada na Figura 4.4.

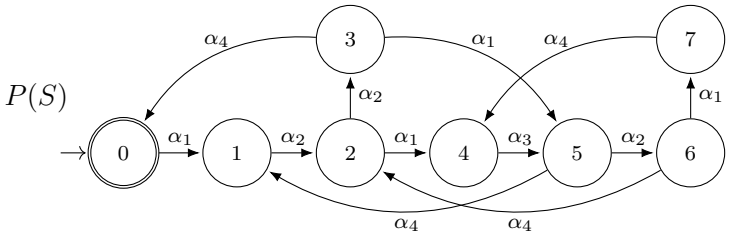


Figura 4.3: Projeção natural do supervisor S sobre o conjunto dos eventos controláveis no SFS.

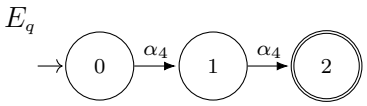


Figura 4.4: Especificação de quantidade (E_q) para um lote de tamanho 2, Definição 4.

A projeção do supervisor da Figura 4.3 possui a propriedade do observador segundo o algoritmo de verificação proposto por Pena et al. (2014). O autômato resultante $P_{Eq} = E_q||P_{\Sigma\to\Sigma_c}(S)$ *(Figura 4.5), representa todas as sequências possíveis de eventos controláveis, que uma vez executados, levam à fabricação de 2 produtos no SFS.*

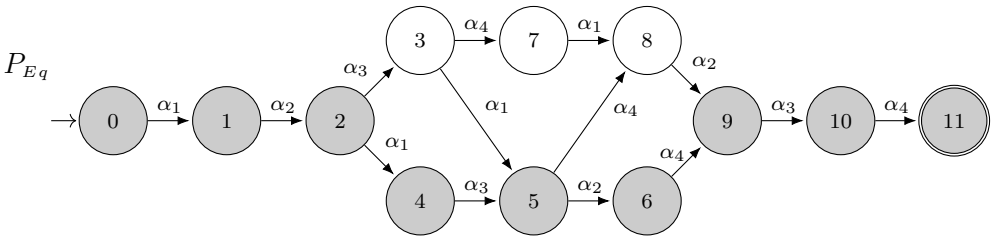


Figura 4.5: Autômato P_{Eq}.

Escolhe-se uma sequência $s_{ot} \in P_{Eq}$*, sendo* $s_{ot} = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\alpha_4\alpha_3\alpha_4$ *(sequência que passa pelos estados em cinza). De acordo com o Teorema 1, a linguagem A é controlável com respeito à* $\mathcal{L}(G)$*. O autômato A pode-se calcular aplicando:* $A = P^{-1}_{\Sigma\to\Sigma_c}(s_{ot}) \cap S$*, Figura 4.6. A adição dos eventos não controláveis (interseção com S) na sequência* s_{ot} *não afeta a controlabilidade da linguagem A.*

As condições apresentadas no Teorema 1 são suficientes para que a linguagem recuperada A para qualquer sequência $s_{ot} \in P_{\Sigma\to\Sigma_c}(S)$ *seja controlável com relação à* $\mathcal{L}(G)$*, mas não são necessárias.*

Essa linguagem A é vista como um novo supervisor, mas que implementa a linguagem composta de todas as cadeias de $\mathcal{L}_m(S/G)$ que projetam em s_{ot}. Como a linguagem A é

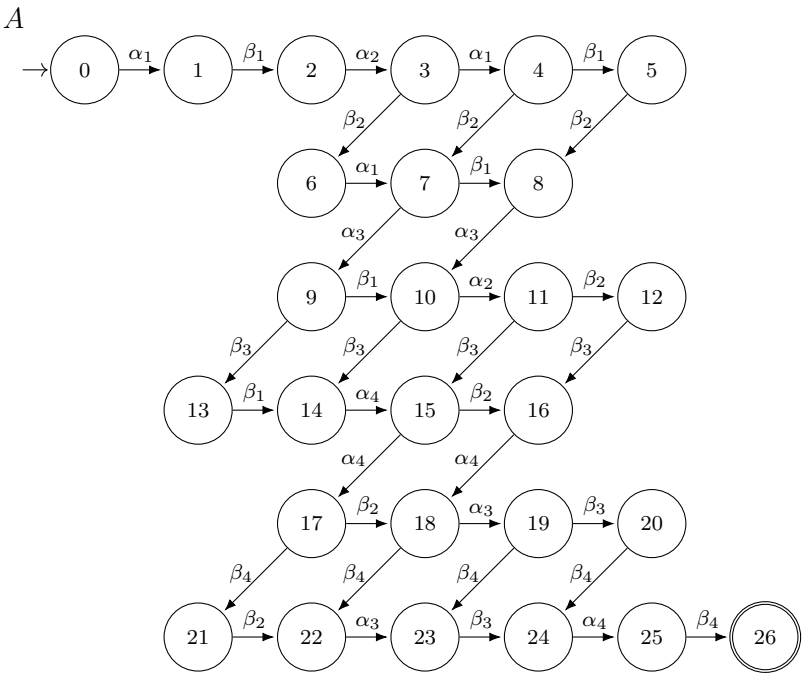


Figura 4.6: Autômato que implementa A.

controlável, pode-se assegurar a implementação da sequência s_{ot}. A condição principal do Teorema 1 é que a $P_{\Sigma\to\Sigma_c}(S)$ seja uma PO-abstração.

A seguir, apresenta-se um exemplo em que as condições para a aplicação do Teorema 1 não estão presentes.

Exemplo 6. *Considere o sistema de manufatura (Zhang et al., 2017) mostrado na Figura 4.7, composto por 2 máquinas (M_1 e M_2) e uma unidade de teste (TU), conectados por um buffer de capacidade 3 (B_1) e um buffer unitário (B_2). A máquina M_1 modelada pelo autômato da Figura 4.8(a) recebe o insumo bruto (evento α_1) e após processá-lo o deposita no buffer B_1 (evento β_1). A máquina M_2 retira as peças (evento α_2) de trabalho do buffer de entrada B_1, processa e deposita as mesmas (evento β_2) no buffer de saída (B_2). A máquina TU, modelada pelo autômato da Figura 4.8(b), testa a qualidade das peças (evento α_3) que foram retiradas de B_2 e aprova (evento β_3) ou rejeita (evento β_4). Desse modo, um insumo bruto deve ser processado pelos 3 subsistemas, em sequência para que torne um produto final.*

A restrição deste problema é evitar underflow ou overflow nos buffers B_1 e B_2 modelados pelos autômatos da Figura 4.9.

Para aplicar o resultado do Teorema 1, é necessário que a projeção natural do com-

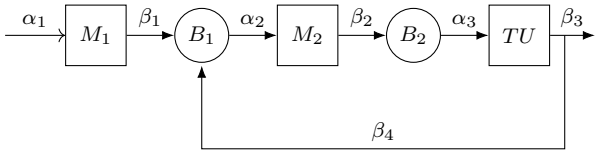
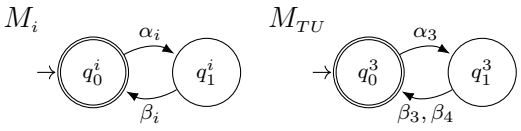
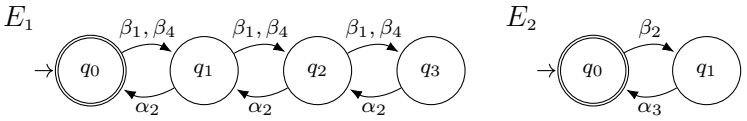


Figura 4.7: Sistema de manufatura com retrabalho.



(a) M_i, $i = \{1, 2\}$. (b) M_{TU}.

Figura 4.8: Modelos das máquinas.



(a) Especificação do *Buffer* B_1. (b) Especificação do *Buffer* B_2.

Figura 4.9: Modelos das especificações.

portamento em malha fechada, projetado no conjunto de eventos controláveis, tenha a propriedade do observador. No caso deste sistema, não acontece, o que é esperado já que a sequência para produção de uma peça depende de eventos não controláveis (β_3 e β_4), ou seja, $\alpha_1\alpha_2\alpha_3$ produz um produto, caso seja seguido de β_3. Alternativamente, quando falha o teste (β_4), a produção de um produto é dada pela sequência $\alpha_1\alpha_2\alpha_3\alpha_2\alpha_3$.

Os conceitos apresentados nesse capítulo e no Capítulo 2 fundamentam a contribuição principal deste trabalho, que será apresentada no próximo capítulo.

5

Resultados

O Teorema 1 apresentado no capítulo anterior refere-se à aplicação da abstração do comportamento em malha fechada pela síntese monolítica. Não obstante, a aplicação dessa abordagem está sujeita à condição principal de que a abstração do comportamento em malha fechada deve possuir a propriedade do observador. Por conseguinte, ao violar essa condição principal, a linguagem recuperada A não será garantidamente controlável com relação à $\mathcal{L}(G)$ porque em alguns do seus estados, eventos não controláveis podem ser desabilitados. Desse modo, a formulação do problema deste trabalho é apresentada na Seção 5.1.

5.1 Formulação do Problema

Seja um sistema físico modelado por um autômato $G = (-, \Sigma, -, -, -)$, com $\mathcal{L}(G) \subseteq \Sigma^*$ e com alfabeto $\Sigma = \Sigma_c \cup \Sigma_u$. Seja o conjunto de restrições modeladas por E, a linguagem desejada para o sistema em malha fechada por $K = E||G$ e o supervisor $S = SupC(K, G)$. Seja $P : \Sigma^* \to \Sigma_c^*$ uma projeção natural. Caso $P_{\Sigma \to \Sigma_c}(S)$ não possua a propriedade do observador que é a condição principal para a aplicação do Teorema 1, propor outra abstração tal que seja possível resolver o problema de planejamento, ou seja, obter a partir desta abstração uma linguagem A que será controlável com relação à $\mathcal{L}(G)$.

Portanto, o objetivo deste trabalho é estender o resultado principal de Vilela & Pena (2016) para lidar com o caso em que a abstração do comportamento em malha fechada não é PO-abstração. A estratégia proposta consiste em estender o conjunto de eventos mantidos na projeção (Bravo et al., 2014), para obter uma PO-abstração e lidar com as modificações necessárias nas definições e teorema, causados pela presença de eventos não-controláveis.

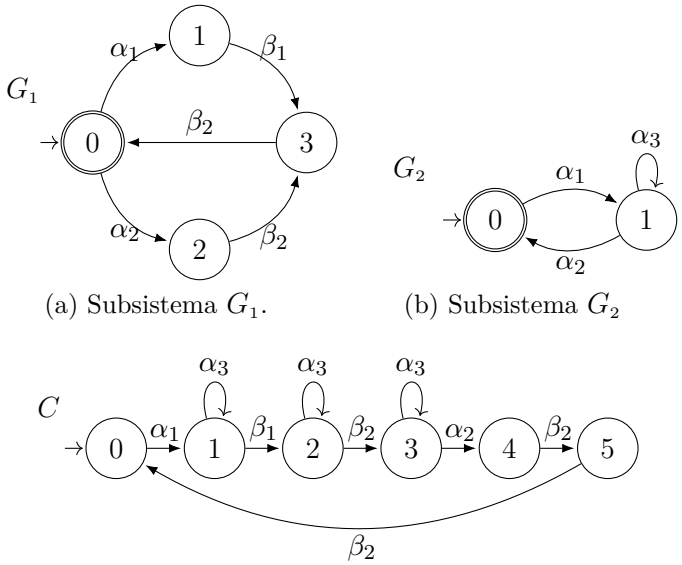
5.2 Definições

Antes de apresentar a principal contribuição deste trabalho, serão apresentadas algumas definições que auxiliarão a demonstração do mesmo. A seguir é apresentada a Definição 5, que calcula o rótulo de uma composição paralela entre dois autômatos.

Definicão 5 (Rótulo de uma Composição)**.** *Seja* $A_i = (-, \Sigma_i, \delta_i, q_i^0, -)$*, com* $i \in I = \{1, ..., m\}$ *e* $A = (Q, \Sigma, \delta, q_0, -)$ *obtido por* $A = \parallel_{i=1}^{m} A_i$*. Seja* $P_i : \Sigma^* \to \Sigma_i^*$*. A função que apresenta o rótulo de cada estado de uma composição síncrona é:* $r_A(q) = ((\delta_1(q_1^0, P_1(s)), \delta_2(q_2^0, P_2(s)), \ldots, \delta_m(q_m^0, P_m(s)))$ *onde* $s \in \Sigma^*$ *e* $q = \delta(q_0, s)$ *para* $q \in Q$*.* □

A função rótulo, segundo a Definição 5, apresenta a informação dos estados de cada autômato A_i que compõem cada estado $q \in Q$ de A. Ela pode ser aplicada a qualquer autômato resultante de uma composição. A fim de ilustrar essa definição, o Exemplo 7 é apresentado.

Exemplo 7. *Sejam os autômatos* $G_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, -), G_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, -)$ *e* $G = (Q_G, -, \delta_G, q_0^G, -)$ *apresentados no Exemplo 2 (são apresentados novamente na Fig. 5.1). Calcular o rótulo do estado* $4 \in Q_G$ *usando a Definição 5.*



(a) Subsistema G_1. (b) Subsistema G_2

(c) Autômato G que implementa $G_1||G_2$

Figura 5.1: Rótulo de uma composição.

O rótulo do estado 4 de G é:

$$\begin{aligned} r_G(4) &= (\delta_1(q_1^0, P_1(\alpha_1\beta_1\beta_2\alpha_2)), \delta_2(q_2^0, P_2(\alpha_1\beta_1\beta_2\alpha_2)) \\ &= (\delta_1(q_1^0, \alpha_1\beta_1\beta_2\alpha_2), \delta_2(q_3^0, \alpha_1\alpha_2)) \\ &= (2, 0) \end{aligned}$$

O rótulo $r_G(4)$ *do autômato G, pela Definição 5, é* $(2, 0)$ *sendo* $s = \alpha_1\beta_1\beta_2\alpha_2$ *tal que* $\delta_G(0, s) = 4$*. Portanto, para todo estado* $q \in Q_G$ *é possível obter* $r_G(q)$*:*

Tabela 5.1: Rótulos do autômato G.

Estado de G	s	$r(q)$
0	ϵ	$(0, 0)$
1	α_1	$(1, 1)$
2	$\alpha_1\beta_1$	$(3, 1)$
3	$\alpha_1\beta_1\beta_2$	$(0, 1)$
4	$\alpha_1\beta_1\beta_2\alpha_2$	$(2, 0)$
5	$\alpha_1\beta_1\beta_2\alpha_2\beta_2$	$(3, 0)$

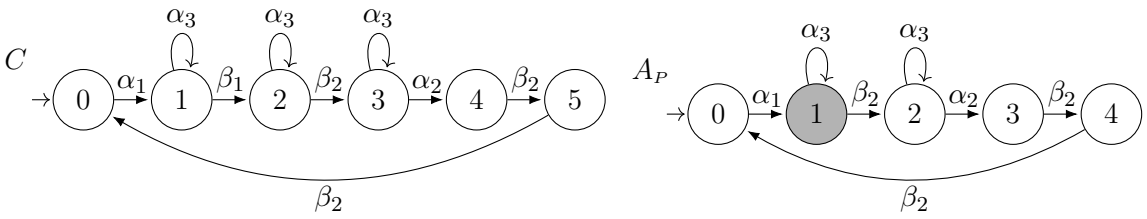
A Definição 5 pode ser usada para calcular o rótulo de cada estado do autômato que implemente $K = \mathcal{L}(G)||\mathcal{L}(E)$ que implementa o comportamento desejado pelo simples fato de K ser resultante de uma composição síncrona. Se K não é controlável, então a máxima sublinguagem controlável e não bloqueante $S = Sup\mathcal{C}(K, G)$ pode ser sintetizada e implementada por um supervisor (Ramadge & Wonham, 1989). Porém, como S é um subautômato de K, então os rótulos calculados para o autômato que implementa K são os mesmos para o autômato que implementa S (depois da eliminação dos maus estados de K).

A seguir apresenta-se a função que calcula o rótulo para cada estado de um autômato resultante de uma projeção natural.

Definição 6 (Rótulo de uma Projeção)**.** *Seja o autômato $A_p = P_{\Sigma_A \to \Sigma_B}(C)$ onde $A_p = (Q_p, -, \delta_p, q_0^P, -)$ e $C = (Q_c, -, \delta_c, q_0^C, -)$. Estabelece-se a função $d' : Q_p \to 2^{Q_C}$ com: $d'(\delta_p(q_0^P, t)) = \{q' \in \{\delta_c(q_0^C, s) \mid s \in P^{-1}(t) \cap \mathcal{L}(C)\}\}$. A função $d(d'(q))$ converte um conjunto de tuplas em uma tupla de conjuntos.* □

A função rótulo segundo a Definição 6, apresenta a informação dos estados do autômato C que compõem cada estado $q \in Q_p$ de A_p. Ela pode ser aplicada a qualquer autômato resultante da operação de uma projeção natural. A fim de aclarar essa definição o Exemplo 8 é apresentado.

Exemplo 8. *Seja o autômato $C = (Q_C, -, \delta_C, q_0^C, -)$ da Figura 5.2(a).*



(a) Autômato C. (b) Projeção natural $A_p = P(G)$.

Figura 5.2: Rótulo do estado de um autômato resultante de uma projeção.

O autômato $A_p = (Q_p, -, \delta_p, q_0^P, -)$ da Fig. 5.2 (b) implementa a linguagem $P_{\Sigma \to \Sigma_1}(C)$ com $\Sigma_1 = \{\alpha_1, \alpha_2, \beta_2\}$. O rótulo do estado 1, $d'(1) = d'(\delta_p(q_o^p, \alpha_1))$ de A_p é obtido identificando, inicialmente, quais estados de C que são alcançados por cadeias que projetam em α_1 (no

caso, $\alpha_1, \alpha_1\beta_1$*, que levam a* $\{1,2\} \subseteq Q_C$*). Portanto, o rótulo do estado* 1 *de* A_p *é calculado como:*

$$
\begin{aligned}
d'(\delta_p(q_0^P, \alpha_1)) &= \{q' \in \{\delta_C(q_0^C, s) \mid s \in P^{-1}(\alpha_1) \cap \mathcal{L}(C)\}\} \\
d'(1) &= \{q' \in \{\delta_C(q_0^C, s) \mid s \in \beta_1^* \alpha_1 \beta_1^* \cap \mathcal{L}(C)\}\} \\
d'(1) &= \{q' \in \{\delta_C(q_0^C, s) \mid s \in \{\alpha_1, \alpha_1\beta_1\}\}\} \\
d'(1) &= \{q' \in \{1,2\}\} \\
d'(1) &= \{1,2\}
\end{aligned}
$$

Por fim, a função $d(d'(q))$ *converte o conjunto de tuplas* $\{1,2\}$ *em uma tupla de conjuntos, assim:* $d(d'(1)) = (\{1\},\{2\})$.

A seguir, apresenta-se a Definição 7, a qual define o sistema de produção adaptado de (Vilela & Pena, 2016). Esta definição diz respeito à forma com que o sistema é modelado.

Definicão 7 (Sistema de Produção)**.** *Seja* G *um sistema composto por* m *subsistemas. Cada subsistema é modelado por um autômato* $M_i = (Q_i, \Sigma_i, \delta_i, Q_i^m, q_i^0)$*,* $i \in I = \{1, ...m\}$ *tal que:*

i) $G = \prod_{i=1}^{m} M_i$*,* $\Sigma = \bigcup_{i=1}^{m} \Sigma_i$ *onde* $\Sigma_i = \Sigma_{i_c} \cup \Sigma_{i_u}$*;*

ii) $\Sigma_i \cap \Sigma_j = \emptyset$*,* $i, j \in I$ *e* $i \neq j$*;*

iii) M_i *é caracterizada por:*

- **a)** $\forall \sigma \in \Sigma_i$ *tal que* $\delta(q_i^0, \sigma) = q_a$*,* $\sigma \in \Sigma_{ic}$*;*
- **b)** $\exists w_{i_k} = \sigma_{i_k}^{\uparrow} t_{i_k}^{\downarrow} \in \mathcal{L}_m(M_i)$ *com* $k \geq 1$*,* $\sigma_{i_k}^{\uparrow} \in \Sigma_{i_c}$ *e* $t_{i_k}^{\downarrow} \in \Sigma_{i_u}$ *tal que* $\delta(q_i^0, \sigma_{i_k}^{\uparrow}) = q_a$*,* $\delta(q_a, t_{i_k}^{\downarrow}) = q_i^0$*, e* q_a *é denominado estado* ***ativo****;*

iv) $\Sigma_{\sigma\uparrow} = \{\sigma_{i_k}^{\uparrow} | \forall w_{i_k} \in \mathcal{L}_m(M_i), i = 1, \ldots, m\}$*,* $\Sigma_{t\downarrow} = \{t_{i_k}^{\downarrow} | \forall w_{i_k} \in \mathcal{L}_m(M_i), i = 1, \ldots, m\}$*;*

v) $\Sigma_{fix}^C = \{\sigma \in \Sigma_c | \forall t \in \Sigma_u \backslash \Sigma_{t\downarrow} \Rightarrow \exists \sigma \in \Sigma_c$*, que retorna à produção*$\}$*;*

vi) $M^{Ru} = \{M_i | \Sigma_i \cap \Sigma_{Ru} \neq \emptyset\}$*;*

vii) $\Sigma_c = \Sigma_{\sigma\uparrow} \cup \Sigma^C_{fix}$, *onde os alfabetos* $\Sigma_{\sigma\uparrow}$ *e* Σ^C_{fix} *não necessariamente são disjuntos.* □

A definição de Sistema de Produção introduzida por (Vilela & Pena, 2016) e apresentada na Seção 4.1 (Definição 2) se diferencia da Definição 7 no que diz respeito às características que deve ter cada subsistema M_i (item *iii*). No item *iii* da Definição 2 cada subsistema deve possuir a propriedade do observador. Essa caraterística não é relevante na Definição 7, pois o objetivo neste trabalho é trabalhar com sistemas que não são PO-abstração.

Em palavras, o Sistema de Produção da Definição 7 é composto por autômatos com conjuntos de eventos disjuntos (item *ii*)). Estes autômatos são sempre iniciados com um evento controlável ($\sigma^{\uparrow}_{i_k}$) e finalizados com um evento não controlável ($t^{\downarrow}_{i_k}$) (item *iii*)b)). w_{i_k} é uma cadeia que representa um ciclo de trabalho de M_i e o índice $k \in \mathbb{Z}^+$ representa o número de tarefas que o subsistema M_i realiza (item *iii*)b)). Em cada subsistema M_i existe pelo menos uma cadeia $w_{i_k} = \sigma^{\uparrow}_{i_k} t^{\downarrow}_{i_k}$. O evento $\sigma^{\uparrow}_{i_k}$ (ligar) dispara o início de w_{i_k} e o evento $t^{\downarrow}_{i_k}$ (desligar) causa o final de w_{i_k}. O item *b*) nesta definição, foi adaptado de (Bravo et al., 2018).

O conjunto $\Sigma_{\sigma\uparrow}$ contém todos os eventos controláveis de liga e $\Sigma_{t\downarrow}$ contém todos os eventos não controláveis que causam o final de uma tarefa (item *iv*)). O conjunto Σ^C_{fix} do item *v*) é um conjunto de eventos controláveis tal que para todo evento não controlável pertencente ao conjunto $\Sigma_u \backslash \Sigma_{t\downarrow}$ existe pelo menos um evento controlável que leva o sistema de volta à condição típica de funcionamento. $\Sigma_u \backslash \Sigma_{t\downarrow}$ são eventos que podem ocorrer no sistema mas não contribuem para a produção de um produto (Seção 5.4). Por último, o conjunto M^{Ru} (item *v*)) contém um conjunto de máquinas M_i que apresentam eventos do conjunto Σ_{Ru}, onde Σ_{Ru} é o conjunto de eventos acrescentados na projeção para obter a PO (Seção 3.5).

A seguir, apresenta-se a Definição 8, a qual apresenta o autômato que implementa uma especificação de quantidade para a produção de um lote de K produtos. A Definição 8 é uma modificação da Definição 4 no que diz respeito ao conjunto de eventos Σ_{E_p}.

Definicão 8. *Um autômato* $E_p = (-, \Sigma_{E_p}, -, -, -)$ *é uma especificação de produção para a produção de um lote de k produtos e tem as seguintes características:*

1. $\Sigma_{E_p} = \{\sigma\}$, *sendo* σ *o último evento da sequência de produção que produz 1 produto,*

seja controlável ou não controlável, segundo o sistema de produção.

2. $\mathcal{L}_m(E_p) = v \subseteq \Sigma^*_{E_p}$, $\mid v \mid = k$, *onde* $k \in \mathbb{Z}^+$ *representa a quantidade de produtos.* □

Como foi dito na Seção 3.5, o algoritmo proposto por (Bravo et al., 2014) busca um conjunto de instâncias de eventos não-controláveis que não podem ser apagados na projeção natural posto que causam a violação da propriedade do observador. Portanto, neste trabalho, esse conjunto de eventos não-controláveis será chamado de Σ_{Ru} com $\Sigma_{Ru} \subseteq \Sigma_u$, que, ao ser mantido na projeção, torna a mesma uma PO-Abstração. Por conseguinte, o novo conjunto de eventos na projeção do comportamento em malha fechada é $\Sigma_c \cup \Sigma_{Ru}$ de modo que $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)$ seja uma PO-abstração.

No resultado anterior (Teorema 1, (Vilela & Pena, 2016)), escolhia-se uma sequência s_{o_t} qualquer da projeção do comportamento em malha fechada e constrói-se a partir dela, uma linguagem $A = P^{-1}_{\Sigma\rightarrow\Sigma_c}(s_{o_t}) \cap S$ que será controlável com relação a $\mathcal{L}(G)$. Nesta extensão, é necessário construir uma linguagem $\mathcal{F}^{\uparrow}$ invés de só escolher uma sequência s_{o_t} da projeção do comportamento em malha fechada para obter uma linguagem $A = P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S$ que será controlável com relação a $\mathcal{L}(G)$. Para mostrar esta ideia da linguagem $\mathcal{F}^{\uparrow}$, o Exemplo 9 é apresentado.

Exemplo 9. *Seja o Sistema de Manufatura Simplificado (SFS) ou pequena fábrica apresentado em (Vilela & Pena, 2016). O sistema consiste de 2 máquinas (M_1 e M_2) e 1 buffer de capacidade unitária (B_1), Figura 5.3. A máquina M_1 modelada pelo autômato da Figura 5.3(a) terá um evento (β_3) que levará o subsistema para um estado de quebra e um evento de conserto (α_3) que retornará o subsistema novamente para o estado de trabalho. A máquina M_2 não terá seus eventos de quebra e conserto representados, Figura 5.3(b).*

Como restrição de segurança, deve-se garantir que não haja a ocorrência de underflow ou overflow no buffer B_1, Figura 5.5. O conjunto de eventos controláveis são $\Sigma_c = \{\alpha_1, \alpha_2, \alpha_3\}$ e o conjunto de eventos não controláveis são $\Sigma_u = \{\beta_1, \beta_2, \beta_3\}$.

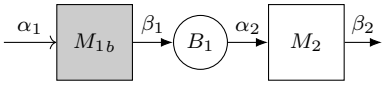
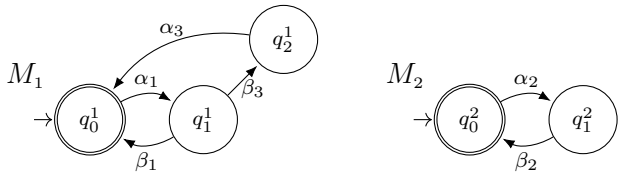


Figura 5.3: Pequena fábrica.



(a) Modelo da Máquina M_1 com quebra.

(b) Modelo da Máquina M_2.

Figura 5.4: Modelo das Máquinas.

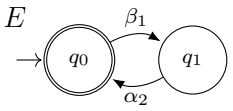


Figura 5.5: Especificação do buffer B_1.

Uma vez que a especificação e as plantas são definidas, elas são usadas para computar o supervisor S e a projeção natural $P_{\Sigma\to\Sigma_c}(S)$ sobre os eventos controláveis é aplicada. A Figura 5.6 apresenta o supervisor e a Figura 5.7 apresenta a projeção do supervisor. Para explicitar as sequências de eventos que levam à produção de um lote de tamanho 2, utiliza-se uma especificação de quantidade conforme a Definição 4, mostrada na Figura 5.8.

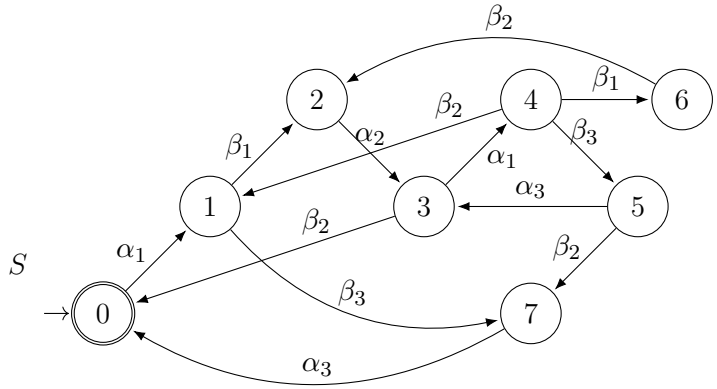


Figura 5.6: Supervisor.

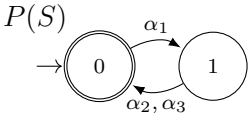


Figura 5.7: Projeção do Supervisor.

A projeção do comportamento em malha fecha mostrada na Figura 5.7 não possui a propriedade do observador. Faz-se então Trim sobre o resultado da composição paralela da abstração do supervisor e da especificação de quantidade. O resultado obtido é mostrado na Figura 5.9.

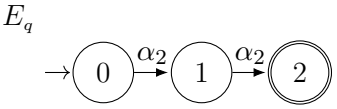


Figura 5.8: Especificação de quantidade para a produção de 2 produtos.

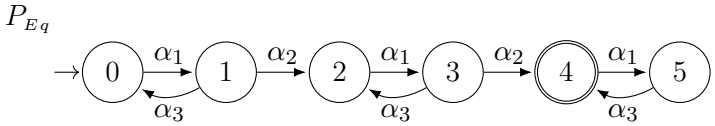


Figura 5.9: Composição paralela entre $P_{\Sigma\to\Sigma_c}(S)$ e a especificação de quantidade para 2 produtos.

O autômato da Figura 5.9 explicita todas as sequências de eventos controláveis que levam à produção de dois produtos. Os ciclos do evento α_3 representam o conserto da quebra na máquina M_1 causada pela ocorrência do evento β_4 mas que foi apagado pela projeção.

Depois de cada ocorrência do evento α_1, existem duas possibilidades de eventos acontecerem: a primeira delas é acontecer o evento α_3 que representa o conserto da falha na máquina M_1 e a segunda possibilidade é acontecer o evento α_2 que representa ligar a máquina M_2. Portanto, seja qual for o evento que vai acontecer (α_3 ou α_2), um deles terá que ser desabilitado. Do autômato da Figura 5.9, tem-se a seguinte sequência de produção:

$$s_1 = \alpha_1\alpha_2\alpha_1\alpha_2,$$

Supondo que a quebra/reparo não faz parte do comportamento de produção. Esse autômato será o modelo do sistema, que é uma das entradas do Planejador da Figura 2.2. O resultado do Teorema 1 será aplicado sobre esse autômato. Suponha que o Planejador tenha retornado como plano ótimo s_{o_t} a sequência s_1. Assim:

$$s_{o_t} = s_1 = \alpha_1\alpha_2\alpha_1\alpha_2$$

A sequência $s_{o_t} = \alpha_1\alpha_2\alpha_1\alpha_2$, não apresenta informação do evento α_3, o qual significa que o evento β_4 que foi apagado na projeção natural está sendo desabilitado. Essa sequência deveria apresentar o evento α_3 para não ter que desabilitar o evento β_4 que representa a ocorrência da quebra da máquina M_1 e o evento β_1 que representa a desliga da máquina

M_1.

O Teorema 1 diz que, caso $P(S)$ seja PO, a linguagem A, que é uma sublinguagem do comportamento em malha fechada que projeta para a sequência s_{o_t}, é controlável. Constrói-se $A = P^{-1}_{\Sigma\to\Sigma_c}(s_{ot}) \cap S$, representada pelo autômato mostrado na Figura 5.10 e percebe-se claramente que A não é controlável, evento β_3 desabilitado nos estados $1, 4$ e 7.

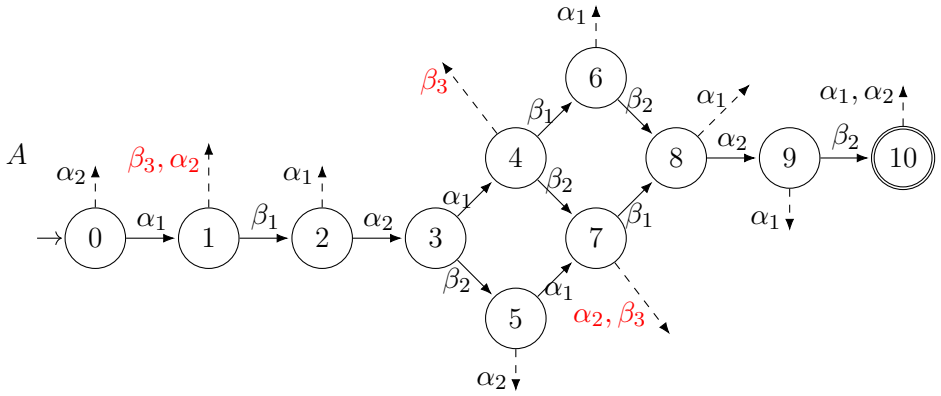


Figura 5.10: Linguagem A.

As setas tracejadas da Figura 5.10 representam as desabilitações de eventos. Como pode ser observado, o evento β_3 está sendo desabilitado nos estados $\{1, 4, 7\}$, mostrando assim a não controlabilidade da linguagem A em relação à planta. Portanto, com este exemplo fica evidenciado que simplesmente uma sequência não é suficiente para obter uma linguagem A controlável com relação à planta em sistemas onde sua $P_{\Sigma\to\Sigma_c}(S)$ não tem a propriedade do observador, portanto, é preciso acrescentar algumas informações na sequência s_{o_t}.

Como pode ser observado no autômato da Figura 5.10, os estados onde o evento β_3 está sendo desabilitado, são aqueles que estão depois da ocorrência do evento α_1 que pertence à máquina M_1 a qual contém a quebra. Em vista disso, a sequência $s_{o_t} = \alpha_1\alpha_2\alpha_1\alpha_2$ deve mostrar depois de cada ocorrência do evento α_1, os futuros dos eventos β_1 e β_3 os quais são diferentes. Quando essa informação é acrescentada na sequência $s_{o_t} = \alpha_1\alpha_2\alpha_1\alpha_2$, deixa de ser uma sequência e começa a tornar-se uma linguagem ($\mathcal{F}^{\uparrow}$).

O primeiro passo é usar o algoritmo OP-search detalhado na Seção 3.5 para obter uma projeção com a propriedade do observador, Figura 5.11. Pode-se observar na Figura 5.11 que os eventos β_1 e β_3 estão agora factíveis depois da ocorrência do evento α_1, os quais não estavam presentes na projeção do supervisor sem a propriedade do observador (Figura

5.7).

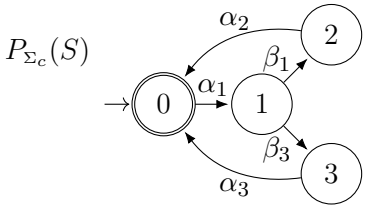


Figura 5.11: Projeção do supervisor com a propriedade do observador.

Depois da aplicação do algoritmo OP-search, os eventos não controláveis β_1 e β_3 não poderão ser apagados na projeção para garantir a propriedade do observador. Esses eventos não controláveis são o conjunto de eventos renomeados $\Sigma_{R_u} = \{\beta_1, \beta_3\}$, portanto, a nova projeção do comportamento em malha fechada com a propriedade do observador é definida com $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)$. Agora com esta projeção é possível escolher uma sequência s e construir uma linguagem $\mathcal{F}^{\uparrow}$. Este processo é descrito na próxima seção.

5.3 Algoritmo que Calcula a Linguagem $\mathcal{F}^{\uparrow}$

O seguinte algoritmo constrói uma linguagem $\mathcal{F}^{\uparrow}$ a partir de uma sequência $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ onde $T_{E_q} = Trim(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$ que será usada no Teorema 2 proposto neste trabalho. O algoritmo consiste de uma série de passos e procedimentos para gerar a linguagem $\mathcal{F}^{\uparrow}$, análoga à cadeia s_{o_t}, usada no Teorema 1. A seguir, os passos do algoritmo são apresentados.

Algorithm 1 Linguagem $\mathcal{F}^{\uparrow}$

Entrada: $s, \Sigma_{\sigma\uparrow}, \Sigma_{t\downarrow}, \Sigma^{C}_{fix}, \Sigma_{Ru}, M^{Ru}$.

1) ***Sequência*** s_{o_t}***:***

   (i) $s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s)$;

   (ii) $\mathcal{L}_m(G_{s_{o_t1}}) \leftarrow \{s_{o_t}\}$.

2) ***Auto-laços de*** Σ_{Ru} ***e*** Σ^{C}_{fix} ***em*** $G_{s_{o_t1}}$:

   (i) $\mathcal{L}_m(G_{s_{o_t2}}) = P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})\to\Sigma_{\sigma\uparrow}}(\mathcal{L}_m(G_{s_{o_t1}}))$;

   (ii.1) se $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$, então:
   $\mathcal{L}_m(G_{s_{o_t3}}) = P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\to(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})}(\mathcal{L}_m(G_{s_{o_t2}}))$.

   (ii.2) se $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$, então:
   Redefine-se δ de $G_{s_{o_t2}}$ tal que: $(\forall q_i \in Q_{s_{o_t2}})(\forall b \in \Sigma^{C}_{fix})$, se $b \notin \Gamma(q_i) \Longrightarrow \delta(q_i, b) = q_i$.
   Assim, obtém-se o autômato $G_{s_{o_t3}}$.

3) ***Auto-laços do evento*** $\sigma^{\uparrow}_i$ ***em cada estado*** $q \in Q$ ***de*** $G_{s_{o_t3}}$ ***onde o subsistema*** $M_i \in M^{Ru}$ ***está no estado ativo*** q_a:

   (i) Redefine-se δ de $G_{s_{o_t3}}$, para incluir $\delta(q, \sigma^{\uparrow}_i) = q$, se o i-ésimo elemento da tupla de conjuntos obtida fazendo $d(d'(q))$ inclui o estado $q_a \in Q_i$ e $M_i \in M^{Ru}$. Assim, obtém-se o autômato $G_{s_{o_t4}}$.

   (ii) $\mathcal{F}^{\uparrow} \leftarrow \mathcal{L}_m(G_{s_{o_t4}})$.

O autômato $T_{E_q} = Trim(P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$, é resultado da composição paralela entre o autômato que implementa a projeção natural do comportamento em malha fechada $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ com a propriedade do observador e a especificação de produção E_p (Definição 8). A linguagem $\mathcal{F}^{\uparrow}$ é construída em três procedimentos. O passo (1) constrói o autômato $G_{s_{o_t1}}$ a partir da sequência $s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s)$ onde $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ (s é uma entrada do algoritmo). A sequência s está composta apenas por eventos de liga $\Sigma_{\sigma\uparrow}$ e desliga $\Sigma_{t\downarrow}$ (lembrando que os eventos de $\Sigma_{t\downarrow}$ são só os que fazem interseção com Σ_{Ru}).

O passo (2) é subdividido em dois subpassos. No subpasso 2(i) faz-se uma projeção inversa gerando a linguagem $\mathcal{L}_m(G_{s_{o_t2}}) = P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})\to\Sigma_{\sigma\uparrow}}(\mathcal{L}_m(G_{s_{o_t1}}))$. O subpasso 2($ii$) que acrescenta autolaços de Σ^{C}_{fix}, é subdividido em dois casos:

O primeiro caso é se $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$, então faz-se, $P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\to(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})}(\mathcal{L}_m(G_{s_{o_t2}})) = \mathcal{L}_m(G_{s_{o_t3}})$.

O segundo caso é se $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$, então redefine-se δ de $G_{s_{o_t2}}$ tal que: $(\forall q_i \in Q_{s_{o_t2}})(\forall b \in$

Σ^{C}_{fix}), se $b \notin \Gamma(q_i) \Longrightarrow \delta(q_i, b) = q_i$. Assim, obtém-se o autômato $G_{s_{o_t3}}$.

Seja o caso $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, o caso $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$, significa que há eventos de Σ^{C}_{fix} na cadeia s. O caso $2(ii.1, ii.2)$ do algoritmo acrescenta autolaços de Σ^{C}_{fix}, sem tornar o autômato não determinístico. O autômato $G_{s_{o_t3}}$ é obtido ao acrescentar autolaços de Σ^{C}_{fix} seja pelo caso $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, $\Sigma^{C}_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$ no autômato $G_{s_{o_t2}}$, o qual significa que a linguagem do autômato $G_{s_{o_t2}}$ continua contida na linguagem do autômato $G_{s_{o_t3}}$.

Por último, o passo (3) acrescenta auto-laços do evento $\sigma^{\uparrow}_{i}$ em cada estado $q \in Q$ do autômato $G_{s_{o_t3}}$ ($G_{s_{o_t3}}$ gerado no passo anterior) onde o subsistema $M_i \in M^{Ru}$ está no estado ativo q_a. Para saber se o subsistema $M_i \in M^{Ru}$ está no estado ativo basta olhar para a tupla de conjuntos do rótulo $d(d'(q))$ em cada estado $q \in Q$. Este último passo, redefine a função δ de $G_{s_{o_t3}}$ para incluir $\delta(q, \sigma^{\uparrow}_{i}) = q$. A linguagem obtida no final do processo é chamada de linguagem $\mathcal{F}^{\uparrow}$.

Com o propósito de mostrar este procedimento do Algoritmo, retoma-se o Exemplo 9. A projeção do comportamento em malha fechada do Sistema de Manufatura Simplificado (Exemplo 9) com a propriedade do observador depois de aplicar o algoritmo OP-search é apresentada na Figura 5.12.

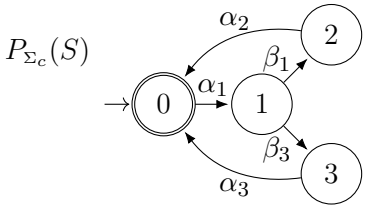


Figura 5.12: Projeção do supervisor com a propriedade do observador.

Observe que foi necessário para obter a propriedade do observador, incluir os eventos β_1 e β_3 na projeção. Estes eventos definem futuros diferentes e portanto devem ser mostrados. O autômato da projeção é composto com E_p e obtém-se o autômato que explicita todas as sequências de eventos que levam à produção de dois produtos, Figura 5.13.

O autômato T_{E_q} apresenta todas as sequências que contribuem com a produção de dois produtos (unicamente com eventos de $(\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})$) e sequências que não contribuem para a produção de dois produtos (sequências onde o evento β_3 de quebra da máquina M_1 está presente). Neste exemplo só existe uma sequência $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ que contribui

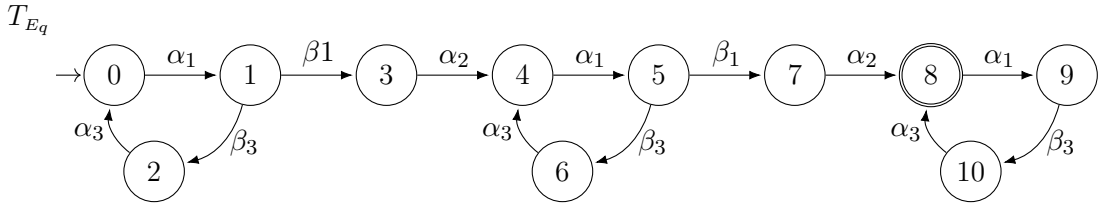


Figura 5.13: Autômato T_{E_q} que implementa $Trim(P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$.

com a produção de dois produtos:

$$s = \alpha_1\beta_1\alpha_2\alpha_1\beta_1\alpha_2.$$

A seguir a aplicação do algoritmo 1 para construir a linguagem $\mathcal{F}^{\uparrow}$ é realizada. Faz-se a projeção da sequência s sobre o alfabeto $\Sigma_{\sigma\uparrow}$ e assim obtém-se a sequência $s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s)$ (passo 1(i)). Obtém-se o autômato $G_{s_{o_t1}}$ e a linguagem $\mathcal{L}_m(G_{s_{o_t1}}) \leftarrow \{s_{o_t}\}$ da sequência s_{o_t} (passo 1(ii)), assim,

$$s = \alpha_1\beta_1\alpha_2\alpha_1\beta_1\alpha_2$$
$$s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s) = \alpha_1\alpha_2\alpha_1\alpha_2.$$

O autômato que implementa a sequência s_{o_t} é mostrado na Figura 5.14.

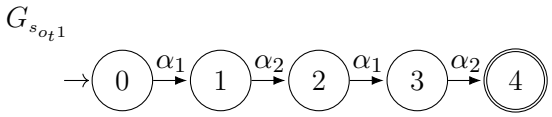


Figura 5.14: Autômato $G_{s_{o_t1}}$ que implementa a sequência s_{o_t} (passo 1(ii)).

O próximo passo (passo 2 do algoritmo) é acrescentar auto-laços do conjunto Σ^C_{fix} e Σ_{Ru} no autômato $G_{s_{o_t1}}$. O conjunto $\Sigma^C_{fix} = \{\sigma \in \Sigma_c | \forall t \in \Sigma_u \backslash \Sigma_{t\downarrow} \Rightarrow \exists \sigma \in \Sigma_c, \text{que retorna à produção}\}$ está composto de eventos controláveis tal que para todo evento t pertencente a $\Sigma_u \backslash \Sigma_{t\downarrow}$ existe pelo menos um evento σ pertencente ao conjunto de eventos controláveis que retorna um produto à linha de produção (Def.7 v). No exemplo, $\Sigma^C_{fix} = \{\alpha_3\}$ pois depois do evento β_3 o evento controlável que retorna à produção é α_3, como pode ser observado na Figura 5.13. No exemplo tem-se que $\Sigma_{Ru} = \{\beta_1, \beta_3\}$.

O passo (2) está subdividido em dois subpassos. O subpasso 2(i) tá definido como, $P^{-1}_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{Ru})\to\Sigma_{\sigma\uparrow}}(\mathcal{L}_m(G_{s_{o_t1}}))$ onde autolaços do conjunto Σ_{Ru} são acrescentados originado a

linguagem $\mathcal{L}_m(G_{s_{o_t2}})$. o autômato $G_{s_{o_t2}}$ é apresentado na Figura 5.15.

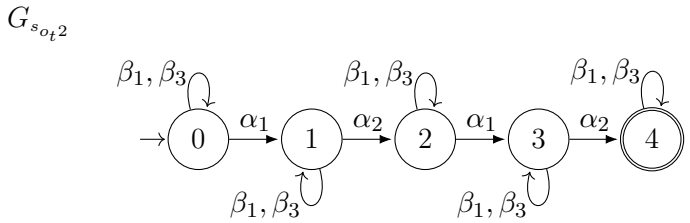


Figura 5.15: Autômato $G_{s_{o_t2}}$ que apresenta Autolaços de Σ_{Ru}.

O subpasso 2(ii) é subdividido em dois casos mas este exemplo executa o caso (ii.1), onde $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$, então faz-se, $P^{-1}_{(\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru} \cup \Sigma^C_{fix}) \to (\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru})}(\mathcal{L}_m(G_{s_{o_t2}})) = \mathcal{L}_m(G_{s_{o_t3}})$. Sendo assim, tem-se que o autômato $G_{s_{o_t3}}$ é:

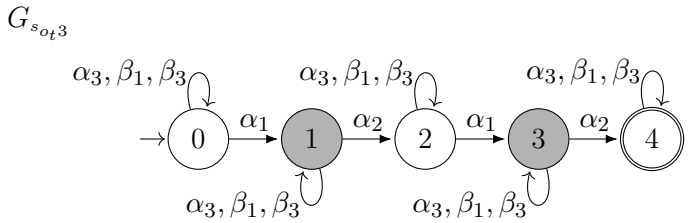


Figura 5.16: Autômato $G_{s_{o_t3}}$ que apresenta Autolaços de Σ^C_{fix} e Σ_{Ru}.

Por último, o passo (3) do algoritmo e o mais importante, pois consiste em identificar, em quais estados $q \in Q$ de $G_{s_{o_t3}}$ as máquinas $M_i \in M_{Ru}$ (M_{Ru} segundo Def.7) estão em estados ativos q_a (q_a segundo Def.7) pela aplicação da função rótulo $d(d'(q))$ e acrescentar auto-laços com os eventos $\sigma^{\uparrow}_{i_k}$. É importante determinar primeiramente neste passo o conjunto M^{Ru} e o rótulo $d(d'(q))$.

O conjunto $M^{Ru} = \{M_i | \Sigma_i \cap \Sigma_{Ru} \neq \emptyset\}$ no passo (3) está composto pelos subsistemas M_i onde os eventos de Σ_{Ru} estão definidos. Portanto, tem-se que $M^{Ru} = \{M_1\}$. Em seguida, determina-se o rótulo $d(d'(q))$ para saber em quais estados o subsistema M_1 está em estado ativo q_a, ou seja, onde o terceiro elemento da tupla de conjuntos possui o estado q^1_1, Tabela 5.2 terceira coluna. Porém, tem-se que os auto-laços acrescentados no autômato $G_{s_{o_t3}}$ serão do evento $\sigma^{\uparrow}_{1_1} = \alpha_1$.

Percebe-se no autômato da Figura 5.16 que os auto-laços do evento $\sigma^{\uparrow}_{1_1} = \alpha_1$ serão acrescentados nos estados $q = 1$ e $q = 3$ pois o primeiro elemento da tupla de conjuntos está

Tabela 5.2: Aplicação de $d'(q)$ e $d(d'(q))$.

q	$d'(q)$	$d(d'(q))$
0	$\{(q_0^1, q_0^2)\}$	$(\{q_0^1\}, \{q_0^2\})$
1	$\{(q_1^1, q_0^2)\}$,$\{(q_0^1, q_0^2)\}$	$(\{q_1^1, q_0^1\}, \{q_0^2\})$
2	$\{(q_0^1, q_1^2)\}$, $\{(q_0^1, q_0^2)\}$	$(\{q_0^1\}, \{q_1^2, q_0^2\})$
3	$\{(q_1^1, q_0^2)\}$, $\{(q_0^1, q_0^2)\}$	$(\{q_1^1, q_0^1\}, \{q_0^2\})$
4	$\{(q_0^1, q_1^2)\}$, $\{(q_0^1, q_0^2)\}$	$(\{q_0^1\}, \{q_1^2, q_0^2\})$

em estado ativo q_1^1 (estados em cinza da Tabela 5.2). Finalmente, o autômato resultante $G_{s_{o_t 4}}$ da execução do passo (3) do algoritmo é apresentado na Figura 5.17, onde a linguagem marcada do autômato $G_{s_{o_t 4}}$ é a linguagem $\mathcal{F}^{\uparrow}$, assim.

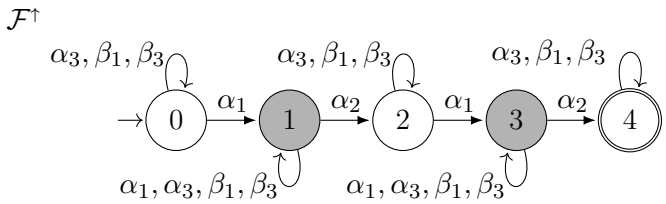


Figura 5.17: Autômato que implementa $\mathcal{F}^{\uparrow} \leftarrow \mathcal{L}_m(G_{s_{o_t 4}})$.

A seguir são apresentados um lema e um corolário que ajudam na demostração do teorema proposto.

Lema 1. *Seja uma cadeia $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$. Seja a linguagem $\mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ que é obtida a partir do Algoritmo 1, onde $\mathcal{F}^{\uparrow}$ é construída a partir de s. $\forall a \in S \subseteq \Sigma^*$ tal que $P_{\Sigma \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))}(a) = s$, então $P_{\Sigma \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))}(a) \in \mathcal{F}^{\uparrow}$.* △

Demostração. Como $s \in T_{E_q}$, pode-se dizer que $s \in \mathcal{L}_m(T_{E_q}) \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$. Por outro lado $s \in (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$. Assim:

$$s \in [(\Sigma_c \cap \Sigma_{\sigma\uparrow}) \cup (\Sigma_{Ru} \cap \Sigma_{t\downarrow})]^* = [\Sigma_{\sigma\uparrow} \cup (\Sigma_{Ru} \cap \Sigma_{t\downarrow})]^* \tag{5.1}$$

$\mathcal{F}^{\uparrow}$ é construído a partir de s em 3 passos. No primeiro passo:

$$P_{(\Sigma_{\sigma\uparrow} \cup (\Sigma_{Ru} \cap \Sigma_{t\downarrow})) \to \Sigma_{\sigma\uparrow}}(s) = s_{o_t}, \quad \text{passo } 1(i) \text{ do algoritmo.} \tag{5.2}$$

O segundo passo é subdividido em dois subpassos.

Subpasso i). $P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(s_{o_t}) = \mathcal{L}_m(G_{s_{o_t2}})$, passo 2($i$) do algoritmo. Substituindo s_{o_t} pela equação 5.2, então temos que, $P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(P_{(\Sigma_{\sigma^{\uparrow}}\cup(\Sigma_{Ru}\cap\Sigma_{t\downarrow}))\rightarrow\Sigma_{\sigma^{\uparrow}}}(s)) = \mathcal{L}_m(G_{s_{o_t2}})$. Portanto, tem-se que:

$$s \in P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(P_{(\Sigma_{\sigma^{\uparrow}}\cup(\Sigma_{Ru}\cap\Sigma_{t\downarrow}))\rightarrow\Sigma_{\sigma^{\uparrow}}}(s)) = \mathcal{L}_m(G_{s_{o_t2}}). \tag{5.3}$$

Já que $\Sigma_{Ru}\cap\Sigma_{t\downarrow}\subseteq\Sigma_{Ru}$. Assim:

$$s \in \mathcal{L}_m(G_{s_{o_t2}}). \tag{5.4}$$

O subpasso ii), é subdividido em dois casos:

O primeiro caso é se $\Sigma^{C}_{fix}\cap\Sigma_{\sigma^{\uparrow}} = \emptyset$, então faz-se:

$$P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\rightarrow(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})}(\mathcal{L}_m(G_{s_{o_t2}})) = \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.5}$$

Substituindo $P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(s_{o_t}) = \mathcal{L}_m(G_{s_{o_t2}})$ em (5.5), então temos que:

$$P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\rightarrow(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})}(P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(s_{o_t})) = \mathcal{L}_m(G_{s_{o_t3}}) \tag{5.6}$$

Usando a propriedade (1) das projeções extraída de Cassandras & Lafortune (2009), onde $t\in L\rightarrow t\in P^{-1}(L)$ em (5.6), então, é sempre verdade que:

$$s_{o_t} \in P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\rightarrow(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})}(P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(s_{o_t})) = \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.7}$$

$$s_{o_t} \in \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.8}$$

Substituindo (5.3) em (5.5), tem-se que:

$$s \in P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru}\cup\Sigma^{C}_{fix})\rightarrow(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})}(P^{-1}_{(\Sigma_{\sigma^{\uparrow}}\cup\Sigma_{Ru})\rightarrow\Sigma_{\sigma^{\uparrow}}}(P_{(\Sigma_{\sigma^{\uparrow}}\cup(\Sigma_{Ru}\cap\Sigma_{t\downarrow}))\rightarrow\Sigma_{\sigma^{\uparrow}}}(s))) = \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.9}$$

O segundo caso é se $\Sigma^{C}_{fix}\cap\Sigma_{\sigma^{\uparrow}}\neq\emptyset$.

Seja o caso $\Sigma^{C}_{fix}\cap\Sigma_{\sigma^{\uparrow}} = \emptyset$ ou, o caso $\Sigma^{C}_{fix}\cap\Sigma_{\sigma^{\uparrow}}\neq\emptyset$, significa que há eventos de Σ^{C}_{fix} na cadeia s, passo 2(ii) do algoritmo. O algoritmo acrescenta autolaços de Σ^{C}_{fix}, sem tornar o

autômato não determinístico. O autômato $G_{s_{o_t3}}$ é obtido ao acrescentar autolaços de Σ_{fix}^C seja pelo caso $\Sigma_{fix}^C \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, $\Sigma_{fix}^C \cap \Sigma_{\sigma\uparrow} \neq \emptyset$ no autômato $G_{s_{o_t2}}$, o qual significa que a linguagem do autômato $G_{s_{o_t2}}$ continua contida na linguagem do autômato $G_{s_{o_t3}}$. Sendo verdade o anteriormente dito, então temos que:

$$\mathcal{L}_m(G_{s_{o_t2}}) \subseteq \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.10}$$

Como $s \in G_{s_{o_t2}}$ segundo a equação 5.4 e com a informação da equação 5.10, então tem-se que:

$$s \in \mathcal{L}_m(G_{s_{o_t3}}). \tag{5.11}$$

Sendo assim (equação 5.11), a cadeia original s continua pertencendo à linguagem implementada pelo autômato resultante. No passo 3 do algoritmo, obtém-se $G_{s_{o_t4}}$ pela inclusão de autolaços de $\sigma_i^{\uparrow} \in \Sigma_c$ em alguns dos estados de $G_{s_{o_t3}}$. As modificações feitas em $G_{s_{o_t3}}$ não removem cadeias existentes. Portanto, a linguagem $\mathcal{L}_m(G_{s_{o_t4}})$, contém todas as cadeias da linguagem $\mathcal{L}_m(G_{s_{o_t3}})$. Sendo isso verdade, Tem-se que:

$$s \in \mathcal{L}_m(G_{s_{o_t3}}) \subseteq \mathcal{L}_m(G_{s_{o_t4}}) = \mathcal{F}^{\uparrow} \subseteq (\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru} \cup \Sigma_{fix}^C)^*. \tag{5.12}$$

Segundo o enunciado deste lema, sabemos que, $P_{\Sigma\to(\Sigma_{\sigma\uparrow}\cup(\Sigma_{t\downarrow}\cap\Sigma_{Ru}))}(a) = s$, e que ao substituir essa informação em 5.12, tem-se que:

$$P_{\Sigma\to(\Sigma_{\sigma\uparrow}\cup(\Sigma_{t\downarrow}\cap\Sigma_{Ru}))}(a) \in \mathcal{F}^{\uparrow} \subseteq (\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru} \cup \Sigma_{fix}^C)^* = (\Sigma_c \cup \Sigma_{Ru})^*. \tag{5.13}$$

Pois $\Sigma_c = \Sigma_{\sigma\uparrow} \cup \Sigma_{fix}^C$, o que demostra o lema. □

Com base no resultado do Lema 1, propõe-se o Corolário 1.

Corolário 1. *Seja uma cadeia $s \in \mathcal{L}_m(T_{Eq}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$. Seja a linguagem $\mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ obtida a partir de s por meio do Algoritmo 1, onde $\mathcal{F}^{\uparrow}$ é construída a partir de s. $\forall a \in S \subseteq \Sigma^*$ tal que $P_{\Sigma\to(\Sigma_{\sigma\uparrow}\cup(\Sigma_{t\downarrow}\cap\Sigma_{Ru}))}(a) = s$, então $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(a) \in \mathcal{F}^{\uparrow}$.* ▲

Demonstração. O lema 1 mostrou que $P_{\Sigma\to(\Sigma_{\sigma\uparrow}\cup(\Sigma_{t\downarrow}\cap\Sigma_{Ru}))}(a) \in \mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ (equação

5.13) e como $(\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})) \subseteq (\Sigma_c \cup \Sigma_{Ru})$, então, rescreve-se (5.13), assim:

$$P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))}(P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a)) = s \in \mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*. \tag{5.14}$$

A partir de (5.14), conclui-se que $s \in (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})^*$. Também de (5.14), tem-se que $s \in \mathcal{F}^{\uparrow} \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$. Assim, pode-se concluir que:

$$s \in P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \subseteq (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))^*. \tag{5.15}$$

Usando (5.14) e (5.15) obtém-se:

$$P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru}))}(P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a)) \in P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow}). \tag{5.16}$$

Aplica-se projeção inversa em (5.16):

$$\begin{aligned} P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a))] \subseteq \\ P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow})]. \end{aligned} \tag{5.17}$$

Da equação (5.17) e usando a propriedade (1) das projeções extraída de Cassandras & Lafortune (2009), onde $L \in P^{-1}(P(L))$, então, é verdade que:

$$\begin{aligned} P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a) &\in P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a))], \\ P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a) &\in P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow})]. \end{aligned} \tag{5.18}$$

A operação realizada entre (5.14) e (5.15) consistiu na eliminação de autolaços na linguagem $\mathcal{F}^{\uparrow}$. Os autolaços removidos são realocados com a operação de projeção inversa e, neste caso:

$$P^{-1}_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}[P_{(\Sigma_c \cup \Sigma_{Ru}) \to (\Sigma_{\sigma\uparrow} \cup (\Sigma_{t\downarrow} \cap \Sigma_{Ru})}(\mathcal{F}^{\uparrow})] = \mathcal{F}^{\uparrow}. \tag{5.19}$$

Então, de (5.18) e (5.19), tem-se que:

$$P_{\Sigma \to (\Sigma_c \cup \Sigma_{Ru})}(a) \in \mathcal{F}^{\uparrow}. \tag{5.20}$$

□

A equação 5.20 demostra o corolário. Com base no resultado do Corolário 1, propõe-se o Corolário 1.

Proposição 1. *Seja S o comportamento em malha fechada. Seja uma cadeia $s \in \mathcal{L}_m(T_{Eq}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$. Seja a linguagem $\mathcal{F}^\uparrow \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ obtida a partir de s por meio do Algoritmo 1, então, $\overline{\mathrm{S}} \cap \overline{P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow)} = \overline{\mathrm{P}^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap \mathrm{S}}$.*

♦

Demonstração. É sempre verdade que $\overline{\mathrm{S}} \cap \overline{P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow)} \subseteq \overline{\mathrm{P}^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap \mathrm{S}}$. Para mostrar a igualdade basta demostrar que:

$$P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\overline{\mathcal{F}^\uparrow}) \cap \overline{\mathrm{S}} \subseteq \overline{\mathrm{P}^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap \mathrm{S}} \tag{5.21}$$

Seja $a_1 \in \overline{\mathrm{S}} \cap \overline{P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow)}$, tem-se que $a_1 \in \overline{\mathrm{S}}$. Portanto, $\exists u \in \Sigma^*$ tal que $a_1 u \in S$. Seja $a = a_1 u \in S$, usando o Corolário 1:

$$P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(a) \in \mathcal{F}^\uparrow \tag{5.22}$$

Assim, $a \in \overline{P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow)}$, portanto:

$$a \in S \cap P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \tag{5.23}$$

Então, demostra-se que::

$$a_1 \in \overline{\mathrm{P}^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap \mathrm{S}}. \tag{5.24}$$

□

A equação 5.24 demonstra a Proposição. Após apresentar o procedimento para obtenção da linguagem $\mathcal{F}^\uparrow$, o Lema 1, o Corolário 1 e a Proposição 1, enuncia-se o Teorema 2, que estende o teorema original de Vilela & Pena (2016).

Teorema 2. *Seja G um sistema, S a máxima sublinguagem controlável contida na linguagem desejada de K e $P : \Sigma^* \to (\Sigma_c \cup \Sigma_{Ru})^*$ a projeção natural com a propriedade do observador. Para toda linguagem $\mathcal{F}^\uparrow \subseteq (\Sigma_c \cup \Sigma_{Ru})^*$ com as características do Algoritmo 1, e $A = P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^\uparrow) \cap S$, como $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ tem a propriedade do observador, então A será controlável com relação a $\mathcal{L}(G)$.* ■

Demostração. Para mostrar que A é controlável em relação a $\mathcal{L}(G)$, $\forall a \in \overline{\mathrm{A}}$ e $\forall \sigma \in \Sigma_u$ com $a\sigma \in \mathcal{L}(G)$, é preciso mostrar que $a\sigma \in \overline{\mathrm{A}}$.

Seja o autômato $T_{E_q} = Trim(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$, onde E_p é segundo a Definição 8. Uma cadeia $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ é escolhida para construir a linguagem $\mathcal{F}^{\uparrow}$, onde $s \in P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)$. Seja uma sequência tal que $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a') = s$. Seja $a \leq a'$, então:

$$a \in \overline{\mathrm{S}}. \tag{5.25}$$

Usando o Corolário 1, $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a') = s$, então $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a) \in \overline{\mathcal{F}^{\uparrow}}$. Portanto, ao aplicar a operação de projeção inversa nos dois lados, tem-se que:

$$\begin{aligned} P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a) &\in \overline{\mathcal{F}^{\uparrow}} \\ P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a)) &\subseteq \overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})} \\ a \in P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a)) &\subseteq \overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})}. \end{aligned} \tag{5.26}$$

Portanto, de (5.26) pode-se concluir que:

$$a \in \overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})}. \tag{5.27}$$

De (5.25) e (5.27), conclui-se que:

$$a \in \overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})} \cap \overline{\mathrm{S}}. \tag{5.28}$$

Como neste caso $\overline{P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow})} \cap \overline{\mathrm{S}} = \overline{\mathrm{P}^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{\mathrm{Ru}})}(\mathcal{F}^{\uparrow}) \cap \mathrm{S}} = \overline{\mathrm{A}}$ segundo a Proposição 1, então:

$$a \in \overline{\mathrm{A}} \subseteq \Sigma^*. \tag{5.29}$$

O próximo passo é mostrar que $a\sigma \in \overline{\mathrm{A}}$. Como S é controlável, tem-se que:

$$\forall a \in \overline{\mathrm{S}}, \forall \sigma \in \Sigma_u, a\sigma \in \mathcal{L}(G) \Rightarrow a\sigma \in \overline{\mathrm{S}}. \tag{5.30}$$

De (5.30), conclui-se que $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma) \in P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\overline{S})$. Como $P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(S)$ possui a

propriedade do observador segundo a Definição 1, então $\exists c \in \Sigma^*$ tal que:

$$P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma c) = P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma)b \text{ e}$$

$$a\sigma c \in S. \tag{5.31}$$

Pelo Corolário 1 e (5.31), tem-se que:

$$P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma c) \in \mathcal{F}^{\uparrow}. \tag{5.32}$$

Aplica-se a operação de projeção inversa em (5.32), assim:

$$P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma c)) \quad \subseteq \quad P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \text{ onde,}$$

$$a\sigma c \in P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(P_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(a\sigma c)) \subseteq P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}). \tag{5.33}$$

Portanto, de (5.33) pode-se concluir que:

$$a\sigma c \in P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}). \tag{5.34}$$

De (5.31) e (5.34) conclui-se que:

$$a\sigma c \in P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S. \tag{5.35}$$

A partir de $A = P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S$ e de (5.35), conclui-se que $a\sigma c \in A$, logo:

$$a\sigma \in \overline{\mathrm{A}}. \tag{5.36}$$

□

Com (5.36), prova-se que a linguagem $A = P^{-1}_{\Sigma\rightarrow(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S$ é controlável com relação à $\mathcal{L}(G)$, então, conclui-se que:

$$\forall a \in \overline{\mathrm{A}}, \forall \sigma \in \Sigma_u, a\sigma \in \mathcal{L}(G) \Rightarrow a\sigma \in \overline{\mathrm{A}}. \tag{5.37}$$

O resultado apresentado no Teorema 2 permite obter uma linguagem A controlável com

relação à $\mathcal{L}(G)$ a partir de uma linguagem $\mathcal{F}^{\uparrow}$. Assim, escolhe-se uma cadeia s e obtém-se, a partir dela, A controlável:

$$s \in \mathcal{L}_m(T_{Eq}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^* \subseteq P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S) \implies A = P^{-1}_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S \text{ é controlável.}$$

Esta ideia é ilustrada na Figura 5.18. A figura ilustra simplificadamente o processo de obtenção da linguagem recuperada A.

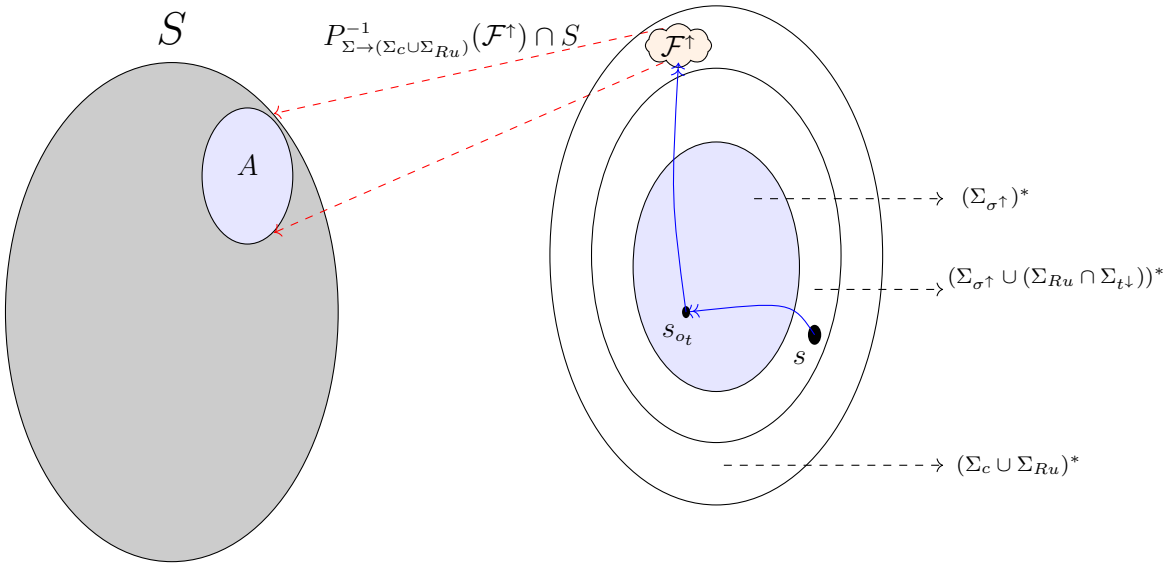


Figura 5.18: Ideia conceitual para construir a linguagem A.

A Figura 5.18 mostra o espaço de busca do supervisor (S) e o espaço de busca da projeção $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p$ que possui a propriedade do observador. Uma sequência s é escolhida na projeção (ponto preto s, elipse externa da direita) e depois projetada no alfabeto $\Sigma_{\sigma\uparrow}$, assim $s_{o_t} = P_{(\Sigma_{\sigma\uparrow}\cup\Sigma_{t\downarrow})\to\Sigma_{\sigma\uparrow}}(s)$ (ponto preto s_{o_t} elipse interna da direita). A linguagem $\mathcal{F}^{\uparrow}$ é obtida a partir da sequência s_{o_t} aplicando o Algoritmo 1. Portanto, com a linguagem $\mathcal{F}^{\uparrow}$, é possível recuperar uma linguagem A que está contida na linguagem que implementa o comportamento em malha fechada (elipse da esquerda).

A próxima seção mostra um estudo de caso da aplicação do resultado principal deste trabalho.

5.4 Estudo de Caso

Seja o sistema apresentado na Seção 4.1 (Exemplo 6), que possui 3 equipamentos, M_1, M_2, M_{TU} e $G = M_1||M_2||M_{TU}$ (Fig 5.19, Fig 5.20 e Fig 5.21). Cada um dos sistemas M_i atende às restrições da Definição 7, ou seja, alfabetos assíncronos (Def.7 (i,ii)), todos os eventos nos estados iniciais são controláveis (Def.7 (iii.a)), há os seguintes ciclos de trabalho: $w_{1_1} = \alpha_1\beta_1$, $w_{2_1} = \alpha_2\beta_2$ e $w_{TU_1} = \alpha_3\beta_3$, um em cada máquina (Def.7 (iii.b)). Pode-se ainda definir os conjuntos de eventos $\Sigma_{\sigma\uparrow} = \{\sigma_1, \sigma_2, \sigma_3\}$, $\Sigma_{t\downarrow} = \{\beta_1, \beta_2, \beta_3\}$ e $\Sigma_C^{fix} = \{\alpha_2\}$ (Def.7 (iv,v)).

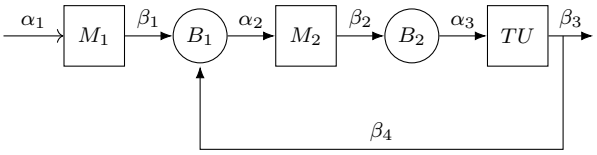
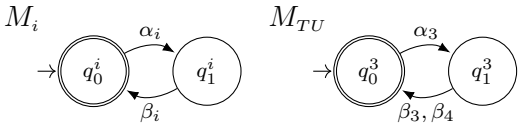
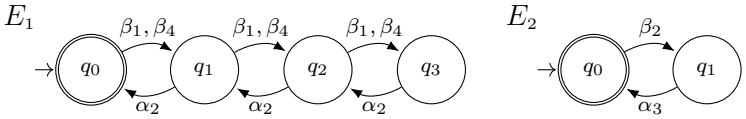


Figura 5.19: Sistema de manufatura com retrabalho.



(a) M_i, $i = \{1, 2\}$. (b) M_{TU}.

Figura 5.20: Modelos das máquinas.



(a) Especificação do *Buffer* B_1. (b) Especificação do *Buffer* B_2.

Figura 5.21: Modelos das especificações.

Suponha que deseja-se realizar a produção de um lote de tamanho 2, no menor tempo possível. Na solução deste problema será utilizada a abstração proposta neste trabalho.

O primeiro passo é obter o comportamento em malha fechada do sistema $S = SupC(K, G)$ (28 estados e 65 transições) e a abstração do mesmo com a propriedade do observador. Como foi dito anteriormente no Exemplo 6, a abstração do comportamento em malha fechada desse sistema sobre os eventos controláveis ($P_{\Sigma\to\Sigma_c}(S)$) não possui a propriedade do observador. Porém, faz-se necessária a aplicação do algoritmo de busca da propriedade do

observador, o qual retorna que todas as instâncias dos eventos β_3 e β_4 devem ser incluídas na projeção natural, ou seja, o conjunto $\Sigma_{Ru} = \{\beta_3, \beta_4\}$ fará parte da projeção, de tal forma que $P : \Sigma^* \to (\Sigma_c \cup \Sigma_{Ru})^*$ é PO-abstração.

O autômato que implementa $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ é apresentado na Figura 5.22 e os estados em cinza são aqueles que possuem os dois eventos $\Sigma_{Ru} = \{\beta_3, \beta_4\}$ possíveis. Para explicitar

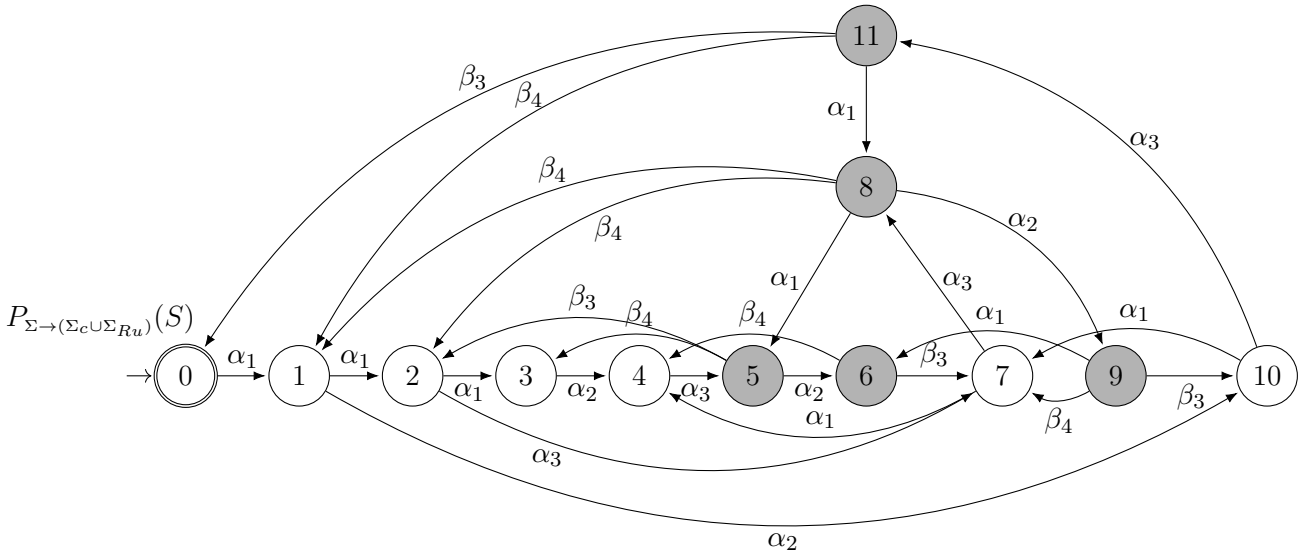


Figura 5.22: Abstração do supervisor com PO.

as sequências de eventos que levam à produção de um lote de tamanho 2, utiliza-se uma especificação de produção conforme a Definição 8, mostrada na Figura 5.23.

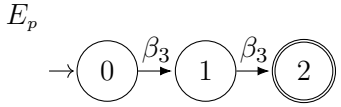


Figura 5.23: Especificação de produção.

A inclusão dos eventos não controláveis faz com que ciclos possam se formar (no caso, quando ocorre o evento β_4, que retorna o produto para a máquina M_2). Como estes eventos, β_3 e β_4, são não-controláveis, não é possível prevenir que eles aconteçam e deve-se lidar com as possibilidades. Considerando o equipamento M_{TU}, há um ciclo $w_{TU} = \alpha_3\beta_3$, $\sigma^{\uparrow}_{TU} = \alpha_3$ e $t^{\downarrow}_{TU} = \beta_3$.

Com os autômatos $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ e E_p, o próximo passo é aplicar o Algoritmo 1 que calcula a linguagem $\mathcal{F}^{\uparrow}$ que será usada no resultado principal deste trabalho (Lema **??**).

Primeiro realiza-se a composição paralela entre $P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)$ e a especificação de produção E_p e faz *Trim* sobre este resultado. Assim, obtém-se o autômato $T_{E_q} = Trim(P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$ mostrado na Figura 5.24.

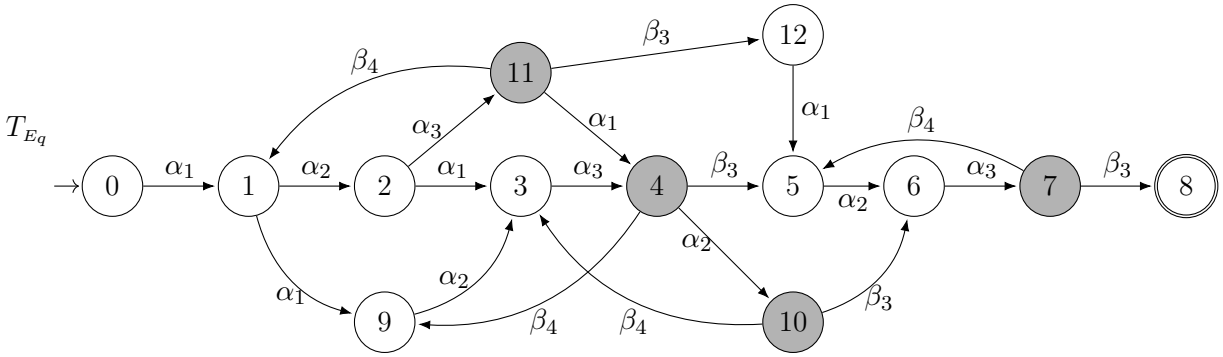


Figura 5.24: Autômato que implementa $T_{E_q} = Trim(P_{\Sigma\to(\Sigma_c\cup\Sigma_{Ru})}(S)||E_p)$.

O autômato da Figura 5.24 explicita todas as sequências $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ que levam à produção de dois produtos. Neste caso tem-se as sequências:

$$s_1 = \alpha_1\alpha_2\alpha_3\alpha_1\beta_3\alpha_2\alpha_3\beta_3$$

$$s_2 = \alpha_1\alpha_2\alpha_3\beta_3\alpha_1\alpha_2\alpha_3\beta_3$$

$$s_3 = \alpha_1\alpha_2\alpha_1\alpha_3\beta_3\alpha_2\alpha_3\beta_3$$

$$s_4 = \alpha_1\alpha_1\alpha_2\alpha_3\beta_3\alpha_2\alpha_3\beta_3$$

$$s_5 = \alpha_1\alpha_1\alpha_2\alpha_3\alpha_2\beta_3\alpha_3\beta_3$$

$$s_6 = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\beta_3\alpha_3\beta_3$$

$$s_7 = \alpha_1\alpha_2\alpha_3\alpha_1\alpha_2\beta_3\alpha_3\beta_3$$

Escolhe-se uma das sequências anteriores $s \in \mathcal{L}_m(T_{E_q}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$, por exemplo a sequência s_{ot6}. Assim,

$$s_6 = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\beta_3\alpha_3\beta_3$$

$$s_{ot6} = P_{\Sigma_s\to\Sigma_c}(s_6) = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\alpha_3.$$

O autômato que implementa a sequência s_{ot6} é mostrado na Figura 5.25. O próximo passo

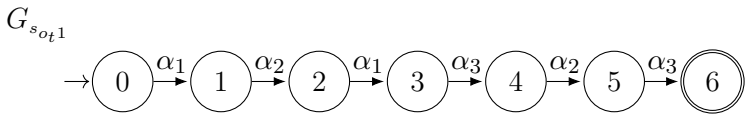


Figura 5.25: Autômato $G_{s_{ot}1}$ que implementa a sequência s_{ot6} (passo 1(ii)).

(passo 2(i, ii) do algoritmo) é acrescentar auto-laços do conjunto Σ^C_{fix} e Σ_{Ru} no autômato $G_{s_{ot}1}$. O conjunto $\Sigma^C_{fix} = \{\sigma \in \Sigma_c | \forall t \in \Sigma_u \backslash \Sigma_{t\downarrow} \Rightarrow \exists \sigma \in \Sigma_c$,que retorna à produção$\}$ está composto de eventos controláveis tal que para todo evento t pertencente a $\Sigma_u \backslash \Sigma_{t\downarrow}$ existe pelo menos um evento σ pertencente ao conjunto de eventos controláveis que retorna um produto à linha de produção (Def.7 v). No exemplo, $\Sigma^C_{fix} = \{\alpha_2\}$ pois depois do evento β_4 o evento controlável que retorna à produção é α_3, como pode ser observado na Figura 5.19. No exemplo $\Sigma_{Ru} = \{\beta_3, \beta_4\}$.

O passo 2(i) acrescenta autolaços de Σ_{Ru} em $G_{s_{ot}1}$ e obtém-se o autômato $G_{s_{ot}2}$ assim, $\mathcal{L}_m(G_{s_{ot}2}) = P^{-1}_{(\Sigma_{\sigma\uparrow} \cup \Sigma_{Ru}) \to \Sigma_{\sigma\uparrow}}(\mathcal{L}_m(G_{s_{ot}1}))$. O passo 2($ii$) está dividido em dois casos. Seja o caso $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, o caso $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$, significa que há eventos de Σ^C_{fix} na cadeia s. Este exemplo é do tipo caso 2(ii.2), quando $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$ pois existe uma interseção que é $\Sigma^C_{fix} = \{\alpha_2\}$. O algoritmo acrescenta autolaços de Σ^C_{fix}, sem tornar o autômato não determinístico. O autômato $G_{s_{ot}3}$ é obtido ao acrescentar autolaços de $\Sigma^C_{fix} = \{\alpha_2\}$ seja pelo caso $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} = \emptyset$ ou, $\Sigma^C_{fix} \cap \Sigma_{\sigma\uparrow} \neq \emptyset$ no autômato $G_{s_{ot}2}$, o qual significa que a linguagem do autômato $G_{s_{ot}2}$ continua contida na linguagem do autômato $G_{s_{ot}3}$. Sendo assim, tem-se que o autômato $G_{s_{ot}3}$ é:

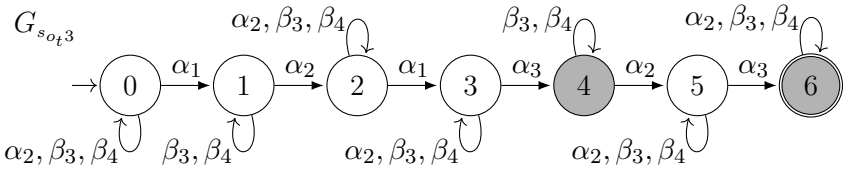


Figura 5.26: Auto-laços de Σ^C_{fix} e Σ_{Ru}, passo (2) do algoritmo.

Pode-se observar que $\alpha_2 \in \Sigma^C_{fix}$ não foi posto como auto-laço nos estados 1 e 4 para não gerar não determinismo (condição segundo o passo 2(ii.2) do algoritmo).

Por último, o passo (3) do algoritmo e o mais importante, consiste em identificar, quais estados $q \in Q$ de $Q_{s_{ot}3}$ as máquinas $M_i \in M_{Ru}$ (M_{Ru} segundo Def.7(vi)) estão em estados ativos q_a (q_a segundo Def.7) segundo pela aplicação da função rótulo $d(d'(q))$ e acrescentar auto-laços com os eventos $\sigma^{\uparrow}_{i_k}$. É importante determinar primeiramente neste passo o

conjunto M^{Ru} e o rótulo $d(d'(q))$.

O conjunto $M^{Ru} = \{M_i | \Sigma_i \cap \Sigma_{Ru} \neq \emptyset\}$ está composto pelos subsistemas M_i onde os eventos de Σ_{Ru} estão definidos. Portanto, tem-se que $M^{Ru} = \{M_{TU}\}$. Em seguida, determina-se o rótulo $d(d'(q))$ para saber aonde o subsistema M_{TU} está em estado ativo q_a, ou seja, onde o terceiro elemento da tupla de conjuntos possui o estado q_1^3. Porém, tem-se que os auto-laços acrescentados no autômato $Q_{s_{o_t 3}}$ serão do evento $\sigma_{3_1}^{\uparrow} = \alpha_3$. A aplicação de $d(d'(q))$ é detalhada na Tabela 5.3.

Tabela 5.3: Aplicação de $d'(q)$ e $d(d'(q))$.

q	$d'(q)$	$d(d'(q))$
0	$\{(q_0^1, q_0^2, q_0^3)\}$	$(\{q_0^1\}, \{q_0^2\}, \{q_0^3\})$
1	$\{(q_1^1, q_0^2, q_0^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_1^1, q_0^1\}, \{q_0^2\}, \{q_0^3\})$
2	$\{(q_0^1, q_1^2, q_0^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_0^1\}, \{q_1^2, q_0^2\}, \{q_0^3\})$
3	$\{(q_1^1, q_0^2, q_0^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_1^1, q_0^1\}, \{q_0^2\}, \{q_0^3\})$
4	$\{(q_0^1, q_0^2, q_1^3)\}$	$(\{q_0^1\}, \{q_0^2\}, \{q_1^3\})$
5	$\{(q_0^1, q_1^2, q_1^3), (q_0^1, q_1^2, q_0^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_0^1\}, \{q_1^2, q_0^2\}, \{q_1^3, q_0^3\})$
6	$\{(q_0^1, q_0^2, q_1^3), (q_0^1, q_0^2, q_0^3)\}$	$(\{q_0^1\}, \{q_0^2\}, \{q_1^3, q_0^3\})$

Percebe-se pela Tabela 5.3, que os estados $4, 5$ e 6 possuem na tupla o estado $\{q_1^3\}$. Como no estado $q = 5$ o evento α_3 já está factível originalmente, então coloca-se auto-laços apenas nos estados 4 e 6 (em cinza). Finalmente, o autômato $G_{s_{o_t 4}}$ resultante do passo (3) do algoritmo é apresentado na Figura 5.27 onde a linguagem marcada de $G_{s_{o_t 4}}$ é a linguagem $\mathcal{F}^{\uparrow}$.

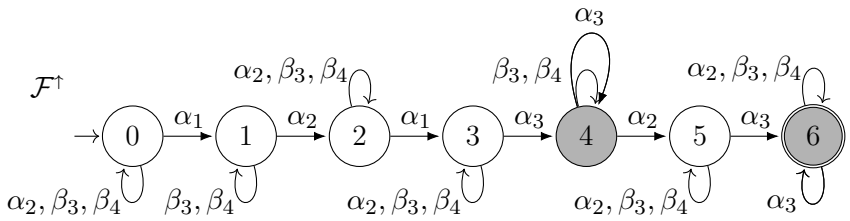


Figura 5.27: Autômato que implementa $\mathcal{F}^{\uparrow} \leftarrow \mathcal{L}_m(G_{s_{o_t 4}})$.

O Lema **??** diz que a linguagem A, que é uma sublinguagem do comportamento em malha fechada que projeta para a linguagem $\mathcal{F}^{\uparrow}$, é controlável. Isso é verdade para qualquer

linguagem $\mathcal{F}^{\uparrow}$ que projeta para uma sequência $s \in \mathcal{L}_m(T_{Eq}) \cap (\Sigma_{\sigma\uparrow} \cup \Sigma_{t\downarrow})^*$ que o planejador da Figura 2.2 possa escolher.

Constrói-se o autômato $A = P^{-1}_{\Sigma\rightarrow(\Sigma_c \cup \Sigma_{Ru})}(\mathcal{F}^{\uparrow}) \cap S$ que contém 39 estados e 62 transições. A linguagem A obtida é uma sublinguagem do comportamento em malha fechada que respeita as restrições e leva à produção de um lote de tamanho 2. Como o controlador só pode atuar sobre os eventos controláveis, então é essencial que todos os eventos que ele deve desabilitar sejam controláveis. Isto equivale a dizer que a linguagem A é controlável com relação à planta G.

A linguagem $\mathcal{F}^{\uparrow}$ deve ser implementada no sistema visando executar a cadeia otimizada $s_6 = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\beta_3\alpha_3\beta_3$. Não se pode implementar somente s_6, pois após de cada ocorrência do evento α_3, existe a possibilidade de ocorrer β_4 no lugar de β_3, ambos não-controláveis. Caso ocorra β_3, segue-se a produção com o restante da cadeia s_6, mas se ocorrer β_4, é preciso que o sistema seja capaz de lidar com a situação. A introdução de $\Sigma_C^{fix} = \{\alpha_2\}$ e dos auto-laços com $\sigma^{\uparrow}_{3_1} = \alpha_3$, garantem o retorno do sistema para a situação de produção e, portanto, garantem a controlabilidade da linguagem resultante A.

6

Conclusões

Este trabalho estabelece as condições necessárias para aplicação de abstrações sobre supervisores pela síntese monolítica para a busca das sequências de solução no problema de planejamento. O propósito principal dessa abstração sobre o comportamento em malha fechada é reduzir o espaço de busca da solução de problemas de planejamento.

Apresenta-se neste trabalho uma extensão do Teorema de Vilela & Pena (2016) para o caso em que a linguagem que implementa a projeção para o conjunto de eventos controláveis do comportamento em malha fechada não possui a propriedade do observador. A inexistência da PO demonstra a importância para a produção de alguns eventos não controláveis. Desta forma, estende-se a abstração para incluir tais eventos. Esta inclusão transforma a cadeia otimizada em uma linguagem $\mathcal{F}^{\uparrow}$ a ser implementada no sistema.

Para obtenção de tal linguagem, $\mathcal{F}^{\uparrow}$, parte-se de uma cadeia otimizada, escolhida entre aquelas que produzem um número de produtos, pressupondo que eventos não controláveis

específicos vão ocorrer. Em seguida, constrói-se a linguagem que garante que mesmo que os não-controláveis previstos não ocorram, é possível finalizar a produção.

A extensão proposta neste trabalho se justifica pelo fato de que existem sistemas de manufatura que por natureza a projeção natural do seu comportamento em malha fechada não apresentam a propriedade do observador, e onde o seu espaço de estados do comportamento em malha fechada é muito grande (explosão de estados). Assim, é legítimo propor a utilização de abstrações estendendo a projeção natural no conjunto de eventos controláveis e um subconjunto de eventos não controláveis sobre o supervisor pela síntese monolítica.

A extensão apresentada neste trabalho, no entanto, não é a solução final para o problema de planejamento, mas torna o problema de otimização mais fácil de ser resolvido.

Uma proposta de continuidade consiste na implementação de um algoritmo que busca a cadeia s que será usada para obter $\mathcal{F}^{\uparrow}$ e a integração desta solução com um protótipo de sistema de manufatura disponível no LACSED/UFMG. Adicionalmente, deseja-se unir resultados obtidos sobre distinção de eventos em sistemas com retrabalho (Cury et al., 2015) com o presente trabalho.

Referências Bibliográficas

Abdeddaim, Y. & Maler, O. (2001). Job-shop scheduling using timed automata? In *International Conference on Computer Aided Verification* (pp. 478–492).: Springer.

Alves, L. V., Bravo, H. J., Pena, P. N., & Takahashi, R. H. (2016). Planning on discrete events systems: A logical approach. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)* (pp. 1055–1060).: IEEE.

Alves, L. V., Martins, L. R., & Pena, P. N. (2017). Ultrades-a library for modeling, analysis and control of discrete event systems. *IFAC-PapersOnLine*, 50(1), 5831–5836.

Alves, M. R. d. C. & Pena, P. N. (2017). Abstrações de supervisores localmente modulares para aplicação na solução de problemas de planejamento. *XIII Simpósio Brasileiro de Automação Inteligente, Porto Alegre-RS*, (pp. 699–704).

Arisha, A., Young, P., & El Baradie, M. (2001). Job shop scheduling problem: an overview.

Ashour, S. (1970). A branch-and-bound algorithm for flow shop scheduling problems. *AIIE Transactions*, 2(2), 172–176.

Baker, K. R. & Trietsch, D. (2013). *Principles of sequencing and scheduling*. John Wiley & Sons.

Bansal, S. (1979). On lower bounds in permutation flow shop problems. *International Journal of Production Research*, 17(4), 411–418.

Bożejko, W., Gnatowski, A., Pempera, J., & Wodecki, M. (2017). Parallel tabu search

for the cyclic job shop scheduling problem. *Computers & Industrial Engineering*, 113, 512–524.

Bravo, H. J., da Cunha, A. E. C., Pena, P. N., Malik, R., & Cury, J. E. (2012). Generalised verification of the observer property in discrete event systems. *IFAC Proceedings Volumes*, 45(29), 337–342.

Bravo, H. J., Pena, P. N., Alves, L. V., & Takahashi, R. H. (2018). Factorization-based approach for computing a minimum makespan controllable sublanguage. *IFAC-PapersOnLine*, 51(7), 19–24.

Bravo, H. J., Pena, P. N., da Cunha, A. E. C., Malik, R., & Cury, J. E. (2014). Generalised search for the observer property in discrete event systems1. *IFAC Proceedings Volumes*, 47(2), 350–355.

Cai, K. & Wonham, W. M. (2013). Supervisor localization of discrete-event systems based on state tree structures. *IEEE Transactions on Automatic Control*, 59(5), 1329–1335.

Canbolat, Y. B. & Gundogar, E. (2004). Fuzzy priority rule for job shop scheduling. *Journal of intelligent manufacturing*, 15(4), 527–533.

Cassandras, C. G. & Lafortune, S. (2009). *Introduction to discrete event systems*. Springer Science & Business Media.

Cavalca, D. L. & Fernandes, R. A. (2018). Hybrid particle swarm algorithm applied to flexible job-shop problem. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–6).: IEEE.

Chen, E. & Lafortune, S. (1991). On nonconflicting languages that arise in supervisory control of discrete event systems. *Systems & Control Letters*, 17(2), 105–113.

Costa, T. A., Pena, P. N., & Takahashi, R. H. (2018). Sco-concat: a solution to a planning problem in flexible manufacturing systems using supervisory control theory and optimization techniques. *Journal of Control, Automation and Electrical Systems*, 29(4), 500–511.

Cunha, A. E. C. & Cury, J. E. R. (2007). Hierarchical supervisory control based on discrete event systems with flexible marking. *IEEE Transactions on Automatic Control*, 52(12), 2242–2253.

Cury, J. E., de Queiroz, M. H., Bouzon, G., & Teixeira, M. (2015). Supervisory control of discrete event systems with distinguishers. *Automatica*, 56, 93–104.

Cury, J. E., Torrico, C. R., & da Cunha, A. E. (2004). Supervisory control of discrete event systems with flexible marking. *European Journal of Control*, 10(1), 47–60.

Dauzère-Pérès, S. & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70, 281–306.

de Queiroz, M. H., Cury, J. E., & Wonham, W. M. (2005). Multitasking supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 15(4), 375–395.

Della Croce, F., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1), 15–24.

Dell'Amico, M. & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations research*, 41(3), 231–252.

Fabre, E. & Jezequel, L. (2009). Distributed optimal planning: an approach by weighted automata calculus. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference* (pp. 211–216).: IEEE.

Feng, L. & Wonham, W. M. (2006). Computationally efficient supervisor design: Abstraction and modularity. In *2006 8th International Workshop on Discrete Event Systems* (pp. 3–8).: IEEE.

Feng, L. & Wonham, W. M. (2008). Supervisory control architecture for discrete-event systems. *IEEE Transactions on Automatic Control*, 53(6), 1449–1461.

Flordal, H. & Malik, R. (2006). Modular nonblocking verification using conflict equivalence. In *2006 8th International Workshop on Discrete Event Systems* (pp. 100–106).: IEEE.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: theory and practice*. Elsevier.

Gohari, P. & Wonham, W. M. (2000). On the complexity of supervisory control design in the rw framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 30(5), 643–652.

Gonzalez, T. & Sahni, S. (1978). Flowshop and jobshop schedules: complexity and approximation. *Operations research*, 26(1), 36–52.

Gupta, J. (1970). M-stage flowshop scheduling by branch and bound. *Opsearch*, 7(1), 37–43.

Hill, R., Cury, J., De Queiroz, M., & Tilbury, D. (2008). Modular requirements for hierarchical interface-based supervisory control with multiple levels. In *2008 American Control Conference* (pp. 483–490).: IEEE.

Hill, R. C., Cury, J. E. R., de Queiroz, M. H., Tilbury, D. M., & Lafortune, S. (2010). Multi-level hierarchical interface-based supervisory control. *Automatica*, 46(7), 1152–1164.

Ignall, E. & Schrage, L. (1965). Application of the branch and bound technique to some flow-shop scheduling problems. *Operations research*, 13(3), 400–412.

Jovane, F., Yoshikawa, H., Alting, L., Boer, C. R., Westkamper, E., Williams, D., Tseng, M., Seliger, G., & Paci, A. (2008). The incoming global technological and industrial revolution towards competitive sustainable manufacturing. *CIRP annals*, 57(2), 641–659.

Kobetski, A. & Fabian, M. (2006). Scheduling of discrete event systems using mixed integer linear programming. In *2006 8th International Workshop on Discrete Event Systems* (pp. 76–81).: IEEE.

Komenda, J. & van Schuppen, J. H. (2005). Modular antipermissive control of discrete-event systems. *IFAC Proceedings Volumes*, 38(1), 97–102.

Lafortune, S. (2019). Discrete event systems: Modeling, observation, and control. *Annual Review of Control, Robotics, and Autonomous Systems*.

Leduc, R. J., Lawford, M., & Wonham, W. M. (2005). Hierarchical interface-based supervisory control-part ii: parallel case. *IEEE Transactions on Automatic Control*, 50(9), 1336–1348.

Liu, S. Q., Kozan, E., Masoud, M., Zhang, Y., & Chan, F. T. (2018). Job shop scheduling with a combination of four buffering constraints. *International Journal of Production Research*, 56(9), 3274–3293.

Malik, R. (2004). On the set of certain conflicts of a given language. *Proceedings of 7th Workshop on Discrete Event Systems (WODES'04)*, (pp. 277–282).

Malik, R. & Pena, P. N. (2018). Optimal task scheduling in a flexible manufacturing system using model checking. *IFAC-PapersOnLine*, 51(7), 230–235.

Malik, R., Streader, D., & Reeves, S. (2004). Fair testing revisited: A process-algebraic characterisation of conflicts. In *International Symposium on Automated Technology for Verification and Analysis* (pp. 120–134).: Springer.

Mastrolilli, M. & Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of scheduling*, 3(1), 3–20.

McMahon, G. & Burton, P. (1967). Flow-shop scheduling with the branch-and-bound method. *Operations Research*, 15(3), 473–481.

Nishi, T. & Wakatake, M. (2014). Decomposition of timed automata for solving scheduling problems. *International Journal of Systems Science*, 45(3), 472–486.

Nowicki, E. & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management science*, 42(6), 797–813.

Oliveira, A. C., Costa, T. A., Pena, P. N., & Takahashi, R. H. (2013). Clonal selection algorithms for task scheduling in a flexible manufacturing cell with supervisory control. In *2013 IEEE Congress on Evolutionary Computation* (pp. 982–988).: IEEE.

Onwubolu, G. & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, 171(2), 674–692.

Panek, S., Engell, S., Subbiah, S., & Stursberg, O. (2008). Scheduling of multi-product batch plants based upon timed automata models. *Computers & Chemical Engineering*, 32(1-2), 275–291.

Panek, S., Stursberg, O., & Engell, S. (2004). Job-shop scheduling by combining reachability analysis with linear programming. *IFAC Proceedings Volumes*, 37(18), 195–200.

Partovi, A. & Lin, H. (2018). Reactive supervisory control of open discrete-event systems. *arXiv preprint arXiv:1804.00037*.

Pena, P. N., Bravo, H. J., da Cunha, A. E., Malik, R., Lafortune, S., & Cury, J. E. (2014). Verification of the observer property in discrete event systems. *IEEE Transactions on Automatic Control*, 59(8), 2176–2181.

Pena, P. N., Costa, T. A., Silva, R. S., & Takahashi, R. H. (2016). Control of flexible manufacturing systems under model uncertainty using supervisory control theory and evolutionary computation schedule synthesis. *Information Sciences*, 329, 491–502.

Pena, P. N., Cury, J. E., & Lafortune, S. (2008). Polynomial-time verification of the observer property in abstractions. In *2008 American Control Conference* (pp. 465–470).: IEEE.

Pena, P. N., Cury, J. E., & Lafortune, S. (2009). Verification of nonconflict of supervisors using abstractions. *IEEE Transactions on Automatic Control*, 54(12), 2803–2815.

Pena, P. N., Cury, J. E., Malik, R., & Lafortune, S. (2010). Efficient computation of observer projections using op-verifiers. In *WODES* (pp. 406–411).

Pinedo, M. (2008). *Scheduling - Theory, Algorithms and Systems. 3a. ed. [S.l.].* Springer.

Pinson, E. (1995). The job shop scheduling problem: A concise survey and some recent developments. *Scheduling Theory and its Applications*, (pp. 277–294).

Queiroz, M. H. & Cury, J. E. (2000). Modular supervisory control of large scale discrete event systems. In *Discrete Event Systems* (pp. 103–110). Springer.

Queiroz, M. H. d. et al. (2004). Controle supervisório modular e multitarefa de sistemas compostos.

Rafael, G. C. (2018). Solution of a scheduling problem using an abstraction of the closed loop behaviour of a discrete event system.

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

Ramanan, T. R., Sridharan, R., Shashikant, K. S., & Haq, A. N. (2011). An artificial neural network based heuristic for flow shop scheduling problems. *Journal of Intelligent Manufacturing*, 22(2), 279–288.

Ruiz, R., Maroto, C., & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34(5), 461–476.

Schmidt, K. & Breindl, C. (2010). Maximally permissive hierarchical control of decentralized discrete event systems. *IEEE Transactions on Automatic Control*, 56(4), 723–737.

Schmidt, K., Moor, T., & Perk, S. (2008). Nonblocking hierarchical control of decentralized discrete event systems. *IEEE Transactions on Automatic Control*, 53(10), 2252–2265.

Schmidt, K. W. & Cury, J. E. R. (2012). Efficient abstractions for the supervisory control of modular discrete event systems. *IEEE Transactions on Automatic Control*, 57(12), 3224–3229.

Smith, L. & Ball, P. (2012). Steps towards sustainable manufacturing through modelling material, energy and waste flows. *International Journal of Production Economics*, 140(1), 227–238.

Su, R. (2012). Abstraction-based synthesis of timed supervisors for time-weighted systems. *IFAC Proceedings Volumes*, 45(29), 128–134.

Subbiah, S., Tometzki, T., Panek, S., & Engell, S. (2009). Multi-product batch scheduling with intermediate due dates using priced timed automata models. *Computers & Chemical Engineering*, 33(10), 1661–1676.

Tao, F., Cheng, Y., Zhang, L., & Nee, A. Y. (2017). Advanced manufacturing systems: socialization characteristics and trends. *Journal of Intelligent Manufacturing*, 28(5), 1079–1094.

Thistle, J. G. (1996). Supervisory control of discrete event systems. *Mathematical and Computer Modelling*, 23(11-12), 25–53.

Vieira, A. D. et al. (2007). Método de implementaçao do controle de sistemas e eventos discretos com aplicaçao da teoria de controle supervisório. Tese submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Doutor em Engenharia Elétrica.

Vilela, J. N. & Pena, P. N. (2016). Supervisor abstraction to deal with planning problems in manufacturing systems. In *Discrete Event Systems (WODES), 2016 13th International Workshop on* (pp. 117–122).: IEEE.

Weckman, G. R., Ganduri, C. V., & Koonce, D. A. (2008). A neural network job-shop scheduler. *Journal of Intelligent Manufacturing*, 19(2), 191–201.

Wong, K. (1998). On the complexity of projections of discrete-event systems. In *Proc. of WODES* (pp. 201–206).

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

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

Wong-Toi, H. & Hoffmann, G. (1991). The control of dense real-time discrete event systems. In *[1991] Proceedings of the 30th IEEE Conference on Decision and Control* (pp. 1527–1528).: IEEE.

Wonham, W. M. (2015). *Supervisory control of discrete-event systems*. Springer.

Wonham, W. M. & Ramadge, P. J. (1988). Modular supervisory control of discrete-event systems. *Mathematics of control, signals and systems*, 1(1), 13–30.

Zhang, J., Ding, G., Zou, Y., Qin, S., & Fu, J. (2019). Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing*, 30(4), 1809–1830.

Zhang, R., Cai, K., & Wonham, W. M. (2017). Supervisor localization of discrete-event systems under partial observation. *Automatica*, 81, 142–147.