UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
DEPARTAMENTO DE ENGENHARIA ELÉTRICA – DEE
COORDENAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA – PPGEEL

Formação: Mestrado Profissional em Engenharia Elétrica

DISSERTAÇÃO DE MESTRADO OBTIDA POR

Moacyr Carlos Possan Junior

MODELAGEM E IMPLEMENTAÇÃO DE SISTEMAS DE CONTROLE SUPERVISÓRIO BASEADOS EM MÁQUINAS DE ESTADOS COM SAÍDAS

JOINVILLE

2009

UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
DEPARTAMENTO DE ENGENHARIA ELÉTRICA – DEE
COORDENAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA – PPGEEL

MOACYR CARLOS POSSAN JUNIOR

MODELAGEM E IMPLEMENTAÇÃO DE SISTEMAS DE CONTROLE SUPERVISÓRIO BASEADOS EM MÁQUINAS DE ESTADOS COM SAÍDAS

Dissertação apresentada à Universidade do Estado de Santa Catarina como requisito para a obtenção do grau de Mestre em Engenharia Elétrica.

Orientador: André Bittencourt Leal, Dr.

JOINVILLE

2009

UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
COORDENAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA - PPGEEL

MODELAGEM E IMPLEMENTAÇÃO DE SISTEMAS DE CONTROLE SUPERVISÓRIO BASEADOS EM MÁQUINAS DE ESTADOS COM SAÍDAS

por

Moacyr Carlos Possan Junior

Esta dissertação foi julgada adequada para a obtenção do título de

MESTRE EM ENGENHARIA ELÉTRICA

Área de Concentração em *Automação de Sistemas*,
e aprovada em sua forma final pelo

CURSO DE MESTRADO PROFISSIONAL EM ENGENHARIA ELÉTRICA
CENTRO DE CIÊNCIAS TECNOLÓGICAS
UNIVERSIDADE DO ESTADO DE SANTA CATARINA

Banca Examinadora:

Dr. André Bittencourt Leal
CCT/UDESC (orientador/presidente)

Dr. Fábio Rodrigues De La Rocha
CCT/UDESC

Dr. Roberto Silvio Ubertino Rosso Junior
CCT/UDESC

Dra. Tatiana Renata Garcia
IST/SOCIESC

Joinville, SC, 15 de dezembro de 2009

FICHA CATALOGRÁFICA

NOME: POSSAN JUNIOR, Moacyr Carlos
DATA DEFESA: 15/12/2009
LOCAL: Joinville/SC, CCT/UDESC
NÍVEL: Mestrado **NÚMERO DE ORDEM: 19** – CCT/UDESC
FORMAÇÃO: Engenharia Elétrica **ÁREA DE CONCENTRAÇÃO**: Automação de Sistemas
TÍTULO: Modelagem e Implementação de Sistemas de Controle Supervisório Baseados em Máquinas de Estados com Saídas **PALAVRAS - CHAVE**: Sistemas a Eventos Discretos, Controle Supervisório, Máquinas de Estados Finitos, Controladores Lógicos Programáveis. **NÚMERO DE PÁGINAS:** xxi, 125 p.
CENTRO/UNIVERSIDADE: Centro de Ciências Tecnológicas da UDESC
PROGRAMA: Pós-Graduação em Engenharia Elétrica - PPGEEL
CADASTRO CAPES: 41002016012P0
ORIENTADOR: André Bittencourt Leal, Dr.
PRESIDENTE DA BANCA: André Bittencourt Leal, Dr.
MEMBROS DA BANCA: Dra. Tatiana Renata Garcia, Dr. Roberto Silvio Ubertino Rosso Junior, Dr. Fábio Rodrigues De La Rocha

A Deus, que é o Alfa e o Ômega, o Princípio e o Fim.
Aos meus pais Moacyr e Zélia.

"Disse-lhe Tomé: Senhor, nós não sabemos para onde vais; e como podemos saber o caminho? Disse-lhe Jesus: Eu sou o caminho, e a verdade e a vida; ninguém vem ao Pai, senão por mim. Se vós me conhecêsseis a mim, também conheceríeis a meu Pai; e já desde agora o conheceis, e o tendes visto."

João 14: 5-7

AGRADECIMENTOS

Primeiramente, meu agradecimento a *Deus*, criador dos céus e da terra, porque d'Ele por Ele e para Ele são todas as coisas.

Aos meus pais, *Moacyr Carlos Possan* e *Zélia Maria Possan* que, mesmo morando distante, sempre me apoiaram e me incentivaram para que alcançasse meus objetivos. Aos meus irmãos, *Gian Carlo Possan* e *Evandro Luiz Possan* que sempre foram meus companheiros. Aos meus demais familiares, pelo auxílio quando precisei de palavras de apoio e pela hospitalidade em me receber em seus lares quando visitei-os.

À querida e doce *Tatiane da Costa*, uma pessoa com um coração sincero, que tem me apoiado e se tornado alguém muito especial para minha vida.

Ao professor *André Bittencourt Leal*, por ter aceitado ser meu *orientador* e acreditado em mim para a realização desse trabalho. Uma grande pessoa com quem me senti à vontade para trabalhar e sempre me apoiou mesmo com as dificuldades encontradas ao longo desse trabalho. Aos membros da banca examinadora, *Tatiana Renata Garcia*, *Roberto Silvio Ubertino Rosso Junior e Fábio Rodrigues de la Rocha*, pelo tempo dedicado à revisão deste trabalho e contribuições para sua melhoria. Aos bolsistas *Denis Silva* e *Alexandre Junkes Pinotti* pelo auxílio na utilização do laboratório de robótica e à *Universidade do Estado de Santa Catarina (UDESC)* em geral.

Aos meus colegas de profissão, com quem tenho convivido a maior parte do meu tempo durante a realização deste trabalho, *Alisson Shigeru Takahashi, Diego Greff, Foad Mobini Kesheh, Hugo Shigeru Nagano, Mathias Cesca Nunes, Rafael Campos do Prado, Leopoldo França, André Bonetto Trindade, Daniel Bernardes, Itamar Loss, Clodoaldo Furtado, Oscar Schweitzer* e demais colegas não mencionados aqui. À *Whirlpool Latin America* pelo incentivo e apoio financeiro para a participação em congressos.

Aos amigos, *Gustavo Jair Riesenberg*, *Sérgio Rodrigo Bernardo, Anderson Dias, Róbson Fernandes, Billy Ferreira, Denílson Cezar Buettcher, Marcelo Weber, Cristiano Santos, Dorval Gottardi Neto, Fernando da Rosa Miranda, Diego Trindade, Otávio Arins, Peter Romancini* e demais que não estão mencionados aqui.

Deixo aqui também minhas sinceras desculpas para aqueles que eu deveria ter citado mas não o fiz.

RESUMO

POSSAN JUNIOR, Moacyr Carlos. **Modelagem e Implementação de Sistemas de Controle Supervisório Baseados em Máquinas de Estados com Saídas**. 2009. 125 p. Dissertação (Mestrado em Engenharia Elétrica – Área: Automação de Sistemas) – Universidade do Estado de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica, Joinville, 2009.

Este trabalho apresenta uma nova metodologia para a modelagem de sistemas de controle supervisório baseados em máquinas de estados com saídas, obtidas a partir dos autômatos que representam os supervisores encontrados com o uso da Teoria de Controle Supervisório (TCS) de Sistemas a Eventos Discretos (SEDs). Procedimentos como este são úteis para criar a documentação relativa à especificações de sistemas de grande porte e à geração de código, assim como permitem que a atualização da documentação e do código seja facilitada quando surgem novos requisitos para variação da lógica de controle em sistemas flexíveis. Além disso, isso abre espaço para a geração de soluções mais confiáveis e também para a possibilidade de geração automática de código. A técnica proposta consiste em obter máquinas de estados finitos com saídas usando como informação os autômatos dos supervisores obtidos por intermédio da TCS e as ações de controle do sistema, onde redundâncias da lógica de controle presentes no modelo do autômato são eliminadas. São propostas abordagens tanto no contexto monolítico quanto no contexto modular local, onde as máquinas obtidas são reduzidas posteriormente com o intuito de obter modelos mais simples, que servem como referência para a implementação em Controlador Lógico Programável (CLP) usando linguagem *Ladder*. A metodologia é demonstrada usando um sistema de manufatura simples como exemplo a fim de facilitar sua compreensão. Além disso, este trabalho trata das dificuldades encontradas na migração da teoria baseada em eventos da TCS na prática baseada em sinais dos CLPs. Após a apresentação da metodologia, é feita a modelagem e implementação para um sistema de maior porte, uma célula de manufatura onde é efetuada uma comparação com uma metodologia já existente que também têm como base a TCS, com o intuito de verificar as vantagens e desvantagens dessa metodologia.

Palavras-chave: Sistemas a Eventos Discretos. Controle Supervisório. Máquinas de Estados Finitos. Controladores Lógicos Programáveis.

ABSTRACT

POSSAN JUNIOR, Moacyr Carlos. **Modeling and Implementation of Supervisory Control Systems Based on State Machines with Outputs**. 2009. 125 p. Dissertation (Master Degree in Electrical Engineering – Field: Systems Automation) – Santa Catarina State University. Post-Graduation Program in Electrical Engineering, Joinville, 2009.

This work presents a new methodology for the modeling of supervisory control systems based on state machines with outputs, obtained from the automata which represent the supervisors found with the usage of the Supervisory Control Theory (SCT) of Discrete Event Systems (DES). Procedures like that are useful to create the documentation which represents the control logic for large scale systems and for the code generation, as well as allows that the documentation and program code updates be easier when new requirements to change the control logic in flexible systems appear. Besides, it makes way for the generation of more reliable solutions and also for the possibility of automatic code generation. The proposed technique consists on the obtaining of finite state machines with outputs using as input information the automata of the supervisors obtained by the SCT and the control actions of the system, where the control logic redundancies existing in the model of the automata are eliminated. Methodologies based on either monolithic or local modular approach are proposed, where the obtained machines are further simplified in order to have simpler models which are used as templates for the implementation in Programmable Logic Controller (PLC) using Ladder language. The methodology is shown using a simple manufacturing system as example to help on its understanding. Besides, this work deals with the difficulties found in the migration from the event based theory in the TCS to the signal based practice for the CLPs. After the presentation of this methodology, it is performed the modeling and implementation for a larger system, a manufacturing cell where a comparison with another existing methodology which also has the SCT as base is performed in order to verify the advantages and disadvantages of such methodology.

Key-words: Discrete Event Systems. Supervisory Control. Finite State Machines. Programmable Logic Controllers.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AFD	Autômato Finito Determinístico
AFN	Autômato Finito Não-Determinístico
CLP	Controlador Lógico Programável
I	Interface
IEC	*International Electrotechnical Commission*
FIFO	*First In First Out*
FSM	*Finite State Machine*
MEF	Máquina de Estados Finitos
P	Planta
PID	Proporcional-Integral-Derivativo
RSP	Representação por Sistema Produto
RW	RAMADGE-WONHAM
SED	Sistema a Eventos Discretos
SC	Sistema de Controle
SO	Seqüências Operacionais
SP	Sistema Produto
TCS	Teoria de Controle Supervisório

Lista de Símbolos

Teoria dos Conjuntos

$\varnothing$	conjunto vazio
$\subseteq$	está contido igual
$\supseteq$	contém igual
$\in$	pertence
$\exists$	Existe
$A \cap B$	A interseção B
$A \cup B$	A união B

Linguagens

Σ	Alfabeto
ε	cadeia vazia
s, t, s_a, s_b	Cadeias
$s \in \Sigma^*$	cadeia sobre Σ
$\|s\|$	comprimento da cadeia s
Σ^*	conjunto de todas as palavras de comprimento finito sobre Σ
St	concatenação das palavras s e t
L_a, L_b	Linguagens
$L \subseteq \Sigma^*$	linguagem sobre Σ
L_aL_b	concatenação das linguagens L_a e L_b
$\overline{L}$	prefixo-fechamento da linguagem L
L^*	fechamento Kleene da linguagem L

Autômatos de Estados Finitos

G	autômato finito determinístico
$(Q, \Sigma, q_0, Q_m, \delta)$	representação de um autômato G
Σ	alfabeto de um autômato
q_0	estado inicial de um autômato
Q	conjunto de estados de um autômato
$\delta: Q \times \Sigma \rightarrow Q$	função de transição de G
Q_m	conjunto de estados marcados de um autômato
$\hat{\delta} : Q \times \Sigma^*$	função de transição estendida para seqüências de eventos $s \in \Sigma^*$
$\sigma \in \Sigma$	Símbolo (evento) em Σ
$\Sigma_G(q)$	conjunto ativo de eventos de G no estado $q \in Q$
G_{nd}	autômato finito não determinístico
$(Q, \Sigma \cup \{\varepsilon\}, q_0, Q_m, \delta_{nd})$	representação de um autômato G_{nd}
δ_{nd}	Função de transição para um autômato não determinístico
2^Q	conjunto de potência de Q

Linguagens Representadas por Autômatos

$L(G)$	linguagem gerada pelo autômato G
$L_m(G)$	linguagem marcada pelo autômato G

Operações Sobre Autômatos e Linguagens

$Ac(G)$	componente acessível do autômato G
$CoAc(\mathrm{G})$	componente co-acessível do autômato G
$Trim(G)$	componente *Trim* do autômato G
$G_1 \parallel G_2$	composição síncrona dos autômatos G_1 e G_2

Expressões Regulares

r, s	expressões regulares
$r + s$	expressão regular representando a união de duas linguagens
rs	expressão regular representando a concatenação de duas linguagens
r^*	expressão regular representando o fechamento Kleene de uma linguagem

Máquinas de Estados com Saídas

Δ	alfabeto de saída (conjunto de símbolos não vazio e finito)
ω	função de saída

Controle Supervisório de SEDs: Abordagem Monolítica

G	SED que representa a planta a ser controlada
Σ	conjunto de eventos da planta G
$\sigma \in \Sigma$	evento da planta G
Σ_c	conjunto de eventos controláveis da planta G
Σ_{uc}	conjunto de eventos não controláveis da planta G
$L(G)$	linguagem gerada pela planta G
$L_m(G)$	linguagem marcada pela planta G
E	especificação de controle para a planta G
K	linguagem alvo de uma planta G
Γ	estrutura de controle para a planta G
$\gamma \in \Gamma$	padrão de controle
R	autômato que modela a linguagem alvo K
S	supervisor
MD	mapa de desabilitações do supervisor
S/G	planta G sob supervisão de S
$L(S/G)$	linguagem gerada pela planta G sob supervisão de S
$L_m(S/G)$	linguagem marcada pela planta G sob supervisão de S
$C(K, L(G))$	linguagens controláveis contidas na linguagem alvo K de uma planta G
$SupC(K, L(G))$	elemento supremo de C, máxima linguagem controlável contida em K

Controle Supervisório de SEDs: Abordagem Modular Local

G_i	subsistema (subplanta) i
E_j	especificação genérica local j
G_{lj}	planta local j associada à especificação genérica local E_j
K_j	linguagem alvo da planta local G_{lj}
SP_k	subplanta assíncrona k
SL_j	supervisor modular local j
MD_j	mapa de desabilitações do supervisor modular local
SLR_j	supervisor modular local reduzido j

Metodologia usando Máquinas de Estados – Operandos, Operadores e Simbologia

/	operando para fazer a distinção transição/ação na máquina de estados
&	operador lógico que representa a condição booleana *AND*
^	operador lógico que representa a condição booleana *OR-Exclusive*
~	operador lógico que representa a condição booleana *NOT*
MM	máquina de *Mealy* na abordagem monolítica
S_c	lista de transições no supervisor devido a eventos controláveis
S_u	lista de transições no supervisor devido a eventos não controláveis
MMS	máquina de *Mealy* reduzida na abordagem monolítica
MO	máquina de *Moore* na abordagem monolítica
MOS	máquina de *Moore* reduzida na abordagem monolítica
MDM	mapa de desabilitações da máquina na abordagem monolítica
MDMS	mapa de desabilitações da máquina reduzida na abordagem monolítica
ML_j	máquina modular local *j*
SL_{jc}	lista de transições no supervisor local *j* devido a eventos controláveis
SL_{ju}	lista de transições no supervisor local *j* devido a eventos não controláveis
MLS_j	máquina modular local reduzida *j*
MDM_j	mapa de desabilitações da máquina modular local *j*
$MDMS_j$	mapa de desabilitações da máquina modular local reduzida *j*

Capítulo 1

Introdução

O crescimento na complexidade dos sistemas automatizados na indústria vem acontecendo de forma significativa nos últimos anos devido ao aumento da demanda de produção, melhorias de qualidade e flexibilidade para reestruturar os processos com o intuito de satisfazer novos procedimentos. Todavia, a evolução dos dispositivos de controle e suas funcionalidades, tais como, velocidade de processamento, memória e comunicação em rede tem evoluído em paralelo com os requisitos da fábrica.

Tendo em vista essa evolução de instrumentação nos processos industriais, uma carência que ainda se verifica é a ausência de métodos formais de *software* para a análise, projeto e implementação de sistemas de controle para Sistemas a Eventos Discretos (SEDs) a fim de reduzir o tempo de desenvolvimento, os investimentos em recursos humanos e que satisfaçam os requisitos de operação para certos processos de uma forma efetiva. Além disso, a ocorrência de falhas de programação que resultam em erros devido à interrupção do processo e prejuízos devido a um *software* mal projetado também é algo evidentemente inaceitável.

Em suma, este trabalho tem como objetivo propor um formalismo para modelagem de sistemas de controle para plantas industriais visando explorar os benefícios provenientes dos sistemas de automação e da metodologia para projeto e controle de SEDs já difundida no meio acadêmico, centrada em autômatos.

1.1 Motivação

Geralmente, os projetos de sistemas de controle supervisório usados na indústria são baseados no conhecimento do projetista, de acordo com a sua experiência em programação. A utilização de métodos formais é bastante escassa, tal que o reuso de documentação e código são comprometidos assim como a disseminação do conhecimento gerado. Entretanto, a automação de sistemas de manufatura tem trazido consigo um aumento na complexidade dos sistemas de controle, tal que elaborar e implementar lógicas de controle robustas e confiáveis

não é uma tarefa trivial. A fim de minimizar os riscos devido a erros de programação e permitir um método formal para modelagem de Sistemas a Eventos Discretos (SEDs), Ramadge e Wonham introduziram em 1989 a Teoria de Controle Supervisório (TCS), que garante uma lógica de controle ótima (não bloqueante e minimamente restritiva) para esses sistemas.

Um Sistema a Eventos Discretos (SED), por sua vez, consiste em um sistema de estados discretos dirigido a eventos, isto é, sua evolução de estado depende da ocorrência de eventos discretos assíncronos no tempo (CASSANDRAS e LAFORTUNE, 2008). São exemplos de eventos o início e o fim de operação de uma máquina e a ativação ou desativação de um sensor de presença.

De acordo com Cassandras e Lafortune (2008), a metodologia para modelagem de SEDs é centrada em autômatos e Redes de Petri e é refinada gradualmente: modelos não temporizados para propriedades lógicas relativas à ordem dos eventos, modelos temporizados para propriedades que envolvem considerações temporais e modelos temporizados estocásticos para propriedades que envolvem um ambiente probabilístico. Este tipo de modelagem ultrapassa áreas específicas de aplicação e permite uma ligação de maneira coerente com outros tópicos para o estudo de sistemas a eventos discretos: teoria de linguagens e autômatos, controle supervisório, teoria de Redes de Petri (MURATA, 1989), Redes de Petri Coloridas (JENSEN, 1997), conjunto de predicados lógicos (HOLLOWAY *et al.*, 1997), assim como álgebra Max-Plus, cadeias de Markov, teoria das filas, simulação de eventos discretos, análise de perturbações e técnicas de estimação concorrente. Existem diversas abordagens para SEDs, destinadas a controle, análise de desempenho, verificação formal de propriedades, entre outras funções.

Dentre as abordagens apresentadas, o interesse deste trabalho tem como base a Teoria de Controle Supervisório (RAMADGE e WONHAM, 1989), que consiste em um formalismo para modelagem e controle de SEDs utilizando linguagens formais e autômatos. Diversas ferramentas, tais como TCT (FENG e WONHAM, 2006), GRAIL (REISER *et al.*, 2006), DESUMA (RICKER *et al.*, 2006), IDES (RUDIE, 2006) e Supremica (AKESSON *et al.*, 2006) foram criadas para efetuar o cálculo de supervisores através da TCS a partir de modelos da planta e especificações de controle, o que auxilia na melhor compreensão da lógica de controle devido à característica visual dos grafos dirigidos denominados autômatos. Além disso, essas ferramentas podem ser usadas para a elaboração de outras ferramentas que permitem a geração de código automático para controladores.

A abordagem monolítica (RAMADGE e WONHAM, 1989) da TCS tem sua aplicação limitada devido ao tamanho considerável do modelo do supervisor obtido para sistemas de grande porte. Isso ocorre, pois tanto o modelo da planta como o modelo das especificações de controle é feito por meio de um único autômato. O procedimento de modelagem da planta consiste na composição síncrona dos diversos subsistemas presentes, resultando em um crescimento exponencial do número de estados de acordo com a quantidade de subsistemas presentes na planta.

A abordagem modular local (DE QUEIROZ e CURY, 2002) é um método de controle supervisório descentralizado e foi proposta com o intuito de reduzir a complexidade na síntese de supervisores para os sistemas de grande porte. Diferente da abordagem monolítica que usa a linguagem da planta completa para gerar um supervisor, a abordagem modular local tem como objetivo explorar a natureza modular do sistema a ser controlado. Essa abordagem usa apenas os subsistemas que são afetados pela especificação correspondente para encontrar os supervisores locais, resultando em diversos supervisores de acordo com o número de especificações presentes no modelo global. Assim, a composição entre autômatos é menor e os supervisores obtidos são menores do que o supervisor obtido através da abordagem monolítica reduzindo, portanto, a complexidade da síntese e prevenindo que ocorra uma explosão combinatória de estados. Embora supervisores menores facilitem o processo de implementação assim como, em geral, resultam em economia no tamanho de código, uma desvantagem desta abordagem consiste na necessidade de realizar o teste de modularidade local dos supervisores. Esse teste realiza a composição síncrona de todos os supervisores locais obtidos, tal que a complexidade computacional é a mesma necessária para obter um supervisor usando a abordagem monolítica. Isso é necessário para garantir que o desempenho da ação conjunta dos supervisores locais seja o mesmo do supervisor obtido a partir da abordagem monolítica.

Os dispositivos de controle comumente usados em sistemas de manufatura são os Controladores Lógicos Programáveis (CLPs). Na prática, a TCS é aplicada em problemas bem restritos, devido principalmente à dificuldade em lidar com problemas complexos e dificuldade na migração da teoria baseada em eventos para a prática baseada em sinais dos CLPs. A implementação de supervisores em CLPs obtidos a partir de modelos baseados em autômatos é destacada nos estudos de (BRANDIN, 1996), (LEDUC, 1996), (DE QUEIROZ e CURY, 2002), (VIEIRA, 2003), (DE QUEIROZ, 2004), (VIEIRA, 2007), (CURZEL, 2008), (LEAL *et al*., 2009), dentre outros. (BRANDIN, 1996) e (LEDUC, 1996) podem ser citados na literatura como estando dentre os primeiros trabalhos a proporem a implementação em

linguagem *Ladder* para CLPs de supervisores sintetizados a partir da TCS. (DE QUEIROZ e CURY, 2002), (VIEIRA, 2003), (DE QUEIROZ, 2004), (CURZEL, 2008) e (LEAL *et al*., 2009) fazem uso da TCS de acordo com a abordagem modular local, onde os supervisores são implementados em CLP usando linguagem *Ladder*. Vieira (2007) propõe uma metodologia de implementação de supervisores modulares locais em CLP por meio de linguagem Grafcet, onde os supervisores podem ser distribuídos em diferentes CLPs.

O uso de linguagens de baixo nível para a programação de CLPs é muito comum nos dias de hoje. Uma pesquisa elaborada pela revista *Control Engineering Magazine* (2007) destaca que aproximadamente 95% dos entrevistados ainda usam linguagem de programação *Ladder* (do inglês, *Relay Ladder Logic*, ou *RLL*). Essa é a principal motivação para uso dessa linguagem neste trabalho, ao invés de usar outras linguagens de programação para CLPs, tais como texto estruturado, lista de instruções, diagrama de blocos de funções e Grafcet. Essas linguagens estão normalizadas de acordo com a *IEC* 61131-3 (IEC 61131-3, 2003), uma norma que visa a padronização das linguagens de programação de CLPs. Além disso, a linguagem Grafcet ou *SFC* (*Sequential Funcion Chart*) está sendo disponibilizada também como recurso de programação por alguns fabricantes de CLPs. Trata-se de uma linguagem derivada das Redes de Petri e da norma *IEC* 848 Grafcet (IEC 60848, 2002), cujo intuito é descrever graficamente o comportamento seqüencial de um programa de controle. Vieira, (2007) propõe a implementação de uma metodologia baseada na abordagem modular local com o uso dessa linguagem.

Além de aplicações em CLPs para a área da manufatura, a literatura apresenta também aplicações da TCS em outros tipos de controladores, com destaque para implementação em microcontroladores (WOOD, 2005), (JONERLAN, 2007), (TEIXEIRA, 2008) e (BARETTA e TORRICO, 2008). Esse apelo deve-se fundamentalmente à busca de maior confiabilidade em *software* embarcado, tendo em vista que falhas na criação de lógicas de controle acarretam em um alto custo para as empresas, devido ao tempo despendido para validação e quantidade de recursos alocados. O projeto de controladores a partir de metodologias formais facilita a identificação de erros durante o desenvolvimento de código e permite também disseminar uma cultura dentro da empresa para que novos projetos reutilizem modelos já prontos e sejam armazenados em bancos de dados, reduzindo assim o tempo de desenvolvimento. Vale ressaltar que a implementação de supervisores em microcontroladores não é objetivo deste trabalho, sendo este voltado para a implementação em CLPs.

Existem diversos métodos, modelos e ferramentas para representar sistemas de controle, tais como fluxogramas, *statecharts* e máquinas de estados. A representação de

sistemas por meio de máquinas de estados é bastante comum atualmente na indústria. Embora a descrição de problemas de controle através de fluxogramas seja possível, os resultados são muito grandes e complexos. Fluxogramas são mais apropriados para descrever uma seqüência de atividades condicionais em programação estruturada, enquanto as máquinas de estados são mais apropriadas para representar programação baseada em eventos. Um Autômato Finito Determinístico (AFD) (HOPCROFT e ULLMANN, 1979) também é considerado uma máquina de estados. As máquinas de estados podem ter saídas associadas a elas, sendo os modelos mais comuns as máquinas de *Mealy* (MEALY, 1955) e de *Moore* (MOORE, 1956). As *statecharts* (HAREL, 1987) por sua vez buscam superar as limitações das máquinas de estado tradicionais mantendo seus benefícios, sendo mais adequadas para representar sistemas complexos e permitindo a modelagem de sistemas onde requisitos de hierarquia, concorrência e comunicação precisam ser satisfeitos (DOUGLASS, 1999), usando variáveis, condições de guarda e ações. Essa metodologia, entretanto, está fora do escopo desta dissertação.

Alguns autores apresentam metodologias para controle supervisório usando modelos de autômatos estendidos com variáveis na tentativa de minimizar a explosão de estados resultante da composição de autômatos. Chen e Lin (2000) assumem que uma variável pode ser atualizada por no máximo um autômato estendido e, a fim de efetuar uma síntese, o espaço de estados precisa ser estendido por estados adicionais. Yang e Gohari (2005) utilizam autômatos com variáveis para implementar um supervisor. Os autores codificam os estados de um dado supervisor usando variáveis *booleanas*. As variáveis são utilizadas em condições de guarda e ações ligadas aos eventos do modelo. Gaudin e Deussen (2007) propõem uma metodologia de controle supervisório por meio de diversos autômatos com variáveis. Para garantir que um supervisor menos restritivo exista, é assumido que todas as variáveis são locais, isto é, não podem ser compartilhadas entre os autômatos. Isto é uma restrição bastante severa, pois as variáveis não podem ser usadas para modelar qualquer interação entre os subsistemas. Skoldstam *et al.* (2008) propõem uma metodologia para transformar modelos estendidos em modelos de Autômatos Finitos Determinísticos (AFDs) tendo o mesmo comportamento, tal que autômatos estendidos podem ser usados para modelagem e AFDs para análise. Além disso, Moura (2009) propõe um método de modelagem usando *statecharts* e implementação desses modelos em CLP usando linguagem *Ladder*.

1.2 Objetivos

Embora a TCS forneça um método automático para síntese de controladores para SEDs, o grande número de estados gerados pelo seu algoritmo torna tais supervisores na prática, ilegíveis, pois impossibilita o entendimento da lógica de controle por inspeção visual do gráfico do supervisor. Este trabalho tem como objetivo utilizar os autômatos padrão dos supervisores modelados a partir das abordagens monolítica e modular local da TCS como parâmetros de entrada a fim de obter modelos de máquinas de estados finitos com saídas para representar os supervisores. Além de facilitar a visualização da lógica de controle se comparada com um AFD, essa abordagem visa reduzir o modelo do supervisor removendo estados e transições redundantes presentes no autômato que representa o supervisor, sem perda de generalidade e mantendo a mesma característica de controle. Essas máquinas passam posteriormente por um processo de redução com o intuito de diminuir ainda mais o número de estados e transições. Além disso, essas máquinas de estados servem como modelos de referência para o projetista efetuar a implementação em CLP, visando facilitar essa tarefa assim como ter um código enxuto e de fácil compreensão. Diversos problemas que surgem na implementação de supervisores em controladores síncronos, como por exemplo, os CLPs, são abordados neste trabalho. Esses problemas surgem devido a migração da teoria baseada em eventos para a prática de controladores síncronos baseados em sinais. A topologia de implementação das máquinas de estados é proposta usando linguagem *Ladder* e leva em conta a resolução desses problemas.

1.3 Contribuições

As principais contribuições deste trabalho são listadas a seguir:

- A criação de formalismos para modelagem de controladores para sistemas a eventos discretos utilizando máquinas de estados com saídas, concebido com base na Teoria de Controle Supervisório, utilizando as abordagens monolítica e modular local para a síntese dos supervisores.
- A concepção de metodologias para a implementação das máquinas de estados em controladores lógicos programáveis utilizando linguagem *Ladder*, levando em consideração os problemas relativos à implementação de supervisores em controladores síncronos.
- A modelagem e proposta de implementação da técnica para uma célula de manufatura, um exemplo de sistema de manufatura de grande porte.

Algumas contribuições deste trabalho foram apresentadas à comunidade científica por intermédio da publicação dos seguintes artigos: *Implementation of Supervisory Control Systems based on State Machines* (POSSAN JUNIOR e LEAL, 2009a) e *Implementação de Sistemas de Controle Supervisório Baseados em Máquinas de Estados* (POSSAN JUNIOR e LEAL, 2009b).

1.4 Organização do Trabalho

Esta dissertação é organizada como segue.

O Capítulo 1 faz uma introdução a respeito da motivação em se utilizar a Teoria de Controle Supervisório (TCS) na indústria e descreve os objetivos e contribuições esperados com este trabalho.

No Capítulo 2 é feita uma descrição dos Sistemas a Eventos Discretos (SEDs) e como eles são representados através da teoria de linguagens e autômatos, visando familiarizar o leitor com esse tipo de abordagem.

No Capítulo 3 é apresentada a metodologia para modelar sistemas de controle supervisório usando máquinas de estados com saídas obtidas a partir dos autômatos que representam os supervisores obtidos a partir da Teoria de Controle Supervisório (TCS), tanto num contexto de controle monolítico quanto modular local.

O Capítulo 4 descreve os problemas encontrados no processo de implementação de supervisores assim como a metodologia para implementação das máquinas de estados em Controladores Lógicos Programáveis (CLPs) usando linguagem *Ladder*.

No Capítulo 5 é realizada a aplicação da metodologia proposta neste trabalho em um sistema de maior porte, uma célula de manufatura, visando avaliar de uma maneira geral as vantagens e desvantagens da metodologia proposta.

Para finalizar, o Capítulo 6 sumariza as conclusões deste trabalho e as perspectivas para trabalhos futuros.

Capítulo 2

Sistemas a Eventos Discretos, Linguagens e Autômatos

Neste capítulo são caracterizados os Sistemas a Eventos Discretos (SEDs), onde é descrito o comportamento desse tipo de sistema e as principais áreas onde eles se encontram. Em seguida, são apresentados formalismos para linguagens formais e autômatos, conceitos provenientes da teoria da computação (HOPCROFT e ULLMANN, 1979) utilizados para representar matematicamente os SEDs. A teoria de linguagens e autômatos consiste em um método formal para estudar a lógica comportamental de SEDs assim como para sintetizar supervisores para esses sistemas. Além disso, o conceito de máquinas de estados com saídas, ou autômatos com entradas e saídas (CASSANDRAS e LAFORTUNE, 2008), é introduzido, dando ênfase às máquinas de *Mealy* e *Moore*. O objetivo disso é auxiliar o leitor na compreensão e começar a familiarizá-lo com o formalismo proposto nesta dissertação para controle supervisório de SEDs baseado em modelos de máquinas de estados com saídas.

A organização do capítulo é descrita a seguir. Na Seção 2.1 faz-se uma caracterização dos sistemas a eventos discretos, apresentando sua concepção dentro da teoria de controle, seus atributos principais, seus modelos de representação e suas aplicações. A Seção 2.2 apresenta a definição e as principais operações sobre linguagens formais. Na Seção 2.3 é apresentada uma definição formal de autômatos determinísticos de estados finitos. A Seção 2.4 descreve os autômatos não-determinísticos de estados finitos. A Seção 2.5 aborda as linguagens que podem ser representadas por autômatos de estados finitos. Na Seção 2.6 são descritas as principais operações sobre autômatos e linguagens a serem utilizadas neste trabalho. A Seção 2.7 trata das linguagens e expressões regulares enquanto a Seção 2.8 define as máquinas de estados finitos com saídas, de *Mealy* e de *Moore*. Por fim, a Seção 2.9 trata das conclusões.

2.1 Caracterização dos Sistemas a Eventos Discretos

Considere um sistema de controle em malha fechada que controla uma planta (ou sistema ou processo), conforme mostrado na Figura 2.1.

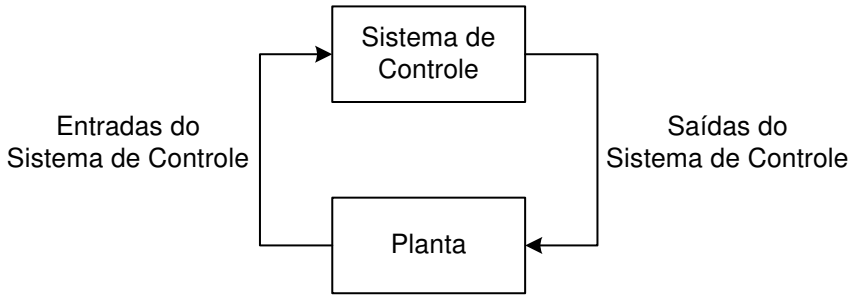


Figura 2.1 – Estrutura Típica de um Sistema de Controle *SC*

O sistema de controle recebe inúmeros estímulos (entradas) da planta e gera inúmeras ações (saídas) para afetar a planta. Existem basicamente duas grandes classes de controladores: controladores de processos contínuos e controladores de processos baseados em eventos discretos. Os primeiros são baseados na Teoria Geral de Controle (OGATA, 2003), tal como o método de controle Proporcional-Integral-Derivativo (PID), por exemplo, e estão fora do escopo de abordagem desta dissertação. O foco deste trabalho é voltado ao controle de processos baseados em eventos discretos.

A diferença entre os sistemas dinâmicos contínuos e os SEDs consiste na enumeração do espaço de estados. Em um sistema contínuo a evolução de estado (variável dependente) é uma função do tempo (variável independente). Um sistema contínuo pode ser modelado por equações diferenciais, caso a variável tempo seja contínua ou por equações a diferenças, caso o tempo seja discretizado.

A Figura 2.2 representa uma trajetória típica de estados de um sistema contínuo de comportamento exponencial, o qual pode ser representado por uma equação diferencial.

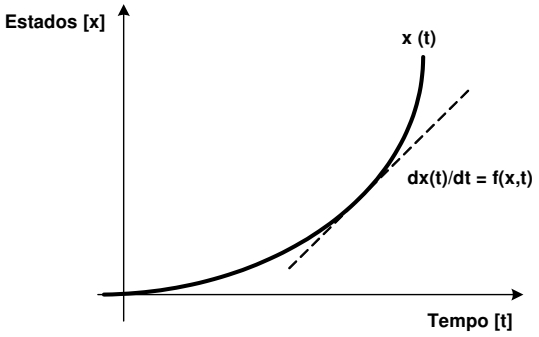


Figura 2.2 – Trajetória de Estados de um Sistema Contínuo

Quando o espaço de estados de um sistema é naturalmente descrito por um conjunto discreto de estados (*0*, *1*, *2*,...) e as transições de estados são observadas somente em pontos discretos no tempo, estas transições de estado são associadas com eventos e o sistema é chamado de Sistema a Eventos Discretos (SED) (CASSANDRAS e LAFORTUNE, 2008).

São exemplos de eventos o início e o término de funcionamento de uma máquina industrial e a percepção de mudança de estado (ativado ou desativado) de um sensor de presença. Estes eventos ocorrem instantaneamente, possuindo um caráter discreto no tempo. O sistema permanece em um determinado estado entre a ocorrência de dois eventos consecutivos. Quando algum evento acontecer, este causa uma transição ou mudança de estado.

Um sistema a eventos discretos evolui de acordo com a trajetória percorrida em seu espaço de estados, como mostra o exemplo da Figura 2.3. Os eventos presentes neste sistema são α, β e λ. As transições de estado ocorrem nos instantes de tempo t_1, t_2, t_3 e t_4 e o espaço de estados é dado por (x_1, x_2, x_3 e x_4). Um mesmo evento pode ter efeitos diferentes na dinâmica do sistema, de acordo com o estado em que ele acontece. Por exemplo, caso o sistema esteja no estado x_1 e o evento α ocorre, o próximo estado será x_3. Agora, se o sistema está no estado x_4 e ocorre o mesmo evento α, o sistema volta para x_1. Observe também neste gráfico como os eventos podem ocorrer em qualquer instante de tempo, caracterizando, portanto um sistema com comportamento assíncrono.

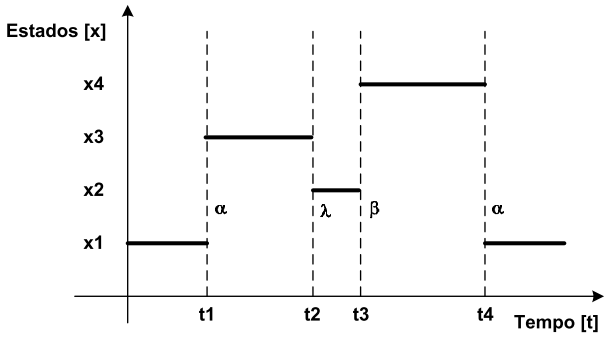


Figura 2.3 – Trajetória de Estados de um SED

Um SED tem sua trajetória de estados caracterizada pela seqüência de eventos percorrida, como por exemplo, $\alpha\lambda\beta\alpha$. Eventualmente, pode-se incluir o tempo em que o evento ocorre, tal como (α, t_1), (λ, t_2), (β, t_3), (α, t_4).

Este trabalho está voltado a sistemas cujo conjunto de eventos possíveis de ocorrer e o número de estados que o sistema pode assumir são finitos. De maneira geral, embora um sistema pode ser representado por um número infinito de estados, a ênfase na modelagem de sistemas de controle é dada a sistemas que contenham um número finito de estados. Alguns

estados são de interesse particular. O estado inicial corresponde à situação de repouso do sistema, onde nenhum evento ocorreu ainda. Na prática, em geral são criados controladores que forcem o sistema a voltar para esse estado inicial, processo que é chamado de reinicialização. Um outro conjunto de estados de relevância para o controle de SEDs são aqueles que correspondem à finalização de uma tarefa ou atividade, sendo denominados estados marcados ou estados finais.

Um sistema de controle para um SED usa condições lógicas simples da forma: SE (condições de entrada) ENTÃO (saídas). As condições de entrada são expressões lógicas formuladas de acordo com a lógica *booleana* (*On, Off, Open, Close,...*) além de operadores lógicos (*AND, OR, NOT,...*). Todos os argumentos só podem assumir duas condições *booleanas*: Verdadeiro (*True*) ou Falso (*False*).

Exemplos de SEDs são encontrados nas mais diversas áreas de atuação e aplicações, tais como, redes de computadores e comunicação, sistemas de manufatura automatizados, sistemas de controle de tráfego aéreo, sistemas de informação, monitoramento avançado e sistemas de controle de automóveis, sistemas de transporte inteligente, sistemas de *software* distribuído, sistemas de logística, e assim por diante.

2.2 Linguagens: Definição e Operações

O ponto inicial para representar SEDs por meio de linguagens consiste no fato de que qualquer SED possui um conjunto de eventos (ou símbolos, de acordo com a teoria da computação) Σ associado a ele. Define-se o conjunto de eventos Σ como um alfabeto e assume-se que Σ seja finito. Uma seqüência de eventos pertencente a este alfabeto forma uma cadeia (ou palavra, ou *string*, em inglês) de eventos. Uma cadeia sem eventos é chamada de *cadeia vazia* e é denotada por ε. O comprimento de uma cadeia é o número de eventos contidos nela, considerando múltiplas ocorrências do mesmo evento. Se s é uma cadeia, seu comprimento é denotado por $|s|$. Por convenção, o comprimento da cadeia vazia ε é zero (HOPCROFT e ULLMANN, 1979).

Uma linguagem definida sobre um conjunto de eventos Σ é um conjunto de cadeias de comprimento finito formado a partir dos eventos em Σ (CASSANDRAS e LAFORTUNE, 2008).

Exemplo 2.1 (CASSANDRAS e LAFORTUNE, 2008) Seja o alfabeto $\Sigma = \{a, b, g\}$. São definidas neste exemplo a linguagem $L_1 = \{\varepsilon, a, abb\}$ consistindo de apenas três cadeias; ou a linguagem $L_2 = \{$todas cadeias possíveis de comprimento 3 iniciando com o evento $a\}$ que contém nove cadeias; ou a linguagem $L_3 = \{$todas cadeias possíveis de comprimento finito iniciando com o evento $a\}$ que contém um número infinito de cadeias.

Seja Σ^* o conjunto de *todas* as cadeias finitas de elementos de Σ, incluindo a cadeia vazia ε, onde o operador * é chamado fechamento Kleene. Observa-se que o conjunto Σ^* tem contagem infinita uma vez que contém cadeias de comprimentos arbitrariamente longos (CASSANDRAS e LAFORTUNE, 2008).

Exemplo 2.2 (CASSANDRAS e LAFORTUNE, 2008) Seja o alfabeto $\Sigma = \{a, b, g\}$. Então, $\Sigma^* = \{\varepsilon, a, b, g, aa, ab, ag, ba, bb, bg, ga, gb, gg, aaa, ...\}$.

Uma linguagem sobre um conjunto de eventos Σ é, portanto um subconjunto de Σ^*.

O conjunto usual de operações, tais como união, intersecção, diferença e complemento com respeito a Σ^*, são aplicáveis a linguagens pois linguagens são conjuntos. Além destas, são apresentadas aqui outras três operações.

2.2.1 Concatenação

Seja $L_a, L_b \subseteq \Sigma^*$, então

$$L_a L_b := \{s \in \Sigma^* : (s = s_a s_b,) \text{ e } (s_a \in L_a) \text{ e } (s_b \in L_b)\} \tag{2.1}$$

Em palavras, a concatenação das linguagens L_a e L_b é o conjunto de cadeias que podem ser formadas ao selecionar qualquer cadeia em L_a e concatenar ela com qualquer cadeia em L_b (HOPCROFT e ULLMANN, 1979).

2.2.2 Prefixo-Fechamento

Seja $L \subseteq \Sigma^*$, então

$$\overline{L} := \{s \in \Sigma^* : \exists t \in \Sigma^* \, (st \in L)\} \tag{2.2}$$

Em palavras, o prefixo-fechamento de L é a linguagem denotada por $\overline{L}$ e que consiste em todos os prefixos de todas as cadeias em L. Em geral, $L \subseteq \overline{L}$.

L é dita ser prefixo-fechada se $L = \overline{L}$. Assim, a linguagem L é prefixo-fechada se qualquer prefixo de qualquer cadeia em L é um elemento de L (CASSANDRAS e LAFORTUNE, 2008).

2.2.3 Fechamento Kleene

Seja $L \subseteq \Sigma^*$, então

$$L^* := \{\varepsilon\} \cup L \cup LL \cup LLL \cup \ldots \tag{2.3}$$

Esta é a mesma operação que a definida acima para o conjunto Σ, exceto que agora é aplicada ao conjunto L cujos elementos podem ser cadeias de comprimento maior que um. Um elemento de L^* é formado pela concatenação de um finito (mas possivelmente extremamente longo) número de elementos de L; isto inclui a concatenação de "zero" elementos, isto é, a cadeia vazia ε (CASSANDRAS e LAFORTUNE, 2008).

Exemplo 2.3 Seja o alfabeto $\Sigma = \{a, b, c\}$ e as linguagens $L_1 = \{\varepsilon, c, cbb\}$ e $L_2 = \{a\}$. Verifica-se que L_1 e L_2 não são prefixo-fechadas. Verifica-se também que $L_1L_2 = \{a, ca, cbba\}$, $\overline{L_1} = \{\varepsilon, c, cb, cbb\}$, $L_2^* = \{\varepsilon, a, aa, aaa, \ldots\}$ e $L_1 \cup L_2 = \{\varepsilon, c, cbb, a\}$.

2.3 Autômatos Determinísticos de Estados Finitos

Um autômato é um dispositivo capaz de representar uma linguagem de acordo com regras bem definidas. Autômatos são representados por grafos direcionados ou diagramas de transição de estados (CASSANDRAS e LAFORTUNE, 2008).

Um Autômato Finito Determinístico (AFD) G é definido por uma 5-upla $G = (Q, \Sigma, q_0, Q_m, \delta)$ tal que:

Q é um conjunto não vazio e finito de estados do autômato,

Σ é um alfabeto finito de eventos[1] σ associados com as transições em G,

$q_0 \in Q$ é o estado inicial do autômato,

$Q_m \in Q$ é um conjunto de estados marcados[2],

$\delta : Q \times \Sigma \rightarrow Q$ é uma função de transição, possivelmente parcial, ou seja, não necessariamente definida para todo estado de Q para todo evento de Σ.

As palavras *gerador* (que explica a notação G) e *máquina de estados* geralmente também são usadas para descrever o dispositivo acima (CASSANDRAS e LAFORTUNE, 2008).

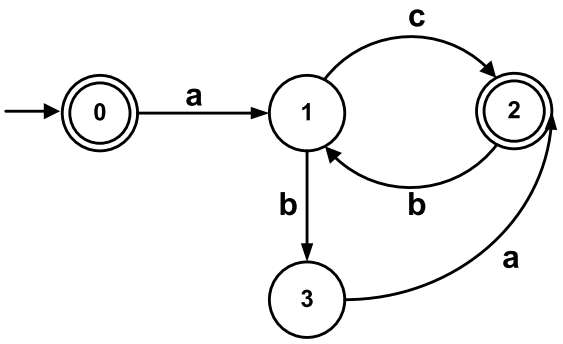
[1] Quando os autômatos são utilizados para modelar o comportamento de um SED, os símbolos são também chamados de eventos.

[2] Também chamados de estados finais.

A função de transição δ, definida para um evento $\sigma \in \Sigma$, pode ser estendida para seqüências de eventos $s \in \Sigma^*$ como a função $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ tal que $q \in Q$, $s \in \Sigma^*$ e $\sigma \in \Sigma$ (PENA, 2007), de modo que $\hat{\delta}(q, \varepsilon) = q$, $\hat{\delta}(q, \sigma) = \delta(q, \sigma)$ e $\hat{\delta}(q, s\sigma) = \delta(\hat{\delta}(q, s), \sigma)$. Esta função de transição estendida $\hat{\delta}$ mapeia para qual estado de Q o sistema transita a partir de um estado de Q com a ocorrência de uma seqüência de eventos $s \in \Sigma^*$.

A seleção apropriada de quais estados marcar é uma decisão de modelagem que depende do problema de interesse. Ao designar certos estados como marcados, pode ser considerado que, ao entrar nestes estados o sistema completou alguma operação ou tarefa (CASSANDRAS e LAFORTUNE, 2008).

Um autômato é representado graficamente por meio de um grafo dirigido, cujos nós representam os estados e os arcos etiquetados representam as transições de estado. O estado inicial é identificado por uma seta apontando para si e os estados marcados são representados por círculos duplos. A Figura 2.4 apresenta um autômato determinístico de estados finitos e sua descrição formal.



$G = (Q, \Sigma, q_0, Q_m, \delta)$
$Q = \{0, 1, 2, 3\}$
$\Sigma = \{a, b, c\}$
$q_0 = 0$
$Q_m = \{0, 2\}$

$\delta(0, a) = 1$
$\delta(1, c) = 2$
$\delta(1, b) = 3$
$\delta(2, b) = 1$
$\delta(3, a) = 2$

Figura 2.4 – Autômato e Descrição Formal

Dado um autômato $G = (Q, \Sigma, q_0, Q_m, \delta)$, define-se o conjunto ativo de eventos de G no estado $q \in Q$, denotado $\Sigma_G(q)$, pelo conjunto $\Sigma_G(q) = \{\sigma \in \Sigma : \delta(q, \sigma)$ é definido$\}$ (DA CUNHA, 2003). No autômato da Figura 2.4, tem-se, $\Sigma_G(0) = \{a\}$, $\Sigma_G(1) = \{b, c\}$, $\Sigma_G(2) = \{b\}$ e $\Sigma_G(3) = \{a\}$.

O estado ativo de um autômato G é definido como sendo o estado que o autômato se encontra após a ocorrência de uma seqüência de eventos $s \in \Sigma^*$, partindo de seu estado inicial. O estado ativo do autômato G ilustrado na Figura 2.4 após a ocorrência da seqüência de eventos *acb*, por exemplo, é o estado *1*.

2.4 Autômatos Não-Determinísticos de Estados Finitos

Na definição de autômatos, as transições de estado descrevem como um evento σ causa uma transição de algum estado q para um novo estado único q'. Considera-se agora que um evento σ no estado q pode causar transições para mais do que um novo estado. Desta forma, $\delta(q, \sigma)$ não representa mais um estado novo específico $q' \in Q$, mas um conjunto de novos estados possíveis. Além disso, é possível incluir a cadeia vazia ε no diagrama de transição de estados de um autômato. Estas duas mudanças referem-se a notação de um *autômato não-determinístico*.

Um autômato determinístico possui a função δ sobre $Q \times \Sigma$. Em contraste, a estrutura de transição de um autômato não determinístico é definida por meio de uma relação sobre $Q \times \Sigma \times Q$ ou, de forma equivalente, uma função de $Q \times \Sigma$ para 2^Q. Dado um conjunto Q, a notação 2^Q significa o conjunto de potência de Q, isto é, o conjunto de todos os subconjuntos de Q (CASSANDRAS e LAFORTUNE, 2008).

Dessa forma, um Autômato Finito Não-Determinístico (AFN) G_{nd} é definido por uma 5-upla $G_{nd} = (Q, \Sigma \cup \{\varepsilon\}, q_0, Q_m, \delta_{nd})$ onde estes objetos possuem a mesma interpretação como na definição de autômatos determinísticos, com a diferença que:

δ_{nd} é uma função $\delta_{nd} : Q \times \Sigma \cup \{\varepsilon\} \rightarrow 2^Q$, isto é, $\delta_{nd}(q, \sigma) \subseteq Q$ toda vez que estiver definida.

O estado inicial pode ser um conjunto de estados, isto é $q_0 \subseteq Q$.

Um exemplo de autômato não-determinístico é ilustrado na Figura 2.5. Percebe-se que quando o evento b ocorre no estado 0, a transição resultante é, ou para o estado 1 ou de volta para o estado 0, caracterizando uma condição de não-determinismo.

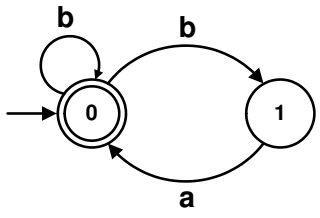


Figura 2.5 – Autômato Não-Determinístico

Para qualquer autômato não-determinístico é possível encontrar um autômato determinístico equivalente, ou seja, um autômato que reconhece a mesma linguagem.

Esse tipo de autômato é apresentado aqui pois serve como referência para as máquinas de estados obtidas na abordagem modular local a serem apresentadas no capítulo 3. Nesse tipo

de modelo de máquina de estados, situações de não-determinismo também aparecem. Durante o processo de redução dessas máquinas, um dos objetivos é eliminar os não-determinismos que existem. Entretanto, vale ressaltar que esse processo de eliminação dos não-determinismos em máquinas de estados, que será apresentado no capítulo 3, difere do processo de determinização convencional utilizado em autômatos.

2.5 Linguagens Representadas por Autômatos

A conexão entre linguagens e autômatos é feita facilmente pela inspeção do diagrama de transição de estados de um autômato. Considere todos os caminhos direcionados que podem ser seguidos no diagrama de transição de estados, começando pelo estado inicial; considere entre esses caminhos todos os que terminam em estados marcados. Isto conduz às noções de linguagens gerada e marcada por um autômato, respectivamente.

2.5.1 Linguagem Gerada e Linguagem Marcada

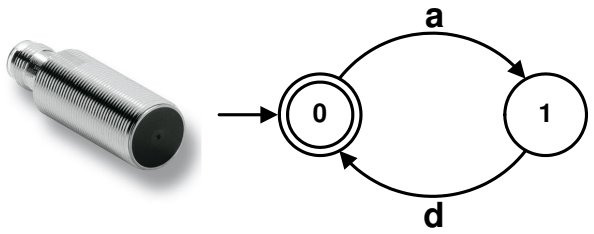
A linguagem gerada de G é $L(G) = \{s \in \Sigma^* : \delta(q_0, s) \text{ é definida}\}$ e a linguagem marcada de G é $L_m(G) = \{s \in L(G) : \delta(q_0, s) \in Q_m\}$ (CASSANDRAS e LAFORTUNE, 2008).

Em palavras, a linguagem $L(G)$ representa todos os caminhos que podem ser seguidos pelo diagrama de transição de estados, partindo do estado inicial; a cadeia correspondente ao caminho é a concatenação dos eventos que representam as transições que compõem o caminho.

A linguagem $L_m(G)$ é o subconjunto de $L(G)$ que consiste somente de cadeias s para as quais $\delta(q_0, s) \in Q_m$, isto é, as cadeias que correspondem aos caminhos que terminam em um estado marcado no diagrama de transição de estados. A linguagem marcada é também chamada de linguagem reconhecida por um autômato e geralmente é dito que um dado autômato é um reconhecedor para uma dada linguagem (CASSANDRAS e LAFORTUNE, 2008).

Segundo (CURY, 2001), um SED pode ser modelado por um autômato G, onde $L(G)$ é o comportamento gerado pelo sistema e $L_m(G)$ é o comportamento marcado ou conjunto de tarefas completas do sistema.

Exemplo 2.4 A Figura 2.6 ilustra um possível autômato para representar o modelo de SED de um sensor indutivo. Dois estados são propostos: o estado inicial com o sensor desativado e o estado onde o sensor está ativado. O conjunto de eventos associados ao sensor é $\Sigma = \{a, d\}$, onde *a* corresponde à ativação do sensor e *d* corresponde à desativação do sensor. Neste caso, a linguagem $L(G)$, que corresponde ao comportamento gerado, consiste de todas as seqüências de eventos que alternam os dois eventos considerados, iniciando com *a* e finalizando com *a* ou *d*. A linguagem vazia $\varepsilon \in L(G)$ e corresponde à situação em que o sensor está no seu estado inicial. Se for considerado como tarefa completa as seqüências que a levam à condição sensor desativado, pode-se afirmar que $L_m(G)$ consiste de todas as seqüências de $L(G)$ que terminam com *d*, acrescida da seqüência ε.



$$L(G) = \{\varepsilon, a, ad, ada, adad, adada, ...\} = (ad)^*(\varepsilon + a)$$

$$L_m(G) = \{\varepsilon, ad, adad, adadad, ...\} = (ad)^*$$

Figura 2.6 – Modelo SED de um Sensor (Autômato e Linguagens)

Um autômato *G* é a representação de duas linguagens: $L(G)$ e $L_m(G)$. Na definição padrão de autômatos de acordo a teoria de autômatos (HOPCROFT e ULLMANN, 1979), a função δ deve ser uma função total e a noção de linguagem gerada não faz sentido pois é sempre igual a Σ^*. Na teoria de SEDs porém, permitir que δ seja parcial trata-se de uma conseqüência do fato que um sistema não possa ser capaz de produzir (ou executar) todas as cadeias em Σ^* (CASSANDRAS e LAFORTUNE, 2008). Em outras palavras, quando o sistema estiver em um determinado estado, em geral nem todos os eventos presentes no alfabeto serão responsáveis por causar transição para outro estado. Alguns eventos simplesmente não tem efeito na dinâmica do sistema dependendo de qual estado esse se encontra.

2.6 Operações sobre Autômatos e Linguagens

Um conjunto de operações é necessário a fim de analisar SEDs modelados por autômatos. Isso é necessário para combinar ou compor, dois ou mais autômatos, assim como operações em um simples autômato a fim de modificar apropriadamente seu diagrama de transição de estados. Nesta seção é dada ênfase à obtenção da componente *Trim* e à

composição síncrona de autômatos, operações relevantes para o processo de síntese de supervisores.

2.6.1 Bloqueio, Acessibilidade, Co-Acessibilidade e Componente *Trim* de um Autômato

Um autômato é dito bloqueante quando existe uma situação onde não é possível atingir um estado marcado a partir de um dado estado. Em se tratando de um SED, a condição de bloqueio é caracterizada por situações indesejadas no funcionamento do sistema. Se um autômato é bloqueante, isso significa que um *deadlock* ou *livelock* pode acontecer. Um *deadlock* correspondente à existência de um estado que ao ser alcançado, impede que o sistema evolua e uma tarefa seja finalizada, ou seja, um estado marcado seja alcançado. Um *livelock* corresponde a um conjunto de estados aos quais a evolução do sistema fica restrita somente a eles, impossibilitando também o sistema de completar uma tarefa.

Para compreensão de acessibilidade, considera-se que os estados que não são acessíveis ou alcançáveis a partir do estado inicial q_0 por alguma cadeia em $L(G)$ podem ser removidos sem afetar as linguagens geradas e marcadas por G. Quando isso é feito, as transições que estão associadas a esses estados também são removidas. Essa operação é denotada $Ac(G)$ onde Ac representa a obtenção da parte acessível do autômato (CASSANDRAS e LAFORTUNE, 2008).

Um estado $q \in G$ é dito ser co-acessível para Q_m, ou simplesmente não bloqueante, se existe uma cadeia em $L_m(G)$ que passa por q; isto significa que existe um caminho no diagrama de transição de estados de G a partir de um estado q até um estado marcado. A operação que consiste em remover todos os estado de G que não são co-acessíveis é denotada por $CoAc(G)$, onde $CoAc$ representa a obtenção da parte co-acessível (CASSANDRAS e LAFORTUNE, 2008).

Um autômato que é acessível e co-acessível é dito ser *Trim*[3]. A componente *Trim* é definida por

$$Trim(G) := CoAc[Ac(G)] = Ac[CoAc(G)] \tag{2.4}$$

onde a comutatividade de Ac e $CoAc$ é facilmente verificada (CASSANDRAS e LAFORTUNE, 2008).

[3] Também chamado de aparado.

Quando uma situação de *deadlock* ou *livelock* acontece, nem todos os estados acessíveis do SED são co-acessíveis e o sistema é dito bloqueante (PENA, 2007). Um SED é não bloqueante se e somente se $L(G) = \overline{L_m(G)}$, isto é, se o autômato correspondente for *Trim*.

A Figura 2.7 ilustra um autômato *G* para exemplificar essas operações (CASSANDRAS e LAFORTUNE, 2008). Se o autômato *G* for utilizado para representar um SED, o estado *5* corresponde a uma condição de *deadlock* enquanto os estados *3* e *4* uma condição de *livelock*, situações em que ocorre bloqueio.

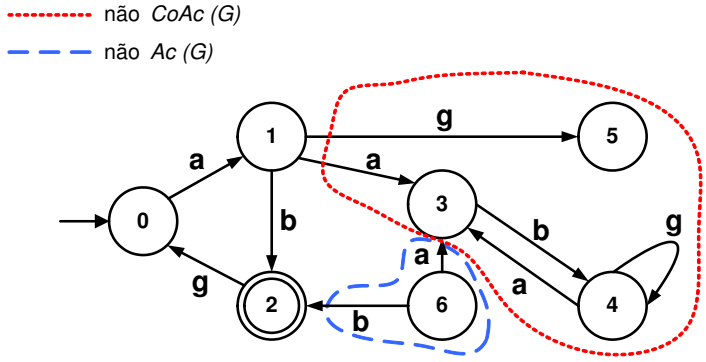


Figura 2.7 – Autômato *G* para Exemplificar Operações (CASSANDRAS e LAFORTUNE, 2008)

A fim de obter a parte *Ac*(*G*) basta remover o estado *6* e as duas transições associadas a ele. Para obter *CoAc*(*G*) é necessário identificar quais são os estados não co-acessíveis (ou bloqueantes) ao estado marcado *2*. Esses são os estados *3*, *4* e *5*. Removendo esses estados e as transições associadas a eles obtém-se a parte *CoAc*(*G*). Nota-se que o estado *6* não é removido uma vez que ele pode alcançar o estado *2*. As componentes acessível *Ac*(*G*), co-acessível *CoAc*(*G*) e *Trim(G)* do autômato *G* são ilustradas na Figura 2.8.

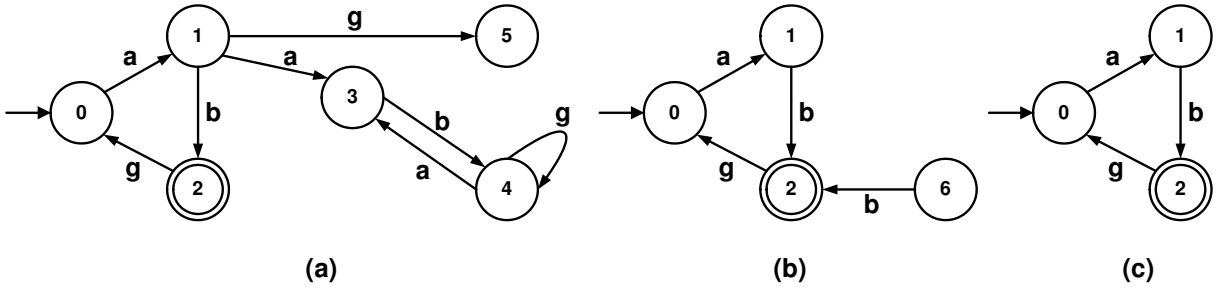


Figura 2.8 – (a) Componente *Ac(G)*, (b) Componente *CoAc(G)*, (c) Componente *Trim(G)*

2.6.2 Composição Síncrona de Autômatos e Linguagens

A composição síncrona (ou paralela) modela um comportamento de ligação para um conjunto de autômatos que operam concorrentemente. Sejam os autômatos $G_1 = (Q_1, \Sigma_1, q_{01}, Q_{m1}, \delta_1)$ e $G_2 = (Q_2, \Sigma_2, q_{02}, Q_{m2}, \delta_2)$, define-se a composição síncrona de G_1 e G_2, denotada $G_1 \| G_2$, pelo autômato

$$G_1 \parallel G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, (q_{01}, q_{02}), Q_{m1} \times Q_{m2}, \delta_{12}) \tag{2.5}$$

onde $\delta_{12} : (Q_1 \times Q_2) \times (\Sigma_1 \cup \Sigma_2) \rightarrow (Q_1 \times Q_2)$ é definida por

$$\delta_{12}\left((q_1, q_2), \sigma\right) = \begin{cases} \left(\delta_1(q_1,\sigma), \delta_2(q_2,\sigma)\right) & \text{se } \sigma \in \Sigma_{G_1}(q_1) \cap \Sigma_{G_2}(q_2) \\ \left(\delta_1(q_1,\sigma), q_2\right) & \text{se } \sigma \in \Sigma_{G_1}(q_1) - \Sigma_2 \\ \left(q_1, \delta_2(q_2,\sigma)\right) & \text{se } \sigma \in \Sigma_{G_2}(q_2) - \Sigma_1 \\ \text{não definida} & \text{caso contrário} \end{cases} \tag{2.6}$$

para $(q_1, q_2) \in Q_1 \times Q_2$ e $\sigma \in \Sigma_1 \cup \Sigma_2$ (CURY, 2001).

Na composição síncrona, um evento comum, isto é, um evento em $\Sigma_1 \cap \Sigma_2$, só pode ser executado se ambos os autômatos executam-no simultaneamente. Assim, os dois autômatos são sincronizados nos eventos em comum. Os demais eventos não estão sujeitos a esta restrição e podem ser executados a qualquer momento, ou seja, ocorrem assincronamente. Neste tipo de interconexão, um autômato pode executar seus eventos particulares sem a participação do outro componente; entretanto, um evento em comum só pode acontecer se ambos os autômatos podem executá-lo (CASSANDRAS e LAFORTUNE, 2008).

Se $\Sigma_1 = \Sigma_2$, então a composição é completamente síncrona, ou seja, todos os eventos entre G_1 e G_2 são sincronizados e tem-se $L(G_1) \parallel L(G_2) = L(G_1) \cap L(G_2)$ e $L_m(G_1) \parallel L_m(G_2) = L_m(G_1) \cap L_m(G_2)$, isto é, as linguagens da composição síncrona são as interseções das linguagens respectivas dos componentes.

Se $\Sigma_1 \cap \Sigma_2 = \varnothing$, então não existem transições sincronizadas e $G = G_1 \parallel G_2$ é o comportamento concorrente de G_1 e G_2. Assim, $L(G) = L(G_1) \parallel L(G_2)$ e $L_m(G) = L_m(G_1) \parallel L_m(G_2)$, isto é, as linguagens da composição síncrona são os produtos síncronos das linguagens respectivas dos autômatos componentes.

A composição síncrona satisfaz também as propriedades comutativa, isto é, $G_1 \parallel G_2 = G_2 \parallel G_1$, e associativa, isto é, $(G_1 \parallel G_2) \parallel G_3 = G_1 \parallel (G_2 \parallel G_3)$. Dessa forma, essa composição pode ser estendida para mais de dois autômatos.

Se todos os alfabetos dos autômatos são assíncronos entre si, o número de estados do autômato resultante cresce exponencialmente com o número de autômatos compostos (PENA, 2007).

Exemplo 2.5 (CASSANDRAS e LAFORTUNE, 2008) Considere os autômatos G_1 e G_2 apresentados na Figura 2.9, que possuem alfabetos $\Sigma_1 = \{a, b, g\}$ e $\Sigma_2 = \{a, b\}$, respectivamente.

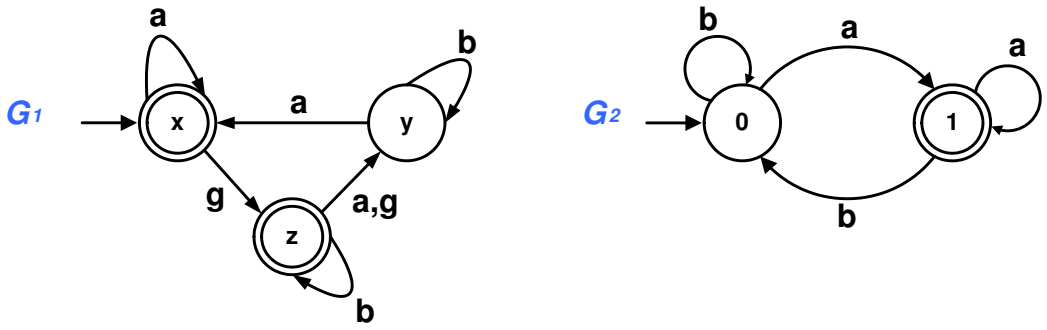


Figura 2.9 – Autômatos G_1 e G_2

A composição síncrona de G_1 com G_2, isto é, $G = G_1 \| G_2$, é ilustrada na Figura 2.10.

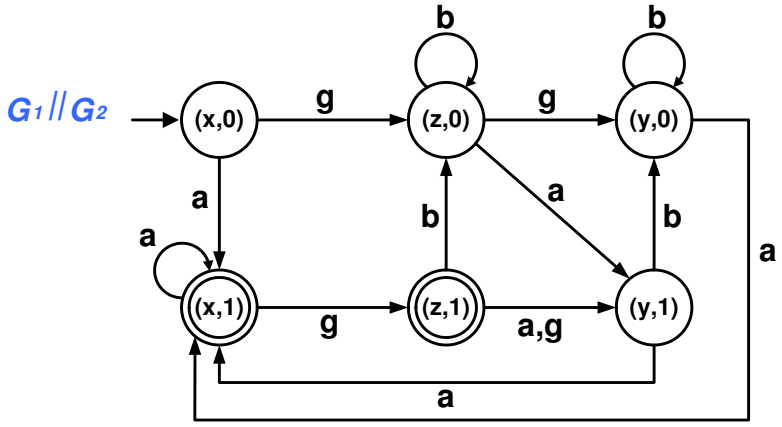


Figura 2.10 – Autômato $G = G_1 \| G_2$

2.7 Linguagens e Expressões Regulares

Representações por autômatos são mais convenientes de usar do que uma enumeração de todas as cadeias presentes em uma linguagem. Entretanto, nem sempre é possível representar uma linguagem por meio de um autômato de estados finitos. Assim, quando uma linguagem pode ser marcada por um autômato de estados finitos está é denominada linguagem regular (CASSANDRAS e LAFORTUNE, 2008).

Outra forma de representar linguagens regulares é através de expressões compactas chamadas expressões regulares. Estas podem ser definidas recursivamente como segue (CASSANDRAS e LAFORTUNE, 2008):

1. $\varnothing$ é uma expressão regular que denota o conjunto vazio, ε é uma expressão regular que denota o conjunto $\{\varepsilon\}$ e σ é uma expressão regular que denota o conjunto $\{\sigma\}$, para todo $\sigma \in \Sigma$.

2. Se r e s são expressões regulares, então rs, $(r + s)$, r^* e s^* são expressões regulares.

3. Não existem expressões regulares a não ser aquelas construídas ao aplicar as regras 1 e 2 acima um número finito de vezes.

As expressões regulares fornecem uma representação finita compacta para linguagens potencialmente complexas com um número infinito de cadeias. Qualquer linguagem que pode ser descrita por uma expressão regular é uma linguagem regular; de forma recíproca, qualquer linguagem regular pode ser descrita por uma expressão regular (CASSANDRAS e LAFORTUNE, 2008).

Os autômatos determinísticos de estados finitos e as expressões regulares possuem o mesmo poder de expressão (DA CUNHA, 2003), pois ambos conseguem representar a mesma classe de linguagens, as linguagens regulares.

Exemplo 2.6 (CASSANDRAS e LAFORTUNE, 2008) Considere o alfabeto $\Sigma = \{a, b, g\}$. A expressão regular $(a + b)g^*$ denota a linguagem $L = \{a, b, ag, bg, agg, bgg, aggg, bggg, ...\}$ que consiste em todas as cadeias que começam ou com o evento a ou o evento b e são seguidas por uma repetição do evento g. Observa-se que embora L contenha infinitos elementos, a expressão regular correspondente fornece uma representação simples de L.

2.8 Máquinas de Estados com Saídas

As máquinas de estados de *Mealy* (MEALY, 1955) e de *Moore* (MOORE, 1956), ou autômatos de *Mealy* e de *Moore* são variantes do Autômato Finito Determinístico (AFD), sendo também denominadas como autômatos com entradas e saídas (CASSANDRAS e LAFORTUNE, 2008) ou simplesmente autômatos finitos com saídas (MENEZES, 1998) ou máquinas de estados com saídas.

O AFD também consiste em uma máquina de estados finitos e a diferença dos autômatos com saídas é que estes possuem a geração de uma palavra de saída associada a eles. No intuito de simplificar a nomenclatura, a partir deste ponto do trabalho os autômatos com saídas serão denominados máquinas de estados de *Mealy* e de *Moore*.

Uma máquina de estados finitos (do inglês, *FSM – Finite State Machine*) modela o comportamento de um sistema, sendo composta por estados, transições e ações. Os estados representam as situações na qual a máquina de estados pode estar em algum momento. Sendo assim, existe um tipo de memória: como a máquina de estados pode ter alcançado a situação

atual. À medida que a aplicação é executada, o estado muda de tempos em tempos e as saídas podem depender do estado atual assim como das entradas (WAGNER *et al*, 2006).

Dentre os tipos de ações existentes, destacam-se as ações executadas durante a transição de estados (*input actions*), isto é, a saída depende do estado atual e das condições de entrada que vão permitir a ocorrência da transição e as ações de entrada (*entry actions*), ou seja, a ação é executada quando entra no estado. Uma máquina de *Mealy* utiliza apenas *input actions* enquanto uma máquina de *Moore* utiliza apenas *entry actions*. Este trabalho trata desses dois tipos de abordagens de máquinas de estado.

Uma máquina de *Mealy* é um autômato finito determinístico com saídas associadas às transições e pode ser representada formalmente pela 6-upla $M = (Q, \Sigma, q_0, Q_m, \delta, \Delta)$ tal que (MENEZES, 1998):

Q é um conjunto não vazio e finito de estados do autômato,

Σ é o alfabeto de entrada (um conjunto de símbolos não vazio e finito σ),

$q_0 \in Q$ é o estado inicial do autômato,

$Q_m \in Q$ é um conjunto de estados marcados,

$\delta : Q \times \Sigma \rightarrow Q \times \Delta^*$ é uma função de transição, possivelmente parcial, ou seja, não necessariamente definida para todo estado de Q para todo evento de Σ,

Δ é o alfabeto de saída (um conjunto de símbolos não vazio e finito).

Assim, as componentes Q, Σ, q_0 e Q_m são como no autômato finito determinístico. A função de transição pode ser representada como um grafo finito direto da mesma forma que nos autômatos convencionais onde é possível adicionar uma saída associada a cada transição existente.

Uma máquina de *Moore* é um autômato finito determinístico com saídas associadas aos estados e pode ser representada formalmente por uma 7-upla $M = (Q, \Sigma, q_0, Q_m, \delta, \Delta, \omega)$ tal que (MENEZES, 1998):

Q é um conjunto não vazio e finito de estados do autômato,

Σ é o alfabeto de entrada (um conjunto de símbolos não vazio e finito σ),

$q_0 \in Q$ é o estado inicial do autômato,

$Q_m \in Q$ é um conjunto de estados marcados,

$\delta : Q \times \Sigma \rightarrow Q$ é uma função de transição, possivelmente parcial, ou seja, não necessariamente definida para todo estado de Q para todo evento de Σ,

Δ é o alfabeto de saída (um conjunto de símbolos não vazio e finito),

$\omega : Q \rightarrow \Delta^*$ é a função de saída, a qual é uma função total.

Portanto, as componentes Q, Σ, q_0 e Q_m são como no autômato finito determinístico e Δ é como na máquina de *Mealy*. A função de transição pode ser representada como um grafo finito direto da mesma forma que nos autômatos convencionais onde se adiciona uma saída associada a cada estado existente.

Uma máquina de *Mealy* pode ser facilmente convertida em uma máquina de *Moore* e vice-versa (CASSANDRAS e LAFORTUNE, 2008).

A máquina de *Mealy* utiliza apenas *input actions*, isto é, a saída depende do estado atual e das condições de entrada. Um exemplo simples de uma máquina de *Mealy* com dois estados é mostrado na Figura 2.11. Considera-se inicialmente que a máquina esteja no estado *0* e a transição *1* acontece. A máquina então executa a ação *2* e vai para o estado *1*. No estado *1* se a transição *2* ocorrer, a ação *2* é executada e a máquina permanece neste mesmo estado. Este tipo de transição é comumente conhecido como *self-loop*. Por fim, se a máquina está no estado *1* e uma condição lógica *AND* entre as transições *1* e *2* ocorre, ela executa a ação *1* e retorna ao estado *0*.

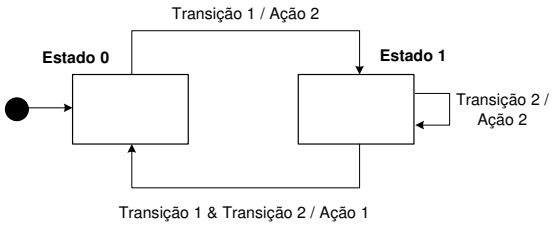


Figura 2.11 – Exemplo de Máquina de Estados de *Mealy*

A máquina de *Moore* utiliza apenas ações de entrada, isto é, a saída depende somente do estado. A vantagem do modelo de *Moore* se comparado com um modelo de *Mealy* é a simplificação do comportamento. Por outro lado, o uso de uma máquina de *Mealy* normalmente leva a uma redução no número de estados se comparado com uma máquina de *Moore*. A máquina de *Moore* equivalente à máquina de *Mealy* mostrada na Figura 2.11 é ilustrada na Figura 2.12. A ação de saída de cada transição na máquina de *Mealy* equivalente corresponde à ação de saída do estado em que a mesma transição entra na máquina de *Moore* (CASSANDRAS e LAFORTUNE, 2008). Considera-se inicialmente que a máquina esteja no estado *0* e a transição *1* ocorre. A máquina então transita para o estado *1* e, ao entrar neste estado, executa a ação *2*. Se a máquina está no estado *1* e a transição *2* ocorre, um *self-loop* faz com que a máquina permaneça no estado *1* e executa a ação *2* novamente. Para finalizar, se a máquina está no estado *1* e a condição lógica *AND* entre as transições *1* e *2* for verdadeira, a máquina transita para o estado *0* e a ação *1* é executada ao entrar neste estado.

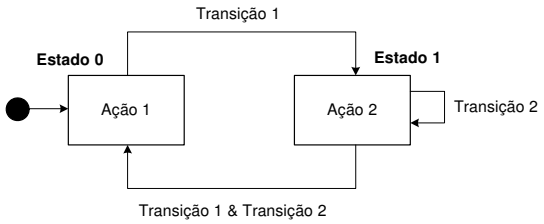


Figura 2.12 – Exemplo de Máquina de Estados de *Moore*

Além disso, aproveitam-se esses exemplos para definir importantes notações para os operandos e também para ilustrar a representação dos modelos de máquinas de estados com saídas a serem utilizados neste trabalho. Um círculo cheio representa o estado inicial da máquina; os estados são representados por retângulos; uma barra "/" é utilizada para fazer a distinção entre transição / ação em uma máquina de *Mealy* e o operador lógico "&" representa uma condição booleana *AND*. Essas notações vão aparecer no decorrer deste trabalho.

2.9 Conclusões

Neste capítulo apresentou-se uma breve revisão de Sistemas a Eventos Discretos (SEDs), assim como sua representação a partir da teoria de linguagens e autômatos. Além disso, foi efetuada uma introdução às máquinas de estados de *Mealy* e *Moore*, também denominadas máquinas de estados com saídas. A compreensão desses conceitos serve como base teórica para o entendimento dos capítulos seguintes.

No próximo capítulo é apresentada a proposta para controle supervisório usando máquinas de estados finitos onde são obtidas máquinas de estados de *Mealy* e *Moore* a partir dos autômatos dos supervisores obtidos através da Teoria de Controle Supervisório (TCS). As abordagens monolítica e modular local da TCS são apresentadas e a modelagem de máquinas de estados com saídas é realizada para esses dois tipos de abordagem.

Capítulo 3

Proposta para Controle Supervisório usando Máquinas de Estados com Saídas

Neste capítulo é descrita a metodologia proposta para a obtenção de máquina de estados com saídas a partir do autômato que modela o supervisor obtido por intermédio da Teoria de Controle Supervisório (TCS) de Sistemas a Eventos Discretos (SEDs). A abordagem monolítica dessa metodologia baseia-se em formalismos para modelar o supervisor através de Máquinas de Estado Finitos (MEF) de *Mealy* e *Moore*. A metodologia fundamentada na abordagem monolítica é então estendida para a obtenção de controladores locais, onde a máquina de estados é obtida a partir dos autômatos que modelam os supervisores modulares locais obtidos a partir da abordagem modular local (DE QUEIROZ e CURY, 2002) de controle supervisório. Dessa forma, uma máquina de estados local é obtida a partir de cada supervisor local presente na planta global.

A organização deste capítulo é como segue. Na Seção 3.1 apresenta-se uma visão geral da metodologia de projeto proposta para controle supervisório usando máquinas de estados com saídas. Na Seção 3.2 é mostrada a abordagem monolítica de síntese de supervisores para controle de SEDs usando a TCS, que serve como base para este trabalho. A Seção 3.3 apresenta um sistema de manufatura simples para, ao longo dessa dissertação, exemplificar a proposta de modelagem utilizando máquinas de estados com saídas. A Seção 3.4 trata da redução de supervisores. Na Seção 3.5 são apresentados algoritmos para obter máquinas de *Mealy* e *Moore* de caráter monolítico, assim como procedimentos para redução dessas máquinas visando obter modelos mais simples. Na Seção 3.6 é apresentada a abordagem modular local para controle de Sistemas a Eventos Discretos (SEDs). Na Seção 3.7 são apresentados algoritmos para obter máquinas de estados com saídas de caráter modular local, assim como procedimentos para redução dessas máquinas. Para finalizar, a Seção 3.8 aborda as conclusões deste capítulo.

3.1 Metodologia de Projeto

A Figura 3.1 ilustra uma visão comparativa entre a metodologia convencional usando a TCS e a metodologia proposta usando máquinas de estados com saídas para síntese e implementação de supervisores para controle de SEDs.

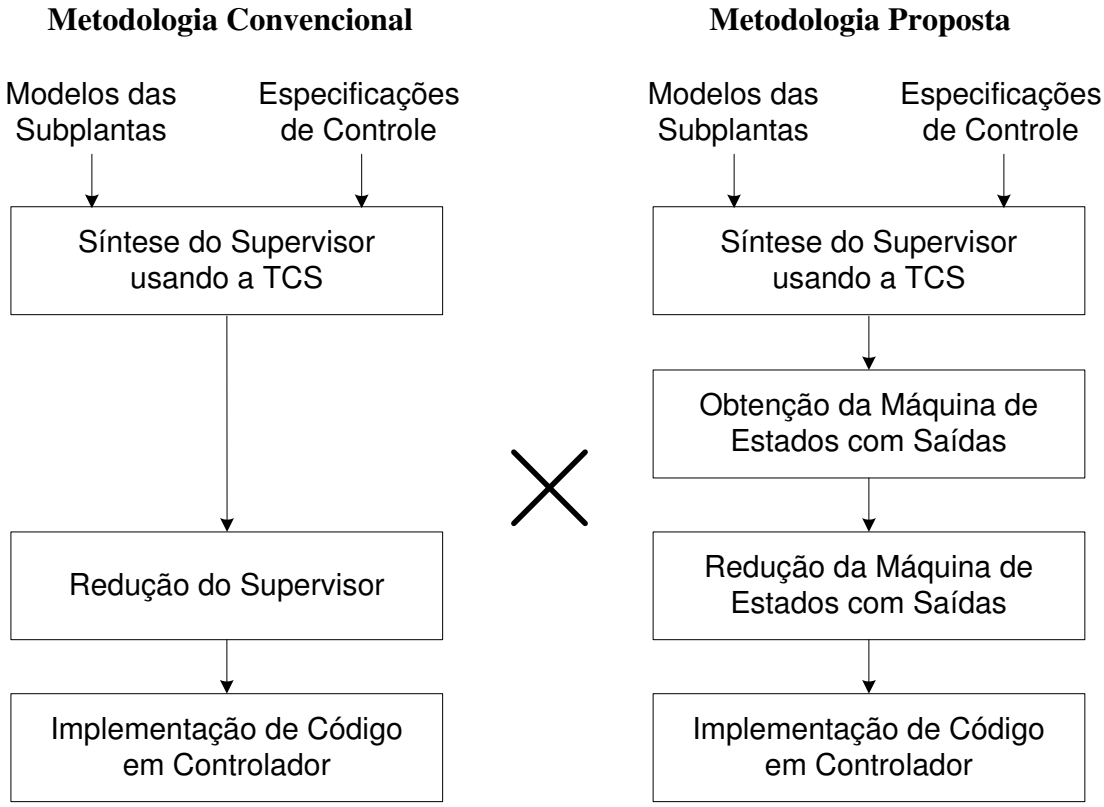


Figura 3.1 –Comparativo entre a Metodologia Convencional e a Metodologia Proposta para Controle Supervisório de SEDs

Na metodologia convencional, primeiramente é realizada a síntese de um supervisor por meio de autômatos usando a TCS. Esse supervisor é obtido a partir dos modelos das subplantas e das especificações de controle. Em seguida, esse supervisor é reduzido a fim de eliminar redundâncias incorporadas ao modelo durante o procedimento de síntese, preservando porém a mesma ação de controle em malha fechada em relação à planta. Dessa forma, o autômato que representa o supervisor reduzido possui menos estados e transições do que o autômato que representa o supervisor original. O supervisor reduzido serve então como base para a implementação de código em um controlador.

Na metodologia proposta, a síntese de um supervisor usando a TCS é feita exatamente da mesma forma que a metodologia convencional. Esse procedimento consiste em uma das propostas deste trabalho, ou seja, se tem como objetivo encontrar um supervisor representado por máquinas de estados com saídas que seja baseado no supervisor original encontrado com

o uso da TCS, tal que a estrutura usada para síntese de supervisores usando a TCS seja mantida a fim de se tirar proveito de suas vantagens. O autômato que representa o supervisor original serve então como parâmetro de entrada para o algoritmo de conversão para obter a máquina de estados com saídas. A máquina com saídas é então submetida a um processo de redução, visando diminuir o número de transições e estados, seguindo assim a mesma linha de raciocínio utilizada para redução de autômatos, visando ter um modelo mais compacto. Além de facilitar a visualização do modelo, esse procedimento é realizado com o intuito de facilitar a implementação de código em uma unidade de controle síncrona. A máquina de estados com saídas em sua forma reduzida serve como base para a implementação de código em um controlador.

Os modelos obtidos são genéricos para qualquer tipo de controlador síncrono, tais como um Controlador Lógico Programável (CLP), computadores, controladores embarcados ou alguma outra unidade de processamento. Entretanto, no contexto deste trabalho aborda-se apenas a implementação de código em CLP usando linguagem *Ladder*, sendo voltado para aplicações no âmbito industrial. Os aspectos relativos à implementação de código neste tipo de controlador serão discutidos no capítulo seguinte.

Procedimento similar é proposto para obtenção de máquina de estados com saídas a partir da abordagem modular local. Nessa abordagem porém, para cada supervisor local é obtido um modelo de máquina local correspondente, responsável pelo controle de uma parte específica sob supervisão.

3.2 Teoria de Controle Supervisório: Abordagem Monolítica

A abordagem monolítica de síntese de supervisor para SEDs foi proposta em (RAMADGE e WONHAM, 1989), sendo também chamada de Abordagem RAMADGE-WONHAM (RW). Nesta abordagem, o comportamento da planta é representado por um autômato G que corresponde ao comportamento não controlado deste SED. A planta em geral consiste num conjunto de subsistemas G_i's atuando em paralelo para realizar uma dada função. Esses subsistemas possuem um comportamento individual e a composição síncrona (CASSANDRAS e LAFORTUNE, 2008) dos autômatos que representam os comportamentos desses subsistemas define o comportamento da planta global. A premissa é de que esse comportamento é insatisfatório e deve ser modificado pelo controlador; modificar o comportamento é entendido como *restringir o comportamento para um subconjunto da linguagem gerada, L(G)* (CASSANDRAS e LAFORTUNE, 2008). Isso é efetuado pois a

planta modela todos os comportamentos possíveis em malha aberta, inclusive comportamentos que são indesejáveis e devem ser evitados, tais como bloqueio, via *deadlock* ou *livelock*, ou estados que são fisicamente inadmissíveis, tal como, por exemplo, a tentativa de colocar uma peça em um *buffer* que já está cheio ou a colisão entre dois robôs.

As restrições do comportamento da planta são feitas por intermédio das especificações de controle. Especificações de controle E_i's podem expressar requisitos funcionais (como impor uma seqüência desejada de eventos), requisitos de segurança (como evitar uma seqüência de eventos indesejada que levaria um sistema a uma condição de colisão), requisitos de decisão (como garantir prioridade ao acesso de um recurso comum), entre outros. Assim, os subsistemas da planta devem atuar de forma coordenada e para isso é introduzido um agente de controle denominado supervisor.

O supervisor interage com a planta numa estrutura em malha fechada, como indicado na Figura 3.2 (RAMADGE e WONHAM, 1989).

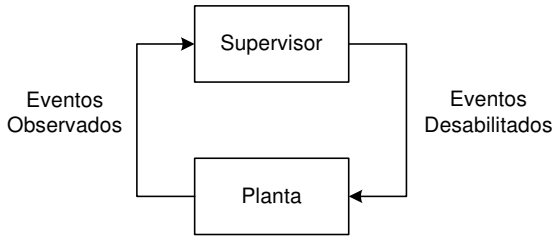


Figura 3.2 – Esquema de Controle Supervisório Monolítico (RAMADGE e WONHAM, 1989)

De acordo com a TCS, a planta gera eventos $\sigma \in \Sigma$ de forma espontânea, enquanto o supervisor observa e é responsável por proibir certos eventos na planta. Esse comportamento é resultado das especificações de controle e serve para impedir que ocorram cadeias de eventos indesejadas.

Um evento pode ser controlável ou não controlável, tal que o conjunto de eventos Σ pode ser particionado em dois subconjuntos distintos de acordo com a natureza de seus eventos: eventos controláveis Σ_c e não controláveis Σ_{uc}, onde $\Sigma = \Sigma_c \cup \Sigma_{uc}$. Os eventos em Σ_c podem ser impedidos de ocorrer, ou em outras palavras, desabilitados enquanto os eventos de Σ_{uc} não podem ser impedidos de ocorrer pela ação de controle, ou seja, são considerados permanentemente habilitados.

O supervisor é responsável por desabilitar ou não os eventos controláveis. A ação de controle que o supervisor aplica sobre a planta corresponde à informação de quais eventos controláveis estão desabilitados (ou, de forma complementar, habilitados). Quando o controle é realizado em malha fechada, o supervisor atua também como um observador de eventos. Ele

observa os eventos gerados pela planta e, baseado nessa informação, aplica uma nova ação de controle à planta informando que eventos estão desabilitados. A ação de controle do supervisor é dita passiva, pois esta indica apenas quais são os eventos habilitados e não força a geração de um evento (DA CUNHA, 2003).

3.3 Um Estudo de Caso: Sistema de Manufatura

Um estudo de caso é apresentado nesta seção com o intuito de ilustrar o uso da TCS e a metodologia para obtenção de máquinas de estados com saídas. Considera-se um sistema simples de manufatura (DE QUEIROZ e CURY, 2000) composto por 3 aparelhos[4] e 2 *buffers* intermediários entre os aparelhos, com capacidade de armazenar uma peça, conforme ilustrado na Figura 3.3. Os aparelhos são representados por A_i, onde i = 1,2,3 e os *buffers* são representados por B_j, onde j = 1,2.

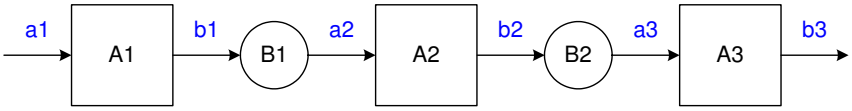


Figura 3.3 – Sistema de Manufatura

Os eventos controláveis que correspondem ao início de operação dos aparelhos são representados por a_i, enquanto os eventos não-controláveis que correspondem ao fim de operação dos aparelhos são representados por b_i, onde i = 1,2,3.

A Tabela 3.1 descreve em detalhes todos os eventos modelados para esse sistema.

Tabela 3.1 – Descrição dos Eventos do Sistema de Manufatura

Equipamento (Subplanta)	**Evento**	**Descrição**
Aparelho A_1	a_1	Início de operação do aparelho *A1*
	b_1	Fim de operação do aparelho *A1*
Aparelho A_2	a_2	Início de operação do aparelho *A2*
	b_2	Fim de operação do aparelho *A2*
Aparelho A_3	a_3	Início de operação do aparelho *A3*
	b_3	Fim de operação do aparelho *A3*

[4] Optou-se por chamar essas subplantas de aparelhos ao invés de máquinas, a fim de evitar confusão de nomenclatura com máquina de estados.

3.3.1 Modelagem da Planta

A modelagem da planta global é feita por meio de um autômato G obtido por intermédio da composição síncrona de suas n subplantas (ou subsistemas)

$$G = G_1 \| G_2 \| G_3 = \mathop{\|}_{i=1}^{n=3} G_i \tag{3.1}$$

A planta pode também ser representada pelas linguagens gerada $L(G)$ e marcada $L_m(G)$, sendo $L(G)$ prefixo-fechada e $L_m(G)$ contida em $L(G)$. Essas linguagens são dadas pelo produto síncrono das linguagens dos n autômatos que modelam os subsistemas.

Para o sistema de manufatura, são considerados apenas os aparelhos como subsistemas da planta global. Os *buffers* são levados em conta somente no modelo das especificações. Três subplantas são modeladas, correspondendo aos três aparelhos presentes no sistema.

O comportamento discreto dos aparelhos é modelado pelo autômato da Figura 3.4, onde os eventos controláveis a_i indicam início de operação do aparelho e os eventos não controláveis b_i indicam fim de operação. Nos diagramas dos autômatos, um evento controlável é representado por um pequeno corte nos arcos correspondentes às transições deste evento (CURY, 2001).

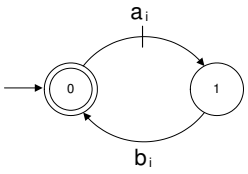


Figura 3.4 – Modelo das Subplantas

3.3.2 Modelagem das Especificações de Controle

A modelagem das especificações de controle é feita por meio de um autômato E obtido pela composição síncrona das m especificações existentes

$$E = E_1 \| E_2 = \mathop{\|}_{j=1}^{m=2} E_j \tag{3.2}$$

Esta especificação E é responsável por restringir o comportamento da planta a comportamentos desejados, que são definidos pelas especificações E_j.

As duas especificações de controle para o sistema de manufatura são restrições de coordenação para evitar *overflow* (sobrecarga) ou *underflow* (retirada de peças de um buffer vazio) nos *buffers*. Estas restrições expressam a idéia de que se devem alternar os eventos b_i e a_{i+1}, iniciando-se pelos eventos b_i. O autômato que modela estas especificações é mostrado na Figura 3.5.

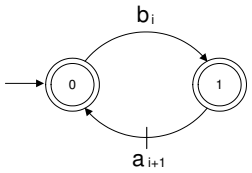


Figura 3.5 – Especificações de Controle

Em se tratando de marcação de estados, geralmente costuma-se marcar todos os estados das especificações e marcar somente os estados que caracterizam uma tarefa completa nas subplantas. Esse é o procedimento mais usual na síntese de controladores para SEDs usando a TCS e que será adotado neste trabalho. Entretanto, isso não é via de regra, ficando a critério do projetista. Existem situações onde opta-se por marcar os estados que representam as tarefas completas nas especificações, e não nas subplantas. Além disso, existem determinados sistemas tais como linhas de transferência com realimentação de peças em *buffers* (CURY, 2001) onde não é permitido marcar todos os estados das especificações. Esse procedimento faria com que existisse uma situação de bloqueio no sistema. Sendo assim, uma análise deve ser efetuada de acordo com o sistema em estudo, a fim de se adotar uma marcação de estados adequada para a síntese de supervisores.

3.3.3 Cálculo da Linguagem Alvo

A linguagem alvo K reflete o comportamento desejado para a planta G operando de acordo com a especificação E, sendo dada por $K = E \parallel L_m(G)$.

3.3.4 Cálculo do Supervisor

Para efetuar o controle da planta G considera-se uma estrutura de controle Γ. Uma opção de controle $\gamma \in \Gamma$ contém o conjunto ativo de eventos habilitados a ocorrerem no sistema (CURY, 2001). Com a definição de controlabilidade de eventos, a estrutura de controle toma a forma

$$\Gamma = \{\gamma \in 2^{\Sigma} : \gamma \supseteq \Sigma_{uc}\}, \tag{3.3}$$

onde a condição $\gamma \supseteq \Sigma_{uc}$ indica que os eventos não controláveis não podem ser desabilitados pela ação de controle (CURY, 2001). Assim, um supervisor representado por um autômato S possui transições de acordo com a ocorrência de eventos na planta G. A ação de controle do supervisor corresponde a desabilitar os eventos controláveis em G que não são permitidos de ocorrer em S após uma cadeia de eventos observada.

Sendo a linguagem $L(S)$ controlável em relação a G, o SED resultante da composição síncrona de S e G ($S \parallel G$) é denominado S/G, onde somente as transições permitidas tanto na planta G quanto no supervisor S são permitidas. Dessa forma, o comportamento em malha fechada é dado por $L(S/G) = L(S \parallel G)$ e $L_m(S/G) = L_m(S \parallel G)$ (CURY, 2001).

Embora a linguagem alvo K reflita o comportamento desejado para a planta, um supervisor não bloqueante S existe para G se e somente se K for controlável (CURY, 2001). Assim, é possível projetar um supervisor S que implemente a máxima linguagem controlável denotada por $SupC(K, L(G))$. Esse supervisor é dito ótimo, pois representa o comportamento menos restritivo possível da planta G que satisfaz a especificação E (CURY, 2001).

O cálculo do supervisor S consiste em um procedimento iterativo que visa identificar maus estados num autômato R que modela a linguagem alvo K. Sendo G, R, $x = \delta_G(x_0, s)$ e $q = \delta_R(q_0, s)$, onde $s \in \overline{K}$ é uma seqüência de eventos, um estado q é um mau estado se existe $\sigma \in \Sigma_{uc}$ tal que $\sigma \in \Sigma_G(x)$ e $\sigma \notin \Sigma_R(q)$.

O algoritmo para cálculo do supervisor S é descrito a seguir (CURY, 2001). Dados G e R, sendo R o autômato que modela a linguagem alvo K:

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

O funcionamento desse algoritmo para calcular o supervisor S será explicado usando o sistema de manufatura como exemplo. A planta G consiste na composição síncrona dos modelos dos autômatos de todas as subplantas, conforme ilustrado na Figura 3.6.

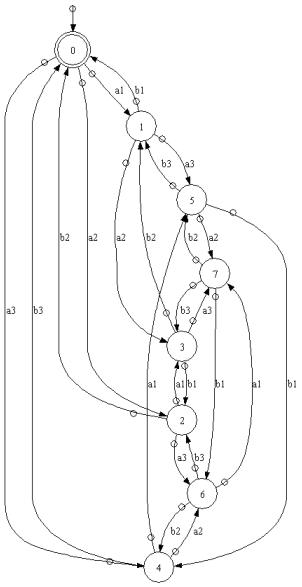


Figura 3.6 – Planta G para o Sistema de Manufatura

e a linguagem alvo *K*, é representada pelo autômato *R* apresentado na Figura 3.7,

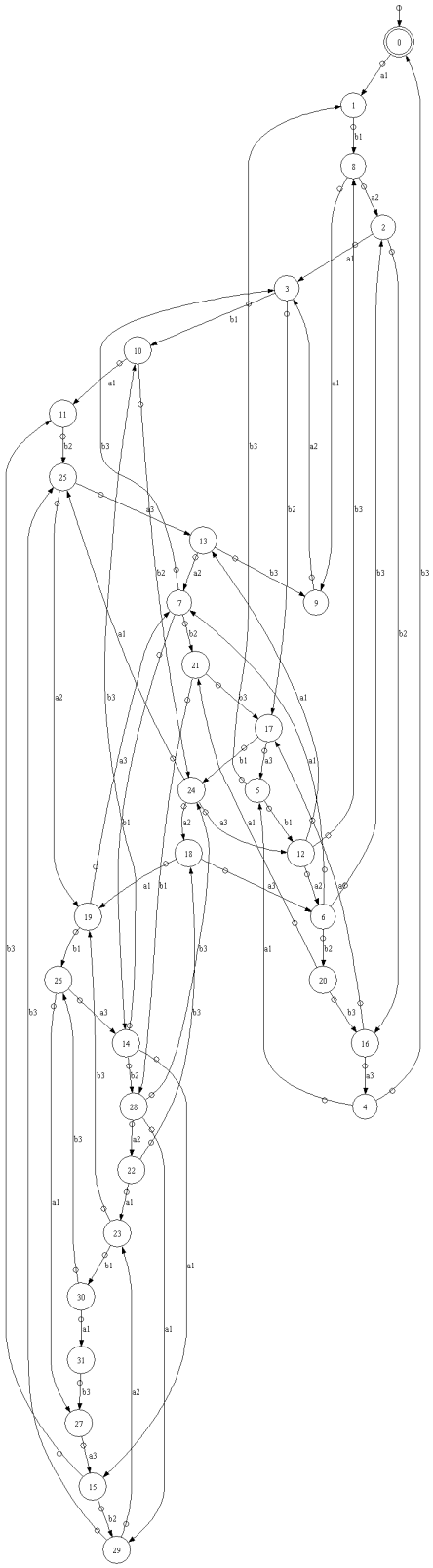


Figura 3.7 – Autômato *R* que Representa a Linguagem Alvo *K* para o Sistema de Manufatura

onde a_1, a_2 e a_3 são os eventos controláveis e b_1, b_2 e b_3 são os eventos não controláveis.

A seguir, é descrita uma iteração do algoritmo utilizado para eliminação dos maus estados, visando obter um autômato *S* que representa a máxima linguagem controlável *SupC(K, L(G))*. Considera-se a seguinte cadeia de eventos: a_1-b_1-a_2-a_1-b_1-a_1. A Figura 3.8 ilustra parte da planta *G* que descreve a cadeia de eventos considerada. Observa-se que a planta atinge o estado 3 ao terminar essa cadeia de eventos.

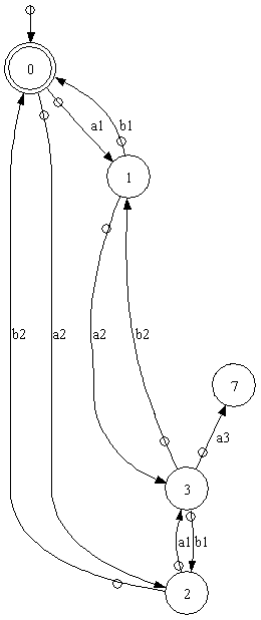


Figura 3.8 – Parte da Planta *G* para mostrar a Cadeia de Eventos Considerada

A Figura 3.9 ilustra parte do autômato *R* que descreve a mesma cadeia de eventos considerada. As transições mostradas sem atingir um estado de chegada são simplificações na Figura 3.9 visando facilitar a compreensão da mesma. Elas foram mantidas com o intuito de ilustrar que essas transições são passíveis de ocorrer nos estados correspondentes.

Observa-se que a cadeia de eventos descrita vai resultar no estado *11* do autômato *R*. Ao comparar o estado *11* de *R* com o estado *3* do autômato da planta *G*, verifica-se que o evento não controlável b_1 que aparece no autômato da planta não aparece no autômato *R*. Assim, de acordo com a definição para maus estados, o estado *11* de *R* caracteriza-se como um mau estado, devendo ser eliminado.

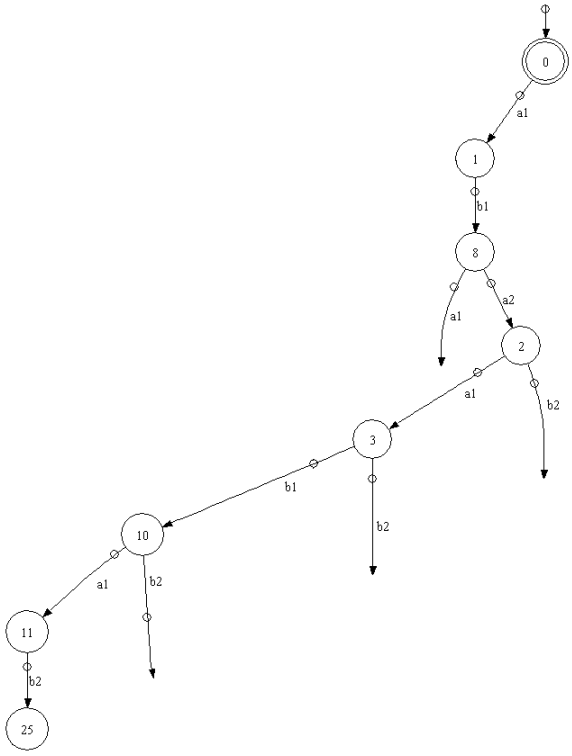


Figura 3.9 – Parte do Autômato *R* para mostrar a Cadeia de Eventos Considerada

Após esse processo iterativo para eliminação de todos os maus estados e cálculo da componente *Trim* de *R*, encontra-se o supervisor *S*. Esses procedimentos para obtenção do supervisor podem ser realizados de forma automática, com o uso de ferramentas computacionais, tais como o GRAIL (REISER *et al.*, 2006) ou o TCT (FENG e WONHAM, 2006).

O autômato que representa o supervisor obtido para o sistema de manufatura possui 18 estados e 32 transições e é mostrado na Figura 3.10. Essa metodologia consiste na abordagem monolítica para controle supervisório. O mapa de desabilitações (*MD*) dos eventos controláveis para cada estado também é ilustrado, onde as desabilitações presentes em cada estado são representadas por linhas tracejadas.

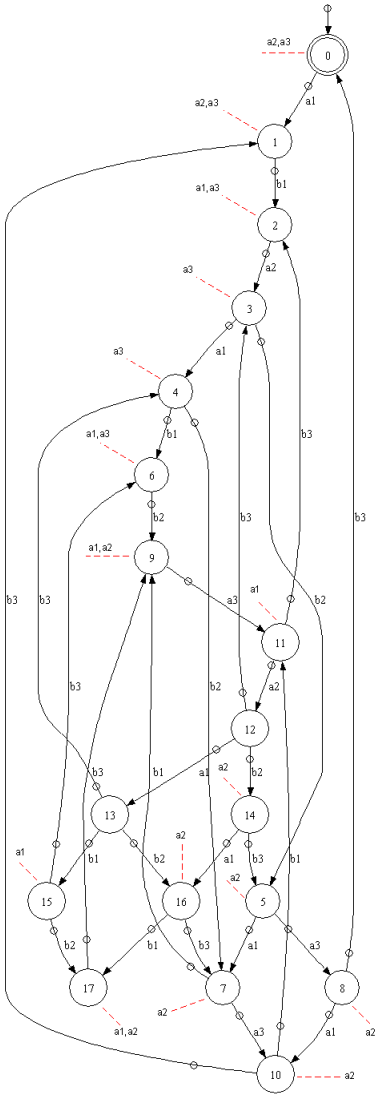


Figura 3.10 – Autômato que representa o Supervisor *S* para o Sistema de Manufatura

3.4 Redução de Supervisores

Quando um supervisor minimamente restritivo é obtido, algumas informações redundantes são incorporadas ao modelo, pois existem restrições que a planta *G* já realiza e não precisariam estar presentes nesse modelo. Isso significa que esse supervisor pode ser reduzido, ou seja, o número de estados é reduzido, preservando a mesma ação de controle em malha fechada em relação à planta *G*. Espera-se que o tamanho de código de uma determinada implementação também seja reduzido ao se obter um supervisor reduzido a partir do supervisor original.

Diz-se que dois supervisores S_1 e S_2 são equivalentes se suas ações de controle sobre a planta *G* produzem o mesmo comportamento, ou seja, se S_1/G for equivalente a S_2/G. Assim, S_2 é um supervisor reduzido para S_1 se (REISER, 2005):

- O número de estados de S_2, denotado por $|S_2|$, for menor que $|S_1|$;

- $L(G) \cap L(S_2) = L(S_1)$

- $L_m(G) \cap L_m(S_2) = L_m(S_1)$

A redução é feita por intermédio da agregação de estados compatíveis entre si em blocos, não necessariamente disjuntos entre si. Durante o processo de agregação, deve-se garantir que as desabilitações dos estados de um bloco não entrem em conflito com os eventos habilitados em outros estados do mesmo bloco e que o supervisor reduzido seja determinístico. De forma a preservar a igualdade das linguagens marcadas, deve-se também garantir que os estados pertencentes ao mesmo bloco estejam consistentemente marcados (REISER, 2005). Para um dado supervisor existe um tamanho mínimo de supervisor reduzido, mas é possível que existam múltiplos supervisores com o número mínimo de estados. Dependendo da aplicação, a busca por um supervisor reduzido com um tamanho mínimo de estados pode ser desnecessária, sendo possível obter supervisores suficientemente reduzidos por meio de algoritmos de menor complexidade (DE QUEIROZ, 2004).

Diversos algoritmos são propostos na literatura a fim de reduzir o tamanho do autômato que representa o supervisor visando encontrar um autômato reduzido que mantenha uma equivalência com o autômato original. Vale destacar os trabalhos de (VAZ e WONHAM, 1986), (SU e WONHAM, 2004) e (SIVOLELLA, 2005). Embora proporcionem uma economia de memória, os supervisores reduzidos geralmente possuem uma lógica de controle de mais difícil compreensão do que os supervisores tradicionais.

A redução de supervisores é utilizada neste trabalho a fim de fazer uma comparação posterior com os modelos obtidos para as máquinas de estados com saídas. Dois algoritmos para redução de supervisores são avaliados: o utilizado no programa TCT (FENG e WONHAM, 2006), desenvolvido por Su e Wonham (2004) e o utilizado no programa GRAIL (REISER *et al.*, 2006), desenvolvido por Sivolella (2005). O supervisor reduzido encontrado para o sistema de manufatura com o uso do algoritmo proposto por Sivolella (2005) possui 12 estados e 26 transições. O supervisor reduzido encontrado com o uso do algoritmo proposto por Su e Wonham (2004) possui 4 estados e 13 transições, sendo mostrado na Figura 3.11.

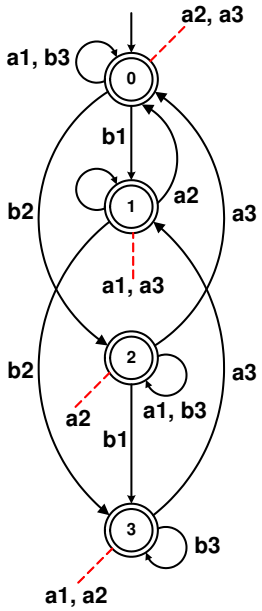


Figura 3.11 – Autômato que representa o Supervisor Reduzido *SR* para o Sistema de Manufatura

Os procedimentos para redução de supervisores podem ser aplicados da mesma forma nos supervisores obtidos a partir da abordagem modular local.

A seguir, é apresentada a metodologia proposta nesta dissertação para obtenção das máquinas de estados com saídas por meio de uma abordagem monolítica. Esse procedimento utiliza o autômato do supervisor monolítico obtido a partir da TCS como informação de entrada para obter as máquinas de estados com saídas.

3.5 Máquinas de Estados com Saídas: Abordagem Monolítica

Este trabalho propõe algoritmos para obtenção de máquinas de estados de *Mealy* e *Moore* a partir do autômato que modela o supervisor monolítico obtido com o uso da TCS. Essa metodologia difere da proposta em (BARETTA e TORRICO, 2008) pois não visa simplesmente transformar os autômatos que representam os supervisores em máquinas de estados com saídas. Neste trabalho, são formalizados algoritmos para transformar autômatos em máquinas de estados com saídas que eliminam as redundâncias presentes nos modelos dos autômatos. Além disso, são propostos também algoritmos para redução dessas máquinas de estados com saídas, visando obter modelos mais simples de supervisores, preservando a lógica de controle original.

3.5.1 Algoritmo para Obtenção da Máquina de *Mealy* na Abordagem Monolítica

O algoritmo proposto para obter a máquina de estados finita de *Mealy* funciona de forma iterativa, varrendo os dados de entrada para obter os estados, transições e ações que compõem esta máquina. A máquina estará completa quando todos os dados de entrada forem processados. Esse algoritmo, assim como os demais algoritmos propostos neste trabalho, poderia ser facilmente implementado em alguma ferramenta computacional para facilitar a síntese de máquinas de estados e evitar assim que seja realizado um procedimento manual, visando maior praticidade, ganho de tempo e menor susceptibilidade a erros para sistemas de maior porte.

O Algoritmo 3.1 é apresentado a seguir.

Algoritmo 3.1 Obtenção da Máquina de *Mealy* na Abordagem Monolítica

```
1: Lê o autômato do supervisor (S) e o mapa de desabilitações (MD)
2: Divide o autômato do supervisor de acordo com a controlabilidade de seus eventos em Su (não controlável) e
   Sc (controlável)
3: Cria Estado Inicial da Máquina = Estado Inicial do Autômato que representa o Supervisor
4: Cria Lista com Estados de Chegada que são obtidos a partir dos Estados de Partida e precisam ser tratados
5: enquanto (Lista de Estados > 0) faça
6:     Lê Estado da Lista e considera agora como um Estado de Partida
7:     Cria Lista de Transições para esse Estado
8:     para (Lista de Transições) faça
9:         Lê Su, MD, Sc
10:        se (Sc evoluiu) então
11:            Atualiza MD e Sc
12:        senão
13:            Cria um Novo Estado de Chegada
14:        fim se
15:        se (Estado já foi criado) então
16:            não faz nada
17:        senão
18:            Adiciona Estado de Chegada na Lista de Estados de acordo com o estado de chegada
               do autômato que representa o supervisor
19:        fim se
20:    fim para
21:    Decrementa Lista de Estados
22: fim enquanto
23: Elimina estados não alcançáveis
24: Salva a Máquina de Estados de Mealy na Abordagem Monolítica MM
```

Os dados de entrada para o algoritmo são o autômato que representa o supervisor e o mapa de desabilitações, enquanto os dados de saída são os estados, as transições e as ações que compõem a máquina de estados de *Mealy*. O autômato que representa a planta não precisa ser utilizado pois a informação do comportamento da planta real já está intrínseca no autômato do supervisor e nas desabilitações.

No processo de transformação do autômato do supervisor para a máquina de *Mealy* os eventos não-controláveis na TCS equivalem às transições entre os estados, enquanto os eventos controláveis equivalem às ações de saída na máquina de *Mealy*.

A fim de criar uma distinção entre as transições e as ações, é necessário primeiro dividir o supervisor em duas partes, de acordo com a controlabilidade dos eventos referentes às suas transições de estado. Define-se por *Su* a parte do supervisor cujas transições se dão por eventos não controláveis e *Sc* a parte cujas transições são devido a eventos controláveis.

Nesse algoritmo, o estado inicial da máquina correspondente ao estado inicial do autômato que representa o supervisor. Essa informação compõe o ponto de partida da máquina, representando a condição onde a operação do processo físico ainda não começou.

O próximo passo é criar uma lista para armazenar os estados que serão obtidos durante a dinâmica do algoritmo. A lista é necessária para que, após o tratamento do estado atual seja finalizado, esses estados sejam tratados. A lista consiste em uma estrutura *FIFO* (*First In, First Out*). Enquanto ainda existirem estados na lista, haverá um processo iterativo para o tratamento de cada um deles. Um *loop* **enquanto** é utilizado para esta finalidade.

Para cada estado, uma lista de transições é criada com o intuito de tratar todas as transições válidas para aquele estado específico. A quantidade de transições válidas para cada estado equivale aos eventos não-controláveis que criam evoluções de estado em *Su* e todas as combinações entre esses eventos. Assim, uma transição entre dois estados na máquina de *Mealy* pode ocorrer por meio de um ou mais eventos não-controláveis no autômato. Um processamento iterativo é realizado até que todas as transições sejam tratadas para todos os estados. Um *loop* **para** é utilizado para cada conjunto de transições a serem tratadas para cada estado.

Após a parte devido a eventos não-controláveis ser tratada, a parte devido a eventos controláveis é processada. Após a ocorrência de uma transição, o estado resultante em *Su* é analisado para verificar quais eventos controláveis são desabilitados pela ação de controle. Enquanto os eventos desabilitados estão proibidos de ocorrer, os remanescentes podem dar origem às ações. As ações válidas para um dado estado são os eventos controláveis que não estão desabilitados no supervisor e que causam evolução de estados em *Sc*. Para cada transição, uma ação de saída pode ser gerada ou não.

Quando uma ação ocorre, o algoritmo verifica se *Sc* evoluiu. Em caso afirmativo, então os eventos desabilitados para o estado de chegada são analisados a fim de verificar se alguma outra ação pode ocorrer. Este passo assegura que todas as ações possíveis de ocorrer

para uma mesma transição são processadas. Isso significa que mais do que uma ação pode ocorrer para a mesma transição.

Caso *Sc* não tenha evoluído então um novo estado é criado. Este estado é comparado com os outros estados e, caso já exista, é descartado. Caso contrário é adicionado à fila de estados para ser tratado posteriormente.

A parte *Su* evolui para cada transição devido a eventos não-controláveis. O mesmo ocorre para a parte controlável na ocorrência de uma ação. Esta metodologia é consistente com a definição de máquina de *Mealy*, onde as saídas (ações) dependem do estado atual e das entradas válidas (transições).

Após o término do procedimento de criar uma transição e as ações correspondentes o algoritmo trata as demais transições presentes na fila de transições. Quando todas as transições nessa fila são tratadas, o algoritmo analisa o próximo estado disponível na fila de estados. Estes processos iterativos são efetuados até que a fila de estados se torne vazia. Isto significa que a máquina de estados finita está completa.

A Figura 3.12 ilustra o modelo de supervisor que seria correspondente à máquina de estados com saídas obtida. Ao comparar com a Figura 3.10 observa-se que alguns estados e transições presentes no modelo do supervisor original deixam de existir nesse modelo. Essa etapa já leva a percepção de uma redução do modelo, onde a ação de controle resultante permanece a mesma.

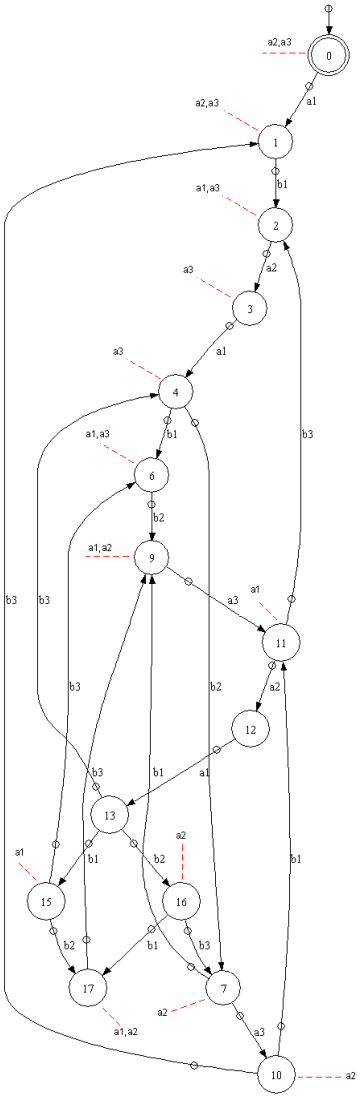


Figura 3.12 – Autômato do Supervisor Modificado

Para compreender melhor como esses estados e transições são eliminados, observe o estado *3* desse modelo. A única transição que acontece a partir dele é o evento controlável a_1, diferente do modelo original onde ocorre também a transição devido ao evento não controlável b_2. A eliminação dessa transição consiste na etapa do Algoritmo 3.1 em que os eventos controláveis (ou ações) são tratados, pois de acordo com o processo iterativo proposto, a transição que chega ao estado *3* é a ação a_2. Assim, apenas a cadeia a_2-a_1 permanece nesse modelo e a transição b_2 pode ser eliminada. Situação similar ocorre para o estado *12*, onde a transição b_2 é eliminada. Além disso, como essa transição que levaria o autômato ao estado *14* é eliminada, não é mais possível alcançá-lo. Esse estado se torna então um estado não-acessível (CASSANDRAS e LAFORTUNE, 2008), podendo ser eliminado do modelo também.

De forma geral, o Algoritmo 3.1 remove os estados que se tornam não-acessíveis no autômato do supervisor original. Para o sistema de manufatura avaliado, os estados *5*, *8* e *14* tornam-se não acessíveis e são, portanto, eliminados.

Além disso, a ocorrência de dois ou mais eventos também é possível desde que isso não afete a ação de controle resultante. Considere que o autômato do supervisor modificado esteja no estado *4*. Independente se ocorrer a cadeia b_1-b_2 ou a cadeia b_2-b_1, o estado resultante e a ação de controle serão os mesmos, isto é, o supervisor atinge o estado *9* e, com a ação de controle a_3, ele chega ao estado *11*. Essa situação é chamada de *insensibilidade ao entrelaçamento* e será discutida em detalhes no próximo capítulo. A ocorrência somente do evento b_1 ou somente do evento b_2 também são levadas em conta na obtenção da máquina de estados.

3.5.2 Representação da Máquina de *Mealy* na Abordagem Monolítica

Utilizando o Algoritmo 3.1 para criação de uma máquina de *Mealy* na abordagem monolítica a partir do autômato do supervisor para o sistema de manufatura, obtém-se a máquina de estados ilustrada na Figura 3.13, composta por 8 estados e 22 transições.

Os estados são nomeados de acordo com os aparelhos que estão ligados em um determinado momento e os *buffers* que estão cheios. As transições são devido aos eventos não-controláveis e as ações, se houver algumas, são separadas das transições por uma barra (/). As ações desabilitadas são representadas por linhas tracejadas. No estado *1* da máquina, por exemplo, as ações a_2 e a_3 estão desabilitadas, isto é, o início de operação dos aparelhos *A2* e *A3* está proibido de ocorrer.

Observa-se também que neste modelo as transições podem ocorrer devido a mais do que um evento não controlável e as ações podem ser devido a mais do que um evento controlável, sendo as condições *AND* representadas pelo operador lógico "&". No estado *2* da máquina, por exemplo, a ocorrência das transições b_1 e b_2 (b_1 & b_2) resulta na execução das ações a_1, a_2 e a_3 (a_1 & a_2 & a_3) e na evolução da máquina para o estado de chegada *5*.

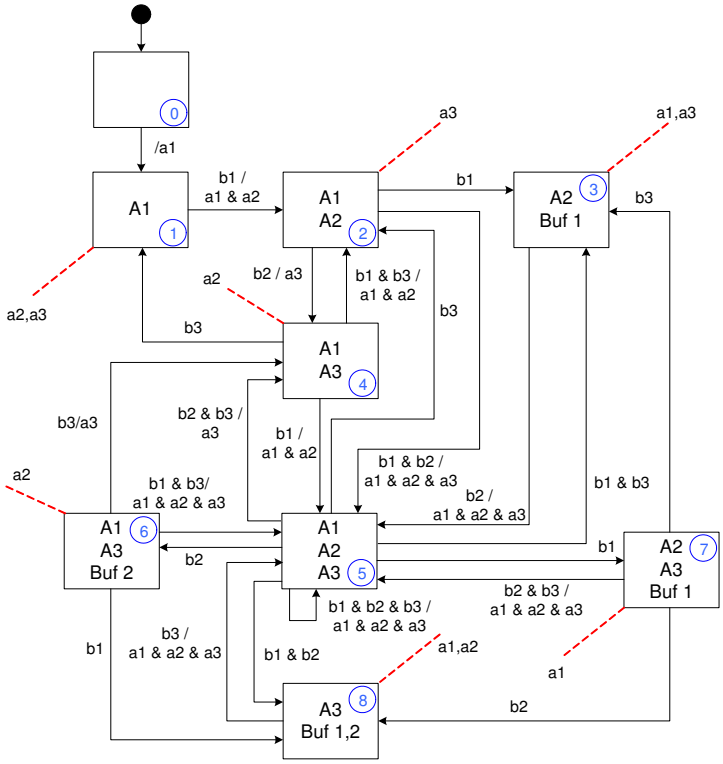


Figura 3.13 – Máquina de Estados de *Mealy MM* na Abordagem Monolítica

3.5.3 Redução da Máquina de *Mealy* na Abordagem Monolítica

A modelagem de máquina de estados com saídas possibilita que ocorra o tratamento de mais de um sinal na entrada física de um controlador síncrono quando estes mudam de valor. Para cada um desses sinais, deve ser relacionada uma variável que corresponde a um evento não controlável. A sinalização da mudança de valor desses sinais ocorre quando eles mudam de valor ao mesmo tempo ou quando eles mudam de valor em instantes de tempo diferentes, porém a leitura deles e conseqüente sinalização de sua ocorrência é efetuada no mesmo instante devido ao caráter síncrono do controlador utilizado[5].

Ao considerar o tratamento de mais do que um evento não controlável no modelo de máquina de estados existe a desvantagem de se resultar em um crescimento exponencial das transições, de acordo com o número de plantas presentes no modelo e quantas dessas podem ser habilitadas ao mesmo tempo. O número de transições criadas para um dado estado é da ordem de 2^n-1, onde n é o número de eventos não-controláveis presentes no modelo e possíveis de ocorrer para aquele estado. Desta forma, para sistemas de grande porte, o tamanho de código para uma determinada implementação seria afetado significativamente

[5] Esse problema será abordado em detalhes na Seção 4.1.2 do Capítulo 4, que trata da detecção de eventos.

para satisfazer essa condição, sendo uma justificativa convincente para não utilizar esse tipo de metodologia.

Uma solução alternativa para isso é considerar uma máquina de estados reduzida onde as transições de estado são representadas somente por transições onde ocorre execução de ações. Transições que não resultem em execução de alguma ação são representadas como *self-loops* dentro de seu estado de partida. Para o estado *2* da máquina de estados ilustrada na Figura 3.13, por exemplo, a transição b_1 poderia ser representada como um *self-loop*, tendo em vista que embora a máquina evolua para o estado *3*, nenhuma ação é executada durante essa transição.

Embora não resultem em ações, essas transições são importantes na representação da dinâmica da planta e não devem ser descartadas durante o processo de implementação. O programa do controlador deve capturar a ocorrência delas e armazenar em alguma variável interna para ser usada na tomada de decisões quando ele detectar a ocorrência de outras transições que vierem a surgir. O Algoritmo 3.2 descreve o processo de redução da máquina monolítica de *Mealy*.

Algoritmo 3.2 Redução da Máquina de *Mealy* na Abordagem Monolítica

```
1: Lê a máquina de Mealy na abordagem monolítica
2: para Estado da Máquina de Mealy = i até n faça   % onde n é o número de estados da máquina
3:     para Transição = i até t faça                % onde t é o número de transições saindo do estado
4:         se (Transição resulta em execução de ação) então
5:         Mantém a transição na máquina reduzida
6:         senão
7:         Cria self-loop representado por linhas tracejadas no estado de partida
8:         fim se
9:     fim para
10: fim para
11: Elimina estados não alcançáveis da máquina resultante
12: Salva a Máquina de Estados de Mealy na Abordagem Monolítica Reduzida MMS
```

É importante salientar que esse procedimento de redução da máquina de *Mealy*, assim como os procedimentos de redução para os demais modelos de máquinas de estados com saídas a serem apresentados posteriormente, não resulta necessariamente em uma máquina de estados mínima.

A Figura 3.14 mostra a máquina reduzida para o sistema de manufatura. Esta máquina possui apenas 4 estados e 7 transições. As transições ilustradas por linhas cheias representam que ocorre execução de ação, independente se houver transição de um estado para outro ou não. As transições ilustradas por linhas tracejadas em formato de *self-loop* dentro de um mesmo estado representam que, embora tenha ocorrido uma transição, nenhuma ação é executada. Considere por exemplo o estado *2* da máquina reduzida. Caso a transição b_2

ocorrer, a máquina evolui para o estado *3*. Caso a transição b_1 ocorrer, o modelo ilustra essa situação como um *self-loop* representado por linhas tracejadas, o que significa que a implementação prática no controlador deve garantir o armazenamento dessa informação em alguma variável interna do código. Quando o evento b_2 ocorrer com a transição b_1 já habilitada, a transição b_1 & b_2 será ativada, tal que a máquina evolui para o estado *4* com a execução das ações a_1 & a_2 & a_3.

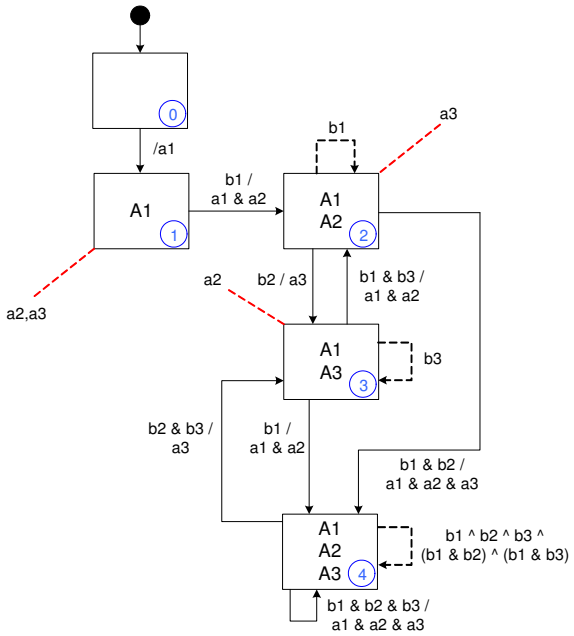


Figura 3.14 – Máquina de Estados de *Mealy* Reduzida *MMS* na Abordagem Monolítica

As desabilitações devem ser representadas nos estados correspondentes para ilustrar quais ações de controle são proibidas de ocorrer. Os estados que se tornaram não alcançáveis se comparados com a máquina de estados original, são eliminados nesse modelo reduzido.

Transições devido a mais do que um evento não controlável e ações devido a mais do que um evento controlável continuam sendo representadas nesse modelo. Essa é uma informação relevante que auxilia o projetista no momento em que for implementar o sistema de controle, a fim de que o programa permita que vários eventos sejam executados dentro de um mesmo ciclo de execução do controlador síncrono.

Além disso, um novo operador lógico aparece neste modelo. O operador "^" representa um *OR-Exclusive*. Para compreender a função desse operador, considere o estado *4* da máquina reduzida. Se qualquer uma das transições listadas no *self-loop* com linhas tracejadas ocorrer, isto é, b_1 ^ b_2 ^ b_3 ^ $(b_1$ & $b_2)$ ^ $(b_1$ & $b_3)$, nenhuma ação será executada e essas transições permanecerão habilitadas. Somente quando as transições b_2 & b_3 ou b_1 & b_2

& b_3 se tornarem válidas é que haverá execução de ações. O operador "^" aparece somente nos casos em que existem *self-loops* representados por linhas tracejadas.

3.5.4 Representação da Máquina de *Moore* na Abordagem Monolítica

Conforme mostrado no exemplo da Seção 2.7, qualquer máquina de *Mealy* pode ser transformada em uma máquina de *Moore* e vice-versa (CASSANDRAS e LAFORTUNE, 2008). Essa premissa continua sendo válida para as máquinas de estados obtidas nesta metodologia. A diferença de uma máquina de *Moore* em relação à uma máquina de *Mealy* consiste em executar as ações quando a máquina entra em um estado (e*ntry action*) ao invés de executar as ações durante as transições, como é realizada pela máquina de *Mealy*. Em outras palavras, independente de qual for a transição que chegar em um determinado estado, a ação será sempre a mesma para a máquina de *Moore*.

A máquina de *Moore* para o sistema de manufatura é mostrada na Figura 3.15 e foi obtida a partir da máquina de *Mealy* apresentada na Figura 3.13. As desabilitações continuam sendo representadas por linhas tracejadas. Embora essa máquina seja bastante similar à máquina de *Mealy* para o sistema de manufatura, ela contém mais estados e transições. Essa característica é, em geral, comum para as máquinas de *Moore* se comparadas com as máquinas de *Mealy*.

Para melhor compreensão, considere, por exemplo, o estado *4* da máquina de *Mealy* original. A ocorrência do evento b_1 leva ao estado *5*, onde todos os aparelhos serão ligados durante a transição. Olhando agora para a representação de *Moore*, o estado equivalente que representa a mesma dinâmica da máquina de *Mealy* também corresponde ao estado *4*. Entretanto, a ocorrência da transição b_1 resulta no estado de chegada *10*, que difere do estado *5* dessa máquina de *Moore*. Embora todos os aparelhos também estejam ligados no estado *10*, pois a ação a_3 já havia sido executada previamente, somente as ações a_1 & a_2 são executadas quando a máquina entra nesse estado. A execução de todas as ações ocorre no estado *5* da máquina de *Moore*, conforme mostrado na Figura 3.15.

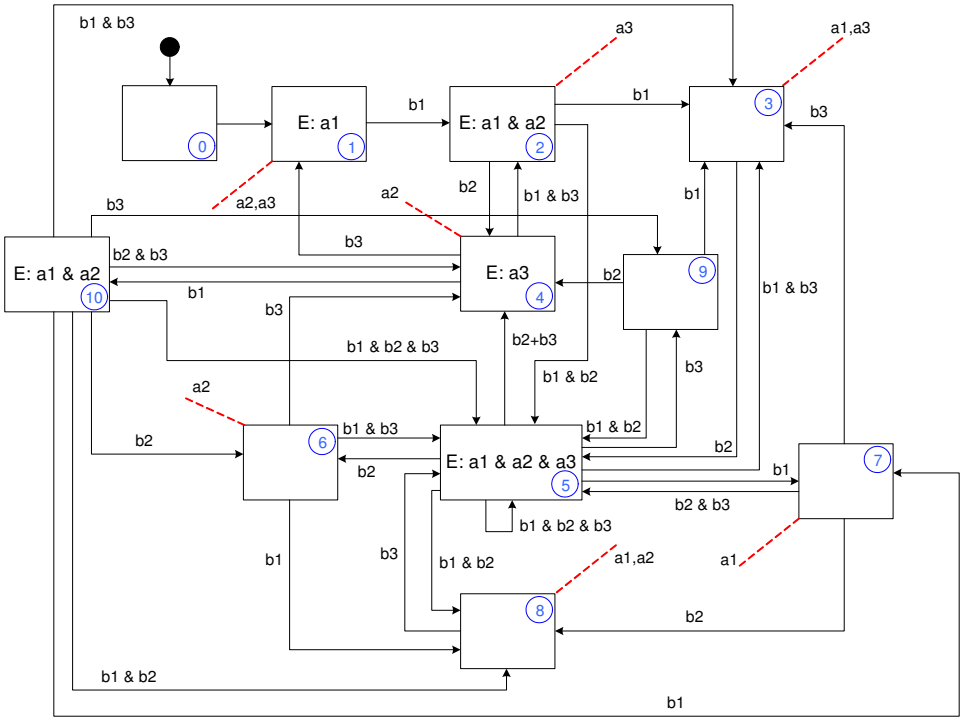


Figura 3.15 – Máquina de Estados de *Moore MO* na Abordagem Monolítica

3.5.5 Redução da Máquina de *Moore* na Abordagem Monolítica

Com o intuito de reduzir a máquina de *Moore* obtida, assim como foi feito para a abordagem de *Mealy*, apresenta-se o Algoritmo 3.3. O procedimento de redução é similar ao utilizado para reduzir a máquina de *Mealy*. Agora, porém, as transições que são representadas por *self-loops* são aquelas que resultam em estados de chegada sem execução de ação.

Algoritmo 3.3 Redução da Máquina de *Moore* na Abordagem Monolítica

1: Lê a máquina de *Moore* na abordagem monolítica
2: **para** Estado da Máquina de *Moore* = *i* até *n* **faça** % onde *n* é o número de estados da máquina
3: **para** Transição = *i* até *t* **faça** % onde *t* é o número de transições saindo do estado
4: **se** (Transição resulta em estado de chegada com execução de ação) **então**
5: Mantém a transição na máquina reduzida
6: **senão**
7: Cria *self-loop* representado por linhas tracejadas no estado de partida
8: **fim se**
9: **fim para**
10: **fim para**
11: Elimina estados não alcançáveis
12: Salva a **Máquina de Estados de *Moore* na Abordagem Monolítica Reduzida *MOS***

A Figura 3.16 ilustra a máquina de *Moore* reduzida para o exemplo proposto. Esta máquina possui 5 estados e 9 transições. Assim como na máquina reduzida de *Mealy*, as transições representadas por linhas cheias são responsáveis por execução de ação enquanto que os *self-loops* representados por linhas tracejadas dentro do seu estado de partida não resultam em execução de ações. Quando mais do uma transição resulta em *self-loop* sem

tomada de ação, elas são representadas pela condição *OR-Exclusive*, pois são independentes umas das outras.

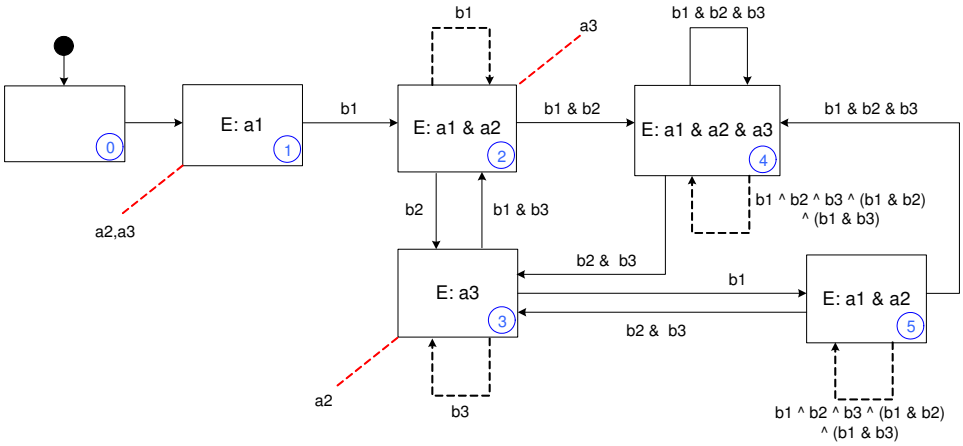


Figura 3.16 – Máquina de Estados de *Moore* Reduzida *MOS* na Abordagem Monolítica

Observe também que as ações a_1 & a_2 são executadas tanto no estado *2* quanto no estado *5*. Entretanto, no estado *2* o aparelho *A3* está desligado, enquanto no estado *5* o aparelho *A3* está ligado.

Ao se comparar com a máquina reduzida de *Mealy* ilustrada na Figura 3.14, observa-se que, para este exemplo, a máquina de *Moore* possui um estado e duas transições a mais.

3.6 Teoria de Controle Supervisório: Abordagem Modular Local

Embora útil para a obtenção de um supervisor para controlar uma planta, a abordagem monolítica (RAMADGE e WONHAM, 1989) da Teoria de Controle Supervisório (TCS) é pouco aceita no ambiente industrial. Segundo (SKOLDSTAM *et al*., 2008) as duas razões principais para isso são a discrepância entre a realidade baseada em sinais e os autômatos que são baseados em eventos[6] e a falta de uma representação compacta para grandes modelos.

A fim de solucionar este segundo problema, a abordagem modular local (DE QUEIROZ e CURY, 2002) surgiu como uma extensão da abordagem de controle modular (WONHAM e RAMADGE, 1988) com o intuito de implementar supervisores a partir de modelos de menor tamanho para sistemas de grande porte. A abordagem modular local visa tirar proveito da característica descentralizada do modelo em malha aberta de sistemas compostos. Dessa forma, essa abordagem permite explorar, além da modularidade das especificações, a modularidade da planta, de forma a diminuir a complexidade computacional da síntese de supervisores e o tamanho das soluções (DE QUEIROZ, 2004).

Diferentemente da abordagem monolítica que usa a linguagem da planta completa para gerar um supervisor, a abordagem modular local usa apenas os subsistemas que são afetados pelas especificações, efetuando um controle local da planta. Assim, o número de supervisores será o mesmo que o número de especificações de controle. Menos autômatos são compostos e os autômatos que representam as linguagens alvo são menores do que o autômato obtido na abordagem monolítica. Sendo assim, a complexidade computacional para síntese de supervisores é reduzida e os supervisores obtidos são menores do que na abordagem monolítica.

3.6.1 Modelagem da Planta

Ao contrário da abordagem monolítica, que utiliza um modelo da planta global por meio de um autômato G obtido por intermédio da composição síncrona de seus n subsistemas, a representação por sistema produto (RSP) (RAMADGE e WONHAM, 1989) consiste na modelagem da planta através de um conjunto de subsistemas completamente assíncronos entre si. Este modelo representa a estrutura descentralizada natural de operações concorrentes

[6] Esse problema será abordado em detalhes na Seção 4.1.2 do Capítulo 4, que trata da detecção de eventos.

em malha aberta para um sistema composto e serve de plataforma para a abordagem modular local (DE QUEIROZ, 2000).

Qualquer planta pode ser modelada por uma RSP. Quando a modelagem inicial da planta possui subsistemas com eventos em comum, pode-se obter uma RSP mais refinada, fazendo-se a composição síncrona desses subsistemas que compartilham eventos.

A Figura 3.17(a) ilustra a planta original e a Figura 3.17(b) ilustra a obtenção de uma RSP mais refinada, cujos subsistemas são agora completamente assíncronos entre si.

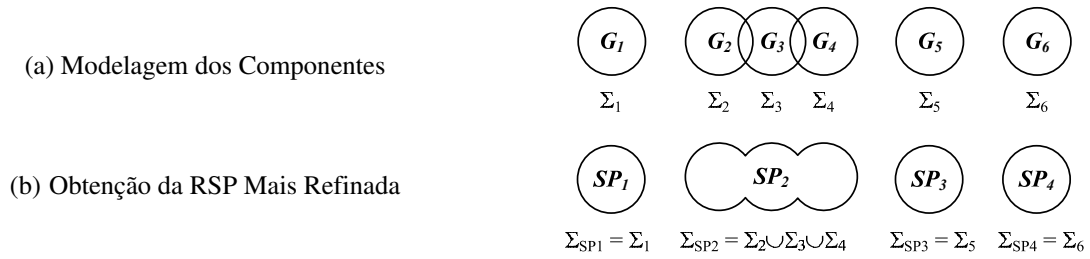


Figura 3.17 – Representação por Sistema Produto (PENA, 2007)

Por outro lado, se todos os subsistemas compartilham eventos entre si, então a RSP mais refinada será a própria planta global. Os resultados obtidos com o uso da abordagem modular local serão melhores à medida que a RSP for mais refinada possível (PENA, 2007).

3.6.2 Síntese dos Supervisores

Supervisores modulares locais podem ser sintetizados a partir da abordagem de controle modular local ao descrever a planta global como uma RSP $\{G_i \mid i \in I = \{1, 2, ..., n\}\}$, onde $G_i = (Q^{Gi}, \Sigma^{Gi}, \delta^{Gi}, q0^{Gi}, Qm^{Gi})$. Os conjuntos de eventos controláveis e de eventos não-controláveis para um subsistema G_i são representados, respectivamente, por Σ_c^{Gi} e Σ_{uc}^{Gi} (VIEIRA, 2007).

Cada uma das especificações de controle $\{E_j \mid j \in J = \{1, 2, ..., m\}\}$ é representada por um autômato correspondente, onde $E_j = (Q^{Ej}, \Sigma^{Ej}, \delta^{Ej}, q0^{Ej}, Qm^{Ej})$. Uma planta local correspondente a cada especificação de controle deve ser obtida, sendo representada por G_{lj}. O procedimento para obter as plantas locais consiste em obter uma RSP por meio do produto síncrono de todos os modelos de autômatos em $\{ G_i \mid i \in I \}$ que possuem algum evento em comum com a especificação sob análise, isto é, os autômatos em que $\Sigma^{Ej} \cap \Sigma^{Gi} \neq \varnothing$ (VIEIRA, 2007).

Para cada uma das especificações de controle, sintetiza-se um supervisor modular local correspondente. Apenas a RSP da planta local G_{lj} é utilizada para a síntese do supervisor local correspondente. Quando pelo menos um desses supervisores desabilitarem um

determinado evento controlável durante a dinâmica de controle, então este evento será proibido de ocorrer nos demais supervisores locais também, resultando em uma ação conjunta que visa manter o sincronismo entre os supervisores.

O procedimento para o cálculo dos supervisores é similar ao adotado na abordagem monolítica. Uma linguagem alvo K_j é obtida para refletir o comportamento da planta local G_{lj} de acordo com a especificação correspondente E_j, tal que $K_j = E_j \parallel L_m(G_{lj})$. A partir de K_j e da planta local G_{lj}, sintetiza-se um supervisor modular local SL_j que implementa a máxima linguagem controlável denotada por $SupC(K_j, L(G_{lj}))$.

Exemplo 3.1 Considere o sistema ilustrado na Figura 3.18. Esse sistema é composto por duas especificações de controle E_a e E_b e três subsistemas assíncronos são descritos por representações em sistema produto. A especificação E_a possui eventos em comum com os subsistemas SP_1 e SP_2, enquanto a especificação E_b possui eventos em comum com os subsistemas SP_2 e SP_3. Como existem duas especificações, devem existir duas plantas locais. A planta local relativa à E_a é $G_a = SP_1 \parallel SP_2$, enquanto a planta local relativa à E_b é $G_b = SP_2 \parallel SP_3$. As linguagens alvo locais K_a e K_b são obtidas fazendo $K_a = E_a \parallel L_m(G_a)$ e $K_b = E_b \parallel L_m(G_b)$. A partir das linguagens alvo locais e das plantas locais, obtém-se os supervisores modulares locais SL_a e SL_b através do cálculo das máximas linguagens controláveis $SupC(K_a, L(G_a))$ e $SupC(K_b, L(G_b))$.

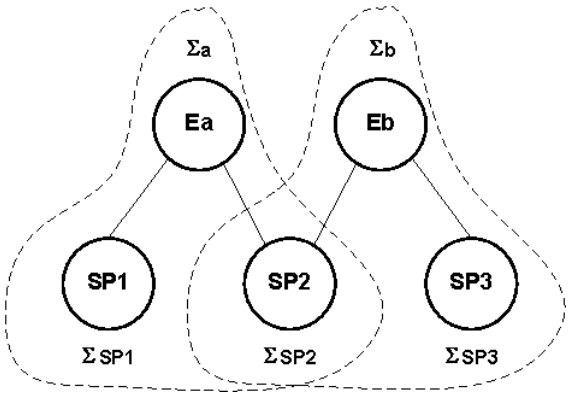


Figura 3.18 – Exemplo para representar a Abordagem Modular Local

A composição síncrona dos modelos das plantas é necessária somente quando duas ou mais plantas possuem eventos em comum. Essa composição não é necessária quando duas ou mais especificações de controle possuem eventos em comum. A abordagem modular local explora a característica modular da planta considerando a existência de diversos subsistemas que são executados de forma concorrente. Cada supervisor modular local resultante de uma determinada especificação vai atuar somente sobre os subsistemas atingidos, como ilustrado na Figura 3.19 (DE QUEIROZ, 2004).

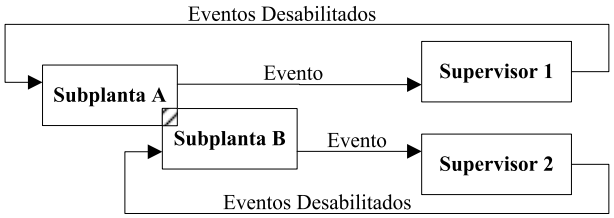


Figura 3.19 – Esquema de Controle da Abordagem Modular Local (DE QUEIROZ, 2004)

Devido ao fato de que cada supervisor utiliza um modelo parcial da planta global sem ter informação das outras especificações a serem implementadas pelos outros supervisores, existe a possibilidade de que essas especificações implementem ações conflitantes. Dessa forma, torna-se necessário realizar um teste para detecção de conflito após o projeto dos supervisores. Considerando a existência de *m* supervisores, esse teste é dado por (DE QUEIROZ, 2004):

$$\|_{j=1}^{m} \overline{L_m(SL_j / G_{lj})} = \overline{\|_{j=1}^{m} L_m(SL_j / G_{lj})} \tag{3.4}$$

Esta propriedade, também chamada de modularidade local, garante que o sistema em malha fechada sob ação dos supervisores modulares seja não-bloqueante, isto é, esteja livre da ocorrência de conflitos.

A principal limitação dessa abordagem consiste na verificação da modularidade local dos supervisores. Para que isso seja feito, a complexidade computacional do teste de modularidade é grande, pois é necessário efetuar a composição síncrona de todos os supervisores locais obtidos, resultando na mesma complexidade computacional necessária para obter um supervisor de acordo com a abordagem monolítica. Assim, se o teste for realizado com sucesso tal que as linguagens não forem conflitantes, então o desempenho da ação conjunta dos supervisores locais será o mesmo de um supervisor obtido através da abordagem monolítica (PENA, 2007).

As principais vantagens da abordagem modular local em relação à abordagem monolítica são apresentadas em (VIEIRA, 2007), sendo descritas a seguir:

- os autômatos que representam os supervisores modulares locais apresentam menor número de estados do que o autômato que representa o supervisor monolítico viabilizando assim uma melhor interpretação desses comportamentos, assim como uma representação reduzida dos supervisores locais e a implementação dos mesmos em algum dispositivo de controle;
- a inclusão ou exclusão de um subsistema afeta somente os supervisores locais que compartilham algum evento com o subsistema em questão, tais que esses supervisores devem ser sintetizados novamente, enquanto os demais permanecem inalterados;

- a inclusão de uma nova especificação não vai afetar os supervisores locais já sintetizados enquanto a remoção de uma especificação resulta na eliminação do supervisor local correspondente, mas também não vai afetar os demais supervisores;
- preserva a modularidade de acordo com a natureza da planta e das especificações de controle.

Considere o sistema de manufatura apresentado na seção 3.3. Duas especificações de controle foram criadas para essa planta. Os autômatos G_1, G_2 e G_3 que representam os modelos dos aparelhos A_1, A_2 e A_3, respectivamente, são assíncronos entre si, isto é, não compartilham eventos em comum. Dessa forma, a RSP é formada pelas próprias subplantas. Ao se utilizar a abordagem modular local, obtêm-se dois supervisores, de acordo com o número de especificações existentes. A Figura 3.20 apresenta os supervisores SL_1 e SL_2 e suas desabilitações, obtidos por intermédio da abordagem modular local.

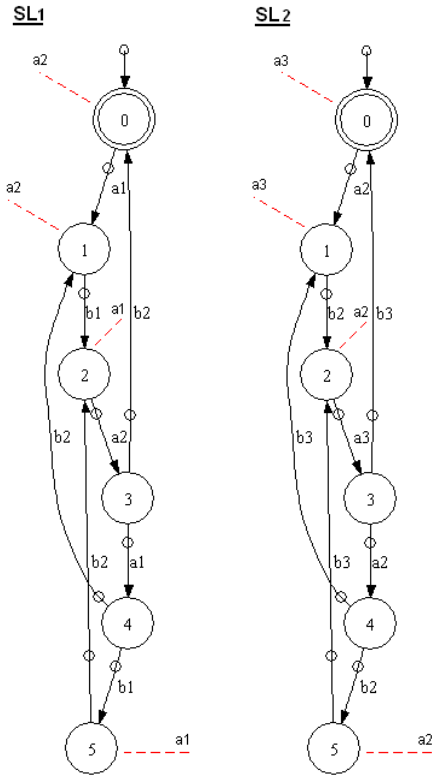


Figura 3.20 – Supervisores Modulares Locais SL_1 e SL_2 para o Sistema de Manufatura

Os supervisores modulares locais reduzidos SLR_1 e SLR_2 obtidos a partir dos supervisores modulares locais SL_1 e SL_2 com o auxílio da ferramenta TCT (FENG e WONHAM, 2006) são ilustrados na Figura 3.21.

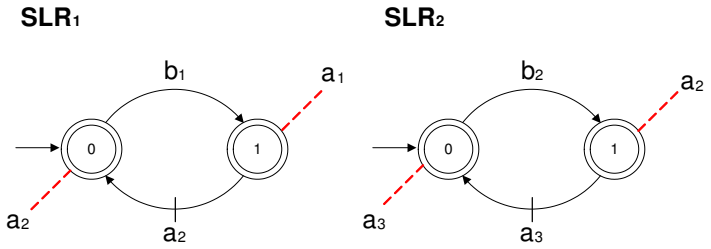


Figura 3.21 – Supervisores Modulares Locais Reduzidos SLR_1 e SLR_2 para o Sistema de Manufatura

3.7 Máquinas de Estados com Saídas: Abordagem Modular Local

Embora as máquinas de estados monolíticas sejam concebidas com o intuito de reduzir o tamanho do modelo do supervisor se comparadas com um Autômato Finito Determinístico (AFD), da mesma forma que os supervisores monolíticos obtidos através da TCS elas podem não ser adequadas para representar sistemas de grande porte. Assim, mesmo com a utilização do modelo monolítico reduzido não se espera obter muitos benefícios em se tratando de redução de código para estes sistemas.

Uma representação de máquina de estados tendo como base o principio de síntese da abordagem modular local (DE QUEIROZ e CURY, 2002) é tratada nesta seção. Da mesma forma que para os supervisores, essa abordagem visa reduzir o tamanho do modelo se comparado com a abordagem monolítica. O objetivo aqui é descrever supervisores modulares locais por intermédio de máquinas de estado, visando explorar a natureza modular da planta sob análise e reduzir a complexidade do modelo do controlador.

3.7.1 Algoritmo para Obtenção de Máquinas de *Mealy* na Abordagem Modular Local

O algoritmo 3.4 apresenta o procedimento para obtenção da máquina de *Mealy* na abordagem modular local. Ele é similar ao algoritmo proposto para obter a máquina de *Mealy* na abordagem monolítica. Agora, entretanto, como parâmetros de entrada são utilizados os supervisores modulares locais e os respectivos mapas de desabilitações, obtidos a partir da síntese dos supervisores modulares locais feita com o uso da abordagem modular local proposta em (DE QUEIROZ e CURY, 2002).

Algoritmo 3.4 Obtenção da Máquina de *Mealy* na Abordagem Modular Local

1: Lê os autômatos dos supervisores modulares locais (SL_j) e os mapas de desabilitações correspondentes (MD_j)
2: Divide os autômatos dos supervisores de acordo com a controlabilidade de seus eventos em *SLju* (não controlável) e *SLjc* (controlável)
3: **para** (Supervisor Local *SLj*, *j* = 1...*m*) **faça** % onde *m* é o número de supervisores modulares locais
4: Cria Estado Inicial ML_j = Estado Inicial de *SLj*
5: Cria Lista com Estados de Chegada que são obtidos a partir dos Estados de Partida e precisam ser tratados
6: **enquanto (**Lista de Estados > 0) **faça**
7: Lê Estado da Lista e considera agora como um Estado de Partida
8: Cria Lista de Transições para esse Estado
9: **para** (Lista de Transições) **faça**
10: Lê *SLju* e *SLjc* para o supervisor *SLj* e *MD* de todos os supervisores locais
11: **se** (*SLjc* evoluiu) **então**
12: Atualiza MD_j e *SLjc*
13: **senão**
14: Cria um Novo Estado de Chegada
15: **fim se**
16: **se** (Estado já foi criado) **então**
17: **não faz nada**
18: **senão**
19: Adiciona Estado de Chegada na Lista de Estados de acordo com o estado de chegada do autômato que representa o supervisor *SLj*
20: **fim se**
21: **fim para**
22: Decrementa Lista de Estados
23: **fim enquanto**
24: Elimina estados não alcançáveis
25: **fim para**
26: Salva a **Máquina de Estados na Abordagem Modular Local *ML***

A fim de que o sincronismo entre os subsistemas seja garantido durante a concepção de uma máquina a partir de um supervisor modular local, devem-se levar em conta as desabilitações presentes nos demais supervisores presentes no modelo do sistema durante o processo de varredura do supervisor a fim de obter os estados da máquina. Dessa forma, ao ser criada a máquina modular local relativa a um supervisor específico, o algoritmo deve prever a evolução dos demais supervisores, visando observar as ações que são permitidas de ocorrer após a evolução de cada supervisor modular local.

Assim, durante a criação dos estados de uma determinada máquina, o mapa de desabilitações relativo a cada supervisor presente no modelo será avaliado de acordo com o estado que esse supervisor se encontra (veja linha 10 do Algoritmo 3.4). Um determinado evento controlável será permitido de ocorrer somente se nenhum dos supervisores locais desabilitá-lo, sendo então representado na máquina de estados em questão como uma ação. Esse processo iterativo de evolução dos supervisores é realizado até que todos os estados presentes nos modelos por autômatos sejam alcançados, resultando então nas máquinas finitas com saídas que representam a abordagem modular local.

3.7.2 Representação da Máquina de *Mealy* na Abordagem Modular Local

A partir do Algoritmo 3.4, duas máquinas de *Mealy* na abordagem modular local são obtidas para o sistema de manufatura, conforme ilustrado na Figura 3.22.

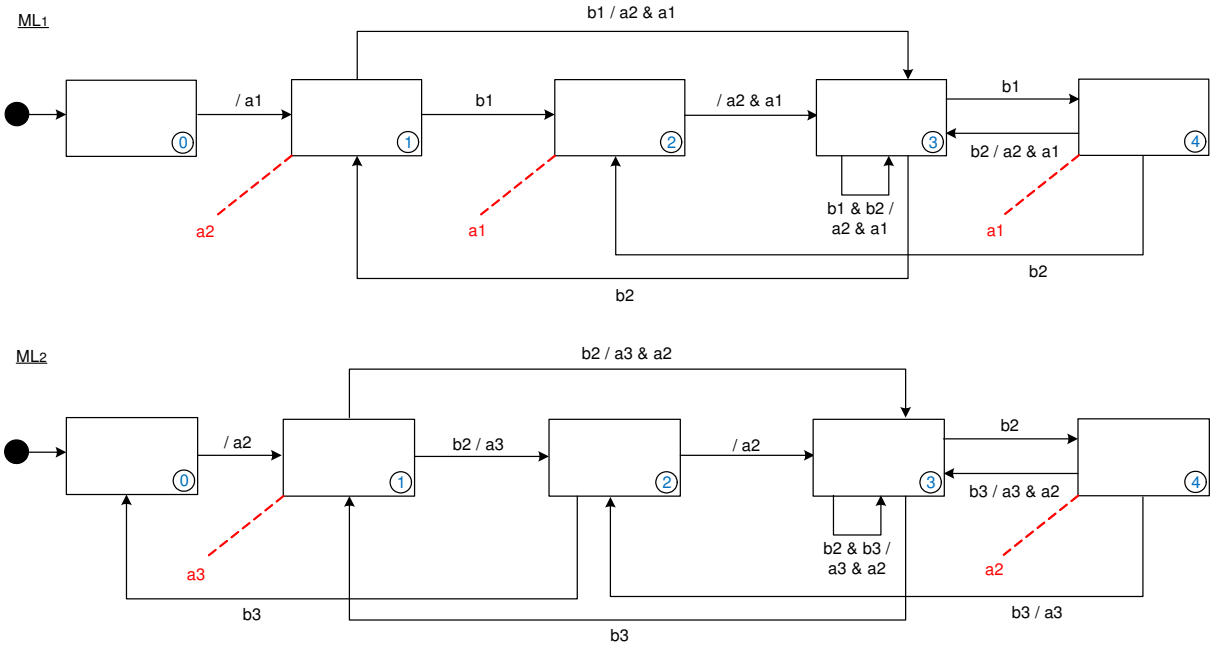


Figura 3.22 – Máquinas de Estados de *Mealy* ML_1 e ML_2 na Abordagem Modular Local

A evolução das máquinas consiste em, a partir do estado inicial, responderem aos eventos não controláveis e transitarem caso eles sejam representados no modelo. Se um mesmo evento não controlável for compartilhado por mais do que uma máquina do modelo (caso de b_2 por exemplo para o sistema de manufatura), ambas máquinas evoluem com a ocorrência deste evento. Após a ocorrência das transições, as desabilitações presentes nos estados de chegada das máquinas serão observadas a fim de avaliar quais ações são permitidas de ocorrer. Caso uma determinada ação também resulte numa transição de estados, a máquina em questão deve evoluir com a execução dessa ação.

As máquinas ML_1 e ML_2 funcionam de forma síncrona e possuem ações conjuntas. No início da operação, a ação a_1 ocorre na máquina ML_1 ligando o aparelho A_1 enquanto a máquina ML_2 permanece em seu estado inicial devido à ação a_2 estar desabilitada pela máquina ML_1. Após a ocorrência da transição b_1, as ações a_2 & a_1 são executadas pois nenhuma das duas máquinas está a desabilitá-las, tal que, além de ligar o aparelho A_2, o aparelho A_1 é ligado novamente. Nesse instante, a máquina ML_1 encontra-se no estado *3* e a máquina ML_2 no estado *1*, onde três condições diferentes podem ocorrer: b_1, b_2 ou b_1 & b_2. Se b_1 ocorrer, a máquina ML_1 transita para o estado *4* e nenhuma ação é executada; se b_2 ocorrer,

a máquina ML_1 transita para o estado *1*, a máquina ML_2 transita para o estado *2* e a ação a_3 é executada, ligando o aparelho A_3; se b_1 & b_2 ocorrer, a máquina ML_1 permanece no estado *3* por meio de um *self-loop* e a máquina ML_2 transita para o estado *3*, devido à transição b_2. Isso ocorre pois nenhuma ação é desabilitada por ambas as máquinas nesses estados, tal que a máquina ML_1 permite a execução de a_2 & a_1 devido ao *self-loop* e a máquina ML_2 executa a_3 & a_2, tal que todos os aparelhos são ligados nesse momento. Considere agora a situação onde b_1 & b_2 ocorrem novamente. A máquina ML_1 permanece no estado *3* gerando um *self-loop* e a máquina ML_2 transita para o estado *4*, devido à transição b_2. Diferente da situação anterior, agora a máquina ML_2 desabilita a ação a_2 ao atingir o estado *4*. Dessa forma as ações a_2 & a_1 não ocorrem na máquina ML_1, tal que nenhuma ação é executada nesta transição e apenas o aparelho A_3 permanece ligado pois já tinha sido ativado anteriormente. Observe que embora somente a ação a_2 esteja desabilitada, ação a_1 também é proibida de ocorrer, pois essa ação será permitida de ocorrer somente depois que a ação a_2 for executada. Seguindo essa linha de raciocínio, as demais transições e ações das máquinas na abordagem modular local são constituídas. Uma composição dessas máquinas de acordo com essa modelagem resulta na máquina de *Mealy* obtida na abordagem monolítica, ilustrada anteriormente na Figura 3.13.

Além disso, ao observar a máquina ML_1, por exemplo, verifica-se uma situação que não ocorre na abordagem monolítica: a máquina possui um caráter não-determinístico devido a uma mesma transição levar a máquina para dois estados distintos. Na máquina ML_1, a transição b_1 sai do estado de partida *1* para dois estados de chegada: estados *2* e *3*. Ao chegar no estado *2*, nenhuma ação é executada; ao chegar no estado *3*, as ações a_2 & a_1 são executadas. De acordo com a evolução dos supervisores durante o processo de obtenção da máquina ML_1, pode acontecer que o supervisor SL_2 desabilite ou não os eventos a_2 e a_1. Dessa forma, surge a questão: como definir qual será o estado de chegada e que ações serão executadas? É necessário olhar as desabilitações nos estados em que as demais máquinas se encontram durante a dinâmica do processo. Para esse exemplo, deve-se observar o estado em que a máquina ML_2 se encontra. Como o evento b_1 é específico da máquina ML_1, ele não ocasiona transição de estados na máquina ML_2. Caso o estado em que a máquina ML_2 se encontra desabilite a ação a_2, então o estado de chegada da máquina ML_1 será o estado *2*. Caso contrário, o estado de chegada da máquina ML_1 será o estado *3*. Assim, a desabilitação da ação a_2 na máquina ML_2 resulta na desabilitação do conjunto a_2 & a_1 na máquina ML_1. A situação de não-determinismo também aparece na máquina ML_2 e o raciocínio é o mesmo do descrito para a máquina ML_1.

Situações de não-determinismo também aparecem em autômatos, como mostrado anteriormente na seção 2.4. Entretanto, na metodologia proposta para máquinas de estados, o processo de determinização é diferente do realizado em autômatos, sendo descrito na próxima seção.

Outra situação que surge nesse modelo correspondente à ocorrência de transições de estados devido a ações. Na abordagem monolítica esse comportamento é observado somente na inicialização do sistema. Na abordagem modular local isso aparece porque durante a concepção de uma máquina ML_j as desabilitações em todos os supervisores modulares locais são avaliadas. Em um determinado momento da varredura dos estados do supervisor SL_j, pode ocorrer que essas ações são permitidas de ocorrer pois nenhum supervisor desabilita-as. Em outro momento, essas ações estarão desabilitadas. Como conseqüência, isso vai aparecer nos modelos das máquinas de estados. Na máquina ML_1, por exemplo, a ação a_2 & a_1 faz com que a máquina transite do estado *2* para o *3*. Situação similar também ocorre para a máquina ML_2, onde a ação a_2 faz com que essa máquina também transite do estado *2* para o *3*.

3.7.3 Redução da Máquina de *Mealy* na Abordagem Modular Local

O procedimento de redução das máquinas de *Mealy* na abordagem modular local possui particularidades que diferem da abordagem monolítica devido à existência de não-determinismos e de transições de estado devido somente a ações. Assim, o objetivo aqui é eliminar estas duas características do modelo, tal que a máquina resultante seja determinística e nenhuma transição de estado seja devido a ações, mas somente devido a eventos não controláveis. Com isso, se espera que tanto o número de estados quanto de transições seja reduzido, diminuindo assim a complexidade do modelo e mantendo a mesma lógica de controle do modelo original. O Algoritmo 3.5 descreve a 1ª etapa da metodologia proposta para obter uma máquina de *Mealy* reduzida na abordagem modular local.

Algoritmo 3.5 Redução da Máquina de *Mealy* na Abordagem Modular Local (1ª Etapa)

1: Lê as máquinas modulares locais (ML_j) e os mapas de desabilitações correspondentes das máquinas (MDM_j)
2: **para** (Máquina Local MLj, j = 1...m) **faça** % onde m é o número de máquinas modulares locais
3: **para** (Estado da Máquina = i até n) **faça** % onde n é o número de estados da máquina
4: **se** (Estado possui ação para outro estado sem uma transição) **então**
5: Une estados de partida e de chegada em um só estado resultante e cria *self-loop* com a ação
6: Transições que chegavam aos estados de partida e de chegada chegam somente ao estado resultante da união desses estados
7: Transições que representam não-determinismos resultam em uma só transição e as ações que eram comum para ambas as transições são mantidas na transição resultante
8: Desabilitações presentes nos estados de partida e de chegada permanecem no modelo
9: **fim para**
10: Salva a **Máquina de Estados de *Mealy* na Abordagem Modular Local Reduzida *MLS'***

O Algoritmo 3.5 é melhor compreendido usando o sistema de manufatura como exemplo. A Figura 3.23 ilustra as máquinas reduzidas na abordagem modular local para o sistema analisado.

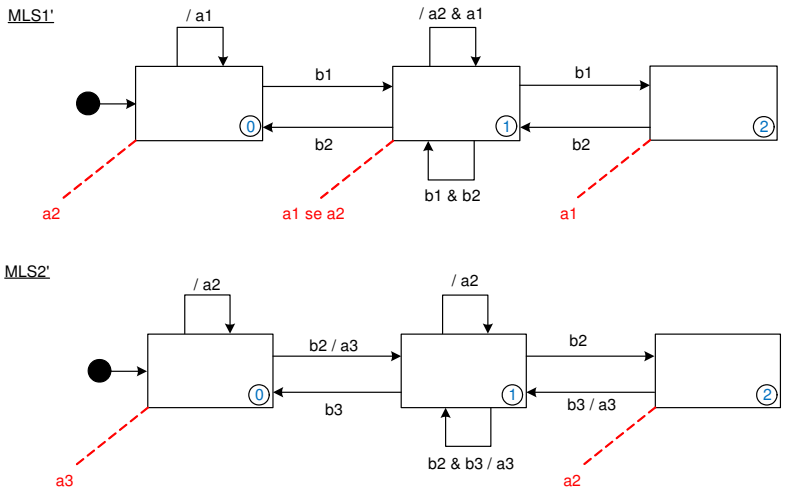


Figura 3.23 – Procedimento de Redução das Máquinas de Estados MLS_1' e MLS_2' na Abordagem Modular Local, 1ª Etapa

Considere a máquina reduzida MLS_1', por exemplo. O estado *0* dessa máquina consiste na união do estado inicial com o estado *1* da máquina original. Essa união é efetuada devido a transição entre eles ser dada originalmente somente pela ação a_1. Embora na abordagem monolítica reduzida essa ação seja representada igual ao modelo monolítico original, na abordagem modular local os estados de partida e de chegada são unidos e essa ação é tratada como um *self-loop* dentro do estado resultante, diferente da máquina original ilustrada anteriormente na Figura 3.22. O mesmo ocorre para a máquina MLS_2' para a ação a_2.

Além disso, o estado resultante *1*, proveniente da união dos estados *2* e *3* da máquina original, é responsável por desabilitar a ação a_1 somente se a ação a_2 também estiver

desabilitada, por meio da máquina MLS_2'. Caso a_2 não estiver desabilitada por MLS_2', então as ações a_2 & a_1 são permitidas para ocorrer, como mostrado na Figura 3.23. Isso ocorre devido à especificação do sistema de que, após o termino de operação do aparelho A_1 (evento b_1), um novo início de operação (evento a_1) só pode ocorrer após o início de operação do aparelho A_2 (evento a_2).

Ainda, na máquina MLS_2' a transição b_3 que antes chegava no estado inicial e no estado *1* agora chega somente no estado resultante *0* do modelo reduzido. O estado resultante *0* mantém também a desabilitação a_3 que estava presente no estado inicial e no estado *1* da máquina original.

Voltando a atenção agora para a máquina original ML_2, a transição b_2 que parte do estado *1* representa uma situação de não-determinismo, tendo como chegada os estados *2* e *3*. O Algoritmo 3.5 resolve esse problema considerando a união dessas transições em uma só e mantendo nessa transição resultante apenas as ações que são comuns para ambas as transições originais. Dessa forma, na máquina reduzida MLS_2', apenas a ação a_3 é mantida com essa nova transição, enquanto a ação a_2 é representada como um *self-loop* no estado de chegada *1*. Isso acontece pois a ação a_2 pode ocorrer ou não na máquina original, dependendo se a máquina ML_1 permitir que ela seja executada ou não. A ação a_3 por sua vez estará sempre habilitada, sendo executada quando qualquer umas das transições que partem do estado *1* de ML_2 ocorrer. Esse comportamento se reflete também no modelo reduzido. Na prática, isso significa que a ação a_2 só pode ser executada desde que a ação a_3 já tenha sido executada. A representação da ação a_2 por meio de um *self-loop* no estado *1* da máquina reduzida MLS_2' significa que essa ação será executada somente se a máquina reduzida MLS_1' não estiver desabilitando-a.

Observa-se também na Figura 3.23 que a ocorrência de ações em *self-loop* poderia ser representada como uma ação de entrada no estado, tal como acontece na máquina de *Moore*. O foco deste trabalho será voltado à representação somente de máquinas de *Mealy* na abordagem modular local, onde a transformação para máquinas de *Moore* poderia ser feita facilmente, de forma similar ao procedimento usado para a abordagem monolítica. Essa decisão decorre do fato do modelo de *Moore* ser bastante similar ao modelo de *Mealy*, assim como acontece na abordagem monolítica.

Ainda na Figura 3.23, observa-se que existem transições nas máquinas que resultam em estados onde não existe a possibilidade de executar ações. Isso aparece na máquina MLS_1' quando a transição b_1 ocorre a partir do estado *1* resultando no estado *2* e na máquina MLS_2' quando a transição b_2 ocorre a partir do estado *1* resultando no estado *2* dessa máquina. Dessa

forma, é possível realizar uma redução similar a adotada na abordagem monolítica: criar *self-loops* representados por linhas tracejadas com as transições que não resultam em execução de ações. Agora porém, é necessário tomar cuidado com as desabilitações existentes nos estados de chegada dessas transições. Na abordagem modular local, elas não podem ser simplesmente omitidas, pois quando esses estados são atingidos eles podem conter desabilitações que afetam diretamente a dinâmica das demais máquinas presentes no sistema. Sendo assim, introduz-se o operador lógico "~", que representa uma condição lógica *NOT*. Na metodologia proposta neste trabalho, esse operador aparece somente na representação reduzida das máquinas de estados da abordagem modular local, onde *self-loops* tracejados representam transições que não resultam na execução de ações, mas são responsáveis pela desabilitação de ações e significam que, quando essas transições ocorrerem, as ações representadas pelo operador "~" são proibidas de serem executadas.

Observe também que a transição b_2 partindo do estado *1* da máquina MLS_1' e chegando ao estado *0* também não possui ação associada a ela. Entretanto, o estado de chegada *0* permite a ocorrência da ação a_1, representada por meio de um *self-loop*, tal que essa transição não pode ser reduzida, assim como foi proposto para as demais.

O Algoritmo 3.6 descreve a 2ª etapa da metodologia para obter uma máquina de *Mealy* reduzida na abordagem modular local, onde os *self-loops* representados por linhas tracejadas são obtidos e os estados que se tornam não alcançáveis são portanto eliminados do modelo.

Algoritmo 3.6 Redução da Máquina de *Mealy* na Abordagem Modular Local (2ª Etapa)

1: Lê as máquinas modulares locais reduzidas (MLS_j) e os mapas de desabilitações correspondentes das máquinas reduzidas ($MDMS_j$)
2: **para** (Máquina Local Reduzida *MLSj*, *j* = 1...*m*) **faça** % onde *m* é o número de máquinas reduzidas
3: **para** Estado da Máquina = *i* até *n* **faça** % onde *n* é o número de estados da máquina
4: **para** Transição = *i* até *t* **faça** % onde *t* é o número de transições saindo do estado
5: **se** (Transição resulta em execução de ação ou em estado destino que executa ação) **então**
6: Mantém a transição na máquina reduzida
7: **senão**
8: Cria *self-loop* representado por linhas tracejadas no estado de partida
9: **fim se**
10: **fim para**
11: **fim para**
12: Elimina estados não alcançáveis da máquina resultante
13: **fim para**
14: Salva a **Máquina de Estados de *Mealy* na Abordagem Modular Local Reduzida *MLS***

As máquinas de estados reduzidas MLS_1 e MLS_2 na abordagem modular local para o sistema de manufatura são ilustradas na Figura 3.24. Trata-se de um modelo bastante compacto onde cada máquina é composta por apenas 2 estados, possui transições devido a

eventos não controláveis, transições em *self-loops* devido a ações e transições em *self-loops* representados por linhas tracejadas devido a eventos não controláveis que aparecem no modelo mas não responsáveis por permitir a execução de ações. Além disso, as desabilitações de ações aparecem de forma convencional nos estados assim como quando ocorrem eventos que não resultam em ações, sendo então representadas pelo operador lógico "~".

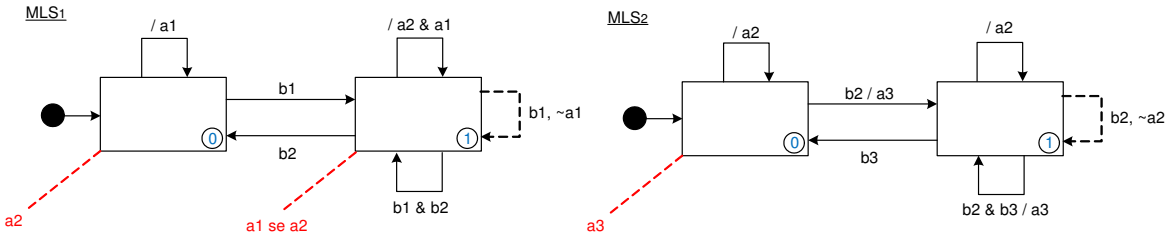


Figura 3.24 – Máquinas de Estados Reduzidas MLS_1 e MLS_2 na Abordagem Modular Local

Para melhor compreensão desse modelo considere, por exemplo, que ambas as máquinas reduzidas estejam no estado *1* e todos os aparelhos, A_1, A_2 e A_3, estejam ligados. Se ocorrerem os eventos b_1 & b_2, tanto a máquina MLS_1 como a máquina MLS_2 permanecem no estado *1*. Todavia, a máquina MLS_2 vai ser responsável pelo *self-loop* com linhas tracejadas devido ao evento b_2 tal que a ação a_2 seja desabilitada. Assim, embora a máquina MLS_1 tenha o *self-loop* devido a b_1 & b_2, as ações a_2 & a_1 não são executadas. Caso em seguida ocorra o evento b_3, ao invés da máquina MLS_2 transitar para o estado *0*, ela permanece no estado *1*. A transição a ser executada é b_2 & b_3 e não a transição b_3, pois a transição b_2 foi armazenada na memória interna quando o *self-loop* com linhas tracejadas ocorreu anteriormente, sendo somente agora executada efetivamente. Assim, a máquina MLS_2 permite a execução da ação a_3 enquanto a máquina MLS_1 permite a execução de a_2 & a_1, pois não existe mais nenhuma desabilitação ativa nesse instante.

3.8 Conclusões

Neste capítulo foi efetuada uma breve revisão das abordagens monolítica e modular local da Teoria de Controle Supervisório (TCS), que serviram como base para apresentar as metodologias para obtenção de máquinas de estados de caráter monolítico e modular local, respectivamente. Apresentaram-se propostas para redução das máquinas de estados sem que ocorresse perda de informações relevantes do modelo que pudessem afetar a lógica de controle estabelecida inicialmente. A obtenção de modelos de acordo com a abordagem modular local possui algumas particularidades que tornam essa tarefa um pouco mais complexa. Isso ocorre devido à necessidade de levar em conta a evolução de todos os

supervisores locais durante a concepção de uma máquina, o que resulta no surgimento dos problemas de transições devido somente a ações e não-determinismo. Entretanto, o procedimento de redução é adotado a fim de eliminar esses problemas e reduzir o número de estados e transições tal que as máquinas reduzidas obtidas são capazes de representar a dinâmica do sistema de uma forma bastante compacta.

Na literatura, muitos autores procuram implementar os supervisores da maneira em que são sintetizados usando a TCS ou, em alguns casos, utilizam algoritmos para redução no número de estados desses supervisores. A metodologia de modelagem proposta neste trabalho serve também como uma forma de reduzir os supervisores obtidos a partir da TCS, onde estados e eventos redundantes para efetuar a lógica de controle são removidos do modelo. Além disso, as máquinas de estados representam modelos diretos para implementação em controladores síncronos. Isso ocorre basicamente devido a se ter uma distinção bem clara entre os eventos não controláveis (representados por transições nas máquinas) e eventos controláveis (representados por ações nas máquinas). Dessa forma, o caráter entrada-saída de uma máquina já é evidente para se fazer a implementação em um controlador que também possui esse comportamento construtivo.

No próximo capítulo, serão primeiramente abordados os problemas encontrados na migração da TCS para a realização prática de supervisores em controladores síncronos. Após isso, serão abordados os aspectos de implementação dos modelos de máquinas de estados obtidos neste capítulo em Controladores Lógicos Programáveis (CLPs) usando linguagem *Ladder* e um modelo baseado na estrutura proposta por (DE QUEIROZ e CURY, 2002) para fins comparativos. A metodologia baseada em máquinas de estados surge como uma proposta genérica para implementação em outros tipos de controladores síncronos. Entretanto, isso está fora do escopo de abordagem deste trabalho.

Capítulo 4

Implementação de Máquinas de Estados com Saídas em Controladores Lógicos Programáveis

O objetivo deste capítulo é inicialmente descrever os problemas encontrados na implementação de supervisores em controladores síncronos, tais como os CLPs. Em seguida, é apresentada uma metodologia de implementação dos modelos baseados em máquinas de estados em Controladores Lógicos Programáveis (CLPs) utilizando linguagem *Ladder*. São apresentadas propostas para implementação de máquinas de estados tanto de acordo com a abordagem monolítica quanto de acordo com a abordagem modular local. Metodologias para implementação de supervisores modulares locais para CLPs são apresentadas em (DE QUEIROZ e CURY, 2002), (VIEIRA, 2003), (DE QUEIROZ, 2004), (CURZEL, 2008), (LEAL *et al*., 2009), dentre outros. Além disso, outras implementações de controle descentralizado utilizando supervisores são propostas na literatura, tal como em (AFZALIAN *et al*, 2008). O sistema de manufatura apresentado no capítulo anterior é usado novamente neste capítulo para exemplificar os procedimentos para implementação.

O capítulo está dividido como descrito a seguir. A Seção 4.1 descreve os problemas encontrados para implementação de supervisores em controladores síncronos, tais como CLPs. Na Seção 4.2 é descrita a implementação das máquinas de estados em CLPs de acordo com a abordagem monolítica. Na Seção 4.3 é ilustrada a implementação de supervisores modulares locais de acordo com a estrutura proposta em (DE QUEIROZ e CURY, 2002) para efeito de comparação. A Seção 4.4 trata dos aspectos de implementação em CLPs considerando as máquinas de estados na abordagem modular local. Para finalizar, a Seção 4.5 aborda as conclusões relativas aos procedimentos de implementação das máquinas de estados em CLPs.

4.1 Problemas Relativos à Implementação de Supervisores em Controladores Síncronos

A solução de problemas de automação de manufatura pode ser obtida com o uso da Teoria de Controle Supervisório (TCS), onde a planta de chão de fábrica é modelada como um Sistema a Eventos Discretos (SED) e autômatos de estado finito são usados para descrever o comportamento da planta em malha aberta, as especificações de controle e o supervisor. Sendo assim, de acordo com Fabian e Hellgren (1998) "a implementação do supervisor é basicamente uma questão de fazer o Controlador Lógico Programável (CLP) se comportar como uma máquina de estados". Entretanto, essa não é uma tarefa simples. Certos problemas surgem durante o processo de implementação em controladores síncronos, tais como CLPs e computadores. Esses problemas existem independentes do modelo usado para representar os supervisores, seja por meio de autômatos, redes de Petri ou redes de Petri coloridas (BASILE e CHIACCHIO, 2007). O foco deste trabalho consiste na implementação de supervisores em CLP, sendo modelados por autômatos.

4.1.1 Causalidade

A TCS assume que todos os eventos são gerados espontaneamente pela planta e que o supervisor observa a cadeia de eventos gerados pela planta e atua na desabilitação de eventos controláveis de modo a evitar qualquer violação nas especificações de controle. Entretanto, na maioria das aplicações práticas, os eventos controláveis não são gerados espontaneamente pela planta física, mas apenas as respostas devido aos comandos enviados. Dessa forma, é necessário responder à questão "quem gera o quê?" (FABIAN e HELLGREN, 1998), isto é, quem é o responsável pela geração de determinados eventos? O supervisor ou a planta? (VIEIRA, 2007).

O esquema de supervisão proposto pela TCS (RAMADGE e WONHAM, 1989) é ilustrado na Figura 4.1-a. Nessa estrutura, a planta é responsável pela geração dos eventos, tanto controláveis quanto não controláveis, enquanto o supervisor apenas observa-os e desabilita um conjunto de eventos controláveis para satisfazer os requisitos de controle.

Entretanto, na maioria das aplicações práticas os eventos modelados como controláveis correspondem a comandos que, na verdade, devem ser gerados pelo elemento de controle e enviados para os atuadores, pois eles não ocorrem espontaneamente. A planta por sua vez gera espontaneamente os eventos não controláveis, como resposta à excitação enviada

pelo CLP por intermédio dos eventos controláveis. A Figura 4.1-b mostra a estrutura de controle normalmente empregada na prática (MALIK, 2002).

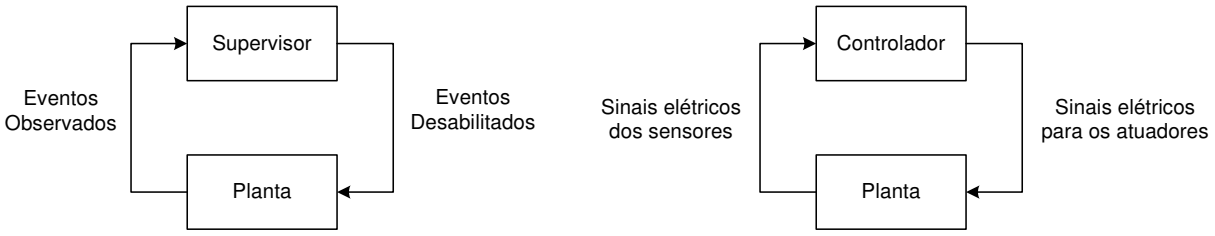


Figura 4.1 – (a) Esquema de Supervisão da TCS (b) Esquema de Controle Normalmente Empregado na Prática

Neste trabalho, o problema de causalidade é resolvido ao se representar o modelo por meio de máquinas de estado ao invés de autômatos. Assim, ao realizar essa transformação, ocorre a mudança da TCS para um modelo de entradas e saídas, como sugerido em (MALIK, 2002). Os sinais provenientes dos sensores correspondem aos eventos não controláveis enquanto os sinais enviados para os atuadores correspondem aos eventos controláveis.

4.1.2 Detecção de Eventos

A natureza síncrona dos CLPs pode levar a problemas na detecção de eventos não controláveis como os descritos a seguir.

Sinais e Eventos

Alguns problemas de implementação devem-se à inexistência de uma maneira trivial para traduzir supervisores baseados em eventos discretos, os quais são simbólicos, assíncronos e ocorrem em instantes discretos no tempo em um universo síncrono e baseado em sinais como o de CLPs (FABIAN e HELLGREN, 1998). Para evitar essa discrepância entre a teoria e a prática, um sinal não pode gerar mais do que um evento $\sigma \in \Sigma_{uc}$ durante um ciclo de execução do CLP (BASILE e CHIACCHIO, 2007).

Efeito Avalanche

Os sinais presentes no domínio dos CLPs são rotulados, podem assumir valores *booleanos* e são amostrados periodicamente. Portanto, a fim de implementar supervisores de acordo com a TCS em CLPs, os eventos são associados com mudanças nos sinais do CLP, o que pode causar o chamado *efeito avalanche*. Este efeito ocorre quando uma mudança de valor em um sinal de entrada do CLP é registrada como um evento não controlável que faz com que o *software* salte por um número arbitrário de estados durante um mesmo ciclo de

execução do CLP. Isto pode ocorrer especificamente se um determinado evento não controlável é utilizado para disparar o gatilho de muitas transições de estado em uma lista, produzindo um comportamento como se fosse uma avalanche.

A Figura 4.2 ilustra um exemplo de efeito avalanche para uma implementação convencional de supervisor em CLP. O supervisor transita do estado *0* para o estado *2* com a ocorrência do evento b_1. Ele deveria transitar do estado *1* para o estado *2* somente com a ocorrência de um novo evento b_1, ou transitar do estado *1* para o estado *3* com a ocorrência de um evento b_2. Entretanto, a implementação proposta no lado direito da Figura 4.2 não restringe que o evento b_1 permita a transição do estado *1* para o estado *2* no mesmo ciclo de execução do CLP. Dessa forma, o efeito avalanche introduz uma transição direta do estado *0* para o estado *2*, tal que, mesmo que o evento b_2 ocorresse, ele não teria efeito na dinâmica do controlador. Isso é claramente insatisfatório.

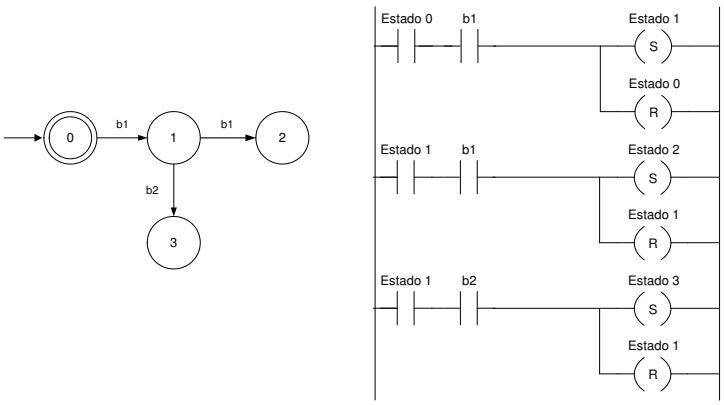


Figura 4.2 – Exemplo de Efeito Avalanche

A fim de solucionar esse problema, Vieira (2007) propõe um procedimento onde uma variável é associada a todas as transições presentes no supervisor. Essa variável é ativada sempre que ocorrer uma transição de estado, sendo desativada no final da seção de código de programa onde a estrutura do supervisor está implementada. Toda transição de estado inclui como condição a negação desta variável. Esta solução resolve o problema, porém tem a desvantagem de permitir que apenas um evento ocorra dentro de um ciclo de execução do CLP.

A solução proposta neste trabalho consiste em desativar a variável correspondente ao evento ocorrido assim que uma transição de estado devido àquele evento acontece. Esta solução será apresentada na Seção 4.2, onde a variável que representa um determinado evento é desativada logo após a ocorrência desse evento, tal que não seja necessária a criação de uma variável adicional para essa finalidade.

Dessa forma, o supervisor desativa aquele evento específico que ocorreu, permitindo que transições devido a outros eventos presentes no modelo possam ocorrer. Se durante um ciclo de execução eventos pertencentes a especificações assíncronas acontecerem, eles serão tratados nesse mesmo ciclo. Essa proposta resolve o problema do efeito avalanche e permite que diversos eventos não controláveis sejam tratados dentro de um mesmo ciclo de execução do CLP.

Além disso, a restrição de que mais do que um evento seja tratado dentro de um mesmo ciclo de execução gera outro problema denominado perda de informação (VIEIRA, 2007). Esse problema ocorre em situações onde diversos eventos ocorrem e somente um é tratado por ciclo de execução. Assim, a informação relativa à ocorrência dos demais eventos é perdida. Esse problema não ocorre na metodologia proposta aqui devido à possibilidade de tratamento dos diversos eventos que ocorreram em um mesmo ciclo de execução. Isso se deve ao fato de não haver uma variável geral que seja ativada quando uma transição de estado ocorrer, mas sim um tratamento específico para cada evento, onde os demais eventos continuam habilitados para que disparem transições.

Incapacidade de Reconhecer a Ordem dos Eventos

Esse problema de simultaneidade foi abordado primeiramente em (FABIAN e HELLGREN, 1998) considerando o processo de implementação de supervisores em CLPs. Entretanto, esse é um problema que não fica restrito somente à implementação em CLPs, podendo aparecer em qualquer outro tipo de controlador onde a leitura de suas entradas seja feita de forma síncrona.

Caso dois ou mais sinais de entrada mudem de valor entre duas leituras da entrada, estas mudanças serão registradas como ocorrência simultânea de eventos não controláveis, não sendo possível identificar qual evento ocorreu primeiro, pois o CLP efetua uma leitura síncrona de suas entradas. Percebe-se que essas mudanças de sinal podem ser simultâneas ou não, mas os eventos serão registrados sempre como simultâneos durante a leitura. Esse problema é denominado *simultaneidade* (FABIAN e HELLGREN, 1998). Dessa forma, a fim de evitar problemas de implementação, o supervisor que dará origem à máquina de estados deve ser tal que a ação de controle não dependa dos diferentes entrelaçamentos dos eventos não controláveis. Fabian e Hellgren (1998) definem esta propriedade como insensibilidade ao entrelaçamento (*interleave insensitivity*) de eventos não controláveis.

Um supervisor *S* é insensível ao entrelaçamento com relação a uma planta *G* (definida sobre um alfabeto $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$), se para $s_1, s_2 \in \Sigma_{uc}^*$ e $\sigma \in \Sigma_c^*$,

$$s\, s_1\, s_2\, \sigma \in L(S/G) \rightarrow s\, (s_1 \,|||\, s_2)\, \sigma \subseteq L(S/G) \tag{4.1}$$

Onde todos os entrelaçamentos de s_1, s_2 consistem em uma linguagem denotada pelo operador $s_1|||s_2$. Em palavras, um supervisor é insensível ao entrelaçamento de eventos não controláveis se sua ação de controle não depender dos possíveis entrelaçamentos de eventos não controláveis que possam ocorrer após qualquer cadeia de eventos *s*.

Caso esta propriedade não seja satisfeita, o supervisor pode adotar uma ação de controle incorreta, em função de não reconhecer a ordem de ocorrência dos eventos não controláveis. Este problema não existe se o supervisor é insensível ao entrelaçamento (FABIAN e HELLGREN, 1998).

A Figura 4.3 apresenta um exemplo de supervisor que é sensível ao entrelaçamento. Ações de controle distintas são executadas caso os sinais de entrada do CLP que correspondem aos eventos não controláveis b_1 e b_2 mudem de valor entre duas leituras da entrada de um CLP. Não existe uma definição clara de qual ação tomar, a_1 ou a_2. A ilustração no lado direito da Figura 4.3 descreve essa variação de nível lógico dos sinais de entrada.

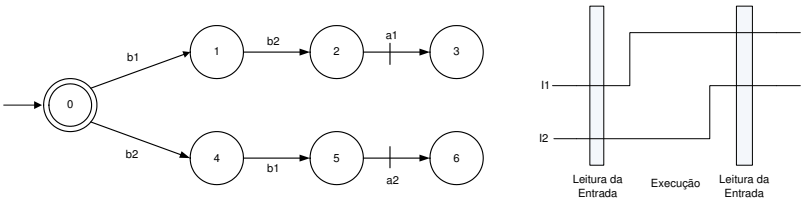


Figura 4.3 – Exemplo de Supervisor Sensível ao Entrelaçamento

A Figura 4.4 apresenta um exemplo de supervisor que é insensível ao entrelaçamento. Neste caso, a mesma ação de controle a_1 é executada independente de qual evento não controlável ocorreu primeiro, b_1 ou b_2.

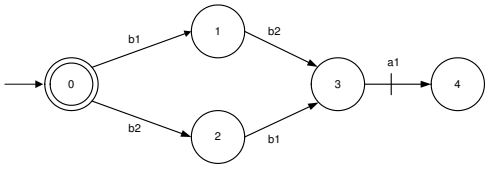


Figura 4.4 – Exemplo de Supervisor Insensível ao Entrelaçamento

Analisando o supervisor obtido para o estudo de caso considerado (ver Figura 3.10), pode-se observar que o mesmo é insensível ao entrelaçamento. Observa-se na Figura 3.10 que, a partir do estado *4* do supervisor, independente se ocorrer a seqüência b_1-b_2 ou b_2-b_1, o

supervisor vai para o estado *9*, permitindo que a mesma ação de controle seja executada (a_3 nesse caso).

O reconhecimento da ordem de ocorrência dos eventos não controláveis não pode ser feito a não ser que, na ocorrência dos eventos, o controlador não seja síncrono e consiga armazenar essa informação de alguma forma para ser tratada posteriormente. Para evitar que esse problema de sensibilidade ao entrelaçamento ocorra em controladores síncronos, é preciso avaliar se os supervisores são insensíveis ao entrelaçamento, o que pode ser verificado através do algoritmo proposto por Fabian e Hellgren (1998).

4.1.3 Resolução de Conflitos

Escolha (ou Conflito)

Esse problema é similar ao de insensibilidade ao entrelaçamento, porém ao invés de ocorrer para eventos não controláveis, ocorre para eventos controláveis. Dessa forma, se num dado estado do supervisor existirem 2 ou mais eventos controláveis habilitados, é necessário escolher que apenas um deles seja gerado, por isso o nome "Escolha". Esse problema é denominado também como conflito (BASILE e CHIACCHIO, 2007). Entretanto, é importante salientar que essa definição de conflito não é relacionada de forma alguma com a noção de linguagens não conflitantes usada na TCS.

De acordo com Fabian e Hellgren (1998), gerar mais do que um evento controlável dentro de um mesmo ciclo de execução pode ser catastrófico. Portanto, as soluções adotadas na literatura seguem a idéia de que "a implementação deve simultaneamente escolher e transitar; e somente um evento simples deve ser escolhido" (FABIAN e HELLGREN, 1998), isto é, somente um evento controlável é gerado em cada ciclo de execução do CLP.

A Figura 4.5 serve para ilustrar esse problema. Quando o supervisor está no estado *1*, o mesmo deve escolher qual evento controlável gerar, a_1 ou a_2. Da maneira que a implementação em linguagem *Ladder* é apresentada no lado direito da Figura 4.5, durante a seqüência de execução do programa, o supervisor gera o evento a_1 e transita para o estado *2*. Assim, a transição do estado *1* para o estado *3* com a geração do evento a_2 nunca ocorre.

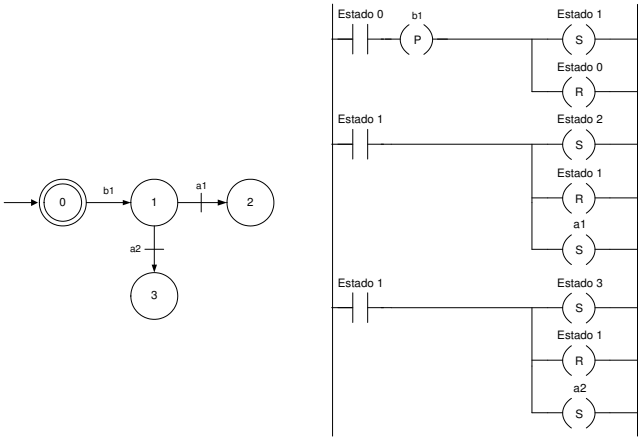


Figura 4.5 – Exemplo do Problema de Escolha

Essa solução prioriza claramente a execução de um evento controlável em detrimento aos demais, o que pode levar a um outro problema. Malik (2002) identificou que, de acordo com a escolha de qual evento executar, a propriedade de não bloqueio do supervisor pode ser perdida. Isso se deve ao fato do controlador limitar a operação apenas para determinadas cadeias de eventos, tal que parte do supervisor não seja utilizado. Assim, estados marcados na parte não utilizada do supervisor jamais serão alcançados, atingindo, portanto uma situação de bloqueio. Esse problema é denominado *problema de determinismo*. Ela define controladores determinísticos como os controladores para os quais cada ação é definida unicamente pela seqüência de eventos ocorridos até então. A fim de gerar um controlador determinístico, é necessário escolher entre eventos controláveis concorrentes que estão habilitados em um determinado estado. Entretanto, ao se derivar um controlador determinístico a partir de um modelo de supervisor, essa propriedade de não bloqueio pode ser perdida e parte do modelo do controlador pode ser bloqueada, conforme mostrado na Figura 4.6. Nesse exemplo, após a ocorrência da cadeia *ac* escolhe-se o evento controlável *f* seguido pelos eventos *h* e *j* (cadeia mostrada em vermelho). A escolha do evento *f* em detrimento ao evento *e* leva a uma ação de controle bloqueante, pois o estado marcado 6 não pode ser mais alcançado.

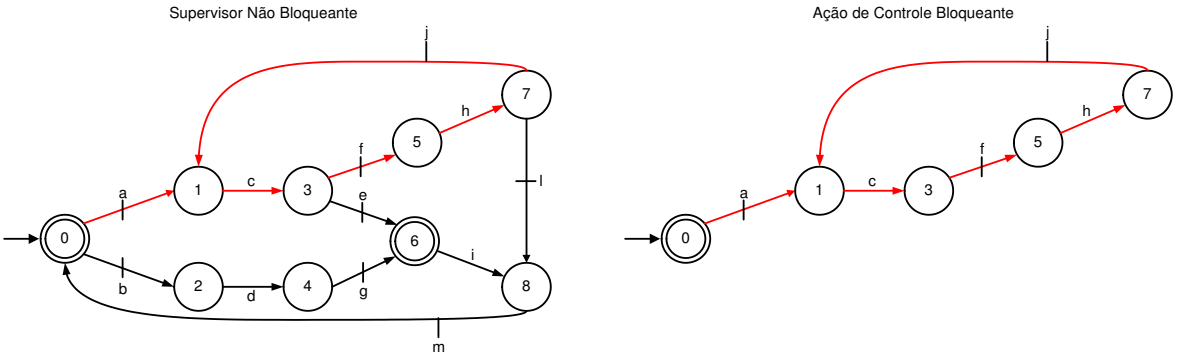


Figura 4.6 – Exemplo do Problema de Determinismo

A fim de evitar que esse problema ocorra, Malik (2002) introduz uma propriedade geral que, quando satisfeita, garante que qualquer controlador para o sistema seja necessariamente não bloqueante.

A metodologia de implementação proposta nesta dissertação tem como intuito tratar mais do que um evento controlável dentro de um mesmo ciclo de execução do programa no CLP. Entretanto, isso é aplicável desde que os eventos controláveis não sejam concorrentes, o que resulta no problema de escolha. Isso significa que, embora mais do que uma ação de controle seja executada, a resultante dessas ações será a mesma independente de qual delas seria executada primeiro no supervisor.

LEAL *et al* (2009) propõem um algoritmo *on-line* para solucionar o problema da escolha entre eventos controláveis por meio de uma abordagem de geração de eventos de forma aleatória. Eles propõem que uma variável responsável pela escolha do evento controlável a ser executado mude de valor lógico a cada ciclo de execução do controlador. Assim, ao identificar a existência do problema de escolha, uma rotina de tratamento deste problema é executada, tal que a decisão de qual evento controlável escolher em um determinado ciclo de execução é completamente aleatória, dependendo exclusivamente de qual valor lógico essa variável assumiu naquele ciclo. Neste trabalho, os autores apresentam um exemplo para resolver o problema de escolha entre dois eventos controláveis.

Abordagens como essa visam minimizar problemas que surgem devido às limitações físicas dos controladores síncronos durante o processo de implementação de supervisores.

Sincronização Inexata

Durante a execução do programa pode ocorrer uma mudança em qualquer sinal de entrada do CLP e, neste caso, esta mudança será reconhecida somente no início do próximo ciclo de execução do programa no CLP. Esse problema pode ocorrer também quando o CLP e a planta estão conectados em rede, tal que atrasos nos canais de comunicação podem causar problemas de comunicação. Assim, a comunicação entre o CLP e a planta está sujeita a atrasos devido à leitura periódica dos sinais de entrada (BALEMI, 1992). A sincronização inexata (FABIAN e HELLGREN, 1998) pode ser um problema quando uma mudança em um sinal de entrada do CLP invalida a ação de controle (a escolha feita pelo programa, que corresponde à geração de um evento controlável), como mostrado no exemplo da Figura 4.7.

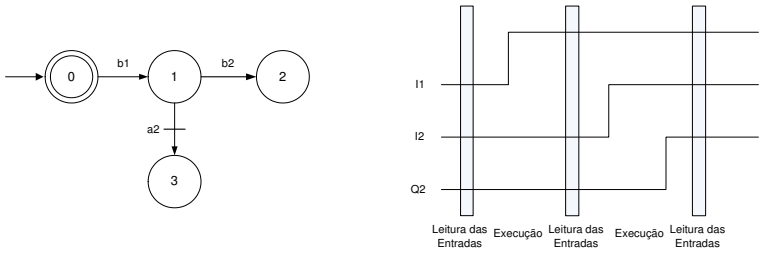


Figura 4.7 – Exemplo de Sincronização Inexata

Neste exemplo, considera-se que o supervisor começa no estado *0* e, quando a entrada I_1 muda de valor, o evento não controlável b_1 é sinalizado, transitando para o estado *1*. Nesse estado, se o CLP não detectar nenhuma mudança de sinal em sua entrada física I_2 que corresponde à sinalização do evento b_2, então o evento controlável a_2 poderá ser executado pelo programa. O problema surge caso, durante esse ciclo de execução onde o evento a_2 é gerado e a saída correspondente Q_2 for energizada, ocorrer uma mudança no sinal de entrada do CLP que corresponde ao evento b_2 antes da geração do evento a_2. Assim, devido à geração do evento a_2 e um novo ciclo de leitura das entradas ser realizado somente depois disso, o programa do CLP interpretará que b_2 ocorreu após a_2, embora na prática tenha acontecido o contrário. A ilustração no lado direito da Figura 4.7 mostra a mudança de sinal nas entradas e na saída física durante 3 leituras das entradas do CLP.

Para que ocorra uma operação adequada sem esse problema, a linguagem gerada pelo supervisor precisa ser insensível ao atraso (*delay insensitive*) (BALEMI, 1992). Malik (2002) introduziu um algoritmo para verificar se o supervisor possui esta propriedade. Torna-se necessário verificar esta propriedade a fim de evitar quaisquer problemas de funcionamento do supervisor implementado devido à sensibilidade ao atraso.

Fabian e Hellgren (1998) ressaltam ainda que os problemas denominados sincronização inexata (*inexact synchronization*) e insensibilidade ao entrelaçamento (*interleave insensitivity*) diferem entre si. Enquanto este último está relacionado à incapacidade de distinguir a ordem de ocorrência de dois ou mais eventos não controláveis gerados no sistema a ser controlado, o primeiro está relacionado à incapacidade de observar a ocorrência de novos eventos não controláveis enquanto a decisão de controle é estabelecida (VIEIRA, 2007).

4.1.4 Duração dos Comandos

Este problema é apresentado em (VIEIRA, 2007) e refere-se ao tempo em que o CLP mantém suas saídas físicas ativas. Caso as saídas físicas sejam ativadas durante um período de

tempo muito curto, como por exemplo, somente um ciclo de execução, esta limitação pode impedir que os elementos de atuação do sistema sejam sensibilizados ou até mesmo que essas saídas sejam detectadas por outro dispositivo de controle. Por outro lado, se a duração desse sinal for excessivamente longa, o elemento de atuação ou outro dispositivo de controle pode interpretar isso como a ocorrência de múltiplos eventos.

A metodologia aqui proposta considera que a saída física referente à ocorrência de um evento controlável de uma determinada subplanta permanece ativa pelo período em que este evento permanecer ativo. Assim, a desativação desse evento se dará no instante em que ocorrer uma evolução de estado nessa subplanta.

4.1.5 Detalhamento da Abstração

Conforme (VIEIRA, 2007), um importante aspecto não abordado na literatura refere-se à abstração aplicada na modelagem do comportamento do sistema a ser controlado. Ele enfatiza que o modelo considera apenas parte dos eventos que podem ocorrer efetivamente, sendo considerados geralmente apenas os eventos necessários para coordenar o comportamento concorrente das diversas subplantas.

Atividades realizadas por uma determinada subplanta denominadas por seqüências operacionais (DE QUEIROZ e CURY, 2002) são um exemplo típico disso. Embora necessárias para caracterizar rotinas de baixo nível intrínsecas a determinadas subplantas, elas não aparecem durante o modelamento do sistema por meio de autômatos. Essa decisão depende do nível de complexidade em que o projetista pretende modelar o sistema, pois isso evita que ocorra uma explosão no número de estados do modelo e do supervisor durante a síntese do mesmo.

A metodologia aqui proposta não se preocupa também com a modelagem dos eventos relativos a falhas que possam por ventura ocorrer no sistema. Diversos estudos vêm sendo realizados na área de diagnóstico de falhas de SEDs, destacando-se o trabalho de ZAD *et al* (2003).

4.2 Implementação de Máquinas de Estados com Saídas em CLP: Abordagem Monolítica

O procedimento de implementação de máquinas de estados de acordo com a abordagem monolítica utiliza como base os modelos das máquinas de *Mealy* e de *Moore* reduzidas obtidas no capítulo anterior (seções 3.5.3 e 3.5.5), onde os supervisores encontrados

para o sistema de manufatura são implementados. Os detalhes da implementação em linguagem *Ladder* para CLPs são descritos nesta seção.

4.2.1 Implementação da Máquina de *Mealy* na Abordagem Monolítica

Com a transformação do autômato do supervisor em uma máquina de *Mealy*, obtém-se um modelo similar a estrutura de controle apresentada por Malik (2002), mostrada na Figura 4.1-b. Sendo assim, a geração da lógica de controle para o CLP fica mais intuitiva do que aquela obtida diretamente a partir de autômatos.

O ciclo de execução do programa de um CLP obedece ao seguinte funcionamento: leitura das entradas, execução da lógica de controle e escrita nas saídas. Esta característica síncrona do CLP obriga que as saídas sejam atualizadas somente no fim do ciclo de execução. Devido a isso, a ativação dos atuadores requer um tratamento especial. Olhando para a estrutura das máquinas de estados, pode acontecer que ao final de operação, um aparelho (ou subsistema, ou subplanta, ou simplesmente planta) pode ser requisitado para um novo início de operação dentro do mesmo ciclo de execução no qual sua operação foi finalizada. Caso isso aconteça, durante a execução da lógica de controle, essa variável vai assumir nível lógico baixo (fim de operação) e voltar ao nível lógico alto (início de operação). Entretanto, como o CLP escreve os valores de sua memória interna nas saídas físicas somente no fim do seu ciclo de execução, ele acaba não reconhecendo o processo de fim/início de operação, tal que sua saída física será mantida ativa o tempo inteiro ao longo do mesmo ciclo de execução. A fim de evitar isso, adicionam-se variáveis para representar a evolução das plantas e garantir o sincronismo durante a dinâmica do sistema. Estas variáveis são chamadas *Planta i*, com *i* variando de 1 até *n* onde *n* é o número de plantas presentes no sistema global. Esta variável é habilitada toda vez que um aparelho termina sua operação. Este procedimento garante que o aparelho não seja requisitado para iniciar sua operação novamente dentro de um mesmo ciclo de execução em que tenha sido detectado o fim de sua operação. Ele será ligado novamente somente no próximo ciclo de execução.

Assim como no modelo, as variáveis b_i representam os eventos não-controláveis (transições) enquanto as variáveis a_i (ações) representam os eventos controláveis na implementação.

O código *Ladder* pode ser dividido em cinco blocos que são chamados por um bloco de organização principal na seguinte ordem: Inicialização, Entradas, Transições/Ações, Desabilitações e Saídas, de acordo com a máquina de *Mealy* reduzida obtida na abordagem monolítica e ilustrada na Figura 3.14.

Inicialização: inicializa a máquina de estados para seu estado inicial e habilita o evento controlável a_1 a fim de iniciar o processo, como mostrado na Figura 4.8. Outras variáveis, tais como os eventos não-controláveis b_i e a evolução das plantas *Planta i*, são desativadas.

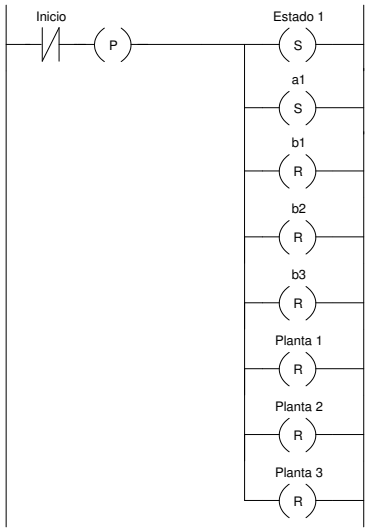


Figura 4.8 – Bloco de Inicialização da Máquina de *Mealy* na Abordagem Monolítica

Entradas: as variáveis de transição b_i serão ativadas no controlador somente quando for identificada uma borda de subida nas entradas correspondentes. Assim, um detector de pulso é necessário para cada entrada a fim de sinalizar o evento correspondente, como mostra a Figura 4.9. As variáveis relativas aos eventos não controláveis são ativadas nesse bloco. Durante a execução do programa, caso forem responsáveis por alguma transição onde alguma ação seja executada, essas variáveis serão desativadas. Caso forem responsáveis apenas por *self-loops* onde não ocorre execução de qualquer ação, sendo representados por linhas tracejadas nas máquinas de estados, essas variáveis permanecem ativas até que alguma transição com execução de ação ocorra posteriormente.

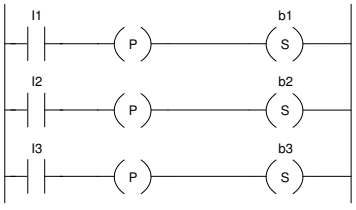


Figura 4.9 – Bloco de Entradas da Máquina de *Mealy* na Abordagem Monolítica

Transições/Ações: o requisito para uma transição ocorrer é a máquina estar em um dado estado e um ou mais eventos não controláveis que permitam a execução de alguma ação ocorrerem. Caso estes requisitos sejam atendidos, um novo estado é ativado e o estado

anterior é desativado. Os eventos não controláveis responsáveis pela transição são desativados para evitar a ocorrência do Efeito Avalanche (FABIAN e HELLGREN, 1998). Caso um evento não controlável (fim de operação de um aparelho) ocorra, a variável correspondente a essa planta, *Planta i*, é ativada a fim de evitar que a ação correspondente ao início de operação daquele aparelho aconteça durante o mesmo ciclo de execução, sendo tratada somente no próximo ciclo de execução. Isto é devido ao tratamento especial necessário para as ações, como descrito previamente nesta seção. As ações, se houver alguma, serão ativadas para permitir que a planta correspondente evolua no mesmo ciclo de execução ou somente no próximo ciclo se proibidas de evoluir no ciclo corrente. A Figura 4.10 mostra esse bloco para o sistema de manufatura. Observe a representação através da máquina reduzida ilustrada na Figura 3.14 para efeito de comparação do modelo teórico com a implementação prática. Considere por exemplo que a máquina esteja no estado *3*. Aqui, duas transições são possíveis: devido aos eventos b_1 & b_3 ou devido somente ao evento b_1. A transição devido aos eventos b_1 & b_3 deve aparecer sempre antes no diagrama *Ladder* devido a esse problema de simultaneidade (FABIAN e HELLGREN, 1998). Isso porque caso os eventos b_1 & b_3 ocorrerem e a transição devido somente ao evento b_1 estiver implementada antes, essa transição será executada e a variável b_1 será desativada. Assim, a transição devido aos eventos b_1 & b_3, por sua vez, nunca acontecerá na prática.

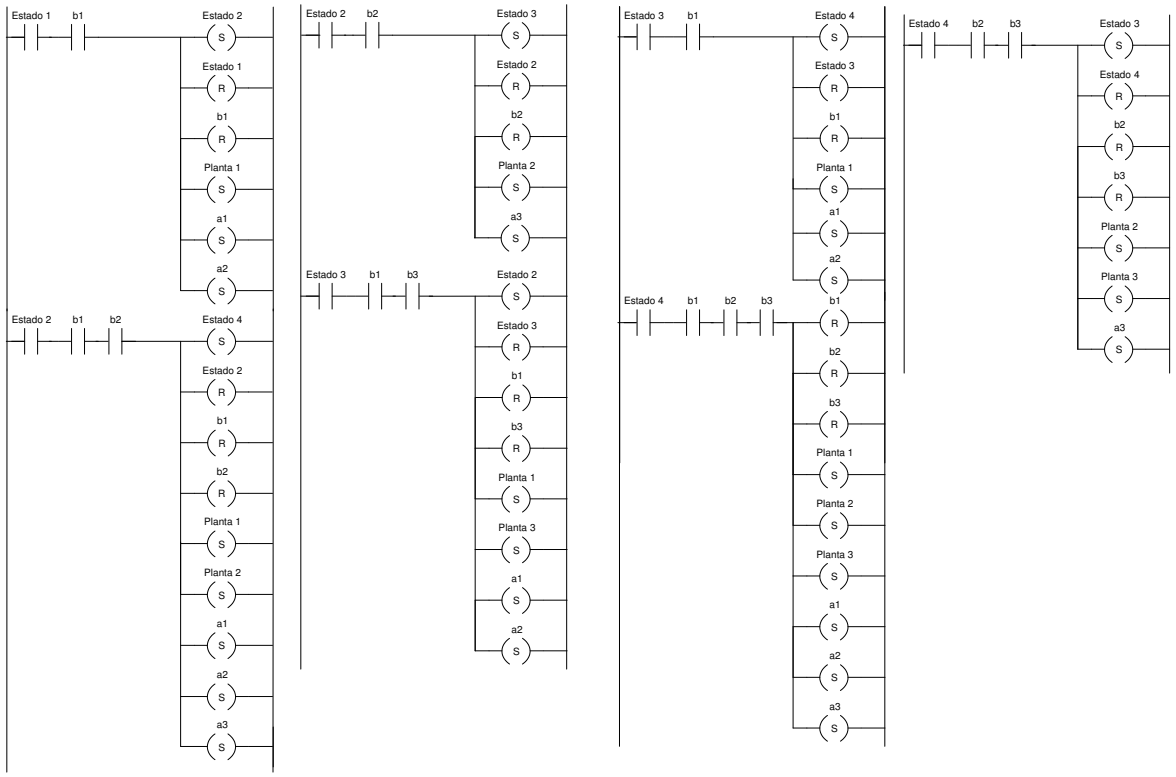


Figura 4.10 – Bloco de Transições/Ações da Máquina de *Mealy* na Abordagem Monolítica

Considere agora que a máquina se encontra no estado *4* e a transição b_1 & b_2 & b_3 ocorre. A máquina permanece no mesmo estado e, como todos os aparelhos terminam sua operação, nenhuma ação será executada no ciclo de execução corrente. Entretanto, como as ações a_1 & a_2 & a_3 são ativadas, no próximo ciclo de execução do CLP todos os aparelhos serão ligados novamente. Assim, essa metodologia garante que é necessário esperar apenas um ciclo de execução para que todas as ações habilitadas de ocorrer sejam executadas.

Desabilitações: este bloco é responsável por desabilitar as ações de controle em determinados estados da máquina. Isso significa que se a máquina alcançar um dado estado, as ações relativas a este estado são proibidas de ocorrer, sendo desabilitadas. A Figura 4.11 mostra o bloco de desabilitações para o exemplo analisado. Os dois primeiros degraus do código *Ladder* representam os estados onde as ações de controle correspondentes são desabilitadas. Além disso, as desabilitações podem ocorrer caso algum evento não controlável esteja sinalizado mas não possibilita a execução de ação (representado por um *self-loop* com linhas tracejadas na máquina de estados reduzida), tal que o início de operação da planta correspondente é proibido. Assim, para que isso seja considerado no diagrama *Ladder*, basta desabilitar a ação relativa a uma determinada *Planta i* quando o evento controlável b_i estiver sinalizado, pois ele não gerou transição na máquina de estados e por conseqüência não foi desativado, como mostram os demais degraus na Figura 4.11.

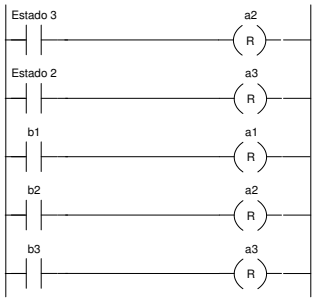


Figura 4.11 – Bloco de Desabilitações da Máquina de *Mealy* na Abordagem Monolítica

Saídas: uma saída física é ativada somente se a ação correspondente relativa a um evento controlável for executada e sua variável correspondente *Planta i* não estiver habilitada, como mostrado na Figura 4.12. Caso isso aconteça, a bobina Q_i que representa a saída física do CLP será energizada. Ainda, no final do programa, todas as variáveis *Planta i* serão desabilitadas a fim de retornar para uma condição inicial antes de um novo ciclo de execução.

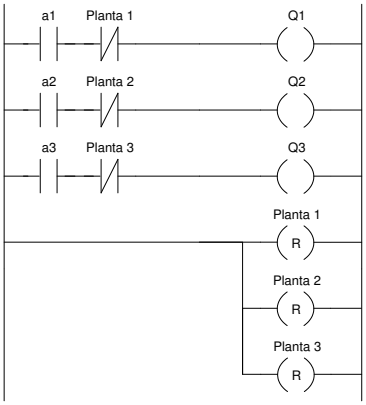


Figura 4.12 – Bloco de Saídas da Máquina de *Mealy* na Abordagem Monolítica

Essa solução tem a vantagem de não restringir que mais do que um evento não controlável seja tratado dentro do mesmo ciclo de execução além de ser um modelo simplificado de implementação.

Para melhor compreensão desta lógica de implementação de controle supervisório, considera-se primeiramente que o início de operação é dado pela execução da ação a_1. Após alguns ciclos de execução do CLP, o aparelho A_1 termina sua operação e a transição devido ao evento não controlável b_1 é gerada. De acordo com a máquina de estados reduzida, as ações a_2 & a_1 são habilitadas para ocorrer e a máquina transita para o estado *2*. Entretanto, como a *Planta 1* terminou sua operação, apenas a saída física Q_2 é ativada nesse mesmo ciclo de execução. A saída física Q_1 será ativada somente no próximo ciclo de execução do CLP. Em seguida, dependendo de qual aparelho terminar sua operação antes a máquina de estados vai possuir um comportamento diferente. Caso o aparelho A_2 termine sua operação, a transição b_2 será gerada, a ação a_3 será executada e a máquina transita para o estado *3*. Caso o aparelho A_1 termine sua operação, a transição b_1 será gerada resultando em um *self-loop* com linhas tracejadas dentro do estado *2*, pois nenhuma ação é executada. Caso o CLP leia em suas entradas mudanças de sinal que representam o fim dos aparelhos A_1 e A_2 (transições b_1 & b_2, respectivamente) então as ações a_1 & a_2 & a_3 serão executadas e a máquina de estados transita para o estado *4*. As demais transições e ações da máquina seguem essa mesma dinâmica como ilustrado no modelo de *Mealy* reduzido da Figura 3.14.

4.2.2 Implementação da Máquina de *Moore* na Abordagem Monolítica

A máquina reduzida de *Moore* para o sistema de manufatura foi obtida no capítulo anterior, sendo mostrada na Figura 3.16. O procedimento para implementação de uma máquina reduzida de *Moore* é bastante similar ao de uma máquina de *Mealy*. A diferença é que as ações e desabilitações estão presentes em um mesmo bloco. Os blocos de entradas e

saídas são exatamente iguais aos da máquina de *Mealy.* O bloco de inicialização é mostrado na Figura 4.13, o bloco de transições é mostrado na Figura 4.14 e o bloco de ações/desabilitações na Figura 4.15.

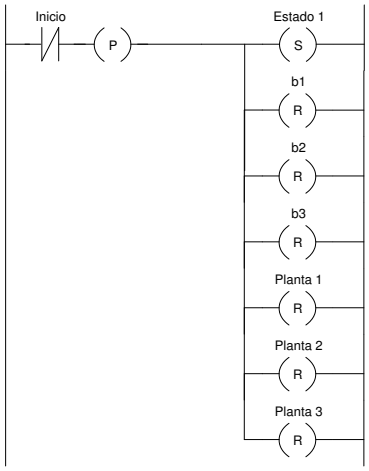


Figura 4.13 – Bloco de Inicialização da Máquina de *Moore* na Abordagem Monolítica

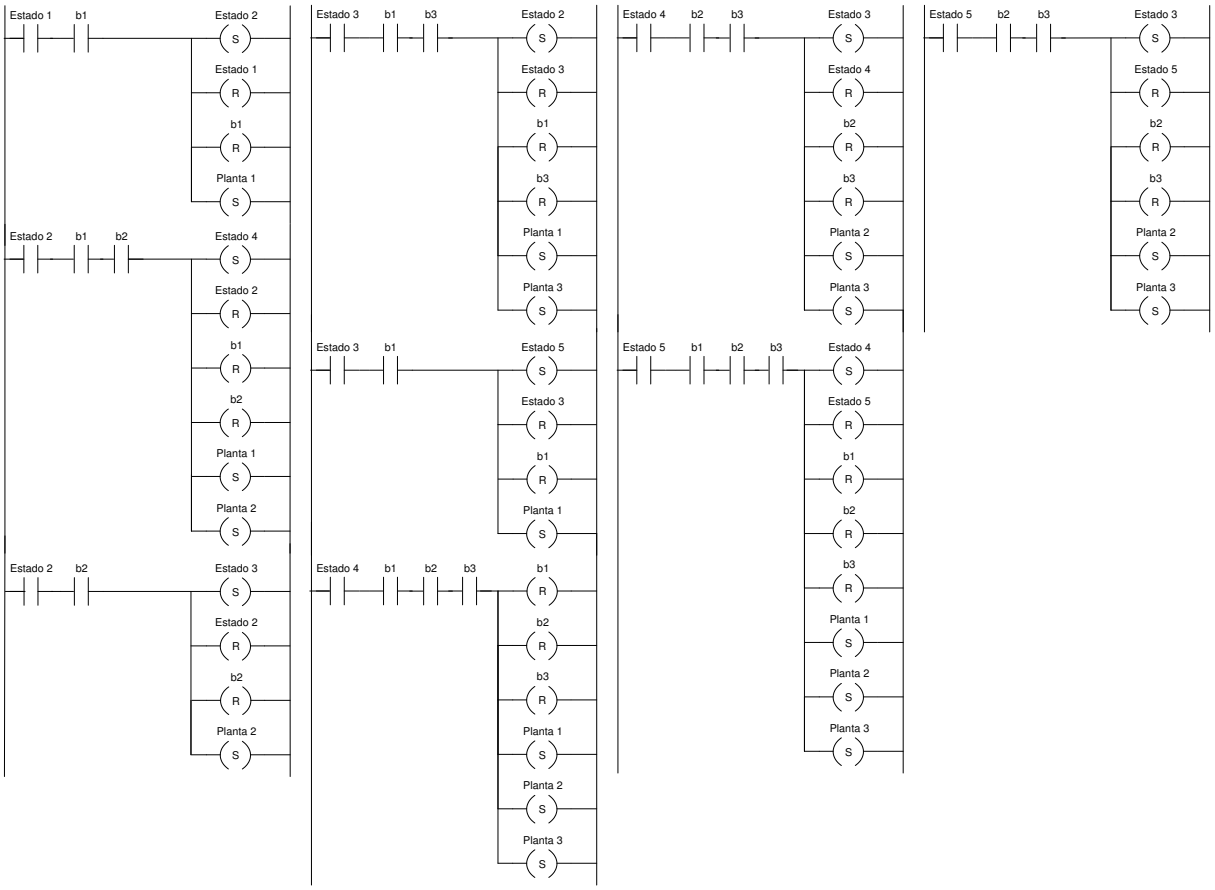


Figura 4.14 – Bloco de Transições da Máquina Monolítica de *Moore*

As ações agora são executadas quando a máquina entra em um determinado estado. Dessa forma, o bloco de inicialização não executa a ação a_1, tal como ocorre com a máquina

de *Mealy*. Essa ação é habilitada no estado *1*, quando o bloco de ações/desabilitações é executado. Além dessa ação, o bloco de ações/desabilitações será responsável também pela execução das demais ações ou pelas desabilitações das mesmas.

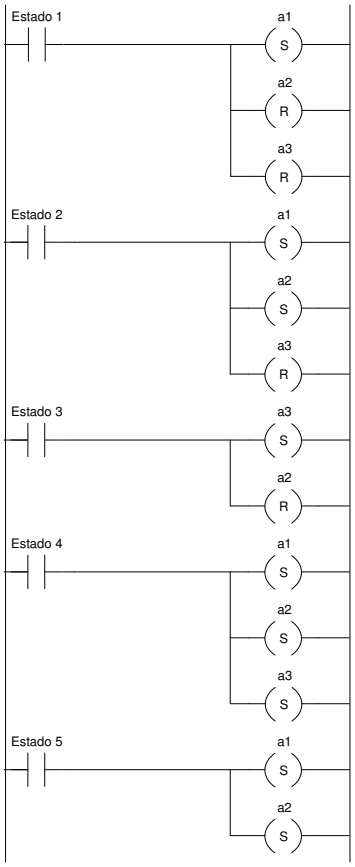


Figura 4.15 – Bloco de Ações/Desabilitações da Máquina de *Moore* na Abordagem Monolítica

Essa metodologia de implementação resulta em um tamanho de código relativamente maior do que a abordagem de *Mealy*, sendo decorrência do maior número de estados e transições do modelo. A vantagem aqui é que o bloco de transições de estados é responsável somente pelo tratamento dos eventos não controláveis. Além disso, obtém-se outro bloco específico para o tratamento dos eventos controláveis, ou seja, a habilitação e desabilitação das ações de controle. As variáveis que representam a evolução das plantas, *Planta i*, possuem exatamente a mesma função das variáveis com mesmo nome presentes na máquina de *Mealy*.

4.3 Implementação de Supervisores Modulares Locais em CLP

Nesta seção, discute-se sobre a implementação de supervisores modulares locais em linguagem *Ladder* para CLPs. Diversos trabalhos na literatura propõem esse tipo de implementação, tais como (DE QUEIROZ e CURY, 2002), (VIEIRA, 2003), (DE QUEIROZ, 2004), (CURZEL, 2008) e (LEAL *et al*., 2009). Essas propostas diferem levemente por causa do comportamento de algumas variáveis durante o ciclo de execução do CLP. Devido ao fato de ser mais difundida no meio acadêmico, a proposta de (VIEIRA, 2003) será utilizada neste trabalho como referência para comparação com a metodologia proposta usando máquinas de estados. A arquitetura usada nesta abordagem é a mesma que foi proposta em (DE QUEIROZ e CURY, 2002), ilustrada na Figura 4.16.

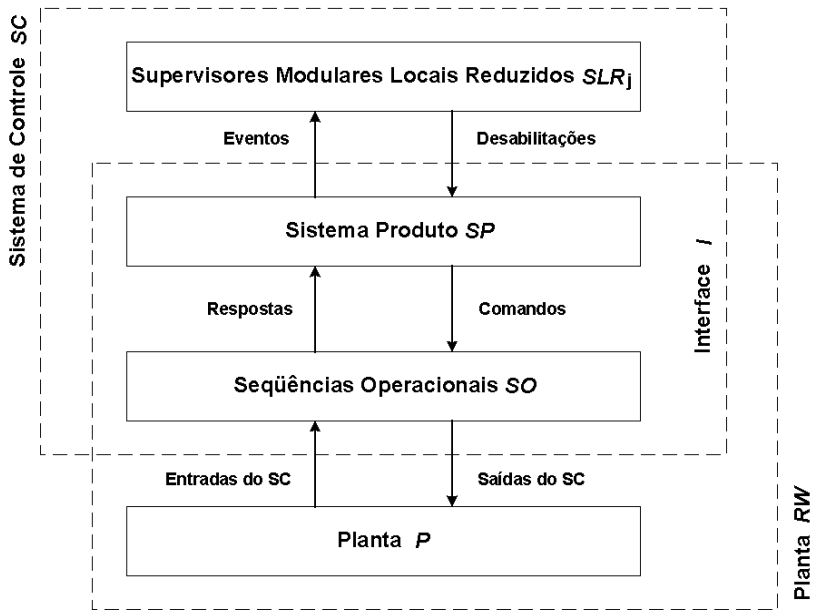


Figura 4.16 – Arquitetura de Implementação Proposta em (De Queiroz e Cury, 2002)

Esta estrutura reproduz no código os modelos abstratos das subplantas por meio do bloco denominado sistema produto *SP*. O sistema produto possui uma interface *I* entre os supervisores modulares locais reduzidos SLR_j e a planta real *P* que, como o próprio nome diz, tem a função de fazer uma interface entre o sistema de controle *SC* (o CLP nesse caso) e a planta física *P*. As saídas do sistema de controle *SC* são definidas de acordo com as informações baseadas nas desabilitações de eventos controláveis provenientes dos SLR_j e dos comandos provenientes do sistema produto *SP*. Assim, para cada evento controlável, existe uma variável de comando e uma variável de desabilitação associadas a ele. Esse evento controlável só pode ser gerado se a variável de desabilitação correspondente não estiver

ativada (VIEIRA, 2007). As saídas do sistema de controle *SC* correspondem portanto aos eventos controláveis e são interpretadas pelo sistema produto *SP* como comandos para a planta real. Os comandos são os sinais responsáveis por acionar as saídas físicas do CLP, relativas ao acionamento dos atuadores da planta. A tradução da variável de comando para sinais reais de saída do CLP é feita por intermédio das seqüências operacionais *SO* de cada subplanta.

As seqüências operacionais *SO* são responsáveis pelas seqüências lógicas que definem a operação de cada subsistema. As rotinas de baixo nível relativas a um determinado subsistema que não foram modeladas por autômatos (como por exemplo, atividades que uma determinada subplanta deve executar após receber um comando para início de sua operação) são tratadas nesse nível. Vale enfatizar que o tratamento dessas rotinas de baixo nível não está em escopo nesta dissertação, sendo levados em conta apenas o evento de início e fim de operação de uma determinada subplanta.

Além disso, as seqüências operacionais *SO* são responsáveis também pelos sinais de entradas para o CLP, relativos a leitura dos sensores presentes na planta. Essas entradas para o sistema de controle *SC* correspondem aos eventos não controláveis e são interpretadas pelo sistema produto *SP* como respostas da planta real. Para cada evento não controlável existe uma variável de resposta associada a ele, sendo sinalizada quando a seqüência operacional *SO* identificar a ocorrência de um evento não controlável. Essa variável de resposta é responsável pela evolução do subsistema correspondente no bloco de sistema produto *SP*, que sinaliza a ocorrência desse evento para o bloco SLR_j , cujos supervisores modulares locais afetados também sofrem uma evolução de estados.

Vale ressaltar que essa estrutura se comporta de acordo com o esquema de supervisão assumido na TCS (RAMADGE e WONHAM, 1989), como mostrado anteriormente na Figura 4.1-a. Observe que os supervisores modulares locais reduzidos SLR_j se comportam como observadores dos eventos gerados pela planta e são responsáveis por desabilitar um conjunto de eventos controláveis para satisfazer os requisitos de controle. Sendo assim, nessa estrutura a planta *RW* é formada pelos blocos *SP*, *SO* e pela própria planta real *P*.

De acordo com De Queiroz (2004), para que seja garantida uma operação adequada desta estrutura de controle é importante considerar os seguintes aspectos:

- para que ocorra um evento controlável em *SP*, as ações dos SLR_j (ou seja, as desabilitações) devem estar atualizadas;

- os eventos não controláveis em *SP* devem ser tratados com prioridade em relação aos eventos controláveis;

- a sinalização da ocorrência de eventos para os SLR_j deve ter um comportamento seqüencial.

A estrutura de implementação considera os supervisores modulares locais reduzidos SLR_j apresentados na Figura 3.21. Um bloco de organização é responsável por chamar os blocos SLR_j, *SP* e *SO*.

O bloco em *Ladder* que corresponde aos supervisores modulares locais reduzidos SLR_j e às desabilitações é mostrado na Figura 4.17. Os supervisores precisam ser inicializados para seus estados iniciais. A transição de estados dos supervisores é realizada por meio de uma máquina de estados. Além disso, as desabilitações devem ser sinalizadas a fim de indicar ao sistema produto *SP* quais eventos controláveis estão habilitados a ocorrer.

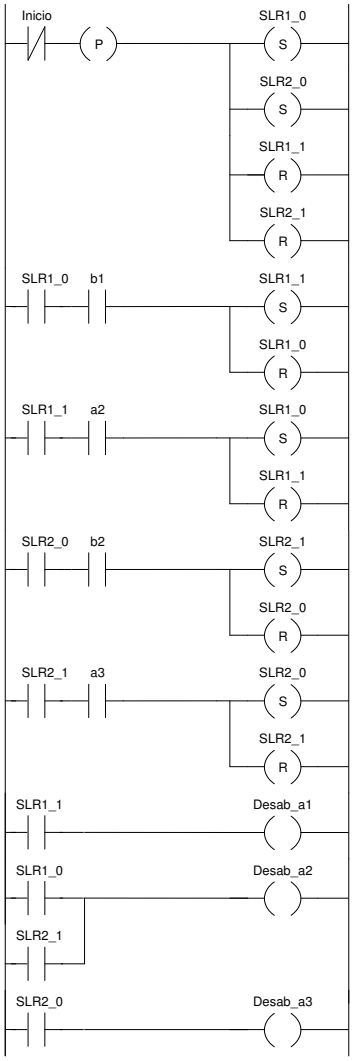


Figura 4.17 – Bloco dos Supervisores Modulares Locais Reduzidos SLR_j e Desabilitações

A Figura 4.18 descreve o bloco do sistema produto *SP*. A transição de estados das subplantas também é realizada por meio de uma máquina de estados. Após a inicialização das

subplantas para seus estados iniciais, o programa deve priorizar a ocorrência dos eventos não controláveis. Assim, os degraus relativos à ocorrência de eventos não controláveis são implementados antes dos degraus relativos aos eventos controláveis. Para que um evento não controlável seja sinalizado, é necessário que a variável b_i*_resposta* esteja ativa e a planta ainda não tenha evoluído naquele ciclo de execução do CLP. Quando a planta evolui, a variável *ev_plt* é habilitada, tal que nenhum outro evento, seja não controlável ou controlável, possa ser executado naquele mesmo ciclo de execução. Isso consiste em uma limitação dessa abordagem, pois ela permite a ocorrência de apenas um evento por ciclo de execução. Por outro lado, isso impede que o problema devido ao efeito avalanche ocorra.

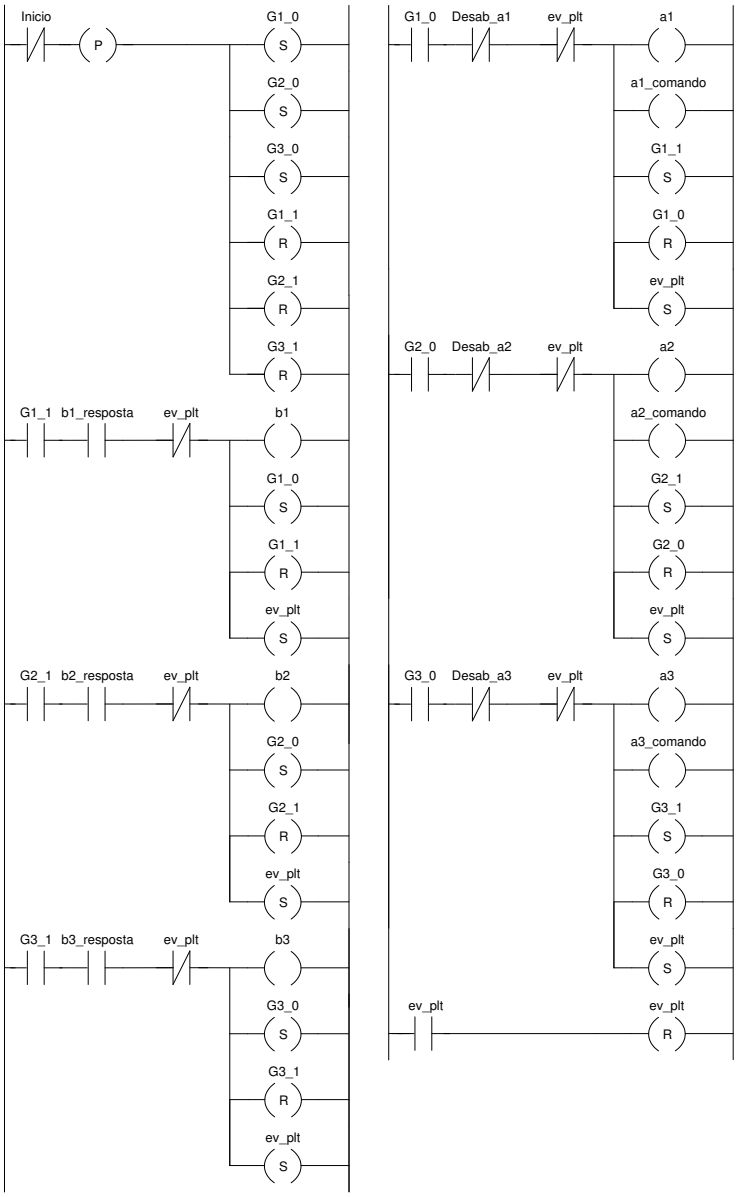


Figura 4.18 – Bloco do Sistema Produto *SP*

Caso a planta esteja habilitada a evoluir devido a um evento não controlável, o evento b_i será sinalizado para o bloco SLR_j. Para que a planta esteja habilitada a evoluir devido a um evento controlável, é necessário que a variável *ev_plt* esteja desabilitada assim como o evento controlável no estado em questão da subplanta correspondente. Caso isso seja verificado, o evento a_i será sinalizado para o bloco SLR_j e o comando a_i*_comando* será sinalizado para o bloco *SO*. Ao final do bloco *SP*, a variável *ev_plt* é desabilitada para representar uma condição inicial para o próximo ciclo de execução.

O último bloco presente nesta topologia é o bloco de seqüências operacionais *SO*. Neste bloco, os comandos a_i*_comando* são responsáveis por acionar as saídas físicas Q_i. Em relação às entradas, quando uma mudança de sinal for detectada em alguma entrada física do CLP I_i, a saída física Q_i correspondente será desativada e a variável b_i*_resposta* será enviada ao bloco *SP* para sinalizar a ocorrência de um evento não controlável.

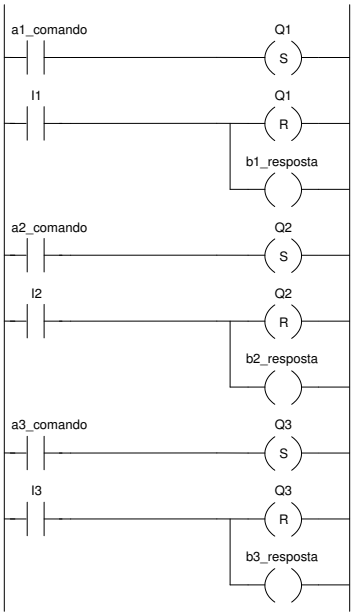


Figura 4.19 – Bloco de Seqüências Operacionais *SO*

4.4 Implementação de Máquinas de Estados com Saídas em CLP: Abordagem Modular Local

A implementação da lógica de controle modular local em CLP é realizada a partir do modelo reduzido obtido no capítulo anterior, representado pela Figura 3.24. Da mesma forma que proposto na abordagem monolítica, torna-se necessário utilizar variáveis denominadas *Planta i*, com *i* variando de 1 até *n* onde *n* é o número de plantas no sistema, para representar a evolução dos subsistemas e garantir o sincronismo durante a dinâmica do sistema.

Além disso, agora não é possível desativar as variáveis que representam eventos não controláveis quando alguma transição devido a eles ocorre, assim como proposto para a abordagem monolítica. Isso porque existem diversas máquinas modulares e se um determinado evento for desativado em uma das máquinas, ele não poderá mais ser responsável por gerar transições nas demais máquinas presentes no sistema dentro de um mesmo ciclo de execução. Outra abordagem é necessária para evitar o problema de efeito avalanche. Assim, uma alternativa seria criar uma variável específica para sinalizar cada evento não controlável em cada uma das máquinas modulares. Entretanto, essa não seria uma prática muito elegante e eficaz, pois diversas variáveis seriam criadas no código e, para sistemas de grande porte, a tendência é que a memória ocupada do CLP seria significativa devido à quantidade de variáveis existentes. A solução proposta neste trabalho consiste em criar variáveis que sinalizam quando as máquinas modulares evoluem, isto é, alguma transição ocorre durante o ciclo de execução. Essas variáveis são denominadas MLS_j_evol. Dessa forma, quando uma transição ocorre na máquina MLS_j, a variável MLS_j_evol é ativada tal que nenhuma transição será mais permitida nessa máquina durante esse ciclo de execução, evitando assim o problema de efeito avalanche. Ao fim da execução das transições nas máquinas, as variáveis MLS_j_evol correspondentes serão desativadas para permitir a ocorrência de transições no próximo ciclo de execução.

As variáveis MLS_j_evol têm um comportamento similar à variável utilizada nos supervisores modulares locais, denominada *ev_plt*. Entretanto, enquanto a primeira permite a evolução de apenas uma transição (que pode conter mais do que um evento não controlável) por máquina modular para cada ciclo de execução do CLP, a segunda permite a evolução de apenas um evento (não controlável ou controlável) para o sistema todo em um ciclo de execução do CLP.

Assim como apresentado no modelo do sistema de manufatura, as variáveis b_i representam os eventos não-controláveis (transições) enquanto as variáveis a_i (ações)

representam os eventos controláveis. A representação dos estados na abordagem modular local é a seguinte: um estado denominado *Estado 1-0* significa que se trata da máquina modular local MLS_1 em seu estado *0* enquanto *Estado 2-1* significa que se trata da máquina modular local MLS_2 em seu estado *1*, por exemplo.

Cinco blocos são criados, sendo chamados por um bloco de organização principal na seguinte ordem: Inicialização, Entradas, Transições, Desabilitações/Ações e Saídas. Os blocos são descritos em detalhes a seguir.

Inicialização: bloco utilizado para inicializar todas as máquinas de estados da abordagem modular local, como mostrado na Figura 4.20. Para o sistema de manufatura em estudo, o *Estado 1-0* é ativado na máquina MLS_1 e o *Estado 2-0* é ativado na máquina MLS_2. As variáveis b_i que representam os eventos, as variáveis *Planta i* que representam a evolução dos subsistemas e as variáveis MLS_j_evol que representam a evolução das máquinas locais do sistema, são desativadas.

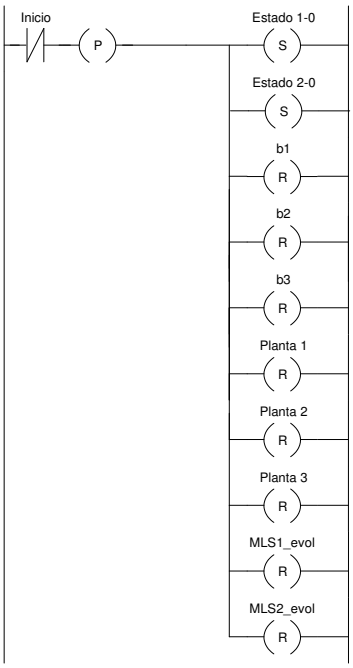


Figura 4.20 – Bloco de Inicialização da Máquina Modular Local

Entradas: bloco igual ao utilizado na abordagem monolítica, onde as variáveis de transição b_i serão sinalizadas somente quando for identificada uma borda de subida nas entradas correspondentes do CLP tal que um detector de pulso é necessário para cada entrada, a fim de capturar o evento correspondente, como ilustrado na Figura 4.21.

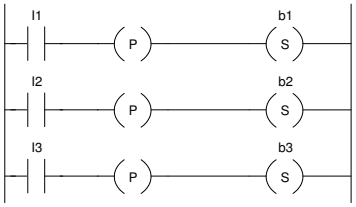


Figura 4.21 – Bloco de Entradas da Máquina Modular Local

Transições: cada máquina local possui um bloco de transições. Caso uma transição ocorra em uma máquina local, a variável *MLS$_j$_evol* correspondente será ativada para informar que essa máquina local evoluiu, tal que nenhuma outra transição será permitida de ocorrer para aquela máquina local no ciclo de execução em questão. Esse procedimento permite que apenas uma transição ocorra por máquina local dentro de um mesmo ciclo de execução do CLP. A Figura 4.22 mostra esse bloco para o sistema de manufatura. Assim como na abordagem monolítica, as transições devido à sinalização de mais do que um evento (b_1 & b_2, partindo do estado *1-1* da máquina *MLS$_1$*, por exemplo) devem aparecer antes do que as transições devido a somente um evento (b_2, também partindo do estado *1-1* da máquina *MLS$_1$*) a fim de evitar o problema de simultaneidade já descrito anteriormente. Após a ocorrência de uma transição, as ações correspondentes serão ativadas, energizando as saídas físicas do CLP no mesmo ciclo de execução ou somente no próximo ciclo, caso a ação ocorra devido ao término da planta correspondente, assim como acontece na abordagem monolítica.

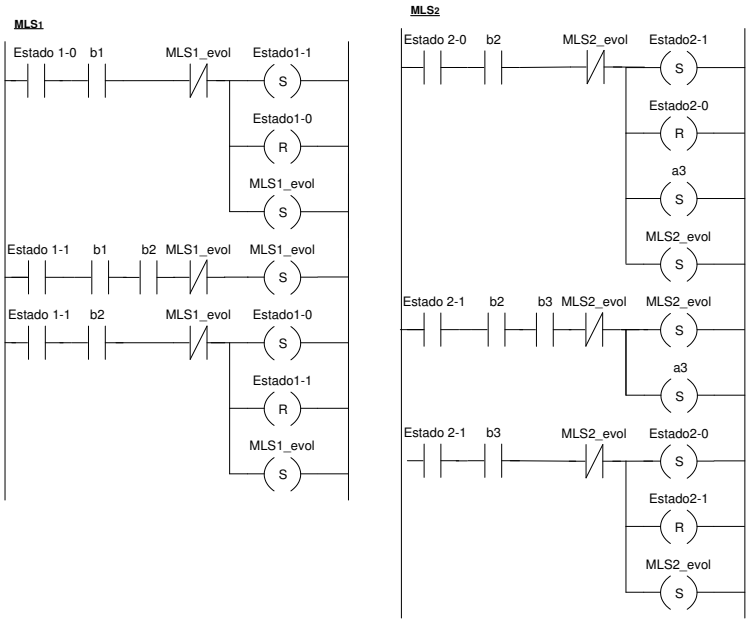


Figura 4.22 – Bloco de Transições da Máquina Modular Local

Ações/Desabilitações: este bloco é o responsável pelos intertravamentos do sistema. Inicialmente, o programa verifica quais ações podem ocorrer em um determinado estado, como ilustrado na Figura 4.23.

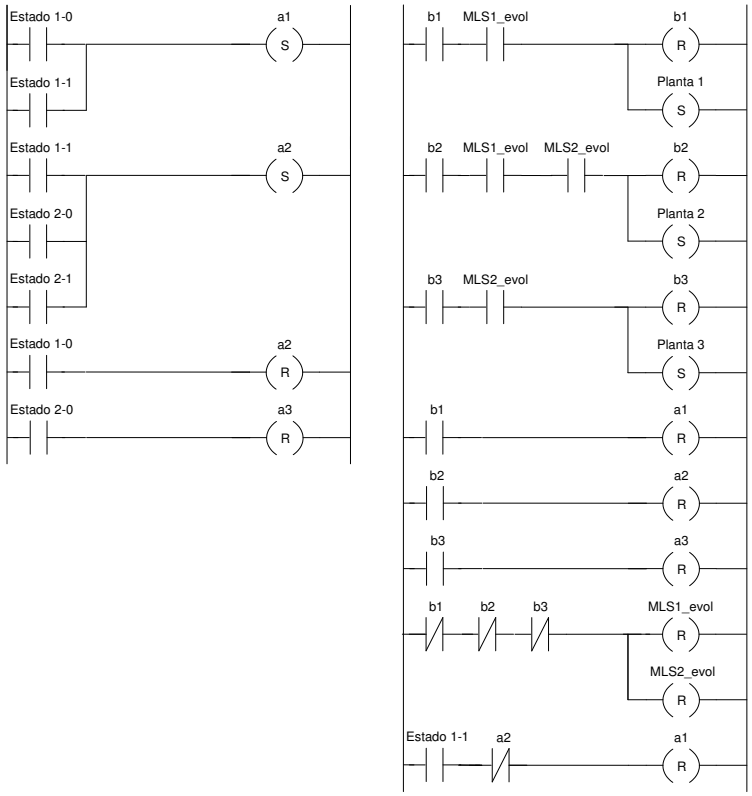


Figura 4.23 – Bloco de Ações/Desabilitações da Máquina Modular Local

Em seguida, são verificadas as desabilitações para cada estado de cada máquina local. Dessa forma, este bloco é responsável primeiramente por habilitar as ações resultantes de *self-loops* dentro de um mesmo estado e, em seguida, efetuar as desabilitações de ações conforme o modelo reduzido obtido. Assim, as ações que permanecerem habilitadas no fim da execução dessa lógica de controle serão as responsáveis por energizar as saídas físicas do CLP no bloco de saídas, desde que a planta correspondente não tenha evoluído nesse mesmo ciclo de execução.

O próximo passo consiste em verificar quais eventos não controláveis ocasionaram transições de estados nas máquinas locais, a fim de desativá-los e ativar as plantas correspondentes. Essa idéia é ilustrada no lado direito da Figura 4.23. Observe que, como o evento b_2 é comum para ambas as máquinas MLS_1 e MLS_2, ele será desativado somente se causar transições em ambas as máquinas. Caso contrário significa que, ou ele não ocorreu naquele ciclo de execução ou ele causou transição em apenas uma das máquinas. Considere a segunda situação. Nesse caso, o que vai aparecer no modelo é um *self-loop* representado por

linhas tracejadas, como acontece, por exemplo, no estado *2-1* da máquina MLS_2 mostrada anteriormente na Figura 3.24. Seguindo esse raciocínio e analisando a continuação do código em *Ladder* na Figura 4.23, os eventos que não foram desativados serão responsáveis por desativar o início de operação da subplanta equivalente. Isso ocorre porque a variável equivalente *Planta i* não foi ativada devido à condição anterior a essa não ter sido válida e, de alguma forma, torna-se necessário evitar que a saída equivalente Q_i seja energizada. Em outras palavras, um *self-loop* representado por linhas tracejadas aguarda a ocorrência de outro evento com o qual existe uma condição *AND* (*&*) para permitir que determinadas ações sejam executadas no sistema. Quando todas as máquinas locais tiverem executado transições, então as variáveis MLS_j_evol serão desativadas novamente para que novas transições aconteçam. Essa condição garante que todos os eventos estarão desativados, como mostrado na Figura 4.23. Esse procedimento é necessário para manter o sincronismo entre as máquinas.

Finalmente, observe o caso particular em que a ação a_1 será desabilitada se a ação a_2 estiver desabilitada também. Esse procedimento reflete o modelo da máquina de estados reduzida MLS_1 mostrada na Figura 3.24 onde a cadeia a_2 & a_1 será permitida de ocorrer somente se a ação a_2 estiver habilitada.

Saídas: o bloco de saídas é exatamente igual ao da abordagem monolítica, onde uma saída será permitida de ocorrer somente se a ação correspondente for executada e sua variável correspondente *Planta i* não estiver habilitada, como mostrado na Figura 4.24.

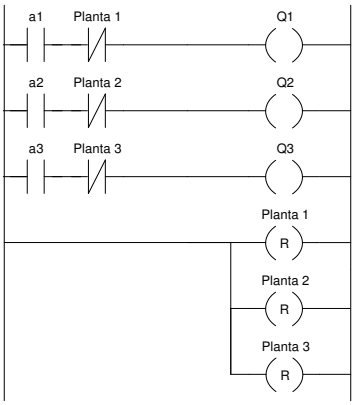


Figura 4.24 – Bloco de Saídas da Máquina Modular Local

Com o intuito de facilitar a compreensão dessa metodologia, considere novamente a máquina reduzida na abordagem modular local mostrada na Figura 3.24. O início do processo acontece com a ativação da ação a_1. Quando o aparelho A_1 termina a operação, o evento b_1 faz com que a máquina local MLS_1 transite para o estado *1-1* enquanto a máquina local MLS_2 permanece no estado *2-0*. Dessa forma, a variável MLS_1_evol é ativada, mas a variável

MLS₂_evol não, pois nenhum evento relativo a essa máquina ocorreu. No início do bloco de *Ações/Desabilitações*, as ações a_1 e a_2 são ativadas, enquanto a ação a_3 permanece desativada. Em seguida, como o evento b_1 aconteceu e a variável *MLS₁_evol* foi ativada, o evento b_1 é então desativado e a variável *Planta 1* é ativada, para impedir que a saída Q_1 seja energizada nesse mesmo ciclo de execução. Como agora todos os eventos estão desativados, as variáveis *MLS₁_evol* e *MLS₂_evol* são desativadas também, permitindo que todas as máquinas locais transitem no próximo ciclo de execução.

Considere agora que após alguns ciclos de execução a transição b_1 & b_2 aconteça. A máquina *MLS₁* permanece no estado *1-1* e a variável *MLS₁_evol* é ativada, enquanto a máquina *MLS₂* transita para o estado *2-1* devido ao evento b_2 e ativa a ação a_3. Ao analisar o bloco de *Ações/Desabilitações* da Figura 4.23, verifica-se que as ações a_1 e a_2 permanecem ativadas. Como ambas as máquinas evoluem, os eventos b_1 e b_2 são desativados e as variáveis *Planta 1* e *Planta 2* correspondentes serão ativadas. Esse comportamento é conforme o esperado, pois essas plantas não podem iniciar uma nova operação dentro desse mesmo ciclo de execução. Por fim, como todos os eventos foram desativados, as variáveis *MLS₁_evol* e *MLS₂_evol* são desativadas novamente. Nesse ciclo, somente a saída Q_3 será energizada, devido à ação a_3 estar ativada e à variável *Planta 3* estar desativada. As saídas Q_1 e Q_2 serão energizadas somente no próximo ciclo de execução, pois as variáveis *Planta 1* e *Planta 2* serão desativas ao final desse ciclo de execução.

Estando agora as máquinas em seus estados *1-1* e *2-1*, respectivamente, suponha que após mais alguns ciclos de execução o CLP detecte que os eventos b_1 e b_2 aconteçam novamente. A máquina local *MLS₁* permanece no estado *1-1* e a variável *MLS₁_evol* é ativada novamente, enquanto a máquina *MLS₂* não sofre nenhuma transição, pois a única transição permitida de ocorrer em seu estado *2-1* com o evento b_2 consiste na condição b_2 & b_3. Assim a variável *MLS₂_evol* permanece desativada e um *self-loop* representado por linhas tracejadas aparece em seu estado *2-1*. Como somente a variável *MLS₁_evol* foi ativada, apenas o evento b_1 é desativado no bloco de *Ações/Desabilitações*. Como o evento b_2 permanece ativo, o programa aguarda o evento b_3 ocorrer para efetuar uma transição na máquina *MLS₂*. Enquanto isso não acontecer, a máquina *MLS₁* estará impedida de transitar, pois a variável *MLS₁_evol* permanece ativa. Esse procedimento é adotado para garantir o sincronismo entre as máquinas. Quando o evento b_3 ocorre, a máquina *MLS₂* transita e a variável *MLS₂_evol* é ativada. Assim, todos os eventos são desativados tal que as variáveis *MLS₁_evol* e *MLS₂_evol* são desativadas também, permitindo que novas transições aconteçam na seqüência da execução do programa.

4.5 Conclusões

Este capítulo considera inicialmente os problemas encontrados no processo de implementação de supervisores em controladores síncronos, como os Controladores Lógicos Programáveis (CLPs), e descreve soluções para esses problemas. Em seguida, é proposta uma lógica para implementação de controle supervisório em linguagem *Ladder* para CLPs baseada nos modelos reduzidos das máquinas de estados com saídas, tanto de caráter monolítico quanto modular local, obtidas com a metodologia proposta no capítulo anterior. Embora o CLP seja caracterizado construtivamente como uma máquina de *Moore*, verifica-se que a concepção da estrutura de código por meio de máquina de *Mealy* pode ser mais conveniente quando se deseja uma menor quantidade de estados e transições. Vale ressaltar que, independentemente se o código for realizado por meio de máquina de *Mealy* ou de *Moore*, o CLP vai atualizar todas as suas saídas ao mesmo tempo, caracterizando fisicamente uma máquina de *Moore*.

Uma topologia de implementação de supervisores modulares locais (VIEIRA, 2003) de acordo com a estrutura proposta em (DE QUEIROZ e CURY, 2002) também é mostrada, com o intuito de se ter uma base comparativa para com as metodologias aqui propostas.

Embora as máquinas modulares locais tendem a reduzir o modelo encontrado, uma desvantagem dessa abordagem se comparado com a proposta em (DE QUEIROZ e CURY, 2002) consiste na ausência dos modelos das subplantas na implementação final. Dessa forma, existe a necessidade de refazer o procedimento de síntese para todas as máquinas caso alguma especificação de controle mude, o que não acontece na estrutura proposta em (DE QUEIROZ e CURY, 2002) onde somente o supervisor que sofreu a alteração precisa ser sintetizado novamente. Entretanto, isso é visualizado como uma vantagem quando o objetivo consiste em se ter um tamanho de código da implementação mais enxuto, resultando em um menor número de variáveis declaradas. Além disso, a visualização da lógica de controle é muito mais simples, onde o diagrama *Ladder* pode ser facilmente comparado com os modelos das máquinas de estados reduzidas.

Este procedimento de implementação já considera a resolução de alguns dos problemas relativos à implementação de supervisores. Além disso, outra vantagem consiste na possibilidade de tratar diversos eventos dentro um mesmo ciclo de execução do CLP, sendo eles controláveis ou não controláveis.

No intuito de avaliar a metodologia proposta, no próximo capítulo é apresentado um problema de controle supervisório para um sistema de maior porte, uma célula de manufatura

para furação de peças. A modelagem por máquinas de estados é realizada tanto para a abordagem monolítica quanto para a abordagem modular local. Por fim, um comparativo geral entre a implementação dessas metodologias e a metodologia proposta em (VIEIRA, 2003) é realizado a fim de se avaliar os pontos positivos e negativos das propostas apresentadas neste trabalho.

Capítulo 5

Projeto do Controlador de uma Célula de Manufatura

Neste capítulo a proposta de projeto de supervisores para Controladores Lógicos Programáveis (CLPs) usando máquinas de estados é aplicada em um sistema de maior porte. Essa planta consiste em uma célula de manufatura responsável pela furação de peças. Esse mesmo sistema já foi abordado em outros trabalhos na literatura, como em (DE QUEIROZ e CURY, 2002), (DE QUEIROZ, 2004), (VIEIRA, 2007) e (MOURA, 2009). O objetivo deste capítulo é demonstrar a metodologia de projeto proposta para modelagem e implementação de máquinas de estados reduzidas para um sistema maior e mais complexo do que o exemplo de sistema de manufatura apresentado no Capítulo 3.

A redução dos modelos dos autômatos que representam os supervisores obtidos pela Teoria de Controle Supervisório (TCS), transformando-os em máquinas de estados que são posteriormente reduzidas a fim de filtrar ações de controle redundantes, tem como objetivo principal a obtenção de modelos mais simples para facilitar o procedimento de implementação e obter um código mais compacto. Diversos trabalhos na literatura, tais como (DE QUEIROZ, 2004), (VIEIRA, 2007) e (TEIXEIRA, 2008) propõem-se a obter supervisores reduzidos a partir da TCS a fim de diminuir a complexidade dos modelos. Os supervisores reduzidos na abordagem modular local, obtidos com o auxílio da ferramenta TCT (FENG e WONHAM, 2006), serão utilizados como critério de comparação com as máquinas de estados reduzidas obtidas para a célula de manufatura. O supervisor reduzido obtido a partir da abordagem monolítica será descrito neste capítulo, mas não será implementado, pois mesmo utilizando os algoritmos de redução, seu tamanho fica relativamente grande se comparado com as demais abordagens.

Este capítulo é organizado da seguinte forma: na Seção 5.1 define-se o problema a ser analisado, seu alfabeto de eventos, sua modelagem e as especificações de controle. Na Seção 5.2 são apresentados os supervisores e máquinas de estados obtidos de acordo com a abordagem monolítica. A Seção 5.3 apresenta os supervisores e máquinas de estados obtidos a

partir da abordagem modular local. Na Seção 5.4 é feita uma análise dos resultados para as diferentes metodologias de implementação descritas neste trabalho. Para finalizar, a Seção 5.5 trata das conclusões deste capítulo.

5.1 Célula de Manufatura

A Figura 5.1 apresenta uma visão geral da célula de manufatura.

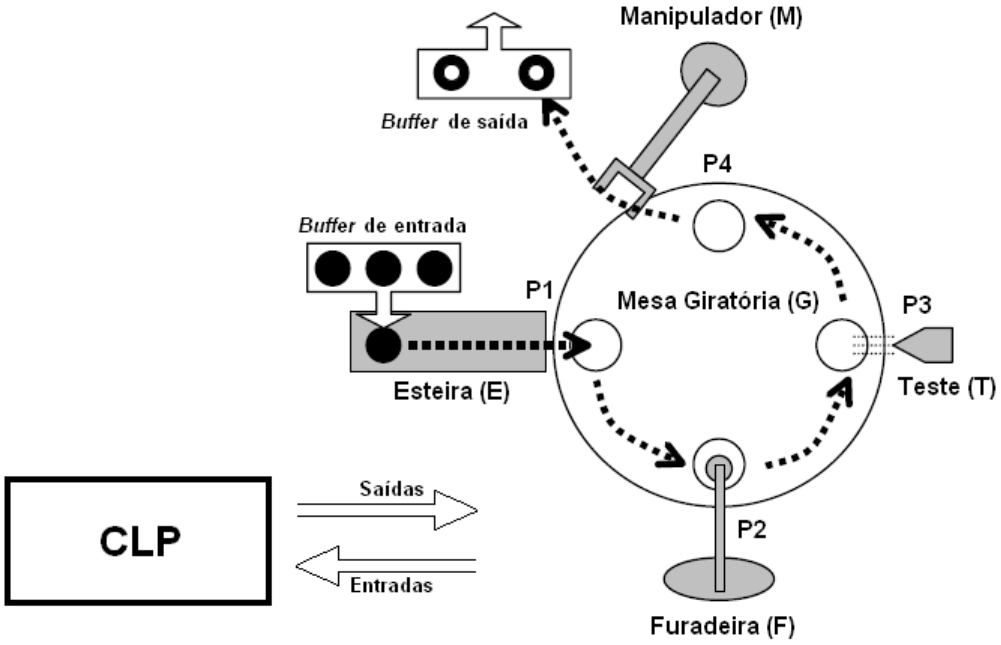


Figura 5.1 –Célula de Manufatura

Esse sistema é composto por uma *Mesa Giratória (G)* com 4 pontos distintos de parada, alcançados a cada giro de 90° para efetuar o trabalho em peças de produção. As peças brutas são inicialmente posicionadas em um *buffer* de entrada que as deposita em uma *Esteira (E)* para transportá-las até a mesa. Após o primeiro giro, uma peça é posicionada para que uma *Furadeira (F)* efetue a perfuração. Após mais um giro de 90°, a peça é submetida a um *Teste (T)* e por fim, após mais um giro da mesa, um *Manipulador (M)* robótico remove a peça da mesa e deposita-a em um *buffer* de saída.

Não existem sensores informando a presença de peças nas posições de parada da mesa (P1, P2, P3 e P4). Espera-se que o sistema opere de forma concorrente e extremamente otimizada tal que, no melhor caso, seja possível ter uma peça em cada uma das posições de trabalho enquanto a mesa se encontrar parada.

A TCS será utilizada para sintetizar uma lógica de controle que atenda esses requisitos, tal que, a partir dos autômatos que representam os supervisores obtidos, máquinas de estados possam ser encontradas de acordo com a metodologia proposta neste trabalho e sirvam como base para que seja feita a implementação em CLP.

5.1.1 Descrição dos Eventos

O modelo desse sistema possui 10 eventos, sendo 5 controláveis e 5 não-controláveis.

A Tabela 5.1 descreve os eventos controláveis.

Tabela 5.1 – Conjunto de Eventos Controláveis

Evento	Descrição
G_inicio	início de operação da *Mesa Giratória*
E_inicio	início de operação da *Esteira*
F_inicio	início de operação da *Furadeira*
T_inicio	início de operação do dispositivo de *Teste*
M_inicio	início de operação do *Manipulador* robótico

A Tabela 5.2 por sua vez descreve o conjunto de eventos não controláveis.

Tabela 5.2 – Conjunto de Eventos Não Controláveis

Evento	Descrição
G_fim	fim de operação da *Mesa Giratória*
E_ fim	fim de operação da *Esteira*
F_ fim	fim de operação da *Furadeira*
T_ fim	fim de operação do dispositivo de *Teste*
M_ fim	fim de operação do *Manipulador* robótico

5.1.2 Modelagem das Subplantas

A planta global *G* é composta por 5 subplantas G_i*'s*, que descrevem o comportamento livre dos componentes da célula de manufatura.

A subplanta G_1 representa um modelo para a *Mesa Giratória (G)*, conforme mostrado no autômato da Figura 5.2.

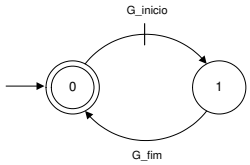


Figura 5.2 – Autômato da Subplanta *G1 (G)*

A subplanta *G2* representa um modelo para a *Esteira (E)*, conforme mostrado no autômato da Figura 5.2.

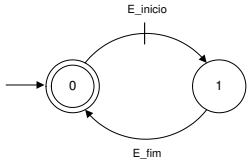


Figura 5.3 – Autômato da Subplanta *G2 (E)*

A subplanta *G3* representa um modelo para a *Furadeira (F)*, conforme mostrado no autômato da Figura 5.4.

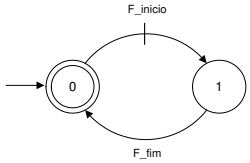


Figura 5.4 – Autômato da Subplanta *G3 (F)*

A subplanta *G4* representa um modelo para o dispositivo de *Teste (T)*, conforme mostrado no autômato da Figura 5.5.

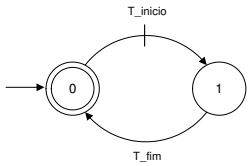


Figura 5.5 – Autômato da Subplanta *G4 (T)*

Por fim, a subplanta *G5* representa um modelo para o *Manipulador (M)* robótico, conforme mostrado no autômato da Figura 5.6.

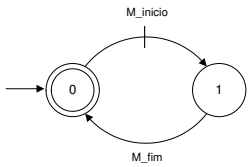


Figura 5.6 – Autômato da Subplanta *G5 (M)*

O estado inicial de todas as subplantas é marcado representando a condição inicial do sistema. Além disso, a marcação define que quando esses estados são alcançados uma tarefa é concluída.

5.1.3 Modelagem das Especificações de Controle

Foram criadas 8 especificações de controle visando restringir o comportamento da planta a um funcionamento desejado e sem que ocorram bloqueios ou conflitos.

A especificação E_1 garante que a *Mesa Giratória* não vai girar à toa (*G_inicio*), isto é, sem que ao menos exista uma peça bruta em P1 (*E_fim*) e/ou uma peça furada em P2 (*F_fim*) e/ou uma peça testada em P3 (*T_fim*). A Figura 5.7 apresenta o autômato dessa especificação.

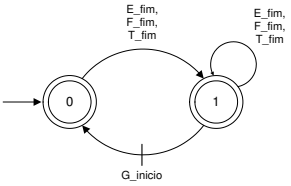


Figura 5.7 – Autômato da Especificação E_1

A especificação E_2 garante que a *Mesa Giratória* não inicia sua operação (*G_inicio*) sem que a *Esteira* termine sua operação (*E_fim*). Da mesma forma, a *Esteira* não inicia sua operação (*E_inicio*) sem que a *Mesa Giratória* termine sua operação (*G_fim*). A Figura 5.8 apresenta o autômato dessa especificação.

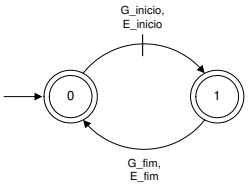


Figura 5.8 – Autômato da Especificação E_2

A especificação E_3 garante que a *Mesa Giratória* não inicia sua operação (*G_inicio*) sem que a *Furadeira* termine sua operação (*F_fim*). Da mesma forma, a *Furadeira* não inicia sua operação (*F_inicio*) sem que a *Mesa Giratória* termine sua operação (*G_fim*). A Figura 5.9 apresenta o autômato dessa especificação.

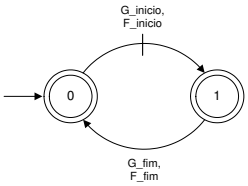


Figura 5.9 – Autômato da Especificação E_3

A especificação E_4 garante que a *Mesa Giratória* não inicia sua operação (*G_inicio*) sem que o dispositivo de *Teste* termine sua operação (*T_fim*). Da mesma forma, o dispositivo de *Teste* não inicia sua operação (*T_inicio*) sem que a *Mesa Giratória* termine sua operação (*G_fim*). A Figura 5.10 apresenta o autômato dessa especificação.

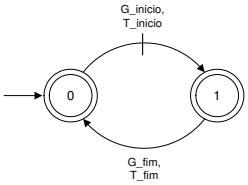


Figura 5.10 – Autômato da Especificação E_4

A especificação E_5 garante que a *Mesa Giratória* não inicie sua operação (*G_inicio*) enquanto o *Manipulador* robótico estiver operando (*M_fim*). Da mesma forma, o *Manipulador* robótico não inicia sua operação (*M_inicio*) sem que a *Mesa Giratória* termine sua operação (*G_fim*). A Figura 5.11 apresenta o autômato dessa especificação.

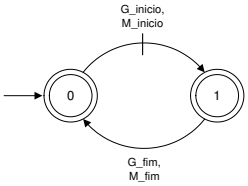


Figura 5.11 – Autômato da Especificação E_5

A especificação E_6 serve para evitar a sobreposição de peças na posição P1 da *Mesa Giratória* (*E_fim*), evitar a furação (*F_inicio*) sem peça bruta na posição P2 da *Mesa Giratória* e evitar girar a *Mesa Giratória* (*G_inicio*) com peça bruta na posição P2. A Figura 5.12 apresenta o autômato dessa especificação.

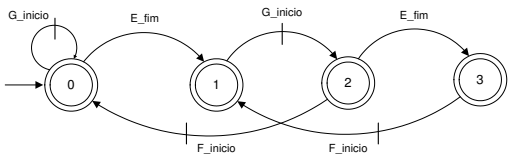


Figura 5.12 – Autômato da Especificação E_6

A especificação E_7 serve para evitar a furação de uma peça 2 vezes (*F_inicio*), evitar o início do *Teste* (*Teste_inicio*) sem peça furada na posição P3 da *Mesa Giratória* e evitar girar a *Mesa Giratória* (*G_inicio*) com peça furada mas não testada na posição P3. A Figura 5.13 apresenta o autômato dessa especificação.

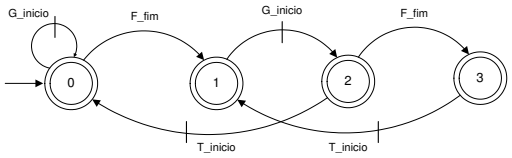


Figura 5.13 – Autômato da Especificação E_7

A especificação E_8 serve para evitar o *Teste* de uma peça 2 vezes (*T_inicio*), evitar o acionamento do *Manipulador* robótico (*M_inicio*) sem peça testada na posição P4 da *Mesa Giratória* e evitar girar a *Mesa Giratória* (*G_inicio*) com peça na posição P4. A Figura 5.14 apresenta o autômato dessa especificação.

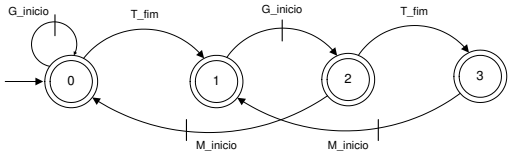


Figura 5.14 – Autômato da Especificação E_8

5.2 Abordagem Monolítica

A obtenção dos supervisores e máquinas de estados para a célula de manufatura a partir da abordagem monolítica é realizada conforme apresentado no Capítulo 3.

5.2.1 Supervisor Monolítico

O supervisor monolítico para a célula de manufatura didática possui 151 estados e 350 transições. Ele não é apresentado aqui devido à dificuldade para sua visualização. Este exemplo já demonstra a dificuldade encontrada para representar sistemas de maior porte por meio de supervisor monolítico.

O supervisor monolítico reduzido obtido com o uso da ferramenta TCT (FENG e WONHAM, 2006) possui 62 estados e 220 transições enquanto o supervisor monolítico reduzido obtido por meio da ferramenta GRAIL (REISER *et al.*, 2006) possui 55 estados e 174 transições.

Devido ao tamanho considerável dos supervisores monolíticos reduzidos, não convém efetuar a implementação dos mesmos em CLP usando linguagem *Ladder* a fim de comparar o tamanho de código com as demais metodologias.

5.2.2 Máquina de *Mealy* na Abordagem Monolítica

A máquina de *Mealy* na abordagem monolítica para a célula de manufatura didática possui 22 estados e 73 transições, sendo ilustrada na Figura 5.15. Embora ainda seja difícil compreender a dinâmica desse sistema com a representação por máquina de estados, percebe-se que o número de estados e transições já foi reduzido significativamente se comparado com o supervisor monolítico.

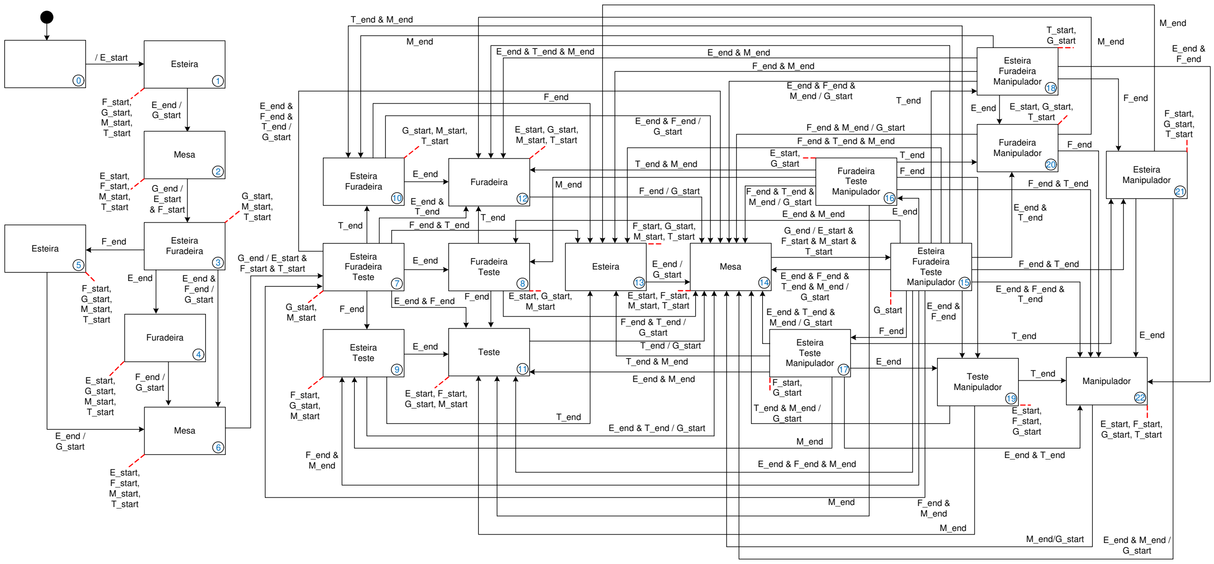


Figura 5.15 – Máquina de *Mealy* na Abordagem Monolítica para a Célula de Manufatura

A maior parte das transições presentes nesse sistema consiste no término de operação de alguma subplanta onde nenhuma ação é executada. Sendo assim, espera-se que com o processo de redução, vários estados e transições sejam eliminados. Reduzindo essa máquina com o uso do Algoritmo 3.2, obtém-se a máquina de *Mealy* reduzida mostrada na Figura 5.16. Observe que a redução de estados e transições foi significativa. Essa máquina contém apenas 7 estados e 7 transições com execução de ação. Várias transições são representadas por linhas tracejadas, pois não são responsáveis pela execução de qualquer ação. Agora, fica muito mais fácil compreender a dinâmica desse sistema. As peças são transportadas até a *Mesa Giratória* através da *Esteira*. Os processos de furação pela *Furadeira*, *Teste* e transporte por meio do *Manipulador* são então realizados. À medida que peças vão preenchendo as posições da *Mesa Giratória*, a máquina transita de estados. Em uma condição de carga máxima, a *Mesa*

Giratória possui peças em todas as posições (estado *7*). Somente quando todas as subplantas terminarem sua operação, é que a *Mesa Giratória* efetua um novo giro (estado *6*). Após atingir esses dois estados, o sistema atinge uma condição de regime, que consiste na alternância entre esses estados discretos.

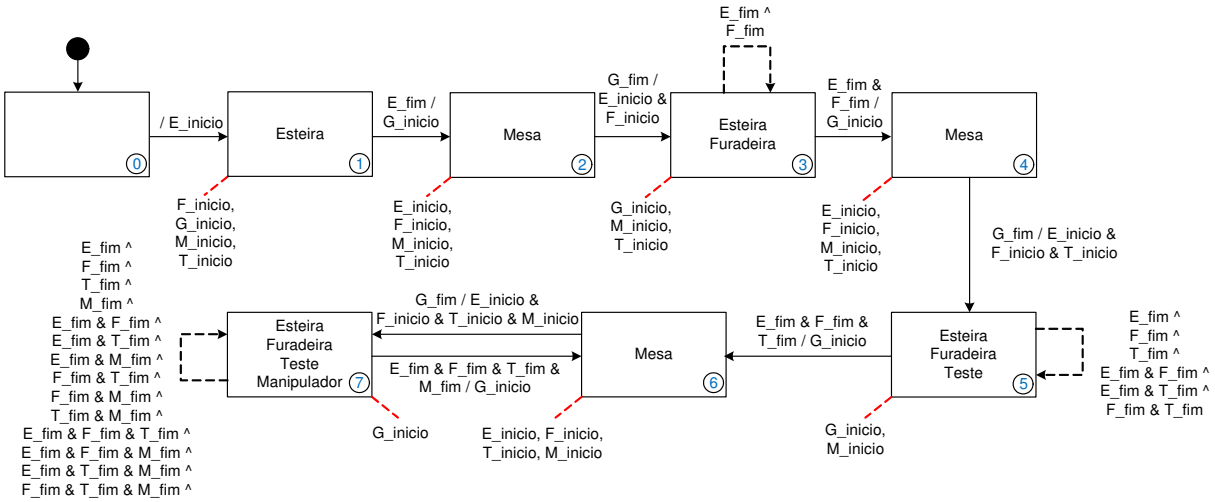


Figura 5.16 – Máquina de *Mealy* Reduzida na Abordagem Monolítica para a Célula de Manufatura

5.2.3 Máquina de *Moore* na Abordagem Monolítica

A máquina de *Moore* reduzida na abordagem monolítica possui exatamente o mesmo número de estados e transições da máquina de *Mealy*. Isto ocorre por se tratar de um sistema de caráter predominantemente serial e onde o único estado de chegada para mais do que uma transição é o estado *6*. Entretanto, a ação executada é a mesma: *G_inicio*. A Figura 5.17 ilustra esse modelo. Devido a grande similaridade com a máquina de *Mealy* reduzida, essa máquina não será implementada para efeito comparativo com as outras metodologias.

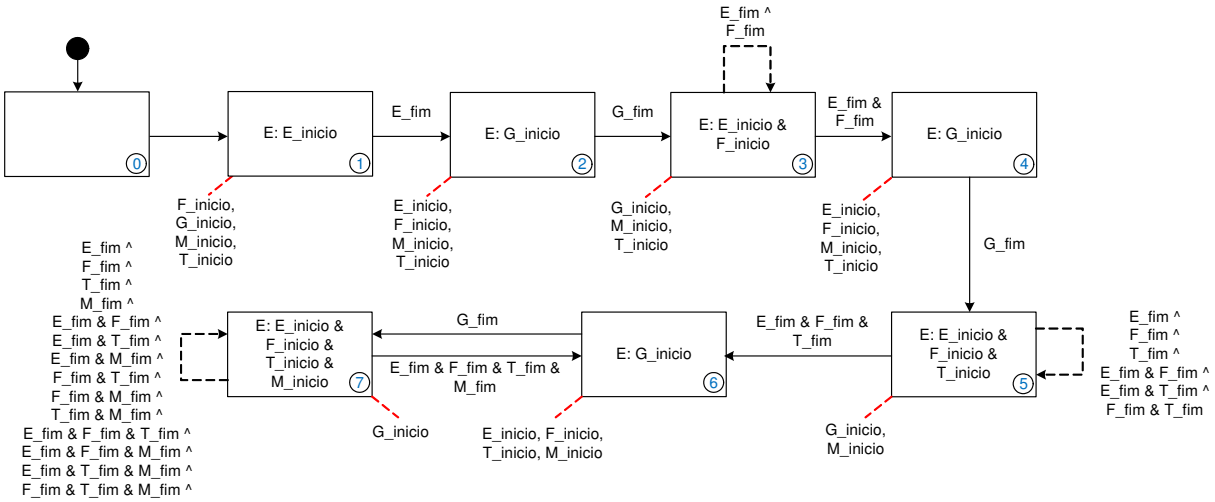


Figura 5.17 – Máquina de *Moore* Reduzida na Abordagem Monolítica para a Célula de Manufatura

5.3 Abordagem Modular Local

A obtenção dos supervisores e máquinas de estados para a célula de manufatura a partir da abordagem modular local é realizada conforme apresentado no Capítulo 3.

5.3.1 Supervisor Modular Local

O número de estados e transições dos supervisores modulares locais obtidos a partir das especificações de controle descritas anteriormente é apresentado na Tabela 5.3. Mesmo na abordagem modular local, alguns supervisores ainda possuem um tamanho relativamente grande, reforçando a necessidade de adotar algum procedimento de redução para reduzir a quantidade de estados e que mantenha a mesma lógica de controle.

Tabela 5.3 – Número de Estados e Transições dos Supervisores Modulares Locais para a Célula de Manufatura

	Supervisor Modular Local - Grail		**Supervisor Modular Local - TCT**		**Supervisor Modular Local Reduzido - Grail**		**Supervisor Modular Local Reduzido - TCT**	
	Número de Estados	**Número de Transições**	**Número de Estados**	**Número de Transições**	**Número de Estados**	**Número de Transições**	**Número de Estados**	**Número de Transições**
SL_1, SLR_1	32	120	32	120	3	22	2	15
SL_2, SLR_2	3	4	3	4	3	4	2	4
SL_3, SLR_3	3	4	3	4	3	4	2	4
SL_4, SLR_4	3	4	3	4	3	4	2	4
SL_5, SLR_5	3	4	3	4	3	4	2	4
SL_6, SLR_6	32	72	24	52	17	62	4	16
SL_7, SLR_7	32	72	24	52	17	62	4	16
SL_8, SLR_8	32	72	24	52	19	70	4	16

Observe que a abordagem em que são obtidos supervisores mais enxutos é utilizando o algoritmo de redução de supervisores presente na ferramenta TCT. Assim, esses supervisores serão implementados de acordo com a proposta de implementação apresentada em (VIEIRA, 2003). Vale lembrar que essa estratégia de implementação requer a implementação dos modelos das plantas (sistema-produto) e da lista de desabilitações para cada supervisor modular local, para que o sincronismo seja mantido e a lógica de controle funcione adequadamente.

5.3.2 Máquina Modular Local

As máquinas modulares locais obtidas com o uso dos Algoritmos 3.4, 3.5 e 3.6 propostos a partir dos supervisores modulares locais, são apresentadas nas figuras a seguir.

A Figura 5.18 ilustra a máquina ML_1, obtida com o uso do Algoritmo 3.4 a partir do supervisor SL_1. A máquina ML_1 é a maior de todas as máquinas obtidas com o uso da abordagem modular local para esse sistema, com 11 estados, sendo resultado do tamanho do supervisor modular local SL_1 do qual a máquina foi obtida, que possui 32 estados.

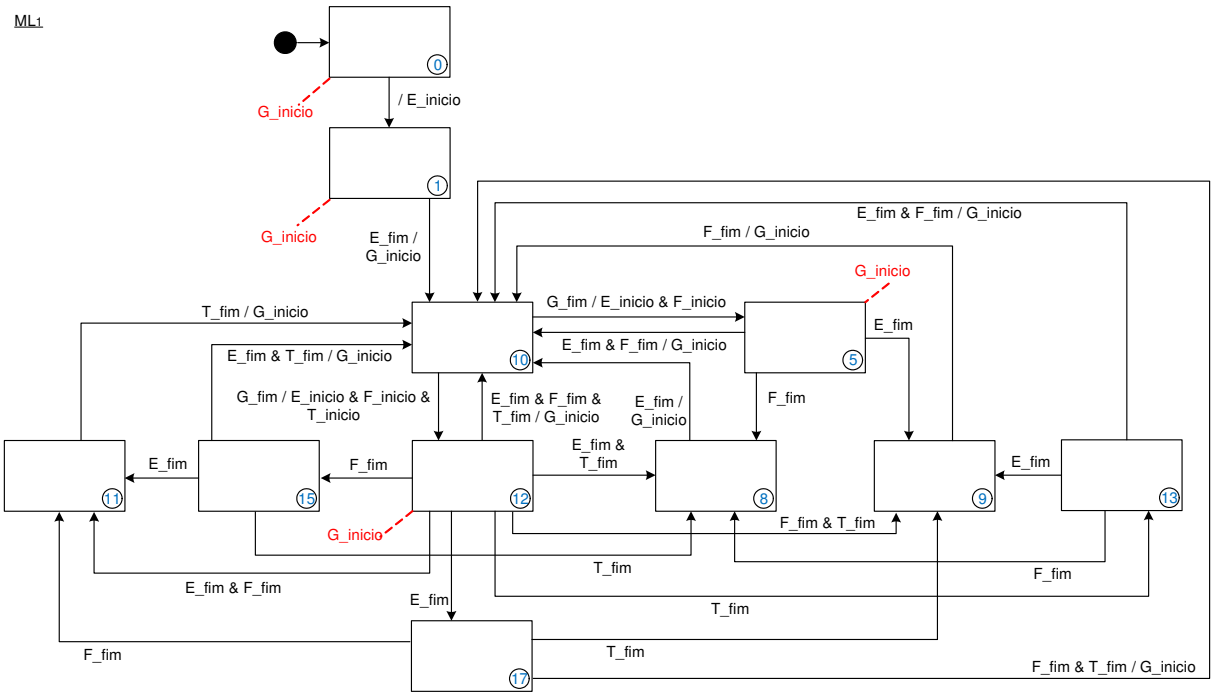


Figura 5.18 – Máquina Modular Local ML_1 para a Célula de Manufatura

Após a redução dessa máquina com o uso do Algoritmo 3.5, obtém-se a máquina reduzida MLS_1' mostrada na Figura 5.19-a. Observe que essa etapa de eliminação de não-determinismos e transições devido à ações já é suficiente para reduzir o número de estados e transições. Após a eliminação das transições que não resultam em ações, criação de *self-loops* representados por linhas tracejadas para essas transições e eliminação dos estados não alcançáveis com o uso do Algoritmo 3.6, obtém-se o modelo final da máquina reduzida MLS_1, ilustrada na Figura 5.19-b. Quando o estado *2* da máquina reduzida é alcançado, a ação *T_inicio* será executada somente se nenhuma outra máquina desabilitá-la. Caso contrário, essa ação não é executada.

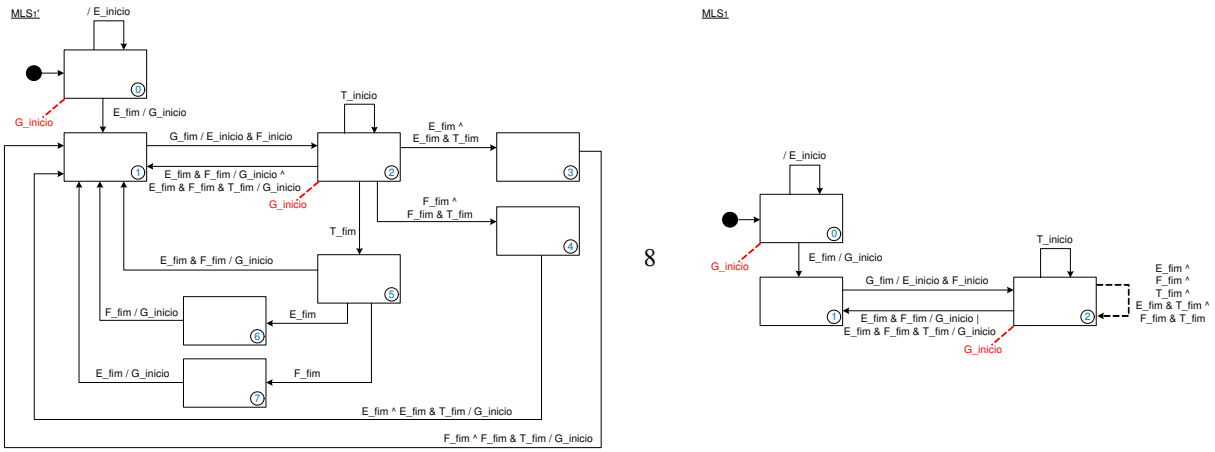


Figura 5.19 – Máquina Modular Local Reduzida para a Célula de Manufatura: (a) *MLS1'* (b) *MLS1*

A máquina *ML2* é ilustrada na Figura 5.20. Uma característica interessante dessa máquina consiste no não-determinismo gerado pelo evento *E_fim* a partir do estado *1*. Ele pode fazer a máquina transitar tanto para o estado *0* quanto para o estado *2*. Para cada um dos estados, uma ação correspondente pode ser executada: no estado *0* a ação *E_inicio* é passível de ocorrer, desde que não seja desabilitada pelas demais máquinas; no estado *2*, a ação *G_inicio* ocorre, pois ela já é executada durante a transição para esse estado.

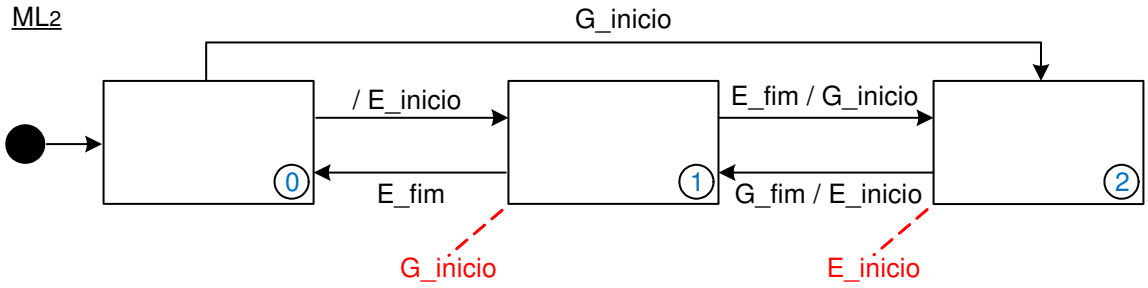


Figura 5.20 – Máquina Modular Local *ML2* para a Célula de Manufatura

O processo de redução da máquina *ML2* resulta na máquina *MLS2*, mostrada na Figura 5.21. Observe que no estado *0* da máquina reduzida existe uma transição devido à ação *E_inicio OU* à ação *G_inicio* (*E_inicio* ^ *G_inicio*), resultado da estrutura da máquina original, onde uma ou outra ação poderia ocorrer ao final do evento *E_fim*. Este por sua vez, deve ser descrito por um *self-loop*, tendo em vista que para essa máquina ele se comporta como uma transição que permanece no mesmo estado, embora ações podem ser executadas após a sua ocorrência.

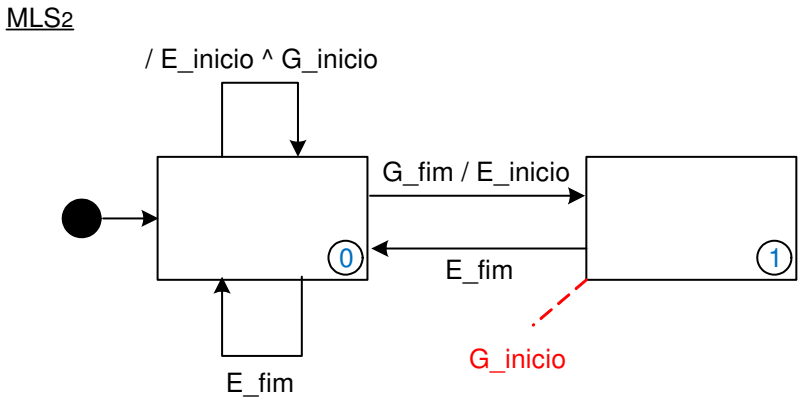


Figura 5.21 – Máquina Modular Local Reduzida *MLS₂* para a Célula de Manufatura

A máquina modular local *ML₃* é ilustrada na Figura 5.22. Essa máquina possui um não-determinismo resultante da transição *F_fim* partindo do estado *1* para os estados *0* e *2*.

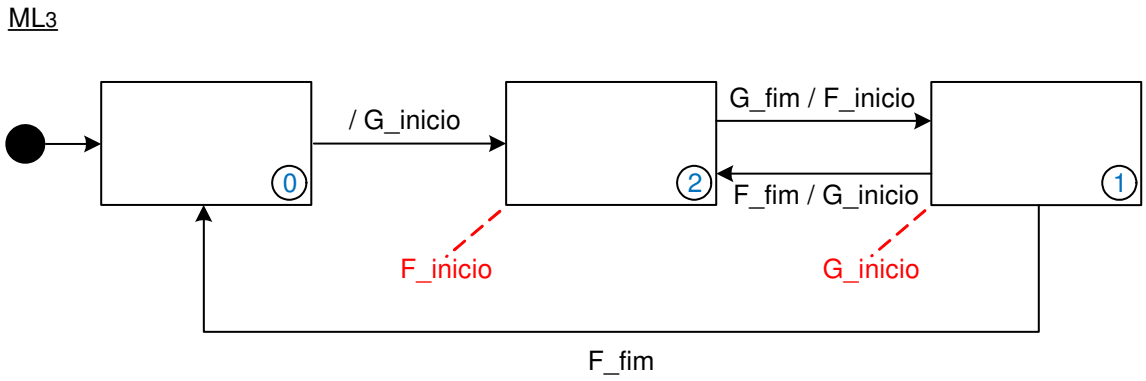


Figura 5.22 – Máquina Modular Local *ML₃* para a Célula de Manufatura

A redução da máquina *ML₃* resulta na máquina *MLS₃*, mostrada na Figura 5.23. Trata-se de um modelo bastante simples de máquina reduzida na abordagem modular local, resultante da interação da *Furadeira* com a *Mesa Giratória*.

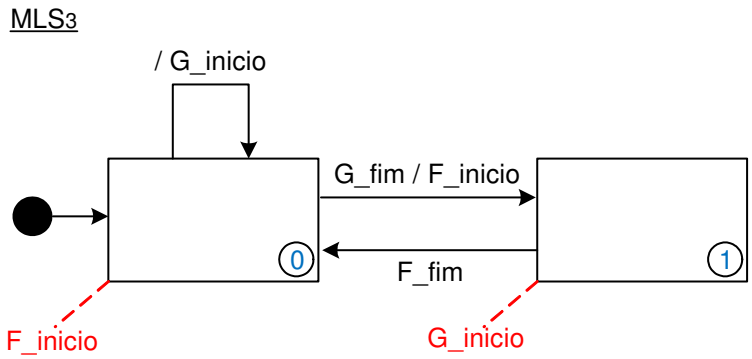


Figura 5.23 – Máquina Modular Local Reduzida *MLS₃* para a Célula de Manufatura

A máquina *ML4* consiste na interação do *Teste* com a *Mesa* Giratória, como mostrado na Figura 5.24. Além do não-determinismo devido ao evento *G_fim* partindo do estado *1*, situação análoga à que ocorre com a máquina *ML2*, essa máquina possui outro não-determinismo devido ao evento *T_fim* partindo do estado *2* e chegando aos estados *0* e *2*.

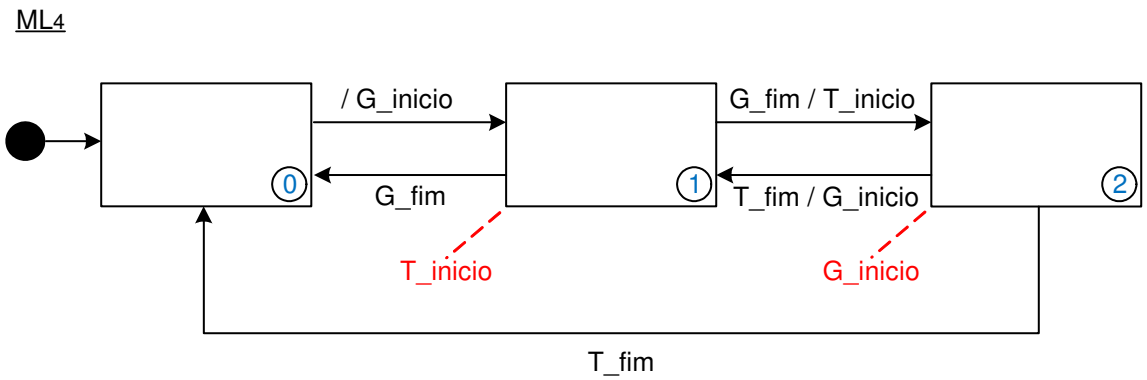


Figura 5.24 – Máquina Modular Local *ML4* para a Célula de Manufatura

A máquina reduzida *MLS4* é apresentada na Figura 5.25. A transição *T_fim* que aparece como um não-determinismo na máquina *ML4*, do estado *2* para os estados *0* e *1*, acaba em suma tendo o mesmo resultado no modelo reduzido, permitindo que a ação *G_inicio* ocorra.

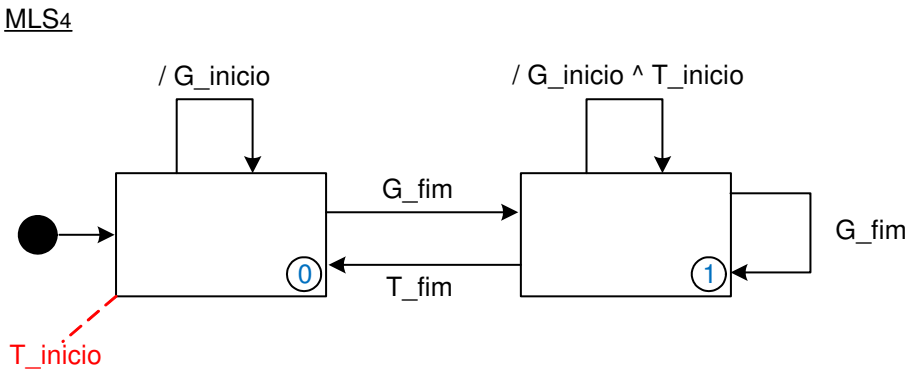


Figura 5.25 – Máquina Modular Local Reduzida *MLS4* para a Célula de Manufatura

A máquina *ML5* é mostrada na.Figura 5.26. Essa máquina representa a interação do *Manipulador* robótico com a *Mesa Giratória*.

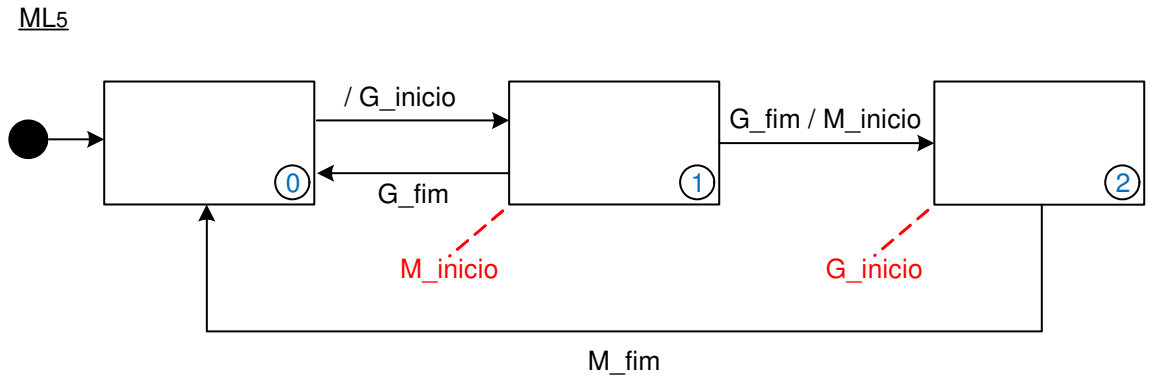


Figura 5.26 – Máquina Modular Local ML_5 para a Célula de Manufatura

A redução da máquina ML_5 resulta na máquina MLS_5, ilustrada na Figura 5.27. Essa máquina possui um comportamento bastante similar às máquinas MLS_2 e MLS_4.

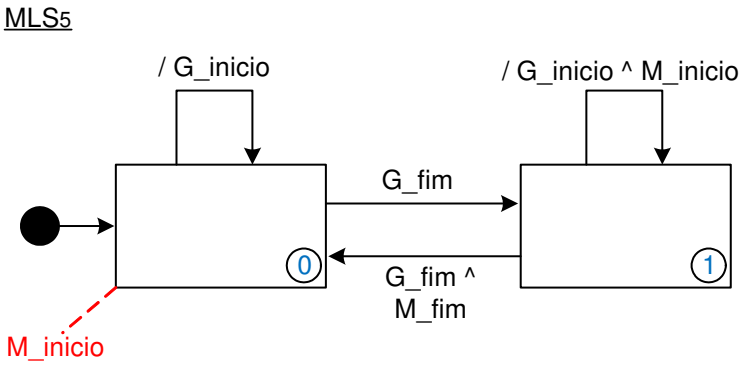


Figura 5.27 – Máquina Modular Local Reduzida MLS_5 para a Célula de Manufatura

A Figura 5.28 mostra a máquina ML_6. Essa máquina já possui um caráter um pouco mais complexo do que as máquinas ML_2 a ML_5, devido à topologia de sua especificação local. Assim como nas demais máquinas, transições devido a ações e não-determinismos também aparecem aqui.

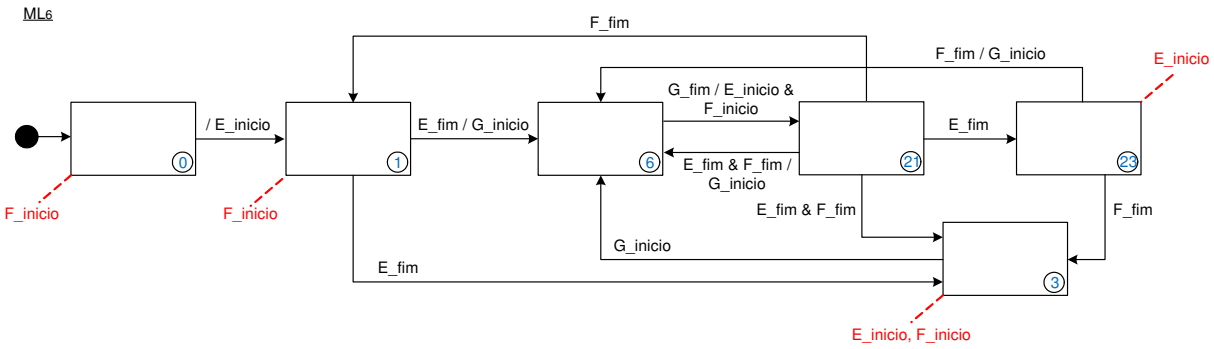


Figura 5.28 – Máquina Modular Local ML_6 para a Célula de Manufatura

O modelo reduzido da máquina ML_6 possui o formato apresentado na Figura 5.29.

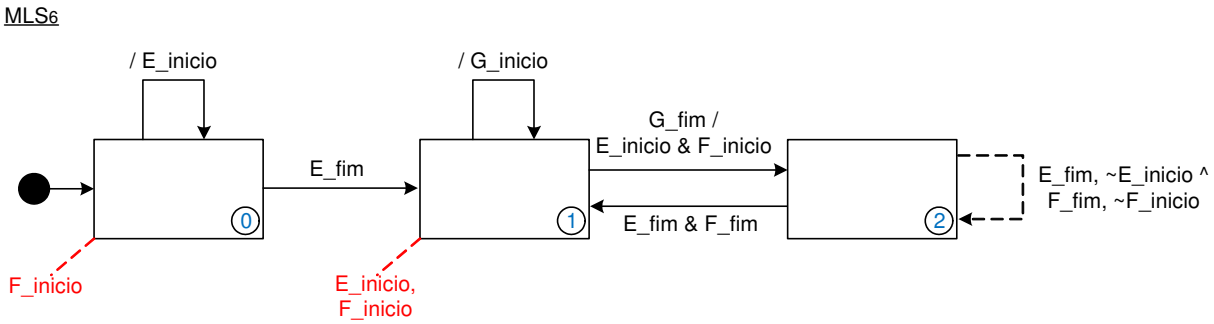


Figura 5.29 – Máquina Modular Local Reduzida *MLS*$_6$ para a Célula de Manufatura

A máquina *ML*$_7$ possui uma estrutura parecida com a máquina *ML*$_6$, sendo mostrada na Figura 5.30.

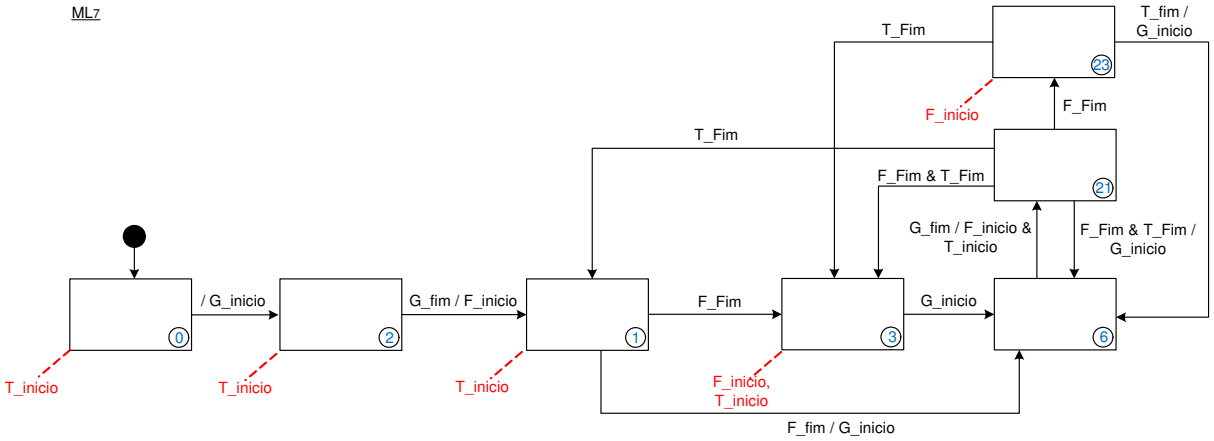


Figura 5.30 – Máquina Modular Local *ML*$_7$ para a Célula de Manufatura

A máquina reduzida *MLS*$_7$ é bastante parecida com a máquina reduzida *MLS*$_6$, possuindo entretanto um estado a mais, como mostra a Figura 5.31.

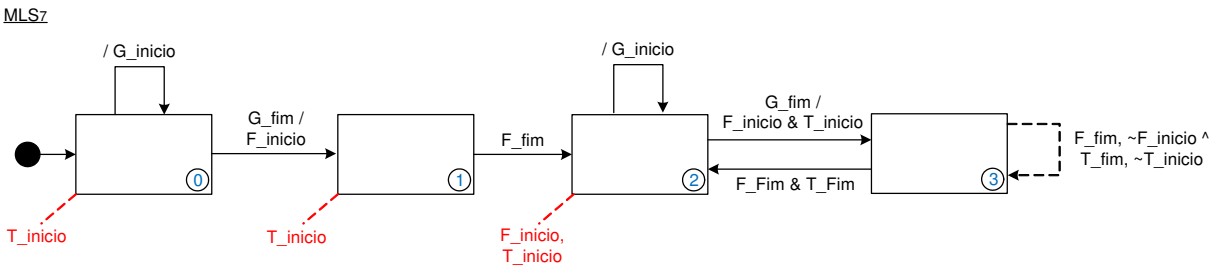


Figura 5.31 – Máquina Modular Local Reduzida *MLS*$_7$ para a Célula de Manufatura

Por fim, a máquina *ML*$_8$ é apresentada na Figura 5.32. A estrutura dessa máquina lembra a estrutura das máquinas *ML*$_6$ e *ML*$_7$, devido à sua especificação local possuir uma topologia exatamente igual a das especificações *E*$_6$ e *E*$_7$, respectivamente.

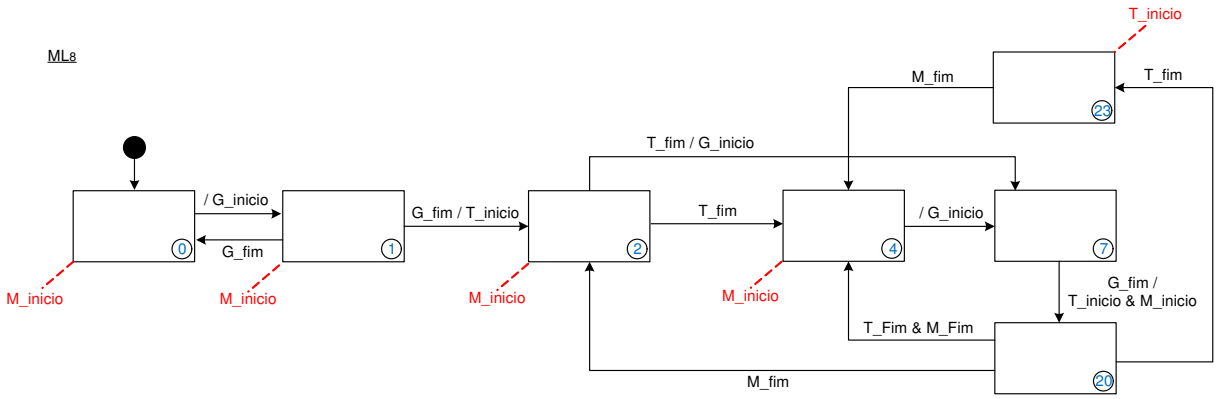


Figura 5.32 – Máquina Modular Local ML_8 para a Célula de Manufatura

A máquina reduzida MLS_8 obtida a partir da máquina original ML_8 é mostrada na Figura 5.33. Observe que essa máquina é bastante similar às máquinas reduzidas ML_6 e ML_7, como era de se esperar.

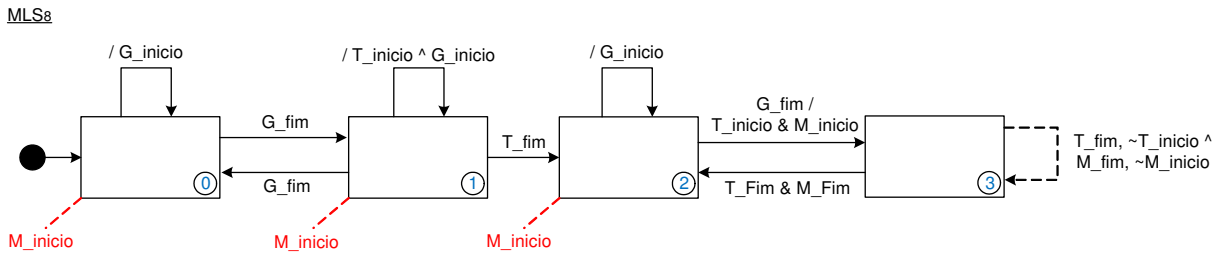


Figura 5.33 – Máquina Modular Local Reduzida MLS_8 para a Célula de Manufatura

A proposta deste trabalho consiste em implementar em *Ladder* as máquinas reduzidas obtidas de acordo com a abordagem modular local.

5.4 Resultados de Modelagem e Implementação para as Diferentes Metodologias

Nesta seção é realizada uma análise dos resultados obtidos para as diferentes metodologias descritas neste trabalho. É feita uma análise comparativa entre os modelos de supervisores e máquinas de estados que possuem menor número de estados e uma análise comparativa entre as implementações desses modelos em CLP.

5.4.1 Comparação entre os Modelos de Supervisores

A Tabela 5.4 mostra um comparativo do número de estados e transições que representam os supervisores para a célula de manufatura obtidos a partir da abordagem

modular local da TCS e das abordagens monolítica e modular local propostas neste trabalho para as máquinas de estados reduzidas.

Tabela 5.4 – Comparação entre as Abordagens de Controle Supervisório para a Célula de Manufatura

	Máquina de Estados Reduzida – Abordagem Monolítica		**Supervisor Reduzido (TCT) – Abordagem Modular Local**		**Máquina de Estados Reduzida – Abordagem Modular Local**	
	Número de Estados	**Número de Transições**	**Número de Estados**	**Número de Transições**	**Número de Estados**	**Número de Transições**
SLR_1, MLS_1	7	7	2	4	3	5
SLR_2, MLS_2			2	4	2	5
SLR_3, MLS_3			2	4	2	3
SLR_4, MLS_4			2	4	2	6
SLR_5, MLS_5			2	4	2	6
SLR_6, MLS_6			4	5	3	5
SLR_7, MLS_7			4	5	4	6
SLR_8, MLS_8			4	5	4	9
TOTAL	**7**	**7**	**22**	**35**	**22**	**45**

Diferente da Tabela 5.3, a Tabela 5.4 mostra para o supervisor modular local obtido a partir do TCT, somente as transições que resultam em mudança de estado. Isso porque não existe a necessidade de considerar as transições que aparecem em *self-loop* nos estados, tendo em vista que essas ações não alteram os estados ativos e, como conseqüência disso, não alteram a ação de controle (DE QUEIROZ, 2004).

Observe que a máquina de estados reduzida na abordagem monolítica possui um número de estados e transições consideravelmente menor do que as abordagens modular local. Conforme comentado anteriormente, devido ao caráter desse sistema ser seqüencial, várias transições no modelo original do autômato que representa o supervisor e da máquina de estados não são responsáveis pela execução de ações. Assim, quando o modelo é reduzido, verifica-se que não existe a necessidade de muitos estados para descrever a lógica de controle proposta. Esse resultado já dá um indício de que a implementação da máquina de estados reduzida na abordagem monolítica vai ser mais compacta e vai ocupar uma quantidade de memória menor do que as demais.

Em relação aos autômatos reduzidos e máquinas de estados reduzidas na abordagem modular local, é possível observar que eles possuem a mesma quantidade de estados, embora a máquina de estados tenha algumas transições a mais. Assim, é de se esperar que o tamanho de código para as implementações desses supervisores seja parecido, mesmo que para a

abordagem por meio de autômatos seja necessário implementar o bloco de sistema produto também.

5.4.2 Comparação entre as Topologias de Implementação em CLP

O ambiente de *software* utilizado para programação em *Ladder* e compilação de código para as diferentes topologias avaliadas é o SIMATIC STEP 7, da Siemens. Essa ferramenta é proposta para ser utilizado em CLPs Siemens da família S7-300. Não foi utilizada nenhuma rotina de programação específica dessa família de CLPs a fim de evitar a perda de generalidade da metodologia proposta.

As diferentes metodologias de implementação foram testadas no *software* S7-PLCSIM, um simulador do ambiente SIMATIC para desenvolvimento de programas e *debug* de erros. A escolha de usar um simulador foi devido à ausência da planta real para efetuar os testes. Os resultados foram satisfatórios para todas as metodologias, onde a lógica para controle supervisório se comportou como planejado, de acordo com as especificações propostas.

A Tabela 5.5 mostra um comparativo do tamanho de código para as diferentes formas de implementação de supervisores para a célula de manufatura a partir dos autômatos encontrados por meio da abordagem modular local da TCS e para as máquinas de estados propostas neste trabalho. Vale ressaltar que a topologia de implementação usada para os supervisores reduzidos de acordo com a abordagem modular local é a proposta em (VIEIRA, 2003). Outras topologias podem sofrer variações nos parâmetros avaliados.

Tabela 5.5 – Comparação entre as Metodologias de Implementação em CLP para a Célula de Manufatura

Parâmetro / **Topologia**	**Máquina de Estados Reduzida – Abordagem Monolítica**	**Supervisores Reduzidos (TCT) – Abordagem Modular Local (VIEIRA, 2003)**	**Máquina de Estados Reduzida – Abordagem Modular Local**
Tamanho do Código (Bytes)	570	842	956
# de Variáveis Declaradas	23 (3 bytes)	59 (8 bytes)	46 (6 bytes)
# Variáveis para Evitar o Efeito Avalanche (Genérico)	-	1	# máquinas locais com saídas
# Eventos por Ciclo de Execução (Genérico)	>= 1	1	>= 1

Tamanho do Código: a máquina de estados reduzida na abordagem monolítica ocupa uma parcela de espaço em memória significativamente menor do que as demais. Embora as abordagens no âmbito modular local tenham surgido como uma proposta para redução do

tamanho de código, isso não se concretiza para o sistema em estudo. Esse resultado acaba sendo influenciado diretamente pela quantidade de transições onde existe execução de ações na máquina monolítica original. Se não for possível ter uma redução considerável de estados, provavelmente o código não será tão compacto tal como o código desenvolvido para a célula de manufatura. Olhando para as demais abordagens, a máquina de estados modular local reduzida possui tamanho de código levemente maior do que o supervisor modular local, devido basicamente à estrutura necessária para habilitar/desabilitar as ações e para garantir o sincronismo das plantas à medida que as máquinas locais evoluem. Entretanto, embora essa estrutura de código seja maior, ela possui vantagem em relação ao número de eventos que podem ser tratados em um mesmo ciclo de execução, como será mostrado adiante.

Número de Variáveis Declaradas: o reflexo no tamanho de código também é visto aqui, de acordo com o número de variáveis declaradas. Observe que a abordagem monolítica para máquina de estados possui exatamente a metade do número de variáveis se comparada com a abordagem modular local.

Número de Variáveis para Evitar o Efeito Avalanche: o número de variáveis necessário para garantir que o efeito avalanche não aconteça é mostrado na Tabela 5.5 para as diferentes metodologias. A máquina de estados na abordagem monolítica não requer nenhuma variável adicional, pois, para cada transição de estados que acontece, os eventos não controláveis responsáveis por essa transição serão desativados imediatamente, tal que nenhuma outra transição ocorra naquele mesmo ciclo de execução do CLP. A abordagem por meio de supervisores modulares locais obtidos a partir da TCS utiliza a variável denominada *ev_plt* para evitar o efeito avalanche. Essa variável é ativada toda vez que um evento ocorre, seja ele controlável ou não controlável. A abordagem modular local usando máquinas de estados por sua vez requer uma variável para cada máquina modular local (MLS_j_evol). Essa variável é ativada toda vez que uma transição é detectada na máquina modular em questão.

Número de Eventos por Ciclo de Execução: a abordagem modular local usando autômatos para representar os supervisores proposta em (VIEIRA, 2003) garante que no máximo um evento ocorra por ciclo de execução, seja ele controlável ou não controlável. Isso acaba tendo uma eficiência pobre, pois pode levar a problemas de comunicação entre a planta de chão de fábrica e o CLP quando diversos eventos são lidos ao mesmo tempo pelo CLP. As abordagens baseadas em máquinas de estados têm a vantagem de tratar diversos eventos em um mesmo ciclo de execução.

5.5 Conclusões

Neste capítulo foi apresentada uma célula de manufatura, que consiste em um sistema de maior porte para avaliação da metodologia proposta nesta dissertação. Para isso, foram obtidos os modelos de acordo com as abordagens monolítica e modular local tanto para os supervisores obtidos através da Teoria de Controle Supervisório (TCS) quanto das máquinas de estados propostas, a fim de fazer uma análise comparativa entre eles.

Além disso, diferentes topologias de implementação foram realizadas a partir desses modelos com o intuito de avaliar parâmetros comparativos, tais como o tamanho de código e algumas propriedades que esses modelos oferecem.

A partir desse exemplo, foi possível avaliar em um âmbito mais geral as vantagens e desvantagens da metodologia proposta neste trabalho, que se baseia em modelos de máquinas de estados com saídas para representação de supervisores para Sistemas a Eventos Discretos (SEDs).

Capítulo 6

Conclusões e Perspectivas

6.1 Conclusões

A metodologia proposta apresenta um novo modelo para representar supervisores para Sistemas a Eventos Discretos (SEDs). A utilização da Teoria de Controle Supervisório (TCS) acaba sendo útil para obter supervisores a partir de modelos mais simples, usando autômatos para representar as subplantas e especificações de controle, tal que, partindo de modelos mais simples sejam obtidos supervisores representados por máquinas de estados com saídas. Em geral, a abordagem por meio de máquinas de estados com saídas consiste em um modelo com bastante riqueza de informações. Convém salientar a importância do processo de modelagem para os supervisores, pois caso alguma especificação de controle mude ou subsistema seja adicionado à planta, existe a necessidade de refazer a síntese das máquinas de estados com saídas.

Os algoritmos propostos para transformação de autômatos que representam supervisores em máquinas de estados com saídas permitem uma redução do número de estados do modelo. Essa abordagem reduzida é utilizada como referência para a implementação do sistema de controle em uma unidade de processamento de dados, devendo apenas levar em conta os aspectos característicos de implementação para o controlador utilizado de acordo com as características construtivas de seu *hardware*. Neste trabalho, foram descritos os problemas de implementação de supervisores em controladores síncronos e as possíveis soluções para eles. A implementação de supervisores em Controladores Lógicos Programáveis (CLPs) usando linguagem *Ladder* é o objetivo final deste trabalho. A estrutura de implementação é obtida a partir dos modelos propostos de máquinas de estados reduzidas com saídas.

O problema de uma célula de manufatura analisado demonstra alguns aspectos relativos à otimização no tamanho de código gerado resultando em economia no uso de memória não-volátil e na possibilidade de tratamento de diversos eventos dentro de um

mesmo ciclo de execução do controlador. Para sistemas de grande porte, isso resulta em uma melhoria na dinâmica temporal do controle quando ocorre a mudança de valor em diversos sinais de entrada e diversas ações de controle devem ser executadas em um mesmo ciclo de execução do CLP, garantindo assim o sincronismo e minimizando os problemas devido à atrasos de comunicação.

6.2 Perspectivas

Como perspectivas futuras, essa abordagem abre campo para o desenvolvimento de ferramentas automáticas para gerar os modelos de máquinas de estados com saídas usando os algoritmos propostos. A automatização desses algoritmos é sem dúvida uma solução muito mais interessante do que a realização de procedimentos manuais, tendo como vantagens a economia no uso de recursos humanos, maior praticidade, menor tempo para obtenção dos resultados e menor susceptibilidade a erros.

O desenvolvimento de ferramentas para geração automática de código em *Ladder* ou alguma outra linguagem de programação para CLPs usando como base os modelos de máquinas de estados reduzidas com saídas também é visto como uma oportunidade em potencial proveniente deste trabalho. Ferramentas poderiam ser criadas para auxiliar desde a síntese do sistema de controle até a criação de código para a aplicação final.

O procedimento apresentado também permite a prática e disseminação nas indústrias de iniciativas como: criação de documentação descritiva da lógica de operação do sistema, armazenamento dos modelos em bancos de dados para posterior reutilização, maior facilidade no aprendizado e comunicação entre os profissionais, flexibilidade para desenvolvimento de lógicas de controle flexíveis e confiáveis, dentre outros.

Essa metodologia abre espaço para novas linhas de estudo considerando a solução de problemas de controle baseadas em máquinas de estados com saídas, onde o cálculo de supervisores seria feito diretamente a partir de modelos de máquinas de estados de subplantas e especificações, eliminando-se assim os modelos de autômatos tal como é usado neste trabalho. Além disso, a modelagem de sistemas poderia ser expandida de máquinas de estados para *statecharts* (HAREL, 1987), similar à metodologia de implementação de *statecharts* em CLPs proposta em (MOURA, 2009). Entretanto, essa metodologia não utiliza como base a TCS, tal como é proposto nesta dissertação.

Outro ramo a ser explorado é a implementação dos modelos de máquinas de estados reduzidas com saídas em outros tipos de controladores, tal como em microcontroladores. Vale

salientar que, de acordo com o tipo de controlador avaliado, deve-se levar em conta durante o processo de implementação as características construtivas intrínsecas dos mesmos.

Além disso, é interessante observar que para o sistema da célula de manufatura, a metodologia usando máquinas de estados reduzidas com saídas em sua abordagem monolítica possui uma representação mais simples do sistema de controle e um tamanho de código significativamente menor do que a metodologia convencional por autômatos para a abordagem modular local. Tendo em vista que uma das propostas da abordagem modular local é a redução do tamanho de código na implementação, esses resultados abrem margem para um estudo mais aprofundado de que tipo de proposta se torna mais adequada para outros tipos de sistemas, tal que seja possível compreender quando é mais conveniente utilizar uma determinada metodologia, não levando em consideração somente o tamanho de código, mas também outros parâmetros de fundamental importância para atender os requisitos de projeto. Assim, seria possível observar se é válido o esforço em explorar a natureza modular da planta sob análise ou simplesmente manter um controlador de caráter monolítico seria a solução mais adequada.

Em se tratando do tipo de máquina de estados com saídas a ser usado quando a metodologia proposta neste trabalho é utilizada, também vale uma ressalva semelhante à considerada para a escolha da metodologia para controle supervisório. Um importante campo de estudo consiste na análise de qual tipo de máquina, de *Mealy* ou de *Moore*, seria mais adequada para representar um determinado tipo de sistema. Dessa forma, o projetista poderia ter um direcionamento sobre qual procedimento é mais apropriado para a síntese e implementação do controle supervisório de acordo com o tipo de SED avaliado.

Referências Bibliográficas

AFZALIAN, A.; NOORBAKHSH, M.; NABAVI, S. A. PLC Implementation of Decentralized Supervisory Control for Dynamic Flow Controller. In: IEEE CONFERENCE ON CONTROL APPLICATIONS, 17th. San Antonio, Texas, USA, 2008.

AKESSON, K.; FABIAN, M.; FLORDAL, H.; MALIK, R. Supremica – An Integrated Environment for Verification, Synthesis and Simulation of Discrete Event Systems. In: IEEE INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS, v. 8, p. 384-385, 2006.

BASILE, F.; CHIACCHIO, P. On the Implementation of Supervised Control of Discrete Event Systems. In: IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, v. 15, n. 4, 2007.

BRANDIN, B. A. The Real-Time Supervisory Control of an Experimental Manufacturing Cell. In: IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, v. 12, pp. 1-14, 1996.

BARETTA, R. D.; TORRICO C. R. C. Máquinas de *Mealy* e *Moore* para Implementação de Controle Supervisório de Sistemas a Eventos Discretos em Microcontroladores. In: XVII CONGRESSO BRASILEIRO DE AUTOMÁTICA - CBA2008, Juiz de Fora, MG, 2008.

BALEMI, S. **Control of Discrete Event Systems: Theory and Application**. PhD thesis, Automatic Control Lab., Swiss Federal Inst. of Technology (ETH), Zurich, Switzerland, 1992

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to discrete event systems**. 2nd Ed. Springer, 2008.

CHEN, Y. -L.; LIN, F. Modeling of Discrete Event Systems using Finite State Machines with Parameters. In: IEEE INTERNATIONAL CONFERENCE ON CONTROL APPLICATIONS, Anchorage, Alaska, USA, 2000.

CONTROL ENGINEERING MAGAZINE. 'Programmable Logic Controllers Product Research', http://www.controleng.com/, 2007.

CURZEL, J. L. **Síntese e Implementação de Controle Supervisório em Uma Célula Flexível de Manufatura Didática**. Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia Elétrica - PPGEEL, Universidade do Estado de Santa Catarina - UDESC, Joinville, SC, 2008.

CURY, J. E. R. Teoria de Controle Supervisório de Sistemas a Eventos Discretos. In: SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, V., Canela, RS, 2001.

DA CUNHA, A. E. C. **Contribuições ao Controle Hierárquico de Sistemas a Eventos Discretos**. Tese (Doutorado) - Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2003.

DE QUEIROZ, M. H. **Controle Supervisório Modular de Sistemas de Grande Porte**. Dissertação (Mestrado) - Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2000.

_____. **Controle Supervisório Modular e Multitarefa de Sistemas Compostos**. Tese (Doutorado) - Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2004.

DE QUEIROZ, M. H.; CURY, J. E. R. Modular Control of Composed Systems. In: Proceedings of the American Control Conference, ACC, Chicago, Illinois, USA, 2000.

_____. Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell. In: INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS, 6th., p. 377-382, Zaragoza, Spain, 2002.

DOUGLASS, B. P. **Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns**. Addison-Wesley, 1999.

FABIAN, M.; HELLGREN, A. PLC-based Implementation of Supervisory Control for Discrete Event Systems. In: IEEE CONFERENCE ON DECISION AND CONTROL, 37th. Tampa, Florida, USA, 1998.

FENG, L.; WONHAM, W. M. TCT: A Computation Tool for Supervisory Control Synthesis. In: INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS, 8th., Michigan, USA, 2006.

GAUDIN, B.; DEUSSEN, P. H. Supervisory Control on Concurrent Discrete Event Systems with Variables. In: IEEE AMERICAN CONTROL CONFERENCE, pp. 4274-4279, New York, NY, USA, 2007.

HAREL, D. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming, Vol. 8, pp. 231-274, 1987.

HOLLOWAY, L. E.; KROGH, B. H.; GIUA A. A Survey of Petri Nets Methods for Controlled Discrete Event Systems. Discrete Event Dyn. Syst.: Theory Appl., Vol. 7, No. 7, pp. 151-190, 1997.

HOPCROFT, J. E.; ULLMANN, J. D. **Introduction to Automata Theory, Languages and Computation**. 1st Ed. Addison Wesley Publishing Company, 1979.

IEC (INTERNATIONAL ELECTROTECHNICAL COMMISSION) 60848. GRAFCET Specification Language for Sequential Function Charts, 2002.

IEC (INTERNATIONAL ELECTROTECHNICAL COMMISSION) 61131-3. International Standard, Programmable Logic Controllers - Part 3: Programming Languages, 2003.

JENSEN, K. **Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use**. Volume 1. 2nd Corrected Printing. New York. Springer Verlag, 1997.

JONERLAN, R. C. **Contribuições a Implementação da Estrutura de Controle Modular Local**. Dissertação (Mestrado) - Engenharia de Produção e Sistemas, Pontifícia Universidade Católica do Paraná - PUC/PR, Curitiba, PR, 2007.

LEDUC, R. J. **PLC Implementation of a DES supervisor for a manufacturing testbeb: an implementation perspective**. M.A.Sc. Thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, 1996.

MALIK, P. Generating Controllers from Discrete-Event Models. In: PROCEEDINGS OF THE MODELLING AND VERIFYING PARALLEL PROCESSES MOVEP 02, 5th. Nantes, France, 2002.

MEALY, G. H. A Method to Synthesizing Sequential Circuits. In: Bell Systems Technical Journal. pp. 1045–1079, 1955.

MENEZES, P. B. **Linguagens Formais e Autômatos**. 5ª Ed. Bookman, 2008.

MOORE, E. F. Gedanken-experiments on Sequential Machines. In: Automata Studies, Annals of Mathematical Studies, (34): 129-153, Princeton University Press, Princeton, NJ, 1956.

MOURA, R. S. **Metodologia para Modelagem, Validação e Programação de Controladores Lógicos Industriais usando *Statecharts* Básicos**. Tese (Doutorado) - Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Universidade Federal do Rio Grande do Norte - UFRN, Natal, RN, 2009.

MURATA, T. Petri Nets: Properties, analysis and applications. In: PROC. IEEE, vol. 77, no. 4, pp. 541-580, Apr. 1989.

OGATA, K. **Engenharia de Controle Moderno**. Prentice-Hall, 2003.

PENA, N. P. **Verificação de Conflito na Supervisão de Sistemas Concorrentes usando Abstrações**. Tese (Doutorado) - Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2007.

POSSAN JUNIOR, M. C.; LEAL A. B. Implementation Of Supervisory Control Systems Based On State Machines. In: IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, IEEE ETFA, XIV., Majorca, Spain, 2009a.

_____. Implementação de Sistemas de Controle Supervisório Baseados em Máquina de Estados. In: SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE - SBAI, IX., Brasília, Brasil, 2009b.

RAMADGE, P.; WONHAM, W. The Control of Discrete Event Systems. In: PROCEEDINGS IEEE - SPECIAL ISSUE ON DISCRETE EVENT DYNAMIC SYSTEMS, v. 77, pp. 81-98, 1989.

REISER, C. **O Ambiente GRAIL para Controle Supervisório de Sistemas a Eventos Discretos: Reestruturação e Implementação de Novos Algoritmos**. Dissertação (Mestrado) - Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2005.

REISER, C.; DA CUNHA, A. E. C.; CURY, J. E. R. The Environment Grail for Supervisory Control of Discrete Event Systems. In: INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS, 8th., Michigan, USA, 2006.

RICKER, L.; LAFORTUNE, S.; Genc, S. DESUMA: A Tool Integrating GIDDES and UMDES. In: INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS, 8th., Michigan, USA, 2006.

RUDIE, K. The Integrated Discrete-Event Systems Tool. In: INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS, 8th., Michigan, USA, 2006.

SIVOLELLA, L. F. **Contribuições à Redução de Supervisores para Sistemas a Eventos Discretos**. Dissertação (Mestrado) - Instituto Militar de Engenharia - IME, Rio de Janeiro, Brasil, 2005.

SKOLDSTAM, M.; AKESSON, K.; FABIAN, M. Supervisor Control Applied to Automata Extended with Variables - Revised. Technical Report. Chalmers University of Technology, Gothenburg, Sweden, 2008.

SU, R.; WONHAM, W. Supervisor Reduction for Discrete-Event Systems. In: PROCEEDINGS OF 2001 CONFERENCE ON INFORMATION SCIENCES AND SYSTEMS, v. 14, p. 31-53, 2004.

TEIXEIRA, C. A. **Aplicação da Teoria de Controle Supervisório no Projeto de Controladores para Eletrodomésticos**. Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia Elétrica - PPGEEL, Universidade do Estado de Santa Catarina - UDESC, Joinville, SC, 2008.

VAZ, A. F.; WONHAM, W. On Supervisor Reduction in Discrete-Event Systems. In: INTERNATIONAL JOURNAL OF CONTROL, v. 44, p. 475-491, 1986.

VIEIRA, A. D. Implementação de Estrutura de Controle de Sistema a Eventos Discretos em Controlador Lógico Programável utilizando a Teoria Controle Supervisório Modular Local. Apostila, 31 páginas, 2003.

_____. **Método de Implementação do Controle de Sistemas a Eventos Discretos com Aplicação da Teoria de Controle Supervisório**. Tese (Doutorado) - Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2007.

WAGNER, F.; SCHMUKI, R.; WAGNER, T.; WOLSTENHOLME, P. **Modeling Software with Finite State Machines: A Practical Approach**. Auerbach Publications, 2006.

WONHAM, W. M.; RAMADGE P. J. Modular Supervisory Control of Discrete Event Systems. Mathematics of Control of Discrete Event Systems, v. 1, n. 1, pp. 13-30, 1988.

WOOD, M. M. **Application, Implementation and Integration of Discrete-Event Systems Control Theory**. Thesis (Master Degree) - Control of Discrete-Event Systems Lab, Queen´s University, Kingston, Canada, 2005.

YANG, Y.; GOHARI, P. Embedded Supervisory Control of Discrete-Event Systems. In: IEEE INTERNATIONAL CONFERENCE ON AUTOMATION SCIENCE AND ENGINEERING, pp. 410-415, Edmonton, Canada, 2005.

ZAD, S. H.; KWONG, R. H.; WONHAM, W. M. Fault Diagnosis in Discrete-Event Systems: Framework and Model Reduction. In: IEEE TRANSACTIONS ON AUTOMATIC CONTROL, v. 48, no. 7, 2003.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo