UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ANDRÉ LUCAS SILVA

CONTROLE SUPERVISÓRIO DE AMBIENTES DINÂMICOS UTILIZANDO AUTÔMATOS FINITOS ESTENDIDOS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO

2016

ANDRÉ LUCAS SILVA

CONTROLE SUPERVISÓRIO DE AMBIENTES DINÂMICOS UTILIZANDO AUTÔMATOS FINITOS ESTENDIDOS

Trabalho de Conclusão de Curso de graduação, apresentado a disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Engenharia de Computação do Departamento Acadêmico de Informática - DAINF - da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para obtenção do título de “Engenheiro em Computação”.

Orientador: Prof. Dr. Marcelo Teixeira

PATO BRANCO

2016

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

Às 9 horas do dia 30 de junho de 2016, na sala V006, da Universidade Tecnológica Federal dão Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Marcelo Teixeira (orientador), Cesar Rafael Claure Torrico e Marco Antonio de Castro Barbosa para avaliar o trabalho de conclusão de curso com o título **Controle supervisório de ambiente dinâmicos utilizando autômatos finitos estendidos**, do aluno **André Lucas Silva**, matrícula 1114840, do curso de Engenharia de Computação. Após a apresentação o candidato foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Marcelo Teixeira
Orientador (UTFPR)

_____
Cesar Rafael Claure Torrico
(UTFPR)

_____
Marco Antonio de Castro Barbosa
(UTFPR)

_____
Beatriz Terezinha Borsoi
Coordenador de TCC

_____
Pablo Gauterio Cavalcanti
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

SILVA, André Lucas. Controle supervisório de ambientes dinâmicos utilizando autômatos finitos estendidos. 81 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2016.

Este trabalho investiga uma limitação da *Teoria de Controle Supervisório* (TCS) de *Sistemas a Eventos Discretos* (SEDs) modelados por Autômatos Finitos Determinísticos (AFDs), no que se refere á adaptação da solução de controle em um ambiente que exige que a ação de controle seja dinâmica, se adaptando em tempo de execução. Tal limitação é ilustrada por meio de um exemplo prático, que também é usado para a proposta de uma possível solução para o problema, baseada em modelagem por Autômatos Finitos Estendidos (AFEs). No presente trabalho são abordadas tanto as etapas de modelagem do sistema por AFEs, quanto a síntese da ação de controle, a implementação do controlador em um microcontrolador, e a visualização da solução de controle por meio de um ambiente gráfico.

Palavras-chave: Ambiente Dinâmico, Autômatos Finitos Estendidos, Sistemas a Eventos Discretos, Teoria de Controle Supervisório

ABSTRACT

SILVA, André Lucas. Supervisory control in dynamic enviroments using finite extended automatas. 81 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2016.

This work deals with a limitation of the Supervisory Control Theory (SCT) of Discrete Event Systems (DES) modelling by Deterministic Finite Automata(DFA), as regards the adjustment of the control solution in an environment that requires a dynamic control action, adapting itself at runtime. This limitation is is ilustrated by a practical example, which is also used to proposing a possible solution to the problem, based in modelling by Extended Finite Automata (EFA). On present paper are both approached the steps of systems modelling by EFAs, as the synthesis of control action, the controler implementation in a microcontroler, and the preview of control solution trough a graphic enviroment.

Keywords: Dynamic Environment, Extended Finite Automata, Discrete Event Systems, Supervisory Control Theory

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

TCS	Teoria de Controle Supervisório
SED	Sistema a Eventos Discretos
AF	Autômato Finito
AFE	Autômato Finito Estendido
STC	Sistema de Tempo Contínuo
AFD	Autômato Finito Determinístico
PCS	Problema de Controle Supervisório
PCS-V	Problema de Controle Supervisório com Variáveis
XML	eXtensible Markup Language (Linguagem de marcação extensível)
UART	Universal Asynchronous Receiver Transmitter (Receptor e transmissor assíncrono universal)
bps	bits por segundo

LISTA DE SÍMBOLOS

Σ	Alfabeto finito
Σ^*	Conjunto de cadeias
ε	Cadeia Vazia
L	Linguagem
$\overline{L}$	Prefixo-fechamento de uma linguagem L
Q	Conjunto de estados de um AF
q°	Estado inicial de um AF
Q^ω	Subconjunto de estados marcados de um AF
$\xrightarrow{\sigma}$	Transição entre estados na ocorrência de um evento σ
$L(G)$	Linguagem gerada por um autômato G
L^ω	Linguagem marcada
$\|$	Composição síncrona entre autômatos
Σ_c	Conjunto de eventos controláveis
Σ_u	Conjunto de eventos não-controláveis
S/G	Ação de controle de um supervisor S sobre uma planta G
$\mathrm{dom}(v)$	Domínio de uma variável v
V	Conjunto de variáveis de um AFE
V'	Conjunto de variáveis de próximo estado de um AFE
Q°	Conjunto de estados iniciais de um AFE
$\mathscr{F}$	Conjunto de fórmulas sobre $V \cup V'$
$x \xrightarrow{\sigma:p} y$	Transição entre os estados x e y em um AFE

SUMÁRIO

1 INTRODUÇÃO

Esse capítulo expõe uma visão geral do assunto a ser tratado neste trabalho, bem como a sua contextualização em relação ao estado da arte. Posteriormente, apresenta-se o problema a ser investigado dentro desse tópico, os objetivos e as justificativas dessa pesquisa, enfatizando a sua importância para o atual estado do conhecimento e, por fim, a estrutura do trabalho.

1.1 CONSIDERAÇÕES INICIAIS

A Teoria de Controle Supervisório (TCS) é um método que define formalmente a síntese de controladores para Sistemas a Eventos Discretos (SEDs). Estruturada sobre a teoria dos de Autômatos Finitos (AFs) e Linguagens Regulares, a TCS provê um mecanismo formal para o cálculo de controladores que detêm certas propriedades qualitativas, como a máxima permissividade e o não bloqueio, características importantes em aplicações práticas. Na TCS, o sistema a ser controlado é modelado por um AF que é denominado planta. As ações de controle a serem exercidas sobre o sistema também são modelados por AFs e são denominadas especificações. A aplicação das especificações na planta é dada por uma entidade denominada supervisor, o qual é calculado a partir de uma operação matemática executada sobre o modelo que representa a composição de plantas e especificações. O resultado dessa operação é um AF, que sintetiza de maneira ótima as ações de controle desejadas (RAMADGE; WONHAM, 1989).

Mesmo em face às vantagens da TCS, a aplicação dessa teoria em escala industrial esbarra em algumas limitações, como a explosão do espaço de estados que ocorre no processamento de AF extensos, problema clássico que é amplamente investigado em controle supervisório por Fei Z. e Lennartson (2012), Ramadge e Wonham (1987), Queiroz e Cury (2000), Teixeira et al. (2015) e Cury et al. (2015). Além disso, a modelagem de especificações de controle pode ser complexa para ser expressa por AFs, particularmente quando envolve a dependência de dados.

Teixeira et al. (2015, p. 132) também afirmam que “enquanto os AF são uma maneira

gráfica de capturar o controle de estado, a dependência de dados é mais naturalmente modelada utilizando variáveis". Uma variação da TCS que incorpora o uso de variáveis por meio de modelos de Autômatos Finitos Estendidos (AFEs) tem sido investigada na literatura Cheng e Krishnakumar (1996), Chen e Lin (2000), Chen e Lin (2001) e Skoldstam et al. (2007). Essa abordagem tem a finalidade de monitorar, por meio do conjunto de variáveis, os elementos do sistema que contribuem para a sua evolução. A atualização do conjunto de variáveis se dá por meio da inserção de fórmulas lógicas junto às transições que modelam a dinâmica do sistema. Da mesma forma, as ações de controle são expressas levando-se em conta os valores das variáveis.

Alguns trabalhos demonstram a utilização dos AFE utilizando tanto a conceituação matemática quanto exemplos de aplicações, com diversas finalidades na ação de controle, como Yang et al. (2008) que propõem a implementação de um controle supervisório para um sistema modelado por máquinas de estados finitas, utilizando variáveis booleanas e funções guardas, que são responsáveis pela ação de controle, permitindo ou não a ocorrência de um evento analisando o valor booleano das variáveis, dentro de um conjunto válido, no momento da ocorrência do evento. Zhao et al. (2012) demonstra uma aplicação prática da modelagem de um SED utilizando AFE aplicado na logística de uma rede de distribuição de energia elétrica.

Recentemente, muitos trabalhos focam na otimização da TCS aplicada a AFE, como Teixeira et al. (2013) que faz o uso de abstração de variáveis visando diminuir o custo da síntese do controlador e aperfeiçoar a ação de controle. Já os benefícios da abordagem de AFE para a redução da complexidade de modelagem de problemas de controle discreto são explorados em Teixeira et al. (2015), juntamente com a introdução de algoritmos de síntese de controladores que suportam técnicas de abstração de variáveis. Shoaei et al. (2012) mostram a abstração de projeção de eventos de um AFE, com resultados significativos na redução do tamanho do controlador, aumentando assim o desempenho computacional em termos de processamento e uso de memória. Em Ouedraogo et al. (2011) é proposto um algoritmo para sintetizar o controlador em plantas e especificações modeladas por um AFE, fazendo o uso das funções de guardas associadas aos eventos, visando manter a segurança na ação de controle do supervisor modelado por um AFE. Em Voronov e Akesson (2009), é apresentado um modelo de verificação das propriedades da TCS para AFE de maneira eficiente em tempo polinomial. Fei Z. e Lennartson (2012) tratam o problema da explosão do espaço de estados em AFE, entretanto utilizado diagramas binários de decisão (AKERS, 1978) para representar a transição do supervisor modelado por um AFE.

1.2 PROBLEMÁTICA E PLANO DE AÇÃO

Grande parte dos estudos relacionados ao tema se dedica a tratar e aperfeiçoar o uso das variáveis ainda no processo de síntese, com a finalidade de simplificar tanto a modelagem, quanto o cálculo do controlador em si. Com isso, pouco vem sendo estudado sobre o impacto do uso de variáveis nas etapas pós-síntese, ou seja, no projeto de implementação do controlador e exploração da dinâmica das ações de controle em tempo real. Em tese, se uma estrutura de variáveis estiver associada ao sistema de controle e o modelo do sistema, assim como os seus requisitos, reconheça essa estrutura, então seria possível permitir a ocorrência ou não de um determinado evento com base no valor atual da variável. Um reflexo imediato dessa hipótese é que a dinamicidade do sistema passa a estar diretamente atrelada à atualização das variáveis. Do mesmo modo, é esperado que as ações de controle sejam personalizadas em tempo de execução, sem requerer uma completa reestruturação na modelagem e nem o cálculo de um novo controlador para cada contexto de controle.

1.3 JUSTIFICATIVA

Visto que o uso de AFEs na TCS de SEDs compõe uma abordagem atual e de relevância prática e científica, o presente trabalho se insere nesse contexto complementando algumas lacunas observadas na literatura, como é o caso da exploração de uma estrutura de variáveis durante os estágios pós-síntese. Pouco também é abordado pela literatura sobre a implementação dos AFE em problemas reais, traduzindo o controlador modelado para alguma linguagem reconhecida por um controlador físico, bem como estudos sobre o real impacto das variáveis na implementação física.

Assim, o trabalho parte do estágio de modelagem e permeia-se por entre os estágios de síntese e implementação computacional da solução de controle, agregando conhecimentos adquiridos durante o curso, principalmente nas áreas de programação, sistemas de controle e automação, sistemas microcontrolados e de teoria computacional, com o conhecimento desenvolvido na literatura, visando abrir mais o campo de pesquisa na questão da implementação.

Com os resultados do trabalho, pretende-se desencadear uma sequência de tópicos para pesquisas futuras na área de modelagem por AFEs, visando simplificar e aperfeiçoar cada vez mais as ações de controle.

1.4 OBJETIVOS

Os objetivos da proposta são divididos em objetivo geral, que remete ao resultado final do trabalho e objetos específicos, que compões os passos intermediários para que se alcance a conclusão do trabalho.

1.4.1 OBJETIVO GERAL

Investigar como um AFE pode ser utilizado para a construção e para o projeto de implementação de um supervisor cuja ação de controle sobre um SED é auto adaptável conforme um conjunto de variáveis. Ilustrar a abordagem por meio do controle de um SED cuja evolução integra diferentes contextos, reconhecidos e adaptados dinamicamente pelo controlador, em tempo de execução.

1.4.2 OBJETIVOS ESPECÍFICOS

- Demonstrar a síntese convencional de controladores para SEDs usando AFs;
- Identificar as limitações da síntese convencional para aplicações que envolvem leituras e chaveamentos de contexto;
- Demonstrar a síntese de controle de SEDs usando AFEs;
- Exemplificar, via modelagem e simulação, a abordagem de síntese com AFEs por meio de um exemplo que simule um ambiente dinâmico;
- Projetar a implementação computacional da solução para o exemplo;
- Ilustrar na prática o comportamento do sistema sob controle.

1.5 ESTRUTURA DO TRABALHO

Este documento se organiza da seguinte forma. O Capítulo 2 apresenta os conceitos que fundamentam o estado da arte em que o trabalho está inserido, com o intuito de oferecer um suporte teórico para o entendimento do trabalho. No Capítulo 3 é descrita a metodologia utilizada para se alcançar os objetivos propostos, bem como os materiais que serão utilizados para o mesmo proposito. O Capítulo 4 descreve as propriedades teóricas desenvolvidas, com a finalidade de resolver o problema proposto e apresenta o desenvolvimento prático do trabalho, envolvendo as etapas de modelagem, simulação, desenvolvimento e aplicações

práticas, encerrando com uma discussão sobre os resultados apresentados. Por fim, o Capítulo 5 apresenta as considerações finais acerca do trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta a conceituação que envolve o estado da arte do tema do trabalho. Inicialmente, define-se “sistema” conforme a literatura e realiza-se um comparativo entre os SEDs e os sistemas dirigidos pelo tempo, na Seção 2.1. Em seguida, os SEDs são explorados de modo a enfatizá-los, por se tratarem do objeto de estudo deste trabalho, com a Seção 2.2 tratando da modelagem de SEDs e apresentando uma justificativa para a escolha de modelagem por autômatos e linguagens. As seções 2.3 e 2.4 introduzem os conceitos de linguagens formais e autômatos, respectivamente. A Seção 2.5 apresenta a teoria de controle supervisório para SEDs modelados por Autômatos Finitos Determinísticos. A Seção 2.6 trata uma limitação da modelagem por AFD, utilizando um ambiente dinâmico de controle, e introduzindo uma alternativa para a síntese de controle para tal situação.

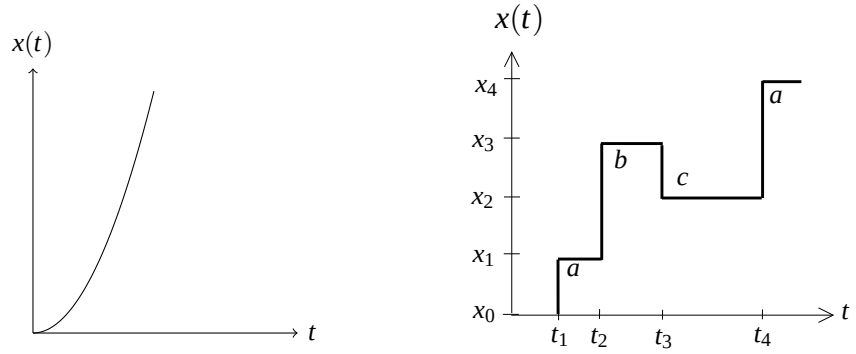
2.1 DUAS IMPORTANTES CLASSES DE SISTEMAS

De acordo com Ogata (2010, p. 2) “um sistema é uma combinação de componentes que atuam em conjunto e realizam um certo objetivo. Um sistema não é limitado apenas a algo físico. O conceito de sistema pode ser aplicado a fenômenos abstratos e dinâmicos”. O avanço tecnológico permitiu o desenvolvimento de sistemas para executar determinadas tarefas pré-estabelecidas para as mais diversas aplicações como, por exemplo, para o controle e monitoramento de sinais analógicos, logística, processamento de dados, robótica, sistemas operacionais e telecomunicações. Em comum, os sistemas têm a característica de envolver um conjunto de estados, denominado *espaço de estados*, representando o contexto do sistema em uma determinada circunstância, e a *transição de estados*, responsável pela evolução do sistema. O que se difere nos sistemas é a maneira como estas transições entre estados ocorrem (TEIXEIRA, 2013, p. 37).

Nos *Sistemas de Tempo Contínuo* (STC), a dinâmica entre os estados são regidas pelo tempo. Assim para cada instante de tempo o sistema se encontra em um determinado estado (OGATA, 1994, p .1), o que remete tais sistemas a modelos de representação baseados no tempo,

como, por exemplo, por meio de equações diferenciais (OGATA, 2010, p. 27) (DORF; BISHOP, 2009, p. 31-34). Os STC podem representar grandezas físicas, como corrente elétrica, pressão, temperatura, etc.

Já nos SEDs, diferentemente dos STC, as transições entre estados são regidas pela ocorrência de estímulos denominados *eventos*, que podem ocorrer de maneira assíncrona e normalmente em intervalos irregulares de tempo (RAMADGE; WONHAM, 1989, p. 81-82). Na ocorrência de um evento, o sistema muda instantaneamente para outro estado, onde permanece até a ocorrência de um próximo evento, que possa ocasionar outra transição. Assim, o sistema não depende da passagem de tempo para mudar de estado (CURY, 2001, p. 7). A Figura 1 apresenta um comparativo entre SEDs e STCs.



(a) Sistema de tempo contínuo (b) Sistema a eventos discretos

Figura 1 – Comportamentos de um STC e de um SED

Na Figura 1(a) é apresentado o comportamento de um determinado STC, com um eixo representando a evolução do tempo, denominado de t, e outro representando o espaço de estados, denominado de $x(t)$. Como pode ser analisado, para cada instante de tempo t o sistema está relacionado a um valor no espaço de estados $x(t)$, demonstrando a continuidade do espaço de estados.

A Figura 1(b) apresenta o comportamento típico de um SED, com um eixo representando a evolução do tempo, denominado de t, e outro representando o espaço de estados, denominado de $x(t)$ e o respectivo conjunto de eventos deste sistema $\{a,b,c,d\}$, responsáveis pelas transições entre estados. Pode-se notar que a ocorrência de um determinado evento, em estados distintos, não leva ao sistema necessariamente a um mesmo estado. Por exemplo, no estado x_0 a ocorrência do evento a leva ao estado x_1. Entretanto, no estado x_2, a ocorrência do mesmo evento a leva ao estado x_4. Outra característica importante analisada, é de que o sistema pode permanecer um tempo arbitrário em determinado estado, realizando uma transição apenas na ocorrência de outro evento, tornando assim o sistema assíncrono. Por fim, o instante de tempo da ocorrência de um evento torna-se indefinido (CURY, 2001, p. 10-11).

2.2 MODELAGEM DE SEDS

Independente da natureza dos sistemas, a realização de análises e experimentos sobre a sua estrutura real pode não ser trivial e nem sempre é possível. É comum que um sistema seja extenso, complexo, ou mesmo insalubre para a ação humana. Ainda, em geral algumas partes do comportamento de um sistema podem ser menos relevantes para determinados contextos, podendo ser abstraídas para fins de análise (TEIXEIRA, 2013, p. 38). Essa representação em níveis de abstração é definida como um *modelo do sistema*, e a ação de representar o sistema por um modelo é denominada de *modelagem*.

Dentre alguns formalismos existentes para a modelagem de SEDs já foram desenvolvidos, citam-se as *Redes de Petri*(MURATA, 1989), (DESROCHERS; AL-JAAR, 1995), a *Teoria das Filas* (BOLCH et al., 2000) e a *Teoria dos Autômatos e Linguagens* (HOPCROFT; ULLMAN, 2001), (ROSA, 2010). Cada formalismo tem suas características e nenhum é aceito de maneira unânime ou como um padrão de modelagem para SEDs.

Este trabalho adota a modelagem por *Autômatos e Linguagens*, que representa o sistema por uma estrutura de transição de estados que facilita especificar quais eventos podem ocorrer ou não em determinado estado. Além disso, essa abordagem fundamenta a sínteses de controladores ótimos, objeto deste trabalho, o que a torna oportuna para ser investigada(CURY, 2001, p. 12).

2.3 TEORIA DE LINGUAGENS FORMAIS

Em um SED, a transição entre estados é regida pela ocorrência de eventos. Sendo assim, uma trajetória que leva o sistema de um estado para outro, é formada por uma sequência finita de eventos denominada de *cadeia*. A análise de uma cadeia gerada pela evolução de um SED, é importante pois descreve o próprio comportamento do sistema, de modo que as operações sobre o modelo de um SED consistem de manipulações sintáticas em suas cadeias (TEIXEIRA, 2013, p. 39). Inerente à ideia de cadeia estão os conceitos que se seguem.

2.3.1 ALFABETO

Um *alfabeto* é o conjunto de eventos que compõem um SED. Denota-se por Σ um alfabeto finito e não-vazio. O conjunto de todas as possíveis cadeias finitas compostas por eventos de Σ é denotado por Σ^*, incluindo a *cadeia vazia*, denotada por ε, que corresponde a uma sequência sem eventos (HOPCROFT; ULLMAN, 2001, p. 28-29).

2.3.2 LINGUAGEM

Para Lewis e Papadimitriou (1998, p. 44) uma *linguagem* L definida sobre um alfabeto Σ é um subconjunto de cadeias em Σ^*. Por exemplo, seja o alfabeto $\Sigma = \{\alpha, \beta, \gamma, \eta\}$, uma possível linguagem sobre Σ seria $L_1 = \{\varepsilon, \beta, \eta\alpha, \alpha\alpha\beta\eta\gamma\}$, sendo que β, $\eta\alpha$ e $\alpha\alpha\beta\eta\gamma$ são cadeias formadas por eventos em Σ. Uma característica importante de uma linguagem é que mesmo que o alfabeto seja finito, o conjunto de cadeias da linguagem pode ser infinito. Para o mesmo alfabeto anterior, pode ser obtida uma outra linguagem $L_2 = \{\varepsilon, \alpha, \beta, \alpha\beta, \beta\beta, \alpha\alpha\eta, \gamma\alpha, \cdots\}$. Assim, a linguagem L_2 é um conjunto infinito formado por cadeias finitas (CASSANDRAS; LAFORTUNE, 2008, p. 55) (HOPCROFT; ULLMAN, 2001, p. 30-31).

Por se estruturar basicamente sobre cadeias de eventos, uma linguagem se torna uma escolha natural para a modelagem formal de SEDs. Como em geral, o comportamento dos SEDs contempla um número finito de elementos, é de interesse que a linguagem que represente um SED também possa ser expressa como um conjunto finito de recursos. Isso remete à classificação das linguagens dentro de duas perspectivas distintas, a saber Lewis e Papadimitriou (1998, p. 50):

- *Regularidade*: Uma linguagem é dita ser regular quando pode ser expressa por uma expressão regular definida sobre um conjunto finito de elementos, mesmo que o número de possíveis cadeias que possam compor essa linguagem seja infinito;
- *Não-regularidade*: Uma linguagem é dita ser não-regular quando nem sempre pode ser representada por alguma estrutura finita de elementos.

Dessa maneira, a representação por uma linguagem regular torna-se apropriada para um SED (TEIXEIRA, 2013, p. 43). Uma maneira de reconhecer uma linguagem regular é por meio dos (AFDs) (HOPCROFT; ULLMAN, 2001),(LEWIS; PAPADIMITRIOU, 1998), (ROSA, 2010), apresentados na Seção 2.4.

2.3.3 OPERAÇÕES SOBRE CADEIAS E LINGUAGENS

Por ser uma linguagem um conjunto, todas as operações convencionais sobre conjuntos, como por exemplo a união e a intersecção, estão automaticamente definidas. A somar com essas operações, também podem ser aplicadas as linguagens operações para lidar com tipos especiais de conjuntos cujos elementos tem a característica de serem cadeias de eventos (CURY, 2001, p. 26) (CASSANDRAS; LAFORTUNE, 2008, p. 56).

2.3.3.1 OPERAÇÕES MORFOLÓGICAS SOBRE CADEIAS

Seja um alfabeto Σ e uma cadeia $s = \alpha\beta\gamma$ com $\alpha, \beta, \gamma \in \Sigma^*$, então:

- α é o prefixo de s;
- β é uma subcadeia de s;
- γ é o sufixo de s;

2.3.3.2 CONCATENAÇÃO DE LINGUAGENS

Sejam duas linguagens $L_1, L_2 \subseteq \Sigma^*$, então a concatenação L_1L_2 das duas linguagens é definida como:

$$L_1L_2 = \{s \in \Sigma^* : (s = s_1s_2) \wedge (s_1 \in L_1) \wedge (s_2 \in L_2)\} \quad (1)$$

Assim, uma cadeia de eventos em L_1L_2 pode ser escrita como a concatenação de uma cadeia em L_1 com uma cadeia em L_2.

2.3.3.3 PREFIXO-FECHAMENTO

Seja uma linguagem $L \subseteq \Sigma^*$ então o seu prefixo-fechamento $\overline{L}$ é definido por:

$$\overline{L} = \{s \in \Sigma^* : (\exists t \in \Sigma^*) \text{ tal que } st \in L\} \quad (2)$$

Em palavras, $\overline{L}$ consiste de todos prefixos de todas as cadeias em L. Em geral, $L \subseteq \overline{L}$, e L é dita prefixo-fechada se $L = \overline{L}$. Assim, L é prefixo-fechada se qualquer prefixo de qualquer cadeia de eventos em L também seja uma cadeia de L (CASSANDRAS; LAFORTUNE, 2008, p. 56). Por exemplo, seja $\Sigma = \{\alpha, \beta, \gamma\}$ e as linguagens $L_1 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta\beta\}$ e $L_2 = \{\varepsilon, \gamma, \gamma\beta\beta\}$. L_1 é prefixo-fechada pois o prefixo de qualquer uma de suas cadeias, também é uma cadeia de L_1. Entretanto, L_2 não é prefixo-fechada pois a cadeia $\gamma\beta$ que é prefixo da cadeia $\gamma\beta\beta$, não pertence a L_2.

2.4 AUTÔMATOS FINITOS DETERMINÍSTICOS

Um autômato finito é um diagrama de transição de estados, que permite modelar, de maneira intuitiva, diversos problemas computacionais . Um AF contém um conjunto de estados, no qual a transição entre um estado para um outro é realizada na ocorrência de um evento (LEWIS; PAPADIMITRIOU, 1998, p. 55-63)(HOPCROFT; ULLMAN, 2001, p.37).

Formalmente um AFD pode se descrito como 5-tupla $\langle \Sigma, Q, q^{\circ}, Q^{\omega}, \rightarrow \rangle$ em que:

- Σ é o alfabeto finito;
- Q é o conjunto finito de estados;
- $q^{\circ} \in Q$ é o estado inicial;
- $Q^{\omega} \subseteq Q$ é o subconjunto de estados finais;
- $\rightarrow \subseteq Q \times \Sigma \times Q$ é a relação de transição de estados.

Uma transição é denotada formalmente por $q_1 \xrightarrow{\sigma} q_2$, representando que a ocorrência de um evento $\sigma \in \Sigma$ move o autômato do estado q_1 para o estado q_2 (TEIXEIRA, 2013, p. 43-44).

O modelo formal descrito acima pode ser ilustrado por um modelo gráfico no formato de grafos dirigidos (BANG-JENSEN; GUTIN, 2007), com os vértices representando os estados e as arestas representando as transições entre estados na ocorrência de um determinado evento. Essa representação visa simplificar a visualização e a analise do autômato (CURY, 2001, p. 32)(CASSANDRAS; LAFORTUNE, 2008, p. 59). Por exemplo, o seguinte AFD pode ser representado graficamente pelo grafo apresentado na Figura 2.

- Alfabeto $\Sigma = \{a, b, c\}$;
- Conjunto de estados $Q = \{Q_0, Q_1, Q_2\}, q^{\circ} = Q_0$ e $Q^{\omega} = \{Q_2\}$;
- Com as seguintes transições: $Q_0 \xrightarrow{a} Q_1, Q_0 \xrightarrow{b} Q_2, Q_1 \xrightarrow{a} Q_2, Q_2 \xrightarrow{c} Q_0, Q_1 \xrightarrow{b} Q_1$

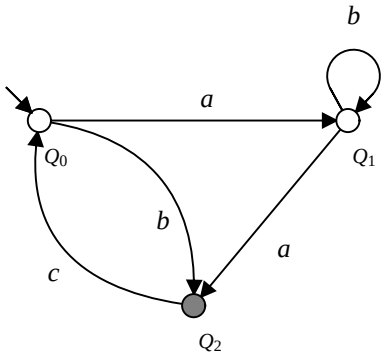


Figura 2 – Exemplo de um AFD.

Com o modelo gráfico, a visualização e a análise da evolução do autômato tornam-se mais simplificadas. Inicialmente, o autômato se encontra no estado Q_0 a partir do qual autômato pode transitar para Q_1 ou Q_2, dada a ocorrência de eventos distintos, a e b, respectivamente. O

estado Q_1 contém um *auto-laço*, no qual a ocorrência do evento b faz o autômato permanecer no estado Q_1. O estado Q_2 é denominado de estado marcado definindo que a tarefa designada pelo autômato está completa (CASSANDRAS; LAFORTUNE, 2008, p. 60).

2.4.1 AUTÔMATOS COMO RECONHECEDORES DE LINGUAGENS

Por implementar a noção de estados, eventos e transições, o comportamento de um autômato pode ser descrito pelas diferentes cadeias de eventos produzidas pela transição entre estados. Assim, o comportamento de um autômato G pode ser expresso em termos de linguagens, classificadas em (BOUZON, 2004, p. 10-11), (TEIXEIRA, 2013, p. 45):

- *Linguagem gerada* $L(G)$: A linguagem gerada envolve todas as possíveis cadeias que podem ocorrer em G, a partir de um estado inicial, independentemente de levar para um estado marcado. Formalmente é definida como:

$$L(G) = \{s \in \Sigma^* : q^\circ \xrightarrow{s} q \in Q\} \tag{3}$$

- *Linguagem marcada* L^ω: A linguagem marcada é o conjunto que contempla todas as cadeias que transitam o automato do estado inicial para algum estado pertencente ao conjunto de estados marcados. Formalmente é definida como:

$$L^\omega(G) = \{s \in \Sigma^* : q^\circ \xrightarrow{s} q \in Q^\omega\} \tag{4}$$

Assim, por exemplo, no autômato da Figura 2, as cadeias $a, ab, abac$ pertencem ao conjunto de linguagens geradas, pois levam a transição do estado inicial Q_0 para qualquer estado que não seja um estado marcado, no caso Q_2. Já as cadeias b, aa, aba fazem parte do conjunto de linguagens marcadas, pois levam o autômato do estado Q_0 para o estado Q_2.

A partir da linguagem obtida do autômato, pode-se analisar a equivalência entre autômatos. Sejam dois autômatos G_1 e G_2, se $L(G_1) = L(G_2)$ e $L^\omega(G_1) = L^\omega(G_2)$, então G_1 e G_2 são ditos autômatos equivalentes (CURY, 2001, p. 36). Outra associação em AFD e linguagens, é que toda linguagem definida por um AFD, pode ser definida por uma linguagem formada por uma expressão regular (HOPCROFT; ULLMAN, 2001, p. 90-91). Cury (2001, p. 36) reforça essa definição a partir do teorema de Kleene: " Se a linguagem L é regular, existe um autômato G com número finito de estados tal que $L^\omega(G) = L$. Se G tem um número finito de estados, então $L^\omega(G)$ é regular."

2.4.2 COMPOSIÇÃO SÍNCRONA DE AUTÔMATOS

Sejam dois autômatos $A = \langle \Sigma_A, Q_A, q_A^\circ, Q_A^\omega, \rightarrow_A \rangle$ e $B = \langle \Sigma_B, Q_B, q_B^\circ, Q_B^\omega, \rightarrow_B \rangle$, a composição síncrona entre A e B, denotado por $A \,\|B$, gera o autômato

$$A \,\| \, B = \{\Sigma_A \cup \Sigma_B, Q_A \times Q_B, (q_A^\circ, q_B^\circ), Q_A^\omega \times Q_B^\omega, \rightarrow\}, \tag{5}$$

tal que a relação de transição no modelo composto é definida como segue:

- $(q_A, q_B) \xrightarrow{\sigma} (q_A^l, q_B^l)$, se $\sigma \in \Sigma_A \cap \Sigma_B$. Ou seja, se o evento σ pertence aos alfabetos de ambos autômatos, então ele levará a uma transição em ambos autômatos;
- $(q_A, q_B) \xrightarrow{\sigma} (q_A^l, q_B)$, se $\sigma \in \Sigma_A \setminus \Sigma_B$. Se o evento σ pertence ao alfabeto de A mas não de B, então ele levará a uma transição apenas no estado gerado a partir de A.
- $(q_A, q_B) \xrightarrow{\sigma} (q_A, q_B^l)$, se $\sigma \in \Sigma_B \setminus \Sigma_A$. Se o evento σ pertence ao alfabeto de B mas não de A, então ele levará a uma transição apenas no estado gerado a partir de B.

Se duas transições a serem compostas forem ambas rotuladas com um mesmo evento σ comum a Σ_A e Σ_B, então as transições são sincronizadas nos autômatos A e B. Do contrário, se σ pertencer a apenas a um dos alfabetos, a sua ocorrência será assíncrona em A e B, ou seja, é executada de modo independente em cada autômato e só evolui aquele cujo alfabeto contempla tal evento. Por exemplo se $\sigma \in \Sigma_A$, porém $\sigma \notin \Sigma_B$, sua ocorrência leva a uma transição no autômato A, porém não tem efeito sobre o autômato B. Se $\Sigma_A \cap \Sigma_B = \emptyset$, os autômatos A e B são ditos assíncronos, sem a existência de eventos síncronos entre os autômatos (CURY, 2001, p. 46) (TEIXEIRA, 2013, p. 46).

O produto síncrono em (5) apresenta operações sobre apenas dois autômatos, entretanto o mesmo pode se aplicar para um número maior de autômatos. Assim, sejam os autômatos $G_i = \langle \Sigma_i, Q_i, q_i^\circ, Q_i^\omega, \rightarrow_i \rangle$, para $i = 1, \cdots, n$, a composição síncrona global pode ser obtida a partir de:

$$G = \Big\|_{i=1}^{n} G_i, \text{ para } \Sigma = \bigcup_{i=1}^{n} \Sigma_i \tag{6}$$

Uma propriedade a ser levada em conta da composição síncrona é que o modelo composto deve preservar os comportamentos individuais das linguagens geradas e marcadas de cada autômato. Assim, a composição síncrona é útil para representar sistemas complexos modularmente, podendo ser modelados por subsistemas, em geral mais simples, e compostos posteriormente para que se obtenha o modelo global do sistema(TEIXEIRA, 2013, p. 45-46).

2.4.3 ACESSIBILIDADE E BLOQUEIO EM AUTÔMATOS

Uma característica importante de um AFD é a acessibilidade de seus estados, ou seja, a partir do estado inicial, é importante saber quais dos estados posteriores podem ser alcançados (CURY, 2001, p.36-37). Dado um AFD $G = \langle \Sigma, Q, q^\circ, Q^\omega, \rightarrow \rangle$, podem ser definidos os seguintes tipo de acessibilidade (CASSANDRAS; LAFORTUNE, 2008, p. 65):

- *Estado acessível*: um estado $q \in Q$ é acessível se $q^\circ \xrightarrow{s} q$ para algum $s \in \Sigma^*$. Em palavras, um estado q é dito acessível se existe uma cadeia que transite o estado inicial q° até q.
- *Autômato acessível*: se $\forall q \in Q$, q for acessível, então G é dito acessível. Ou seja, se todos os estados forem acessíveis, então o autômato é acessível como um todo.
- *Autômato co-acessível*: se cada cadeia $s \in L(G)$ pode ser completada por algum $r \in \Sigma^*$ tal que $sr \in L^\omega(G)$, então G é dito ser co-acessível. Em outras palavras, um autômato é co-acessível se a partir de qualquer estado, exista ao menos um caminho que leve a um estado marcado. A co-acessibilidade também pode ser descrita por $L(G) = \overline{L^\omega(G)}$ e essa propriedade também é conhecida como não-bloqueio.
- *Autômato Trim*: G é dito *trim* se ele é acessível e também co-acessível.

A partir de G, pode ser obtida a sua componente acessível G_{ac} e co-acessível G_{co}. G_{ac} pode ser obtida a partir da remoção dos estados não-acessíveis de G, bem como as transições associadas a este estado. De maneira semelhante, eliminando os estados não co-acessíveis de G, bem como suas transições associadas, se obtém G_{co}(CURY, 2001, p. 37).

2.5 TEORIA DE CONTROLE SUPERVISÓRIO

Estruturada na teoria dos autômatos e linguagens, a *Teoria de Controle Supervisório* (TCS) (RAMADGE; WONHAM, 1989) considera que um conjunto de autômatos é usado para descrever o comportamento de um SED (planta) e suas especificações de controle. Então, operações matemáticas são processadas sobre a composição desses autômatos, resultando em uma lógica de controle que possa supervisionar o sistema de maneira tal que a sua interferência no sistema seja minimamente restritiva e não-bloqueante. Portanto, o principal ponto da TCS é permitir a síntese automática de controladores (supervisores) ótimos para SEDs (TEIXEIRA, 2013, p. 49).

2.5.1 PLANTA DO SISTEMA

Para aplicar a TCS em um SED, primeiramente deve ser levado em conta os componentes que compões o sistema. Esses componentes são definidos por Teixeira (2013, p. 50) como “elementos individualmente organizados dentro de um arranjo lógico, de modo que a execução conjunta desses elementos determina o comportamento global do sistema, conhecido como planta”. Sem nenhuma interferência externa ou ação de controle, é dito que a planta representa o comportamento do sistema em malha aberta.

Uma maneira de se obter o modelo da planta de um sistema é modelando os seus componentes individuais na forma de subsistemas, de modo tal que a composição destes subsistemas resulte no comportamento global desejado para a planta. Operacionalmente, utilizando a modelagem por autômatos, os subsistemas podem ser representados por AFDs, e a planta global ser obtida através de uma operação automatizada de composição síncrona entre esses autômatos (CURY, 2001, p. 53-54) (TEIXEIRA, 2013, p.50). Por exemplo, sejam dois autômatos G^1 e G^2, ambos com 2 estados e 2 transições. Então $G = G^1 \parallel G^2$ representa a planta do sistema em malha aberta. A Figura 3 ilustra um exemplo, com os autômatos G^1 e G^2 representando dois subsistemas e o autômato G representando a planta composta, contendo 4 estados e 8 transições.

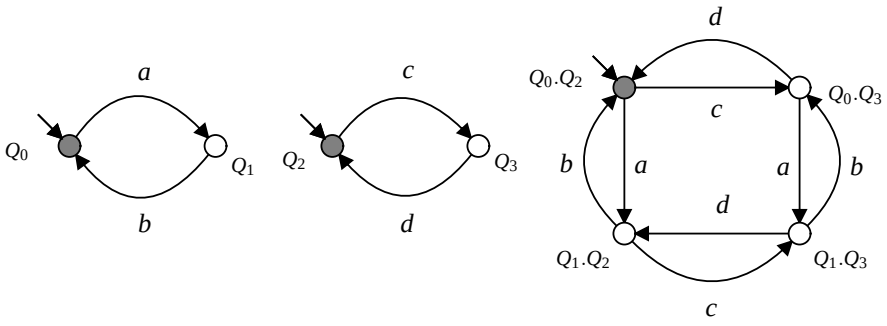


Figura 3 – Modelagem da planta de um SED.

A planta G obtida na Figura 3 irá conter uma linguagem gerada denotada por $L(G)$ que representa o comportamento em malha aberta do sistema.

2.5.2 ESPECIFICAÇÕES

O comportamento em malha aberta em alguns casos pode não ser satisfatório para determinada ação de controle pois a linguagem $L(G)$ de uma planta G pode conter cadeias indesejáveis que podem violar condições de acessibilidade e de não-bloqueio, desejadas para o sistema, ou até mesmo apresentar um comportamento inadequado fisicamente, como por

exemplo a colisão de dois robôs em uma linha de automação (CASSANDRAS; LAFORTUNE, 2008, p. 133). Assim, para se obter um comportamento desejado, são implementadas sobre a planta especificações de controle que executam restrições a serem cumpridas quando o sistema estiver em funcionamento. Ao ser associada ao modelo de uma planta, uma especificação interfere em determinados eventos possíveis em malha aberta, adequando o comportamento da planta aos requisitos de controle. No contexto da TCS, uma especificação representa uma ação proibitiva que observa os eventos da planta e desabilita pontualmente eventos considerados proibidos, quando estes puderem ocorrer (RAMADGE; WONHAM, 1989, p. 85). O comportamento do sistema após a ação de uma especificação sobre uma planta é denominado de malha fechada. Dada a planta G da Figura 3, a Figura 4 apresenta uma possível especificação E que atua sobre G.

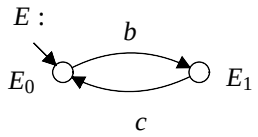


Figura 4 – Modelagem de uma especificação para um SED

Na Figura 4 a especificação E controla a ocorrência dos eventos b e c. Assim, ao ser aplicada sobre a planta G, E vai restringir a ocorrência de c apenas após a ocorrência de b. A especificação também define a *exclusão mútua* de ambos eventos. No estado E_1, o evento b é proibido pela especificação, permitindo assim apenas a ocorrência de c. Da mesma maneira, o estado E_0 proíbe a ocorrência do evento c.

2.5.3 SUPERVISOR

A ação de uma especificação E sobre uma planta G resulta na proibição de determinados eventos a partir da desabilitação destes eventos em G. Entretanto, a planta pode conter alguns eventos cuja ação de controle é ineficaz e assim não podem ser desabilitados de maneira direta. Por exemplo, a quebra de uma máquina em uma linha de produção não depende de qualquer ação de controle, tal que uma possível desabilitação desse evento por parte de E poderia ocasionar em uma inconsistência semântica entre o sistema e sua ação de controle (TEIXEIRA, 2013, p. 52). Portanto, é necessário realizar uma partição no conjunto de eventos, a fim de que ação de controle tenha conhecimento dos eventos passíveis de desabilitação e daqueles que não podem ser desabilitados diretamente.

Assim, a TCS define o particionamento do conjunto de eventos de uma planta em (RAMADGE; WONHAM, 1989, p. 85):

- *Eventos controláveis* denotado por Σ_c é o conjunto de eventos que podem ser desabilitados na planta no momento em que ocorrem.
- *Eventos não-controláveis* denotado por Σ_u é o conjunto de eventos que não podem ser controlados diretamente, portanto não podem ser desabilitados de maneira direta.

Como consequência da partição do conjunto de eventos, o comportamento do sistema em malha fechada também se altera, pois a ação de controle fica agora a cargo de um agente denominado *supervisor*, denotado por *S*. Basicamente, um supervisor sintetiza e integra a controlabilidade de eventos e as leis de controle impostas pelas especificações sobre a planta (CASSANDRAS; LAFORTUNE, 2008, p. 135). Formalmente, um supervisor S é um mapa $S : L(G) \rightarrow 2^{\Sigma}$, associado a uma linguagem $L_S \subseteq L^{\omega}(G)$ que, após qualquer cadeia $s \in L(G)$, observa eventos em G e informa, dentre os eventos possíveis de ocorrer no estado atual, quais devem ser habilitados. Portanto, a ação de controle de S sobre G, denotada por S/G consiste na habilitação de eventos do conjunto de eventos habilitados $S(s) \subseteq \Sigma$ (TEIXEIRA, 2013, p. 52) (CURY, 2001, p. 49). A figura 5 demonstra a ação de controle do supervisor sobre a planta em malha fechada.

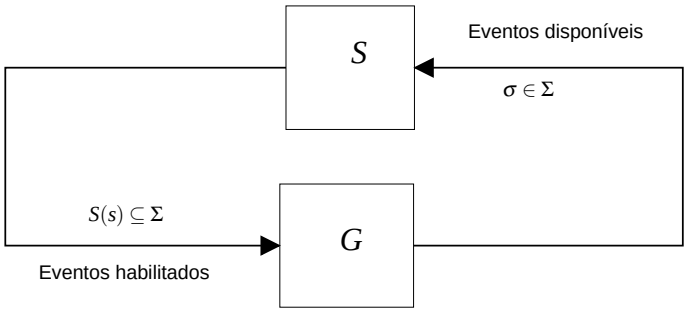


Figura 5 – Ação de controle de um supervisor S sobre uma planta G

Se um evento $\sigma \in \Sigma$ for observado após uma cadeia qualquer $s \in L(G)$, S atualiza o conjunto de eventos habilitados $S(s)$. Caso um evento não faça parte do conjunto $S(s)$ mas seja possível de ocorrer na planta G, então ele está sendo inibido pela ação de controle do supervisor.

O ação de controle S/G apresenta um comportamento gerado em malha fechada, dado por uma linguagem denotada por $L(S/G)$, sendo que $L(S/G)$ é um subconjunto de $L(G)$. O comportamento marcado em malha fechada é dado por $L^{\omega}(S/G) = L(S/G) \cap L^{\omega}(G)$. Assim, $L^{\omega}(S/G)$ consiste de todas as cadeias de $L^{\omega}(G)$ que sobrevive sob a ação de controle S (CASSANDRAS; LAFORTUNE, 2008, p. 136). Caso $L(S/G) = \overline{L^{\omega}(S/G)}$, então o supervisor S é dito não-bloqueante (CURY, 2001, p. 50-51). Dessa maneira, sempre que uma cadeia sobrevive após a ação de controle, essa cadeia é um prefixo de uma cadeia marcada, garantindo que a evolução do sistema sobre o controle do supervisor sempre

leva a finalização de uma tarefa, assim nunca bloqueando o sistema (TEIXEIRA, 2013, p. 53)(CASSANDRAS; LAFORTUNE, 2008, p. 137) .

2.5.4 O PROBLEMA DE CONTROLE SUPERVISÓRIO

O *Problema de controle supervisório* (PCS) (RAMADGE; WONHAM, 1989) é definido em Teixeira (2013, p. 53) como:

"Seja uma planta G, com eventos em Σ*, e uma especificação* $E \subseteq \Sigma^*$*, definindo um comportamento desejado* $K = E \cap L^{\omega}(G)$*, encontre um supervisor não-bloqueante S tal que* $L^{\omega}(G/S) \subseteq K$*".*

Em palavras, deseja-se encontrar um supervisor S que controle a planta G, tal que S seja livre de bloqueio, respeitando um conjunto de especificações modeladas por E, e que, ao mesmo tempo, exerça uma ação de controle minimamente restritiva sobre G

2.5.5 CONTROLABILIDADE

Para a definição de uma solução para o PCS, é necessário o conceito de *controlabilidade* de linguagens. Uma linguagem $K \subseteq \Sigma^*$ é dita ser *controlável* em relação a L se

$$\overline{K}\Sigma_u \cap L \subseteq \overline{K} \tag{7}$$

K é controlável se e somente se $\overline{K}$ for controlável, portanto, a controlabilidade é uma propriedade do prefixo-fechamento de uma linguagem. Assim, a expressão formal de definição de controlabilidade também pode ser escrita como (CASSANDRAS; LAFORTUNE, 2008, p. 147):

$$\forall s \in \overline{K}, \forall e \in \Sigma_{\mathrm{u}}, se \in L \text{ então } se \in \overline{K} \tag{8}$$

As condições apresentadas em (7) e (8) mostram que para qualquer prefixo de uma cadeia em K, se um evento não-controlável é observado em L, a cadeia resultante segue sendo um prefixo de K. Assim, garante-se que nenhum evento não-controlável possível em L seja desabilitado pela ação de controle (RAMADGE; WONHAM, 1989, p. 86). Caso a linguagem K, tal que $K \subseteq L^{\omega}(G)$, seja controlável, então existe um supervisor não-bloqueante S, tal que $L^{\omega}(S/G) = K$ (TEIXEIRA, 2013, p. 53)(CURY, 2001, p. 51).

Entretanto, pode ocorrer que a linguagem K não seja controlável, tornando assim necessário o cálculo de uma sub-linguagem controlável que contenha a maior proximidade possível com K, denominada de *máxima linguagem controlável*.

Seja $\mathscr{C}(K,G) = \{L \subseteq K \mid L \text{ é controlável em relação a } L(G)\}$, é possível determinar um elemento supremo único, denotado por $\sup\mathscr{C}(K,G)$ que representa a máxima linguagem controlável de K em relação a G. Na prática tal linguagem é o comportamento menos restritivo possível de ser implementado por um supervisor não bloqueante S na planta G, respeitando o conjunto de especificações (CURY, 2001, p. 51-52). Assim, $L^{\omega}(S/G) = \sup\mathscr{C}(K,G)$ e $L(S/G) = \overline{\sup\mathscr{C}(K,G)}$ é dito como uma solução ótima para o PCS. No caso de $\sup\mathscr{C}(K,G)$ não poder ser obtido, então o PCS não tem solução para o sistema representado pela planta G (TEIXEIRA, 2013, p. 53).

(RAMADGE; WONHAM, 1984) apresenta um algoritmo para a obtenção da máxima linguagem controlável, em que o $\sup\mathscr{C}(K,G)$ é obtido através de um processo iterativo que identifica os *maus estados* do autômato que representa as especificações.

Um estado x é dito ser um mau estado se, após alguma cadeia $s \in \Sigma^*$ ser verificada em uma planta G e uma especificação E , existe $\mu \in \Sigma_u$, tal que $q \in Q \xrightarrow{s} x \xrightarrow{\mu}$ na planta G e $q \in Q \xrightarrow{s} x$ na especificação E. Em palavras, dada a ocorrência de uma cadeia s qualquer na planta G que leve ao estado x, o estado x contém uma transição caso ocorra um evento não-controlável μ qualquer. Porém o autômato da especificação está inibindo a ocorrência de μ no estado x, o que não pode ser realizado pela ação de controle, pois μ é um evento não-controlável.

Assim, sendo uma planta $G = \langle \Sigma_G, Q_G, q_G^{\circ}, Q_G^{\omega}, \rightarrow_G \rangle$ e uma especificação $E = \langle \Sigma_E, Q_E, q_E^{\circ}, Q_E^{\omega}, \rightarrow_E \rangle$, tal que $L^{\omega}(E) = K \subseteq L^{\omega}(G)$, o algoritmo que calcula $S = \sup\mathscr{C}(K,G)$ é obtido a partir dos seguintes passos:

(i) Identificar maus estados em E. Caso não existam, $S = E$ e fim;

(ii) Caso exista um mau estado, removê-lo em E e em seguida atualizar E sem a presença do estado removido;

(iii) Testar o bloqueio em E verificando se E é acessível ou co-acessível, ou seja, é $trim$. Caso não seja, retornar para o passo (i). Caso seja, $S = E$ e fim.

A execução do algoritmo resulta em $L^{\omega}(S) = \sup\mathscr{C}(E)$ e $L(S) = \overline{L^{\omega}(S)}$, assim o comportamento do supervisor é minimamente restritivo e não-bloqueante e, portanto, uma solução ótima para o PCS.

2.5.6 EXEMPLO DA TCS EM UM SED MODELADO POR AFD

Suponha que o sistema de manufatura de uma pequena fábrica seja composto por duas máquinas, M_1 e M_2, que são interligadas por um *buffer* que suporta estocar até 5 peças empilhadas. Tal sistema é mostrado na Figura 6.

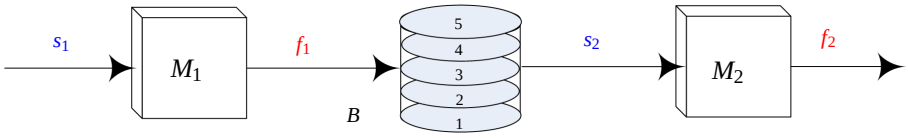


Figura 6 – Ilustração do sistema de manufatura com buffer

Nesse exemplo, a máquina M_1 tem o seu inicio de operação na ocorrência do evento s_1 e o final na ocorrência do evento f_1, que também é responsável por armazenar uma peça no *buffer*. Quando o evento $s2$ ocorrer, a máquina M_2 liga e retira uma peça da pilha do *buffer* e após o processamento dessa peça, M_2 é desligada pelo evento $f2$. M_1 e M_2 são modeladas individualmente como mostra a Figura 7, com ambos modelos contendo 2 estados e 2 transições.

Os modelos M_1 e M_2 consideram apenas os eventos de fim e inicio de cada máquina. Por exemplo, a máquina M_1 inicia desligada no seu estado inicial, e após a ocorrência do evento s_1 a máquina liga. Após a máquina ligar o único evento possível de ocorrer no autômato é f_1 que sinaliza o desligamento da máquina e retorna o autômato para o estado que representa a máquina desligada, que no caso é o estado inicial. O estado inicial também é definido como um estado marcado, pois a máquina somente completa uma tarefa se estiver desligada. Não é desejável assumir que a máquina estando ligada e executando também esteja completando uma tarefa.

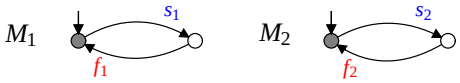


Figura 7 – Modelo das máquinas M_1 e M_2 respectivamente

A planta global do sistema, apresentada na Figura 8, é obtida através da composição $M = M_1 \,||\, M_2$, com o conjunto de eventos $\Sigma = \{s1, s2, f1, f2\}$ em que se assume $\Sigma_c = \{s1, s2\}$ e $\Sigma_u = \{f1, f2\}$.

A planta M da Figura 8, contendo 4 estados e 8 transições, apresenta o comportamento em malha aberta do sistema, sem nenhuma ação de controle. Entretanto, esse comportamento é insatisfatório devido às condições de funcionamento esperadas do sistema. Como objetivo de controle, espera-se evitar o *overflow* e o *underflow* no *buffer*, ou seja, evitar que as máquinas em algum momento tente colocar peças no *buffer* quando estiver cheio, ou retirar quando estiver

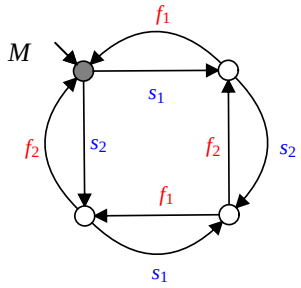


Figura 8 – Modelo da planta global *M* do sistema

vazio. Para isso modela-se a especificação E apresentada na Figura 9, contendo 6 estados e 10 transições.

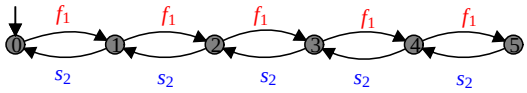


Figura 9 – Especificação de controle *E*

Após definir as especificações de controle, é necessário fechar a malha do sistema, realizando a composição síncrona entre M e E, resultando no comportamento a malha fechada denotado por K, resultando no autômato da Figura 10, contendo 24 estados e 44 transições, sendo 6 estados marcados (sinalizados pela cor cinza) e 2 estados proibidos .

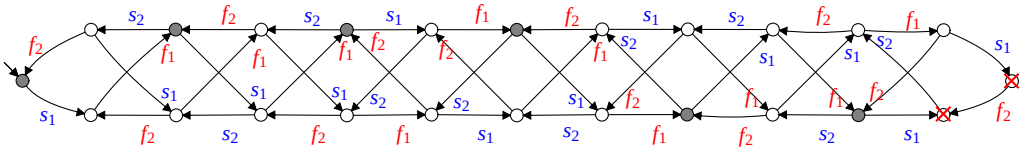


Figura 10 – Ação de controle em malha fechada

Devido aos estados proibidos, é necessário sintetizar de K a máxima linguagem controlável $\sup\mathscr{C}(K)$, representada pelo autômato da Figura 11, em que é obtido a ação de controle desejada não-bloqueante e minimamente restritiva. O autômato de $\sup\mathscr{C}(K)$ contém 22 estados e 40 transições, devido a remoção dos estados proibidos.

2.6 LIMITAÇÕES DA TCS UTILIZANDO AFD

A TCS utilizando AFDs é útil para definir lógicas de controle para diversas aplicações reais. Convencionalmente, o comportamento do controlador é definido já na etapa de modelagem, quando o engenheiro constrói o modelo das especificações para atuar sobre a planta. Assim a semântica de controle é definida na etapa de projeto, antes mesmo da obtenção do controlador, e a ação de controle é estática em tempo de execução. Em algumas situações, porém, é desejável que o comportamento do controlador possa se adaptar ao longo da execução

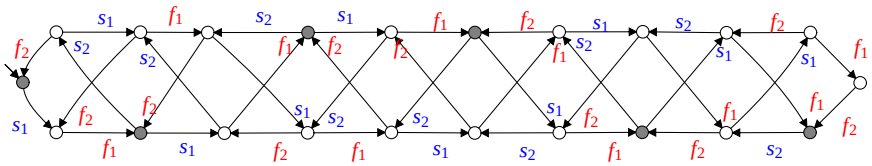


Figura 11 – Modelo da solução ótima de controle para o sistema

do processo, de modo a reconhecer peculiaridades na planta que interfiram nas políticas de controle.

Nesses casos, um controlador sintetizado conforme a TCS convencional pode ser inadequado para aplicações cuja natureza da ação de controle é dinâmica. Sem a característica de auto-adaptação, seria necessário que diversos controladores fossem obtidos previamente, de maneira a tentar antever as diversas soluções de controle que o ambiente poderia exigir. Além disso, é possível que a dinâmica do processo real só seja conhecida em tempo de execução, o que inviabiliza implementações estáticas obtidas *a priori*.

Para investigar tais aspectos em mais detalhes, apresenta-se a seguir um exemplo em que o ambiente pode, em certos casos, exigir um comportamento dinâmico do controlador. Com base nesse exemplo, será motivada a proposta de pesquisa para a etapa conclusiva do trabalho.

2.6.1 UM EXEMPLO MOTIVACIONAL

Este exemplo se inspira no processo apresentado em (RAMADGE; WONHAM, 1989, p. 87), o qual é reproduzido na Figura 12.

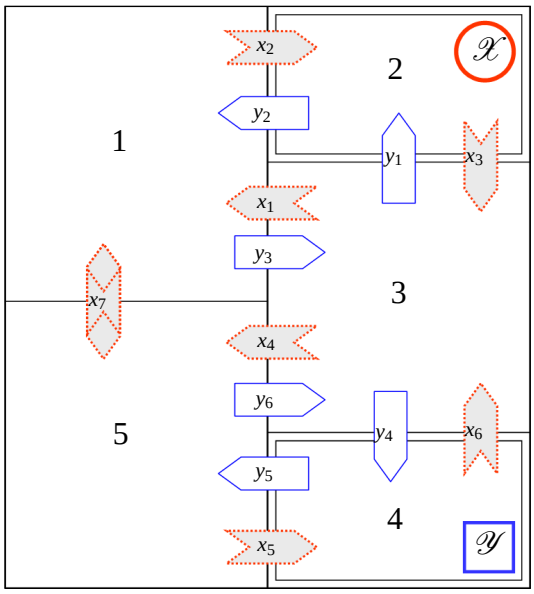


Figura 12 – Ilustração do ambiente

O ambiente explorado é composto de cinco setores conectados por onde transitam dois

robôs, $\mathscr{X}$ e $\mathscr{Y}$. Os robôs inicialmente são posicionados nos setores 2 e 4, respectivamente, que também são os setores marcados para a finalização das tarefa de ambos. As conexões identificadas pela cor cinza, $x_i,\ i = 1,\cdots,7$, são as permitidas para o robô $\mathscr{X}$, e as identificadas pela cor branca $y_j,\ j = 1,\cdots,6$ são permitidas para o robô $\mathscr{Y}$. Todas as transições são unidirecionais, exceto a transição entre os setores 1 e 5 que é bidirecional para o robô $\mathscr{X}$, permitindo que ele transite livremente pelos setores através do evento bidirecional x_7. Assim, para fins de modelagem, assume-se que x_7 é um evento não-controlável enquanto todos os outros são controláveis.

As Figuras 13 e 14 apresentam respectivamente as versões modulares dos autômatos representando o comportamento dos robôs $\mathscr{X}$ e $\mathscr{Y}$ em malha aberta. O comportamento modular de cada planta é semelhante. Por exemplo, a planta $G_{\mathscr{X}}[3]$ representa a movimentação do robô $\mathscr{X}$ pelo setor 2. Na ocorrência do evento x_2, $\mathscr{X}$ transita para o setor 2, representado pelo estado marcado em $G_{\mathscr{X}}[3]$. A saída desse setor é dada pelo evento x_3, retornando $G_{\mathscr{X}}[3]$ para o estado inicial que representa que $\mathscr{X}$ se encontra fora do setor 2.

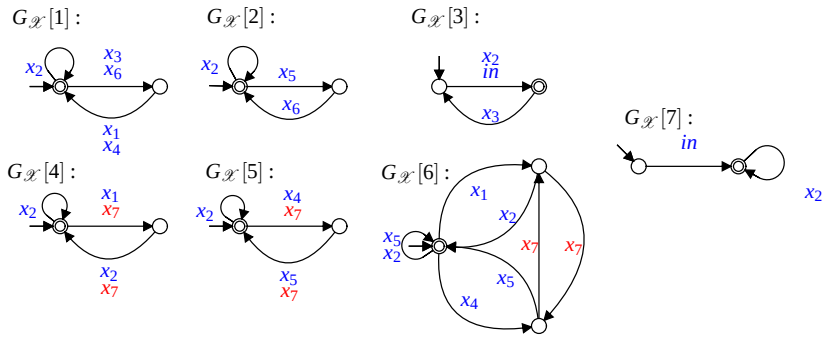


Figura 13 – Autômatos modulares $G_{\mathscr{X}} = \|_{i=1,\cdots,7}\ G_{\mathscr{X}}[i]$ modelando $\mathscr{X}$

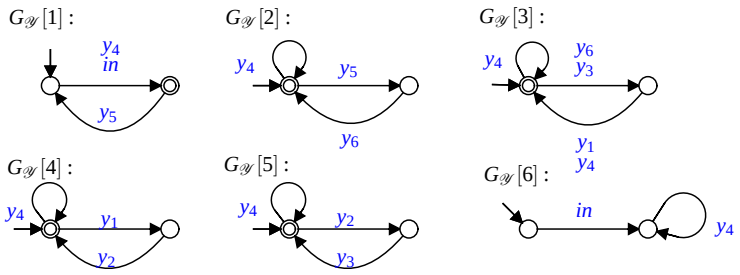


Figura 14 – Autômatos modulares $G_{\mathscr{Y}} = \|_{j=1,\cdots,6}\ G_{\mathscr{Y}}[j]$ modelando $\mathscr{Y}$

A planta do sistema, formada pela composição das plantas modulares, é modelada pelo autômato $G = G_{\mathscr{X}} \,\|\, G_{\mathscr{Y}}$ com 36 estados e 96 transições.

Assume-se que o objetivo de controle para o ambiente é realizar a exclusão mútua dos robôs em cada setor, ou seja, $\mathscr{X}$ e $\mathscr{Y}$ não podem ocupar o mesmo setor, ao mesmo tempo. Para isso, é realizada a modelagem individual de cada especificação para cada setor, obtendo-se

assim as especificações E_i para $i = 1, \cdots, 5$ representando o respectivo setor, como apresenta a Figura 15.

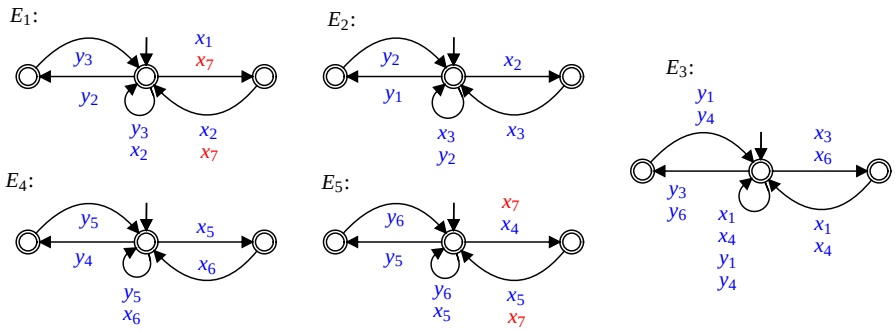


Figura 15 – Especificações para exclusão mútua

Cada especificação E_i expressa a exclusão mútua do respectivo setor i. Por exemplo, a especificação E_2 aplica a exclusão mútua ao setor 2. Na ocorrência do evento y_1, $\mathscr{Y}$ adentra ao setor 2 e a especificação permite apenas a ocorrência de x_2, que representa a entrada de $\mathscr{X}$ no mesmo setor, após $\mathscr{Y}$ deixar o setor na ocorrência do evento y_2.

A composição das especificações $E = \|_{i=1,\cdots,5} E_i$ é um autômato contendo 174 estados e 632 transições. A composição $K = E \,\|\, G$ contém 29 estados e 60 transições, que pode conter um supervisor $\sup\mathscr{C}(K,G)$ modelado por um autômato S contendo 17 estados e 28 transições, como apresentado na Figura 16, representando a solução ótima de controle para este problema.

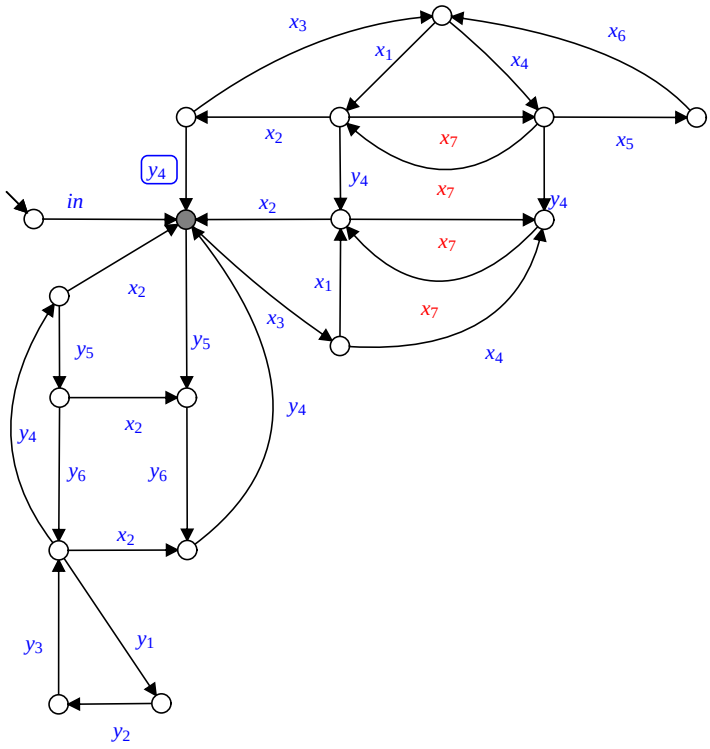


Figura 16 – Ótima solução de controle para a exclusão mútua

2.6.2 UM PROBLEMA MOTIVACIONAL

Para o exemplo anterior, suponha que $\mathscr{X}$ e $\mathscr{Y}$ são robôs coletores que visitam cada setor carregando e descarregando material. Assume-se que apenas os setores 2, 3 e 4 contém material para carga e descarga, enquanto os setores 1 e 5 são apenas para transição e podem ser compartilhados pelos robôs a qualquer instante.

Para esse novo cenário, suponha que o ambiente da Figura 12 pode assumir dois contextos distintos. Quando ambos os robôs $\mathscr{X}$ e $\mathscr{Y}$ estão vazios, eles podem compartilhar os setores 2, 3 e 4 até que ocorra algum carregamento, o que é modelado pelos eventos $l_i^{\mathscr{X}}, i = 2,3,4$ e $l_j^{\mathscr{X}}, j = 2,3,4$. Ao ocorrer o carregamento, é aplicada a exclusão mútua em todos os setores até que ocorra um descarregamento, o que é modelado pelos eventos $u^{\mathscr{X}}$ e $u^{\mathscr{Y}}$, quando os robôs podem voltar a compartilhar o mesmo setor.

Observe que, na ocorrência de um carregamento, o comportamento da ação de controle é similar a apresentada anteriormente, com o intuito de realizar a exclusão mútua. Entretanto, ao ocorrer a descarga, o compartilhamento dos setores é permitido aos robôs, e a ação de controle deve suspender a exclusão mútua.

A solução de controle esperada, para este caso, deve ser flexível e auto-configurável, de tal maneira que inicie a exclusão mútua dos robôs quando um evento de carregamento é observado, e desabilite a exclusão mútua quando um evento de descarregamento é observado, representado uma troca de contexto de controle.

Na prática, a implementação de um controle supervisório com troca de contexto pode ser dispendioso em termos de esforço de modelagem e de custo computacional, pois a ação de controle pode envolver diversas trocas de contextos e, para cada contexto, as especificações de controle se alterem de uma maneira que exija a síntese de um supervisor diferente. Assim, torna-se desejável a obtenção de um supervisor que detenha característica de dinamismo, o que não é o caso do supervisor S obtido anteriormente, que se limita à prática de apenas uma única ação de controle.

Diante desse novo contexto assumido pelo problema de controle, observa-se que a solução obtida conforme a TCS convencional se torna degradada pela necessidade de dinamismo do controlador ao se adaptar ao contexto da planta. Esse tipo de ambiente é mais naturalmente representável utilizando variáveis associadas ao autômato que modela a planta, com a finalidade de processar o contexto de determinada informação que interfira diretamente na ação de controle. A abordagem que incorpora variáveis na TCS é então sugerida como forma de tratar do problema, e é apresentada no Capítulo 4.

3 MATERIAIS E METODOLOGIA

Este capítulo descreve a metodologia utilizada para alcançar os objetivos propostos para o trabalho, bem como a apresentação dos materiais utilizados para alcançar tais propositos.

3.1 METODOLOGIA

Inicialmente, foi realizada a modelagem das plantas e especificações envolvidas no exemplo proposto na Sessão 2.6.2. O exemplo visa coordenar múltiplos robôs coletores, sujeitos a políticas de controle que se alteram conforme o contexto da planta.

Em seguida, foi utilizado o software Supremica [1] para obter um supervisor que sintetiza o comportamento desejado para o sistema sob controle. Relembrando que tal comportamento não apenas deve preservar a controlabilidade e o não-bloqueio, como também atender às especificações e apresentar o dinamismo do supervisor conforme a troca de contexto de controle que a aplicação exige. Ao final da etapa de modelagem, o supervisor foi testado manualmente no Supremica, via simulação, a fim de analisar se de fato reflete o comportamento desejado de controle.

Se o comportamento apresentado não for o esperado, as etapas de modelagem e síntese podem ser revistas, porém, caso seja o esperado, passa-se então para o projeto de implementação do supervisor em um microcontrolador, responsável por controlar efetivamente o sistema. Para isso, é necessário traduzir o comportamento do supervisor para a linguagem de programação *C*, interpretada por um microcontrolador, utilizando-se o software DESLab [2] . Salienta-se que não há na literatura suporte ao projeto de implementação de um supervisor que seja representado por um AFE e nem mesmo o DESLab suporta AFE. Portanto, ao que se estima, essa etapa do trabalho é de inovação científica, o que naturalmente agrega riscos ao projeto.

Caso esta implementação seja satisfatória, tendo a ação de controle obtida e gravada no microcontrolador, passa-se a ilustrar o ambiente da Figura 12, sendo possível visualizar

[1] http://www.supremica.org/

[2] https://sites.google.com/site/controlediscreto9/instaladores

a ação de controle de maneira gráfica. Para isso, na ocorrência de cada evento no sistema, o microcontrolador enviará para a interface gráfica o contexto atual do ambiente, como por exemplo, o setor atual de cada robô, o status de carga e descarga de cada robô e as transições possíveis para cada robô. Ao receber tais informações, a interface ilustra o ambiente de maneira gráfica, evidenciando o contexto atual do sistema e as possibilidade de transições que podem ocorrer com os robôs.

A Figura 17 resume a metodologia do trabalho usando um diagrama.

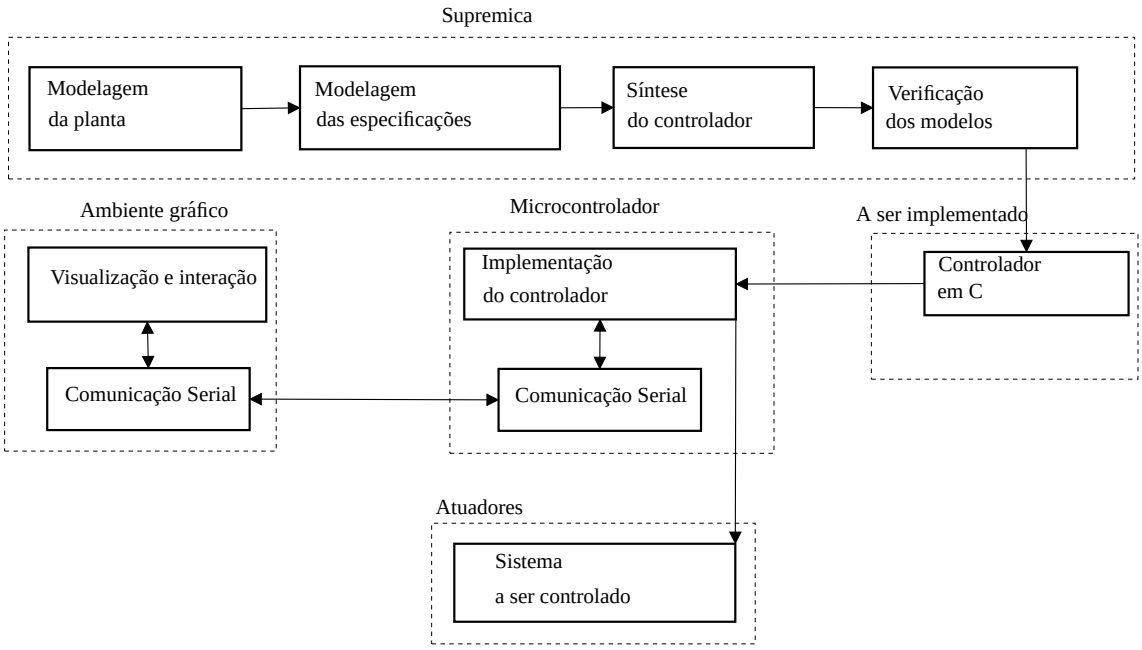


Figura 17 – Diagrama da metodologia do trabalho

3.2 MATERIAIS

Os materiais apresentados abaixo foram adotados no escopo deste trabalho.

3.2.1 AMBIENTE DE SÍNTESE E SIMULAÇÃO

O software *Supremica* (AKESSON et al., 2003) é um ambiente de modelagem e verificação de sistemas. A ferramenta é livre para uso educacional e de pesquisa, escrita na linguagem Java, e será utilizada neste trabalho para a modelagem de plantas, especificações e para a síntese de supervisores. A ferramenta ainda será explorada para a verificação formal de características desejáveis do supervisor, como a controlabilidade e o não bloqueio, importantes para caracterizar adequadamente o trabalho proposto.

Outra propriedade importante do Supremica, é que ele permite simular graficamente

um modelo, facilitando assim a visualização da ocorrência de certos eventos e de como o sistema se comporta como um todo. Isso possibilita a verificação empírica do comportamento desejado e até a descoberta de possíveis falhas antes de projetar a implementação do sistema e de verificá-lo formalmente. Além das características citadas acima o Supremica ainda possui o diferencial de suportar a associação de variáveis ao modelo do sistema, bem como a implementação das fórmulas de guarda e de atualização de variáveis (AKESSON et al., 2006).

3.2.2 INTERPRETAÇÃO DO SUPERVISOR

Uma vez obtido no Supremica um autômato (AFE) que representa o supervisor para o sistema de controle, esse autômato deve ser traduzido para o código interpretável por um microcontrolador. Para tal tarefa, será utilizado o software DESLab(TORRICO, 2016), desenvolvido na *UTFPR*, que permite a construção de autômatos finitos que modelem SEDs e, sobre eles, permite realizar as operações básicas de composição e síntese de supervisores. O Deslab gera saídas textuais e gráficas para visualização dos autômatos e também traduz autômatos para a linguagem de programação *C*, compatível com diversas famílias de microcontroladores.

3.2.3 MICRONCONTROLADOR

O microcontrolador será o dispositivo usado neste trabalho para implementar a ação de controle para ser executada sobre o sistema. O modelo utilizado é o microcontrolador *MSP430G2553* [3], da Texas Instruments, por ser um dispositivo de baixo custo e que pode atender satisfatoriamente às necessidades do projeto. O código interpretável neste modelo de microcontrolador é descrito em linguagem de programação *C*.

3.2.4 SISTEMA SUPERVISÓRIO

Uma vez implementado em nível de hardware, o sistema de controle está apto a ser transposto para o ambiente operacional, de modo que as sequências operacionais dos componentes de atuação passam a ser comandadas por esse supervisor. Logo, uma das formas de ilustrar os resultados deste trabalho seria por meio da observação do próprio sistema real sob controle.

No entanto, exemplificar a proposta usando um sistema real pode ser caro e consumir um tempo demasiadamente grande para o escopo do trabalho. Além disso, essa abordagem

[3] http://www.ti.com/product/msp430g2553

envolveria etapas de construção cujo enfoque ultrapassa os limites do trabalho, como a movimentação efetiva de robôs, sistemas de visão, e demais aspectos operacionalmente importantes, mas menos relevantes para o esquema de coordenação proposto.

Uma alternativa que abrevia a construção física do sistema, mas que preserva a representatividade dos resultados, é a utilização de um simulador de ambiente que representa graficamente e em tempo real o ambiente e seus componentes, coordenados por meio da comunicação com o controlador. Como parte desse trabalho, foi desenvolvido o ambiente apresentado na Figura 12, o qual pode ilustrar o sistema suscetível às ações de controle.

4 CONTROLE SUPERVISÓRIO DE AMBIENTES DINÂMICOS

Este capítulo apresenta a base teórica-formal que sustenta o desenvolvimento do trabalho. A Seção 4.1 introduz o conceito de AFEs, suas propriedades e operações. Algumas definições resultaram desse trabalho enquanto outras assumem e estendem resultados anteriores sobre o tema, em particular as operações de síntese de controle, apresentadas na Sessão 4.2. A Seção 4.3 adapta os AFEs à problemática que envolve o trabalho e apresenta a etapa de modelagem e síntese de controle e por fim, a Seção 4.4 apresenta a etapa de codificação e implementação de controle e desenvolvimento da interface gráfica.

4.1 AUTÔMATOS FINITOS ESTENDIDOS

Um *Autômato Finito Estendido* (AFE) é uma estrutura de estados, similar aos AFs, mas estendida com fórmulas sobre as transições do modelo, responsáveis por atualizar valores de variáveis nos estados alcançados após a transição (OUEDRAOGO et al., 2011, p. 561) (TEIXEIRA et al., 2013, p. 132).

4.1.1 CONJUNTOS DE VARIÁVEIS

Uma *variável* v é uma entidade associada a um domínio $\mathrm{dom}(v)$ e um valor inicial $v^{\circ} \in \mathrm{dom}(v)$. Um conjunto de variáveis $V = \{v_0, \ldots, v_n\}$ contém o domínio $\mathrm{dom}(V) = \mathrm{dom}(v_0) \times \cdots \times \mathrm{dom}(v_n)$, e um elemento de $\mathrm{dom}(v)$ é denotado por $\overline{v} = (\overline{v_0}, \ldots, \overline{v_n}) \in \mathrm{dom}(V)$ com $\overline{v_i} \in \mathrm{dom}(v_i)$, denotado da *valoração*, sendo que uma valoração é uma estrutura que atribui para cada variável $v \in \mathrm{dom}(V)$ um valor correspondente ao domínio.

Por exemplo, seja $V = \{a, b, c\}$ um conjunto de variáveis com $\mathrm{dom}(a) = \mathrm{dom}(b) = \mathrm{dom}(c) = 0, \ldots, 10$. Uma possível valoração (em um caso determinístico) para $\mathrm{dom}(V)$ pode ser, por exemplo $(0,0,0)$, que também é a valoração inicial, ou também $(2,6,4)$. Caso uma variável seja booleana, uma valoração atribui apenas valores *verdadeiro* ou *falso*.

Um segundo conjunto de variáveis, denominado *variáveis de próximo estado* e

denotado por $V' = \{ v' \mid v \in V \}$ com $\text{dom}(V') = \text{dom}(V)$, é usado para descrever como as variáveis são atualizadas pelas transições (TEIXEIRA et al., 2013, p. 132). Assim, em um AFE, quando uma transição é disparada e uma fórmula implementada sobre essa transição atualiza alguma variável, a transição na verdade está alterando o valor de uma variável associada ao estado corrente para um novo valor a ser associado ao próximo estado (TEIXEIRA, 2013, p. 118).

4.1.2 ESTRUTURA LÓGICA DE ATUALIZAÇÃO DE VARIÁVEIS

Seja V um conjunto de variáveis associadas a um AFE, a implementação de fórmulas lógicas para mapear o valor atual da variável definido em V com um valor de próximo estado definido em V' (SKOLDSTAM et al., 2007, p. 3389), na ocorrência das transições, leva em conta a semântica que se deseja expressar. Tal implementação dispõe dos recursos de constantes, operados lógicos, matemáticos e relacionais, além de dispor da própria combinação de variáveis (TEIXEIRA, 2013, p. 119).

Por exemplo, seja:

- α uma variável que representa números inteiros;
- $\text{dom}(\alpha) = \{0, \cdots, 9\}$;
- $\alpha^{\circ} = 0$.

Uma transição com a fórmula de atualização $\alpha' = \alpha + 1$, atualiza no próximo estado o valor atual da variável acrescentada de 1, caso α seja menor que 9. Caso $\alpha = 9$, a adição de uma unidade ao valor 9 está fora de $\text{dom}(\alpha)$, assim a atualização não pode ser efetivada e a transição é desabilitada.

Já uma transição com a fórmula de atualização $\alpha' = 5$ atualiza no próximo estado o valor da variável para 5, independente do seu valor atual. Neste caso, como $5 \in \text{dom}(\alpha)$, esta transição sempre é habilitada. Em contraste, uma fórmula como $\alpha = 5$ compara o valor atual de α com 5 no estado atual, sem realizar nenhuma atualização. Neste caso, se a comparação for verdadeira, a transição é habilitada, caso o contrário, não.

4.1.3 DEFINIÇÃO FORMAL E EXPLÍCITA DE UM AFE

Formalmente, um AFE pode ser descrito por uma 6-upla $G_{\text{v}} = \langle \Sigma, V, Q, Q^{\circ}, Q^{\omega}, \rightarrow \rangle$, sendo:

- Σ o alfabeto de eventos;
- $V = \{v_1, \ldots, v_n\}$ o conjunto de variáveis;
- Q o conjunto finito de estados;
- $Q^\circ \subseteq Q$ o conjunto de estados iniciais;
- $Q^\omega \subseteq Q$ o conjunto de estados marcados;
- $\rightarrow \subseteq Q \times \Sigma \times \mathscr{F} \times Q$ a relação de transição entre estados, onde $\mathscr{F}$ é o conjunto de fórmulas sobre $V \cup V'$.

Denota-se por $x \xrightarrow{\sigma:p} y$ uma transição em $_{\mathrm{v}}$ que parte do estado x para o estado y, com o evento $\sigma \in \Sigma$ e a fórmula $p \in \mathscr{F}$ (TEIXEIRA, 2013, p. 120).

Um AFE $_{\mathrm{v}}$ também pode ser interpretado de maneira explícita como um AF, em que os valores das variáveis são descritos explicitamente como parte de cada estado. Assim, seja um AF $G = \langle \Sigma, Q, Q^\circ, Q^\omega, \rightarrow \rangle$ em que:

- $Q = Q \times \mathrm{dom}(V)$;
- $Q^\circ = Q^\circ \times \{(v_1^\circ, \ldots, v_n^\circ)\}$;
- $Q^\omega = Q^\omega \times \mathrm{dom}(V)$;
- $\rightarrow$ é tal que $(x, \overline{v}) \xrightarrow{\sigma} (y, \overline{v'})$ para $\overline{v}, \overline{v'} \in \mathrm{dom}(V)$, se há $x \xrightarrow{\sigma:p} y$ tal que $p(\overline{v}, \overline{v'}) = \mathit{verdade}$.

A relação de transição é definida com base na relação em G_{v}, porém levando em conta os efeitos das fórmulas lógicas implementadas sobre as variáveis(TEIXEIRA et al., 2013). Por exemplo, seja o AFE da Figura 18 com $v \in V$ sendo $\mathrm{dom}(v) = \{0, 1\}$ e $v^\circ = 0$.

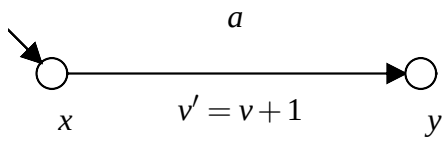


Figura 18 – Exemplo de relação de transição em um AFE

Em x, o valor de v é 0 e na ocorrência do evento a, a fórmula $p : v' = v + 1$ atualiza o valor de v para 1, que pertence a $\mathrm{dom}(v)$. Assim, $p(v, v') = \mathit{verdade}$ portanto a transição $(x, v) \xrightarrow{a} (y, v')$ é habilitada e pode ser equivalentemente reescrita na forma explícita $(x, 0) \xrightarrow{a} (y, 1)$

Na interpretação explícita, os valores de uma variável são descritos explicitamente, como parte de cada estado. Assim, essa representação evidência em termos de números de estados a real dimensão da estrutura do AFE (TEIXEIRA, 2013, p. 120). Por exemplo, a variável b descrita anteriormente também pode ser representada por um autômato, demonstrado na Figura 19, em que o valor de b inicia com 0 no estado inicial do autômato e é alterado para 1 na ocorrência do evento a. O conjunto de estados do autômato da Figura 18 denotado por A, explicitamente é definido como $Q_A = Q \times \mathrm{dom}(b) = \{(x,0),(x,1),(y,0),(y,1)\}$. Como o comportamento de A não permite $b = 1$ no estado x e nem $b = 0$ no estado y, então $Q_A = \{(x,0),(y,1)\}$ com $Q_A^\circ = (x,0)$.

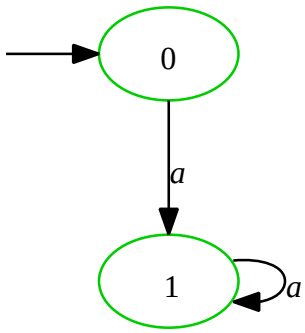


Figura 19 – Automato explícito da variável v, no contexto de A

A relação de transição também pode ser estendida para cadeias em Σ^*, assim para um AFE G_v:

- $(x,\overline{v}) \xrightarrow{\varepsilon} (x,\overline{v})$ para todo $(x,\overline{v}) \in Q$;
- $(x,\overline{v}) \xrightarrow{s\sigma} (x'',\overline{v}'')$ se $(x,\overline{v}) \xrightarrow{s} (x',\overline{v}') \xrightarrow{\sigma} (x'',\overline{v}'')$ para algum $(x',\overline{v}') \in Q$.

Denota-se por $G_\mathrm{v} \xrightarrow{s} (x,\overline{v})$ o estado $(x,\overline{v})$ que pode ser alcançado a partir do estado inicial de G_v. Assim o AFE G_v pode ter linguagens representando o seu comportamento gerado e marcado, definidas por (TEIXEIRA, 2013, p 121) (TEIXEIRA et al., 2013, p. 133):

- $L(G_\mathrm{v}) = \{s \in \Sigma^* \mid q^\circ \in Q^\circ \xrightarrow{s} (x,\overline{v}) \in Q\}$;
- $L^\omega(G_\mathrm{v}) = \{s \in \Sigma^* \mid q^\circ \in Q^\circ \xrightarrow{s} (x,\overline{v}) \in Q^\omega\}$

4.1.4 PROPRIEDADES E OPERAÇÕES DE UM AFE

Tal como os AFs, algumas propriedades podem ser definidas para os AFEs e algumas operações podem ser realizadas, conforme descritas a seguir.

4.1.4.1 PROPRIEDADES DE FÓRMULAS

Em um AFE, é comum que a implementação de uma fórmula use apenas algumas variáveis em um conjunto de variáveis V. Quando uma variável é usada para implementar uma fórmula $f \in \mathscr{F}$, então a variável é dita ser um *alvo* em f. Variáveis alvo podem serem coletadas de fórmulas como se segue.

Definição 1 *Para um AFE* $G_{\mathrm{v}} = \langle \Sigma, V, Q, Q^{\circ}, Q^{\omega}, \rightarrow \rangle$*, seja* $\mathscr{F}$ *o conjunto de fórmulas implementadas em* G_{v} *utilizando variáveis de* $V \cup V'$*. Para uma fórmula* $p \in \mathscr{F}$*, uma variável alvo na fórmula p é definida como*

$$\tau(p) = \{v \in V \mid v \text{ é implementada em } p\}$$

e

$$\tau'(p) = \{v' \in V' \mid v' \text{ é a variável de próximo estado atualizada por } p\}.$$

Por exemplo, dada uma fórmula $p = (x' = y + 1)$, então $\tau(p) = \{x, y\}$ e $\tau'(p) = \{x\}$. Def. 1 pode ser generalizada para todas fórmulas implementadas em uma AFE G_{v} por $\tau(\mathscr{F})$ ou simplesmente por $\tau(G_{\mathrm{v}})$.

As fórmulas em $\mathscr{F}$ podem ser classificadas de acordo com a maneira em que elas interferem nas variáveis. Uma fórmula pode utilizar uma variável para testar alguma condição lógica de acordo com o valor atual da variável, sem alterar este valor. Em contrapartida, uma fórmula também pode atualizar o valor atual de uma variável para algum outro valor definido em seu domínio, sem realizar nenhum teste condicional. Formalmente, tal conceito pode ser expresso como se segue.

Definição 2 *Dado um AFE* $G_{\mathrm{v}} = \langle \Sigma, V, Q, Q^{\circ}, Q^{\omega}, \rightarrow \rangle$*, com fórmulas definidas em* $\mathscr{F}$*, uma fórmula* $p \in \mathscr{F}$ *é dita ser uma* guarda *se* $\tau'(p) = \emptyset$*.*

Nesse caso, a fórmula p não atualiza nenhuma variável. Por exemplo, para a fórmula $p = (x = y + 2)$, têm-se $\tau(p) = \{x, y\}$ e $\tau'(P) = \emptyset$ e, portanto, p é uma guarda. Caso todas as fórmulas em uma AFE forem guardas, a definição que se segue pode ser estabelecida.

Definição 3 *Um AFE* G_{v} *é dito ser* puro *se* $\tau'(G_{\mathrm{v}}) = \emptyset$*.*

Se um AFE é puro, as fórmulas associadas a suas transições são todas guardas e não há nenhum atualização de valor das variáveis. AFE puros são particularmente úteis para

a modelagem de especificações, que visam unicamente restringir um comportamento, sem atualizações.

Por outro lado, para os modelos das plantas é esperado apenas atualização das variáveis, sem nenhuma restrição. Para conseguir tal efeito, pode ser atribuído aos modelos uma estrutura de fórmulas de atualização (sem guardas) implementadas nas transições. Após a ocorrência de uma transição, uma atualização altera o valor da variável e, desde que os valores das variáveis são parte da semântica do sistema, tais alterações de valores acaba por interferir em todo o contexto do sistema.

Ao se trabalhar com atualizações nas plantas, algumas situações incomuns poder ocorrer e interferir na síntese de controle. Por exemplo, pode acontecer de um dado evento σ ser associado com fórmulas que atualizem variáveis distintas, tal como:

$$x_1 \xrightarrow{\sigma:a=a+1} y_1 \text{ e } x_2 \xrightarrow{\sigma:b=b+1} y_2. \tag{9}$$

Ou, ainda, σ pode ser associada com fórmulas que implementam atualizações que sejam divergentes para a mesma variável, como:

$$x_1 \xrightarrow{\sigma:a=a+1} y_1 \text{ e } x_1 \xrightarrow{\sigma:a=a-1} y_1. \tag{10}$$

Para fins práticos de uma ação de controle, é conveniente que situações como (9) e (10) sejam evitadas.

No caso de (9), tal comportamento torna-se indesejado pois interfere na definição de um comportamento semântico desejado para o sistema. Já no caso de (10), não faz sentido a modelagem de uma planta que atualiza uma variável para algum valor não definido. Além que na ocorrência de um composição, as fórmulas de atualização seriam combinadas por conjunção e na maneira que se apresentam, a combinação seria marcada como falsa.

Para prevenir a ocorrência de tais situações na síntese de um controlador, são definidas as propriedades de *normalidade* e de *satisfatibilidade*, como se segue:

Definição 4 *Um AFE* G_{v} *é dito ser* normal *se, para todo* $x_1 \xrightarrow{\sigma:P_1} y_1$ *e* $x_2 \xrightarrow{\sigma:P_2} y_2$*, então* $\tau'(P_1) = \tau'(P_2)$.

Em palavras, um AFE é dito ser normal se as fórmulas implementadas em suas transições associadas a um mesmo evento, sempre atualize o mesmo conjunto de variáveis.

Definição 5 *Seja G_{v} um AFE e seja $\mathscr{F}$ o conjunto de fórmulas implementadas em G_{v} utilizando variáveis de $V \cup V'$. Uma fórmula $f \in \mathscr{F}$ é dito ser* satisfatível *se é verdadeira para pelo menos uma valoração definida em* dom(V)*. Caso todas as fórmulas de G_{v} forem satisfatíveis, então G_{v} é dito ser um autômato satisfatível.*

Por exemplo, seja $V = \{a, b, c\}$ um conjunto de variáveis com $\mathrm{dom}(a) = \mathrm{dom}(b) = \mathrm{dom}(c) = 0, \ldots, 10$ e com a valoração inicial $(0,0,0)$, uma fórmula como $a' = a + 1$ é satisfatível, pois associa o próximo estado a' com a valoração $(1,0,0)$, que é definida em $\mathrm{dom}(V)$

No caso do AFE ser satisfatível, então garante-se que o comportamento do AFE é livre de restrições quanto a atualização das variáveis, assim cada estado é associado ao menos com uma valoração. Equivalente a esta expressão, é dito que em todo estado, toda variável é associado com pelo menos um valor. Caso as variáveis sejam associadas particularmente com um único valor, então é dito que os valores que as variáveis podem assumir é determinístico.

4.1.4.2 DETERMINISMO

O determinismo de um AFE $A_{\mathrm{v}} = \langle \Sigma, V, Q, Q^{\circ}, Q^{\omega}, \rightarrow \rangle$ pode ser definido em função do estado, do valor das variáveis associadas ao estado, ou da conjunção de ambos, conforme definido a seguir.

- A_{v} é *Q-determinístico* se $|Q^{\circ}| = 1$ e para duas transições quaisquer $x \xrightarrow{\sigma:p_1} y_1$ e $x \xrightarrow{\sigma:p_2} y_2$, tal que $p_1 \wedge p_2$ são satisfatíveis, é sempre verdade que $y_1 = y_2$, para todo $x, y_1, y_2 \in Q, \sigma \in \Sigma$ e $p_1, p_2 \in \mathscr{F}$.

  Em palavras, caso em um estado x houver uma transição σ associada a duas fórmulas distintas p_1 e p_2 que levem respectivamente a estados y_1 e y_2, A_{v} só será Q-determinístico se $y_1 = y_2$, não importando quantas atualizações satisfatíveis são combinadas, e a cardinalidade do seu conjunto de estados iniciais for igual a 1. Assim a transição pode ser descrita por $x \xrightarrow{\sigma:p_1,p_2} y$. Caso o contrário, e a ocorrência do evento σ transitar A_{v} para estados distintos($y_1 \neq y_2$) ou a cardinalidade do conjunto inicial não for 1, então A_{v} não é determinístico em função de seus estados;

- A_{v} é *V-determinístico* se $(x, \overline{v}) \xrightarrow{\sigma} (y, \overline{w}) \wedge (x, \overline{v}) \xrightarrow{\sigma} (y, \overline{w'})$ implica em $\overline{w} = \overline{w'}$, para todo $x, y \in Q, \sigma \in \Sigma$ e $\overline{v}, \overline{w}, \overline{w'} \in \mathrm{dom}(V)$.

  Em palavras, se em um estado $(x, \overline{v})$ o evento σ leva a dois estados distintos $(y, \overline{w})$ e $(y, \overline{w'})$, A_{v} apenas será V-determinístico se $\overline{w} = \overline{w'}$. Caso $\overline{w} \neq \overline{w'}$ então há transições distintas a

partir do mesmo estado x dada a ocorrência do mesmo evento σ que transite para estados distintos. Neste caso, A_{v} não é V-determinístico.

Por fim, diz-se que A_{v} é determinístico caso for Q-determinístico e V-determinístico (TEIXEIRA, 2013, p. 122).

4.1.4.3 COMPOSIÇÃO

Sejam dois AFEs $A_{\mathrm{v}} = \langle \Sigma_A, V_A, Q_A, Q_A^{\circ}, Q_A^{\omega}, \rightarrow_A \rangle$ e $B_{\mathrm{v}} = \langle \Sigma_B, V_B, Q_B, Q_B^{\circ}, Q_B^{\omega}, \rightarrow_B \rangle$, a composição síncrona de A_{v} e B_{v} é tal que

$$A_{\mathrm{v}} \,\|\, B_{\mathrm{v}} = \langle \Sigma_A \cup \Sigma_B, V_A \cup V_B, Q_A \times Q_B, Q_A^{\circ} \times Q_B^{\circ}, \rightarrow \rangle, \tag{11}$$

em que a relação de transição do modelo composto é definida por:

- $(x_A, x_B) \xrightarrow{\sigma : p_A \wedge p_B} (y_A, y_B)$ se $\sigma \in \Sigma_A \cap \Sigma_B, x_A \xrightarrow{\sigma : p_A} y_A, x_B \xrightarrow{\sigma : p_B} y_B$. Em palavras, se o evento σ pertence ao alfabeto de ambos os autômatos, então a transição sofre a fusão e as fórmulas sofrem conjunção e levará a uma transição em ambos autômatos dos estados x para y;
- $(x_A, x_B) \xrightarrow{\sigma : p_A} (y_A, x_B)$ se $\sigma \in \Sigma_A \setminus \Sigma_B, x_A \xrightarrow{\sigma : p_A} y_A$. Se o evento σ pertence ao alfabeto de A_{v} mas não ao de B_v, então σ está associado apenas às fórmulas da transição do estado gerado a partir de A_{v};
- $(x_A, x_B) \xrightarrow{\sigma : p_B} (x_A, y_B)$ se $\sigma \in \Sigma_B \setminus \Sigma_A, x_B \xrightarrow{\sigma : p_B} y_B$. Se o evento σ pertence ao alfabeto de B_{v} mas não ao de A_{v}, então σ está associado apenas às fórmulas da transição do estado gerado a partir de B_{v};

Portanto, a definição de composição síncrona entre AFEs é similar a de AFDs apresentada na Seção 2.4.2, assim como a característica de sincronismo e assincronismo entre eventos. A distinção fica por conta das fórmulas implementadas sobre as transições nos AFEs. As fórmulas são combinadas por conjunção quando os eventos são síncronos, como no primeiro caso, e são preservadas quando os eventos são assíncronos, como nas situações seguintes (TEIXEIRA, 2013, p.123).

4.1.4.4 SUBAUTÔMATO

Sejam dois AFEs, $A_{\mathrm{v}} = \langle \Sigma, V, Q, Q^{\circ}, Q^{\omega}, \rightarrow \rangle$ e $B_{\mathrm{v}} = \langle \Sigma, V, Q, Q^{\circ}, Q^{\omega}, \rightarrow \rangle$, A_{v} é dito ser um subautômato de B_{v} denotado por $A_{\mathrm{v}} \subseteq B_{\mathrm{v}}$ se:

- $\Sigma_A = \Sigma_B$;
- $V_A = V_B$;
- $Q_A \subseteq Q_B, Q_A^\circ \subseteq Q_B^\circ, Q_A^\omega \subseteq Q_B^\omega$;
- Se $x \xrightarrow{\sigma:p_A}_A y$ então existe uma transição $x \xrightarrow{\sigma:p_B}_B y$ tal que a fórmula p_A implica em p_B.

Em palavras, um subautômato $A \subseteq B$ resulta da remoção de alguns estados e transições ou da expansão das fórmulas em B_v. $A \subseteq B$ também implica em $L(A) \subseteq L(B)$ e $L^\omega(A) \subseteq L^\omega(B)$ (TEIXEIRA et al., 2013, p. 133).

Exemplos: O AFE $x \xrightarrow{\sigma:v'=1} y$ é um subautômato do AFE $x \xrightarrow{\sigma:v'=1} y \xrightarrow{\alpha:v'=2} z$. O AFE $x \xrightarrow{\sigma:v'=1\backslash v=0} y$ é um subautômato do AFE $x \xrightarrow{\sigma:v'=1} y$. Em ambos, a variável v é atualizada no próximo estado dada a ocorrência do evento σ. Entretanto, no primeiro caso a transição é irrestrita. Já no segundo, a transição só ocorre quando o valor de v no estado atual for igual a zero.

4.2 TEORIA DE CONTROLE SUPERVISÓRIO COM AFE

Caso a planta e a especificação de um sistema sejam modelados por um AFE, a condição de controlabilidade também passa a ser estendida para considerar as variáveis, denotada por *V-controlabilidade* (TEIXEIRA et al., 2013, p. 134). Assim, seja uma planta $G_\mathrm{v} = \langle \Sigma, V, Q, Q^\circ, Q^\omega, \rightarrow \rangle$ e uma especificação $E_\mathrm{v} = \langle \Sigma, V, Q, Q^\circ, Q^\omega, \rightarrow \rangle$, então E_v é V-controlável em relação a G_v se a seguinte implicação for verdadeira:

Para todo $s \in \Sigma^*, \mu \in \Sigma_\mathrm{u}, x_e \in Q_E, x_g, x_g' \in Q_G$ e $\overline{v}, \overline{v'} \in \mathrm{dom}(V)$: se $E_\mathrm{v} \xrightarrow{s} (x_e, \overline{v}) \wedge G_\mathrm{v} \xrightarrow{s} (x_g, \overline{v}) \xrightarrow{\mu} (x_g', \overline{v'})$ então existe um $x_e' \in Q_E$ tal que $E_\mathrm{v} \xrightarrow{s} (x_e, \overline{v}) \xrightarrow{\mu} (x_e', \overline{v'})$.

Em palavras, uma especificação que seja modelada por E_v é V-controlável se não proíbe nenhum evento não controlável μ na planta G_v (TEIXEIRA, 2013, p. 125). A V-controlabilidade se difere da controlabilidade convencional, apresentada na Seção 2.5.5, pelo fato que a especificação não devem apenas habilitar todos os eventos não controláveis que são possíveis na planta, mas também mantem o comportamento de atualização das variáveis do mesmo modo que a planta o faz. Entretanto, para eventos controláveis que podem ser desabilitados pela especificação, a especificação pode desabilitar a atualização das variáveis.

Com o conceito de V-controlabilidade, a síntese de controle para AFEs pode ser definida. Porém, a definição é feita utilizando subautômatos, ao invés de linguagens como na

síntese de controle para AFDs. Assim, seja um comportamento esperado sob controle $E_{\mathrm{v}} \parallel G_{\mathrm{v}}$ (equação 11), define-se o conjunto:

$$\mathscr{C}_V = \{K_{\mathrm{v}} \subseteq E_{\mathrm{v}} \parallel G_{\mathrm{v}} | K_{\mathrm{v}} \text{ é V-controlável em relação a } G_{\mathrm{v}}\} \tag{12}$$

$\mathscr{C}_V$ contém um elemento supremo (AFE) denotado por $\sup\mathscr{C}_V(E_{\mathrm{v}}, G_{\mathrm{v}})$, que representa o comportamento mais permissivo possível de ser implementado em G_{v} atendendo as especificações de controle E_{v}. Além da controlabilidade, é desejado que o comportamento $\sup\mathscr{C}_V$ seja não bloqueante. De maneira formal, o não-bloqueio de um autômato $A_{\mathrm{v}} = \langle \Sigma, V, Q, Q^{\circ}, Q^{\omega}, \rightarrow \rangle$ pode ser assim definido:

Para todo $s \in \Sigma^*, t \in \Sigma, x, y \in Q_A, y \in Q_A^{\omega}, \overline{v}, \overline{w} \in V$,

$$A_{\mathrm{v}} \xrightarrow{s} (x, \overline{v})$$

implica em

$$(x, \overline{v}) \xrightarrow{t} (y, \overline{w}),$$

para algum $(y, \overline{w}) \in Q_A^{\omega}$.

Se A_{v} for não-bloqueante, então sua linguagem $L(A_{\mathrm{v}})$ também é não-bloqueante (TEIXEIRA, 2013, p. 127). A característica de não-bloqueio de um AFE é semelhante a um AFD, assim uma caso uma cadeia $s \in L(A_{\mathrm{v}})$ e também $s \in L^{\omega}(A_{\mathrm{v}})$, então o AFE é não-bloqueante.

4.2.1 PROBLEMA DE CONTROLE SUPERVISÓRIO COM VARIÁVEIS

O PCS apresentado na Seção 2.5.4 pode ser refórmulado para o controle supervisório de AFE. Teixeira (2013, p. 128) define o Problema de Controle Supervisório com Variáveis(PCS-V) como:

Dados os AFEs Q-determinísticos E_{v} e G_{v} que modelam respectivamente a especificação e a planta de um SED, encontre um subautômato não-bloqueante $K_v \subseteq E_{\mathrm{v}} \parallel G_{\mathrm{v}}$ que seja V-controlável em relação a G_{v}.

Assim, se $\sup\mathscr{C}_V(E_{\mathrm{v}}, G_{\mathrm{v}})$ é não-bloqueante, então ele é a solução ótima para o PCS-V e pode ser usado para implementar um supervisor que desabilita todos os eventos controláveis possíveis em G_{v} que não são elegíveis em $\sup\mathscr{C}_V(E_{\mathrm{v}}, G_{\mathrm{v}})$ (TEIXEIRA et al., 2013, p. 134). A solução do PCS-V também pode ser associada a PCS desde que seja garantida a equivalência da representação do SED e suas especificações.

Seja G e E AFDs que modelam respectivamente a planta e a especificação de um SED, definindo um comportamento desejado K. Seja G_{v} a versão explícita de um AFE modelando a planta de um SED e E_{v} um AFE que apresenta os mesmos requisitos de controle que E, tal que com $L(G) = L(G_{\mathrm{v}})$, $L(E \parallel G) = L(E_{\mathrm{v}} \parallel G_{\mathrm{v}})$ e $L^{\omega}(E \parallel G) = L^{\omega}(E_{\mathrm{v}} \parallel G_{\mathrm{v}})$. Então é verdade que $L(\sup\mathscr{C}_V(E_{\mathrm{v}}, G_{\mathrm{v}})) = \sup\mathscr{C}(K, G)$ (TEIXEIRA, 2013, p. 128-129).

Dessa maneira garante-se a equivalência da solução de controle seja o sistema modelado por um AFE ou por um AF que represente um AFE de maneira explícita.

4.2.2 TCS COM AFES APLICADA A UM EXEMPLO

Para grau de comparação, utiliza-se o mesmo exemplo apresentado na Figura 6 na Seção 2.5.6. O primeiro passo na modelagem de um SED por AFEs é a identificação do conjunto de variáveis que estarão associadas ás transições do autômato. Para isso, deve ser levado em conta qual informação do sistema deve ser monitorada pela ação de controle do sistema. Neste exemplo, a especificação de controle atua diretamente sobre o *buffer* de armazenamento das peças, com os eventos f_1 e s_2 responsáveis por qualquer ação de carga ou descarga do *buffer* e o objetivo de controle é evitar o *overflow* e o *underflow* do *buffer*. Assim, cada evento de carga e descarga deve prover informações sobre o *status* do *buffer*. Uma variável v, tal que $\mathrm{dom}(v) = 0,1,2,3,4,5,6$ e $v^{\circ} = 0$, é criada para este fim. Os valores 0 e 6 no domínio representam, respectivamente, a possibilidade da ocorrência de *overflow* e *underflow* na planta em malha aberta.

Agora, deve ser definida a semântica de v estabelecendo a maneira como ela será atualizada e em quais transições devem ser associadas as atualizações. O evento f_1 é responsável por depositar peças no *buffer*. Assim, a cada ocorrência deste evento, o valor de v deve ser incrementado em 1. Já o evento s_2 é responsável pela retirada de peças do *buffer*. Logo, a cada ocorrência deste evento o valor de v deve ser decrementado em 1.

Definido o conjunto de variáveis, o domínio e a semântica de cada variável, pode-se então modelar a planta do sistema, representando o comportamento em malha aberta. Esse modelo se difere da modelagem por AFD, da Figura 7 da pag. 31, apenas no fato de que, agora, algumas transições no modelo estão associadas a fórmulas lógicas.

A transição com o evento f_1 contém a fórmula $v' = min((v+1), 6)$. Assim a cada disparo desse evento, v é atualizada no próximo estado com o valor mínimo entre o seu valor atual acrescentado de 1 e 5. Já a transição s_2 contém a fórmula $v' = max((v-1), 0)$. Assim a cada disparo desse evento, v é atualizada no próximo estado com o valor máximo entre o seu valor atual decrementado de 1 e 5. Ambas as fórmulas são satisfatíveis e, portanto, o modelo da

planta também é satisfatível, garantindo que não haja restrições no modelo da planta, condição teórica indispensável para a síntese de controle, conforme discutido na Subseção 4.1.4. A Figura 20 ilustra o modelo da planta.

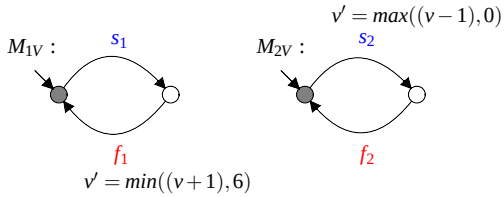


Figura 20 – Modelos das máquinas M_{1v} e M_{2v} representadas por AFE

Com o comportamento da planta definido, a variável *v* associada a planta passa também a apresentar uma semântica definida, representada explicitamente pelo autômato da Figura 21.

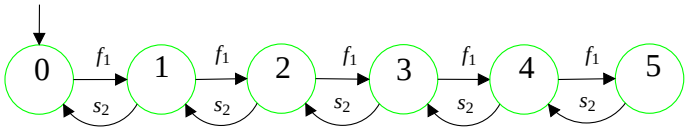


Figura 21 – Autômato explícito contendo a variável *v* utilizada no exemplo

A planta global do sistema é obtida da composição $M_v = M_{1v} \,||\, M_{2v}$. Considerando que a variável *v* já faz parte de ambos os modelos devido às fórmulas de atualização, o seu comportamento está implícito na planta, por construção. M_v resulta em um AFE explícito, apresentado na Figura 22, contendo 24 estados e 44 transições. Nota-se que agora cada estado em M_v está associado a um estado x_i para $i = 1, \cdots, 4$ tal que $x_i \in Q_{M_{1V}} \times Q_{M_{2V}}$ e associado a algum valor de $v \in \text{dom}(v)$. Por exemplo, a transição em M_{1v}, $x \xrightarrow{f_1 : v' = min(v+1,5)} y$ para $v = 0$ é equivalente a transição em M_v, $(x_2, 0) \xrightarrow{f_1} (x_1, 1)$.

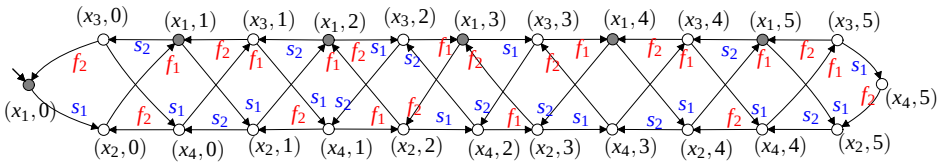


Figura 22 – Modelos da planta global modelando o comportamento de M_1 e M_2

A Figura 23 apresenta a especificação de controle E_v responsável por evitar o *overflow* e o *underflow* do *buffer*. A especificação não deve interferir sobre o valor de *v*, apenas utilizar o valor da variável para validar uma condição lógica no estado atual (fórmulas de guarda). Para absorver essa ideia, a especificação é modelada por um AFE puro.

Assim, sempre que ocorrer a transição f_1, a especificação deve conferir se a fórmula $v < 5$ é verdadeira ou falsa. Caso seja verdadeira, o evento f_1 é habilitado e a fórmula da planta atualiza o valor de *v*. Caso a resposta da avaliação da guarda seja falsa, isso significa que o

valor de v é 5 porém a planta está tentando incrementar uma unidade no *buffer*. Neste caso, a guarda impede a ocorrência de f_1 evitando o *overflow*. Para o evento s_2 o efeito da guarda é semelhante, entretanto evitando o *underflow*.

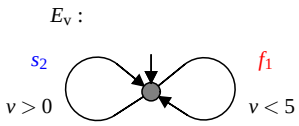


Figura 23 – Especificação de controle E_v modelada por um AFE

Para grau de comparação, a especificação E_v da Figura 23 apresenta um modelo mais simplificado e reduzido do que a especificação E da Figura 9 da pag. 32 devido a redução do número de estados da especificação. Em E são 6 estados e 10 transições para tratar a carga e a descarga no *buffer*. Já em E_v, a carga e descarga é tratada por um único estado associado a uma transição em *loop* implementando as fórmulas de guarda. Neste caso, é permitido a ocorrência de f_1 sempre que a guarda $v < 5$ for verdadeira e a transição s_2 é permitida sempre que a guarda $v > 0$ for verdadeira.

O comportamento em malha fechada $K_v = E_v \parallel M_v$ é apresentado na Figura 24, por um AFE representado de maneira explícita, como um AF, contendo 24 estados e 44 transições, sendo 6 estados marcados. Para a imagem ficar mais clara, o nome dos estados e valores das variáveis em cada estado não são apresentados.

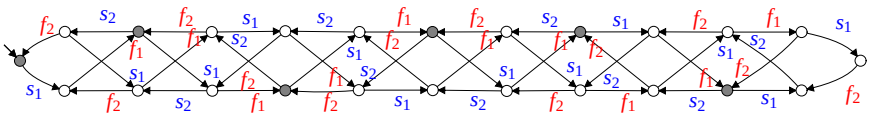


Figura 24 – Comportamento em malha fechada K_v

A ação de controle minimamente restritiva e não-bloqueante é obtida a partir de $\sup\mathscr{C}_V(E_v, M_v)$, representada pelo autômato da Figura 25, contendo 22 estados e 40 transições.

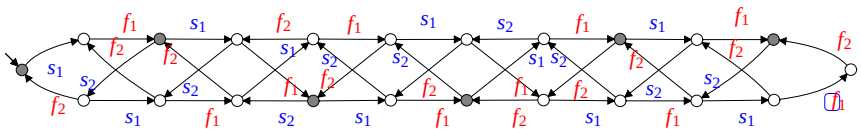


Figura 25 – Solução ótima de controle do sistema modelado por um AFE

A tabela 1 apresenta uma comparação entre os modelos do sistema e os elementos de síntese, representados por AFD e AFE.

Tabela 1 – Número de (estados,transições) dos modelos por AFD e AFE

M : (4,8)	E : (6,10)	K: (24,44)	sup$\mathscr{C}$: (22,40)
M_{v} : (28,54)	E_{v} : (1,2)	K_{v}: (24,44)	sup$\mathscr{C}_V$: (22,40)

Como pode ser analisado, a especificação E_{v} sofre uma redução no número de estados em comparação a especificação E. Entretanto, o modelo da planta M_{v} contém mais estados e transições do que o modelo M, devido ao acréscimo da variável v e das respectivas fórmulas de atualização sobre as transições de M_{v}. Na verdade, a simplificação no modelo da especificação foi compensada pela adição de variáveis à planta, e vice e versa. Em geral, o que se espera é que os elementos de síntese K e K_{v} (comportamentos desejados) e sup$\mathscr{C}$, sup$\mathscr{C}_V$ (supervisores) sejam correspondentes e possuam o mesmo número de estados. Essa relação, no entanto, depende da equivalência de modelagem discutida na Subseção 4.2.1.

4.3 SISTEMA DE CONTROLE COM RECONHECIMENTO DE CONTEXTO

O problema motivador introduzido na Seção 2.6.2 pode agora ser revisto. O grande entrave na modelagem por AFD era pautado na falta de dinamismo do controlador resultante, que não apresentava flexibilidade e auto-configuração conforme a especificação de controle tivesse de ser alterada em tempo de execução. Uma alternativa para agregar tais características ao controlador é a modelagem por AFE. Neste caso, as variáveis seriam utilizadas para monitorar algum contexto do sistema como, por exemplo, se determinado robô está carregado ou não, ou para o controlador ter conhecimento de qual setor determinado robô se encontra, a fim de aplicar a exclusão mútua em determinados momentos, ainda que em outros, não.

Assim, estima-se que as variáveis agregariam poder às ações de controle, pois o controlador poderia utilizá-las como condicionais para a tomada de decisão analisando o contexto do conjunto de variáveis em determinado instante sendo que, na ocorrência de algum evento, o controlador conheceria o contexto e determinaria a ação de acordo. Por exemplo, se fosse utilizado variáveis para monitorar a carga e descarga dos robôs, ao ser efetuada uma carga, o controlador teria conhecimento de que o seu comportamento posterior a este evento é de exclusão mútua nos setores, até que ocorra a descarga, permitindo novamente o compartilhamento de setores.

4.3.1 DEFINIÇÃO DO SISTEMA

Para o ambiente apresentado na Figura 26, considera-se agora a modelagem de plantas e especificações para o sistema, bem com as operações de síntese, levando em conta a modelagem por AFE e suas operações e propriedades apresentadas na Seções 4.1 e 4.2. Todos os modelos e operações a serem apresentados nessa seção foram realizados na ferramenta *Supremica*.

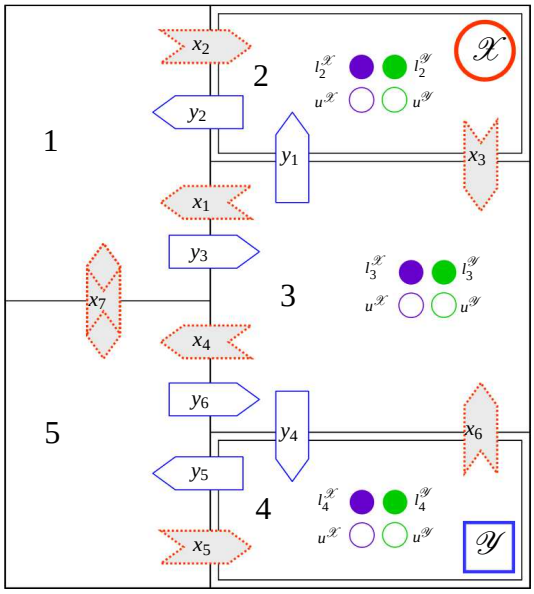


Figura 26 – Ilustração do ambiente considerando $\mathscr{X}$ e $\mathscr{Y}$ como robôs coletores

4.3.1.1 MODELAGEM DA PLANTA

Primeiramente, reforça-se que a estrutura do ambiente da Figura 26 é a mesma das Figura 12. Sendo assim, as transições dos robôs $\mathscr{X}$ e $\mathscr{Y}$ entre os setores permanecem as mesmas e os autômatos $G_{\mathscr{X}}$ (Figura 13) e $G_{\mathscr{Y}}$ (Figura 14) são utilizados para a modelagem do comportamento dos robôs em malha aberta.

Entretanto, agora deve ser levado em consideração também os 8 novos eventos relativos à carga e descarga de material, que passaram a ser observáveis no ambiente. Os eventos são $l_i^{\mathscr{X}}$, $l_i^{\mathscr{Y}}$, $u^{\mathscr{X}}$, $u^{\mathscr{X}}$, para $i = 2,3,4$, em que o valor de i identifica o setor em que é realizado o carregamento, l e u representam a carga e a descarga, respectivamente, e $\mathscr{X}$ e $\mathscr{Y}$ identificam o robô ao qual o evento se refere.

Como foi apresentado na Secão 2.6.2, a ação de carga e descarga dos robôs é responsável pela troca de contexto da ação de controle. Assim a flexibilidade do controlador deve atuar na ocorrência de algum evento que ocasione a carga ou descarga de algum dos robôs.

Com esta finalidade, associa-se um conjunto de variáveis ao modelo da planta para armazenar o contexto do sistema dada a ocorrência dos eventos $l_i^{\mathscr{X}}$, $l_i^{\mathscr{Y}}$, $u^{\mathscr{X}}$, $u^{\mathscr{X}}$.

Assim, são utilizadas duas variáveis, $\mathrm{v}^{\mathscr{X}}$ e $\mathrm{v}^{\mathscr{Y}}$, declaradas com um domínio booleano $\mathrm{dom}(\mathrm{v}^{\mathscr{X}}) = \mathrm{dom}(\mathrm{v}^{\mathscr{Y}}) = \{V, F\}$, com valor inicial $\mathrm{v}^{\mathscr{X}^\circ} = \mathrm{v}^{\mathscr{Y}^\circ} = F$, indicando que ambos robôs iniciam descarregados.

A semântica de atualização de valores de $\mathrm{v}^{\mathscr{X}}$ e $\mathrm{v}^{\mathscr{Y}}$ é imposta perante a ocorrência dos eventos de carga e descarga. Sendo assim, é necessário associar às transições que contenham estes eventos, fórmulas satisfatíveis que atualizem as variáveis. Com essas premissas, constrói-se a planta da Figura 27, que é um AFE é Q-determínistico, V-determinístico, satisfatível e normal.

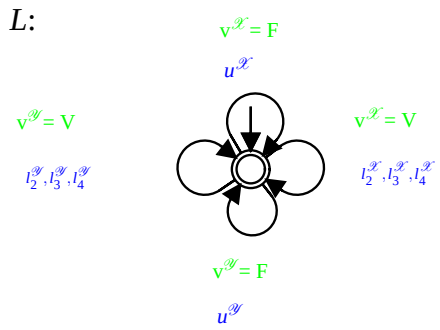


Figura 27 – Submodelo da planta responsável pela troca de contexto

Por exemplo, quando algum evento $l_i^{\mathscr{X}}$ ocorre, $\mathrm{v}^{\mathscr{X}}$ é atualizada para *verdadeiro* pela formula $\mathrm{v}^{\mathscr{X}} = V$, indicando que o robô $\mathscr{X}$ está carregado. Ao ocorrer um evento $u^{\mathscr{X}}$, o robô $\mathscr{X}$ está novamente descarregado, e a variável $\mathrm{v}^{\mathscr{X}}$ é atualizada para o valor *falso*. O mesmo comportamento ocorre aos eventos relacionados ao robô $\mathscr{Y}$.

Agora, o comportamento global da planta do sistema é modelado por $\mathscr{G} = G_{\mathscr{X}} \parallel G_{\mathscr{Y}} \parallel L$, que corresponde a um autômato com 144 estados e 1596 transições.

4.3.1.2 MODELAGEM DAS ESPECIFICAÇÕES

Primeiramente, considera-se que o modelo L da Figura 27 necessita ser sincronizado na ocorrência dos eventos de carga e descarga, para a correta implementação da semântica de atualização de $\mathrm{v}^{\mathscr{X}}$ e $\mathrm{v}^{\mathscr{Y}}$. Este comportamento é representado pelos modelos de especificações Fl^1 e Fl^2, apresentados na Figura 28. Por exemplo, de início ambos os robôs se encontram descarregados, portanto, o estado inicial da especificação deve proibir os eventos $u^{\mathscr{X}}$ e $u^{\mathscr{Y}}$, permitindo apenas os eventos de carga $l_i^{\mathscr{X}}$ e $l_i^{\mathscr{Y}}$. Na ocorrência de alguma carga, o autômato evolui para um estado que agora permite os eventos $u^{\mathscr{X}}$ e $u^{\mathscr{Y}}$ e inibe a ocorrência dos eventos

$l_i^{\mathscr{X}}$ e $l_i^{\mathscr{Y}}$.

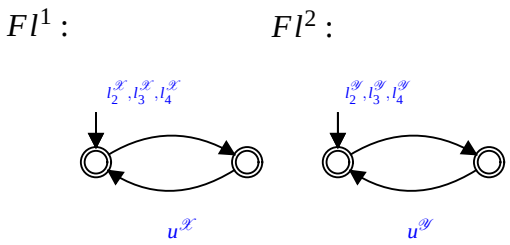


Figura 28 – Modelos de coordenação para carga e descarga de cada robô

Também é necessário realizar uma sincronização entre o comportamento da planta L com a planta dos robôs $\mathscr{G} = G_{\mathscr{X}} \parallel G_{\mathscr{Y}}$, para que os eventos de carga $l_i^{\mathscr{X}}$ e $l_i^{\mathscr{Y}}$ apenas sejam permitidos quando cada robô estiver presente no respectivo setor i. Esta especificação é representada pelos autômatos $UL^{\mathscr{X}} = \|_{j=2,3,4} UL_j^{\mathscr{X}}$ e $UL^{\mathscr{Y}} = \|_{j=2,3,4} UL_j^{\mathscr{Y}}$ da Figura 29.

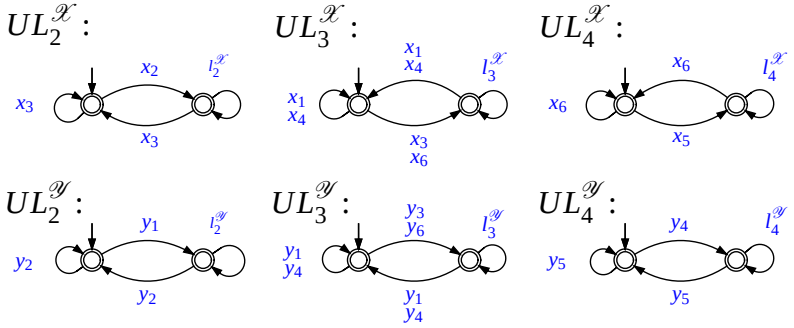


Figura 29 – Modelos especificando a coordenação de carregamento nos setores 2, 3 e 4

Por exemplo, o autômato $UL_2^{\mathscr{X}}$ está permitindo a ocorrência do carregamento do robô $\mathscr{X}$ no setor 2 apenas quando o mesmo adentrar este setor (evento x_2). Ao sair do setor (evento x_3), o evento $l_2^{\mathscr{X}}$ passa a ser proibido pelo modelo. Assim, o evento de carga $l_2^{\mathscr{X}}$ está associado respectivamente e unicamente ao setor 2. A mesma semântica segue para os outros setores em ambos robôs.

Entretanto, as especificações apresentadas nas Figuras 28 e 29 tratam apenas o carregamento nos setores 2, 3 e 4, não atuando no comportamento de carga e descarga dos robôs, quando os mesmos estiverem presentes nos setores 1 e 5. Vale ressaltar que estes setores foram definidos apenas para transição, assim não é possível que os robôs carreguem ou descarreguem material nestes setores. A principio, as especificações da Figura 29 já associam os eventos de cargas aos setores 2, 3 e 4, impedindo que um carregamento ocorra nos setores 1 ou 5, entretanto, nada é especificado em relação a descarga de material, assim, podendo ocorrer um carregamento no setor 3 e um descarregamento no setor 1, por exemplo, para qualquer robô.

Por esta razão, foram modeladas as especificações $U_{1,5}^{\mathscr{X}}$ e $U_{1,5}^{\mathscr{Y}}$ apresentadas na Figura 30. Por exemplo, a especificação $U_{1,5}^{\mathscr{X}}$ permite a ocorrência do evento $u^{\mathscr{X}}$ até que ocorra algum

evento que represente a entrada do robô $\mathscr{X}$ nos setores 1 ou 5 (x_1, x_4, x_7). Na ocorrência de algum desses eventos, $u^{\mathscr{X}}$ passar a ser proibido pela especificação até que $\mathscr{X}$ deixe os setores 1 ou 5, transitando assim para os setores 2, 3 ou 4, permitindo novamente a ocorrência de $u^{\mathscr{X}}$.

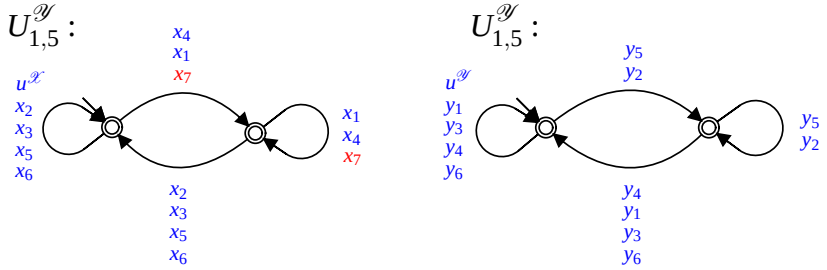


Figura 30 – Modelos especificando a ação proibitiva de descarregamento nos setores 1 e 5

Por fim, pode ser introduzido novas versões para a exclusão mútua de cada setor, de uma maneira que seja incorporado o contexto expresso pela planta através das variáveis. A exclusão mútua apenas deve ser aplicada quando pelo menos um dos robôs está carregado.

Como a planta $\mathscr{G}$ carrega informações sobre o contexto das variáveis, o modelo da especificação precisa apenas checar o valor das variáveis para então validar determinados eventos, conforme o contexto apresentado. Isso tende a simplificar substancialmente os modelos de especificação, em relação à abordagem sem variáveis.

Os AFEs $\mathscr{E}^2, \mathscr{E}^3, \mathscr{E}^4$ da Figura 31 são puros, i.e., eles contém apenas fórmulas de guarda, e podem ser usados para especificar a exclusão mútua dos setores.

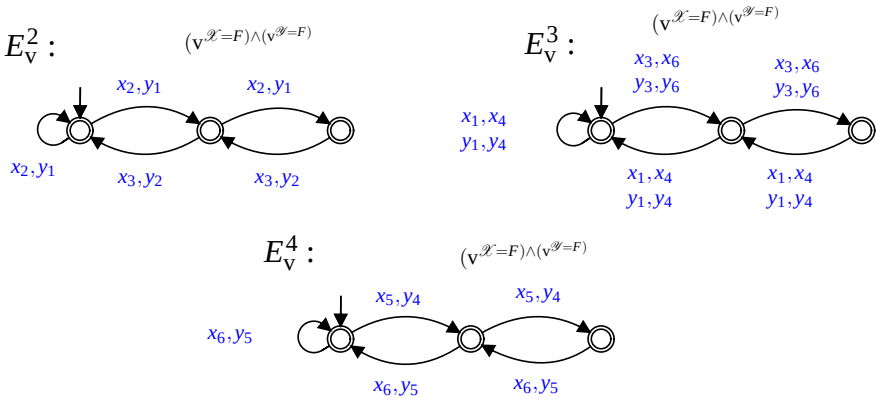


Figura 31 – Modelos de especificações E^2_{v}, E^3_{v} e E^4_{v}

Para explicar a ação das especificações da Figura 31, toma-se como exemplo a especificação E^2_{v}, responsável por realizar a exclusão mútua no setor 2. No estado inicial, a primeira ocorrência de algum dos eventos x_2 ou y_1 (representando que algum dos robôs entrou no setor), é sempre habilitada, pois até então o setor se encontrava vazio. Ao adentrar no setor, o robô pode deixá-lo a qualquer instante, pelo eventos x_3 (caso $\mathscr{X}$ esteja dentro) ou y_2 (caso $\mathscr{Y}$

esteja dentro), retornando ao estado inicial. Entretanto, neste mesmo estado, caso um robô já esteja dentro do setor e o outro robô tentar entrar (segunda ocorrência dos eventos x_2 ou y_1), a especificação irá analisar o contexto das variáveis de carga naquele instante, através da fórmula de guarda $(\mathrm{v}^{\mathscr{X}} = F) \wedge (\mathrm{v}^{\mathscr{Y}} = F)$.

A guarda tem a função de checar se ambos os robôs estão descarregados. Caso a avaliação da guarda seja V, então ambos robôs estão vazios e transição é permitida. Do contrário, se a avaliação da guarda for F, então pelo menos um dos robôs se encontra carregado, assim não é permitida uma ocorrência de x_2 ou de y_1, a menos que o robô carregado (ou ambos) se descarreguem ou algum deles deixe o setor (eventos x_3 ou y_2).

A composição das especificações apresentadas é modelada pelo AFE $\mathscr{E} = E^2_{\mathrm{v}} \,||\, E^3_{\mathrm{v}} \,||\, E^4_{\mathrm{v}} \,||\, Fl^1 \,||\, Fl^2 \,||\, UL^{\mathscr{X}} \,||\, UL^{\mathscr{Y}}$, que corresponde a um autômato com 1368 estados e 16994 transições.

4.3.2 SÍNTESE DO CONTROLADOR

Realizadas as etapas de modelagem das plantas e das especificações para o exemplo proposto, assim como as sincronizações dos modelos modulares de ambas, parte-se então para a operação de síntese de um controlador que possa coordenar adequadamente o ambiente conforme as regras impostas pelos requisitos.

Para tal finalidade, foi utilizado o software Supremica para realização de todas as operações de produto síncrono e de síntese. As propriedades assumidas como requisitos de síntese foram o não bloqueio e a controlabilidade do supervisor $\sup\mathscr{C}_V(\mathscr{E},\mathscr{G})$. Para o exemplo, o controlador obtido resultou de um AFE contendo 157 estados e 722 transições.

Além dos resultados teóricos, que garantem as propriedades da solução de controle, foram também executados testes empíricos no Supremica, envolvendo a simulação da ação do controlador sobre a planta. Esses testes sugerem um comportamento sob controle que condiz com o desejado. Devido ao elevado número de estados e transições, o AFE que modela o supervisor obtido não será apresentado explicitamente nesse trabalho. Entretanto será apresentada a versão desse supervisor traduzido para a linguagem de programação *C* e essa será implementada em um microcontrolador que atuará sobre uma versão gráfica do ambiente, que servirá então ao propósito de interfaceamento com o usuário, para validação empírica da ação de controle.

4.4 IMPLEMENTAÇÃO DO CONTROLADOR

Após as etapas de modelagem e síntese, parte-se então para a implementação do comportamento do supervisor em um controlador, com a finalidade de demonstrar na prática a ação de controle, por meio de um sistema de supervisão.

4.4.1 TRADUÇÃO DO SUPERVISOR PARA LINGUAGEM DE PROGRAMAÇÃO

Para a tradução da estrutura de controle para uma linguagem de programação, que possa ser compreendida por algum controlador, foi adotada a linguagem de programação *C*, devido à sua compatibilidade com diversos modelos de controladores disponíveis no mercado.

Para realizar essa tarefa, utilizou-se o software *DESLab*(TORRICO, 2016), que permite executar operações sobre modelos de autômatos, síntese de ação de controle, verificação de controlabilidade, verificação de bloqueio e também a geração do código relativo ao controlador, em linguagem *C*. Assim, a ferramenta transpõe um resultado, até então é teórico, para uma linguagem compatível com diversas famílias de microcontroladores, dentre elas a do MSP430, utilizada neste trabalho.

Por característica de interfaceamento, o DESLab requer o trabalho manual de inserção de estados e transições para a criação de um autômato, ou seja, a entrada da ferramenta é a versão textual do autômato. Assim, tomando como base o modelo gráfico do supervisor apresentado pelo Supremica, contendo 157 estados e 722 transições, se tornaria complexo e oneroso construir o autômato do supervisor no DESLab. Nesse sentido, foi coletada no Supremica a versão do supervisor gerada pela ferramenta no formato XML(*eXtensible Markup Language*), contendo toda a estrutura dos autômatos das plantas, especificações e do supervisor. Por exemplo, o autômato $G_{\mathscr{X}}[1]$ (Figura 13), é descrito da seguinte forma em XML:

```
1  <Automaton name="Q3_RX" type="Plant">
2          <Events>
3                  <Event id="0" label="x1"/>
4                  <Event id="1" label="x2"/>
5                  <Event id="2" label="x2.{LoadX==N}"/>
6                  <Event id="3" label="x3"/>
7                  <Event id="4" label="x3.{LoadX==N}"/>
8                  <Event id="5" label="x4"/>
9                  <Event id="6" label="x6"/>
10                 <Event id="7" label="x6.{LoadX==N}"/>
11         </Events>
12         <States>
13                 <State id="0" name="Q24" initial="true" accepting="true"/>
14                 <State id="1" name="Q3"/>
15         </States>
16         <Transitions>
```

```
17                  <Transition source="0" dest="1" event="3"/>
18                  <Transition source="0" dest="1" event="4"/>
19                  <Transition source="0" dest="1" event="6"/>
20                  <Transition source="0" dest="1" event="7"/>
21                  <Transition source="0" dest="0" event="1"/>
22                  <Transition source="0" dest="0" event="2"/>
23                  <Transition source="1" dest="0" event="0"/>
24                  <Transition source="1" dest="0" event="5"/>
25          </Transitions>
26  </Automaton>
```

A linha 1 apenas declara o nome do autômato (conforme definido no Supremica) e o seu tipo, podendo ser uma planta, uma especificação ou um supervisor. Neste exemplo, o tipo é *Plant* (planta).

Na sequência são declarados os eventos. Para isso, cada evento recebe um número, utilizado para a sua identificação, e um rótulo, denominado de *label*, conforme designado no Supremica. Os estados são declarados entre as linhas 12 e 15, recebendo um número para identificação e contendo informações sobre estado inicial e marcado do autômato. Por fim, a partir da linha 16 são declaradas as transições. Cada transição é associada a um estado de inicio (*source*), um estado de destino (*dest*) contendo um evento (*event*), sendo essas estruturas mapeadas pelos identificadores numéricos designados previamente.

Nota-se que alguns eventos, como $x2, x3, x6$, são duplicados e uma de suas instâncias é associada a um rótulo que remete à variável de monitoramento de carga e descarga dos robôs. Por exemplo, $LoadX == N$ equivale à fórmula $\mathrm{v}^{\mathscr{X}} = F$ do autômato da Figura 27. Tais eventos são duplicados justamente por estarem associados à entrada do robô $\mathscr{X}$ em setores onde é possível realizar a carga de material e, assim, é necessário realizar a exclusão mútua do setor, eventualmente. Sendo assim, quando $\mathrm{v}^{\mathscr{X}} = F$, a ação de controle faz o uso do evento contendo o rótulo $LoadX == N$, já quando $\mathrm{v}^{\mathscr{X}} = F$, a ação de controle utiliza o evento sem o rótulo.

As linhas 17 e 18 correspondem às transições do estado 0 para o estado 1 utilizando os ponteiros de eventos 3 e 4, que remetem ao evento $x3$. Utilizando a propriedade de Q-determinismo apresentada na Sessão 4.1.4.2, considerando, por exemplo, $\sigma = x3, p_1 =$ e $p_2 = LoadX == N$ e os estados $x = 0$ e $y_1 = y_2 = 1$. Como o estado de destino (estado 1), partindo do mesmo inicio (estado 0), é o mesmo independentemente da formula(p_1 ou p_2) associada a transição (evento $x3$), então o comportamento apresentado é determinístico, segundo a definição apresentada. O mesmo ocorre para os eventos $x2$ (linhas 21 e 22) e $x6$ (linhas 19 e 20), para os quais também, tanto o evento rotulado como o não rotulado, partem do mesmo início para o mesmo destino. Assim, sendo a transição rotulada ou não, ela sempre evoluirá para o mesmo estado. Portanto, todo o autômato apresenta um comportamento determinístico.

Como o DESLab também utiliza identificadores numéricos para estados e eventos, torna-se relativamente simples implementar os modelos dos autômatos das plantas e especificações. A partir dos modelos criados no DESLab, foram realizadas todas as operações de síntese que foram realizadas no Supremica e, como esperado, obteve-se o mesmo modelo do Supervisor, contendo os mesmos 157 estados e 722 transições.

Os eventos presentes no supervisor são declarados no arquivo XML da seguinte forma:

```
<Events>
        <Event id="0" label="x1"/>
        <Event id="1" label="x2"/>
        <Event id="2" label="x2.{LoadX==N}"/>
        <Event id="3" label="x3"/>
        <Event id="4" label="x3.{LoadX==N}"/>
        <Event id="5" label="x4"/>
        <Event id="6" label="x5"/>
        <Event id="7" label="x5.{LoadX==N}"/>
        <Event id="8" label="x6"/>
        <Event id="9" label="x6.{LoadX==N}"/>
        <Event id="10" label="x7" controllable="false"/>
        <Event id="11" label="loadX2"/>
        <Event id="12" label="loadX3"/>
        <Event id="13" label="loadX4"/>
        <Event id="14" label="loadY2"/>
        <Event id="15" label="loadY3"/>
        <Event id="16" label="loadY4"/>
        <Event id="17" label="y1"/>
        <Event id="18" label="y1.{LoadY==N}"/>
        <Event id="19" label="y2"/>
        <Event id="20" label="y3"/>
        <Event id="21" label="y3.{LoadY==N}"/>
        <Event id="22" label="y4"/>
        <Event id="23" label="y4.{LoadY==N}"/>
        <Event id="24" label="y5"/>
        <Event id="25" label="y6"/>
        <Event id="26" label="y6.{LoadY==N}"/>
        <Event id="27" label="unloadX"/>
        <Event id="28" label="unloadY"/>
    </Events>
```

Nota-se a duplicação de todos os eventos responsáveis pela entrada dos robôs nos setores 2, 3 e 4, sendo os eventos x_2, x_3, x_5, x_6 para o robô X, rotulados com o estado da variável LoadX, e os eventos y_1, y_3, y_4, y_6 para o robô Y, rotulados com o estado da variável LoadY. Também há a presença dos eventos de carga e descarga de ambos os robôs.

Em seguida, a versão textual do supervisor foi exportada para código em linguagem *C* e então transferida para um microcontrolador da família MSP430. O DESLab permite a geração do código fazendo o uso de uma máquina de estados no modelo de Mealy (saídas são declaradas em cada evento de transição) ou de Moore (saídas são declaradas em cada estado).

Neste trabalho optou-se pelo modelo de Mealy, por duas razões: primeiramente pela simplicidade na codificação. Uma máquina de Moore necessitaria da declaração do contexto em cada um dos 157 estados. Na máquina de Mealy, basta declarar o contexto na ocorrência de cada um dos eventos do supervisor. O segundo motivo foi de que os eventos são bastante restritos em relação ao comportamento do supervisor. Por exemplo, o evento $y1$ sempre realizará a transição do robô $G_{\mathscr{Y}}$ do setor 3 para o setor 2, sem interferir no robô $G_{\mathscr{X}}$ e este comportamento se mantém independente do estado em que o autômato se encontra. Sendo assim, torna-se mais natural implementar as saídas com base nos eventos, ao invés de considerar os estados.

4.4.2 COMUNICAÇÃO ENTRE MICROCONTROLADOR E INTERFACE GRÁFICA

A transição entre os estados da máquina de Mealy que representa o comportamento do supervisor é realizado na prática pelo envio de uma informação por parte da interface gráfica, que represente a ocorrência de um determinado evento. A comunicação entre interface e microcontrolador é realizada por meio de uma comunicação serial. No microcontrolador, a comunicação serial é interpretada através do periférico de comunicação UART(*Universal Asynchronous Receiver Transmitter* (Receptor e transmissor assíncrono universal)), que é configurado a partir de uma função declarada como *void config_UART(void)* cujo escopo é apresentado a seguir:

```
1  void config_UART(){
2      //Configuração de frequências
3      DCOCTL = 0;
4      BCSCTL1 = CALBC1_1MHZ; //Frequência de 1MHz
5      DCOCTL = CALDCO_1MHZ; //Frequência de 1MHz
6      //Configuração das portas
7      P1DIR |= RXLED + TXLED; //Pinos P1.0 e P1.6 serão saídas(leds)
8      P1OUT &= 0x00; //Saída atríbuidas com nível lógico 0
9      P1SEL |= RXD + TXD;
10     P1SEL2 |= RXD + TXD; //Define a função UART para os pinos P1.1 e P1.2
11     P2DIR = 0xFF; //Pinos da Porta 2 definidos como saída
12     P2OUT &= 0x00; //Saída atríbuidas com nível lógico 0
13     //Configuração da UART
14     UCA0CTL1 |= UCSSEL_2; //Define SMCLK (1MHz) como fonte de clock
15     UCA0BR0 = 104; //1000000 / 9600 ~= 104. Assim, utiliza-se 104 para baurate de 9600
16     UCA0BR1 = 0;
17     UCA0MCTL = UCBRS0; //Modulação UCBRSx = 1
18     UCA0CTL1 &= ~UCSWRST; //Inicializa a máquina de estados da USCI
19     IE2 |= UCA0RXIE; //Habilita interrupção de recebimento por parte da UART
20     __bis_SR_register(CPUOFF+GIE); //LPM0 quando não estiver em operação
21 }
```

A função *void config_UART(void)* configura os poucos parâmetros necessários para a operação da UART para comunicação serial. Primeiramente são definidas as funções de

recepção (RX) e transmissão (TX) de dados para as portas P1.1 e P1.2, respectivamente, do microcontrolador (linhas 9 a 12). Na sequência é definida a taxa de transmissão (*baudrate*) de 9600 bps (bits por segundo) (linhas 15 e 16). Por fim, a instrução da linha 19 define que o microcontrolador estará sempre aguardando o recebimento de alguma informação por parte da interface, que é tratado por meio de uma interrupção, apresentada no trecho de código a seguir:

```
1  #pragma vector=USCIAB0RX_VECTOR
2  __interrupt void USCI0RX_ISR(void)
3  {
4    data = UCA0RXBUF;
5    automato(data);
6  }
```

Na interrupção, uma variável global denominada de data do tipo char, recebe o valor armazenado no buffer de recepção da UART (UCA0RXBUF) que contém a informação recebida da interface gráfica representando a ocorrência de um evento. Em seguida, a variável data é passada como parâmetro para outra função, denominada de *automato*. A função automato é declarada como *void automato(char g)* e recebe como parâmetro de entrada um caractere que representa a ocorrência de um evento, armazenado na variável local *g*. No recebimento de um caractere, a função se utiliza de uma estrutura condicional do formato de *switch case*, testando o valor da variável *g*, para relacionar um caractere recebido com um evento de transição do autômato. A estrutura condicional é apresentada no seguinte trecho de código:

```
1                    switch(g){
2                     case('0'):
3                         occur_event=1;//Evento X1
4                         break;
5                     case('1'):
6                         occur_event=3;//Evento X2
7                         break;
8                     case('2'):
9                         occur_event=5;//Evento X2N
10                        break;
11                    case('3'):
12                        occur_event=7;//Evento X3
13                        break;
14                    case('4'):
15                        occur_event=9;//Evento X3N
16                        break;
17                    case('5'):
18                        occur_event=11;//Evento X4
19                        break;
20                    case('6'):
21                        occur_event=13;//Evento X5
22                        break;
23                    case('7'):
24                        occur_event=15;//Evento X5N
25                        break;
26                    case('8'):
```

```
27                          occur_event=17;// Evento X6
28                          break;
29                      case('9'):
30                          occur_event=19;// Evento X6N
31                          break;
32                      case('A'):
33                          occur_event=21;// Evento Y1
34                          break;
35                      case('B'):
36                          occur_event=23;// Evento Y1N
37                          break;
38                      case('C'):
39                          occur_event=25;// Evento Y2
40                          break;
41                      case('D'):
42                          occur_event=27;// Evento Y3
43                          break;
44                      case('E'):
45                          occur_event=29;// Evento Y3N
46                          break;
47                      case('F'):
48                          occur_event=31;// Evento Y4
49                          break;
50                      case('G'):
51                          occur_event=33;// Evento Y4N
52                          break;
53                      case('H'):
54                          occur_event=35;// Evento Y5
55                          break;
56                      case('I'):
57                          occur_event=37;// Evento Y6
58                          break;
59                      case('J'):
60                          occur_event=39;// Evento Y6N
61                          break;
62                      case('K'):
63                          occur_event=41;// Evento LoadX2
64                          break;
65                      case('L'):
66                          occur_event=43;// Evento LoadX3
67                          break;
68                      case('M'):
69                          occur_event=45;// Evento LoadX4
70                          break;
71                      case('N'):
72                          occur_event=47;// Evento LoadY2
73                          break;
74                      case('O'):
75                          occur_event=49;// Evento LoadY3
76                          break;
77                      case('P'):
78                          occur_event=51;// Evento LoadY4
79                          break;
80                      case('Q'):
```

```
81                          occur_event=53;//Evento UnloadX
82                          break;
83                      case('R'):
84                          occur_event=55;//Evento UnloadY
85                          break;
86                      case('S'):
87                          occur_event=57;//Evento X7
88                          break;
89                  }
```

Por exemplo, caso a interface gráfica envie para o microcontrolador o caractere *0*, a variável *occur_event* recebe o valor de *1*, indicando a ocorrência do evento x_1 e finalizando a execução da estrutura condicional. Na sequência, outra estrutura condicional *switch case* é executada, representando agora o comportamento de transição entre os estados do autômato. Neste caso, a estrutura é definida como *switch(current_state)*, sendo *current state* uma variável global do tipo inteira, que armazena o estado atual do autômato. Por exemplo, o estado inicial do autômato é denominado de 29 (current_state = 29) e seu *case* é apresentado da seguinte forma:

```
1   case(29):
2                   if(occur_event == 3)
3                   {
4                       event_3();
5                       current_state = 16;
6                   }
7                   if(occur_event == 5)
8                   {
9                       event_5();
10                      current_state = 16;
11                  }
12                  if(occur_event == 31)
13                  {
14                      event_31();
15                      current_state = 28;
16                  }
17                  if(occur_event == 33)
18                  {
19                      event_33();
20                      current_state = 28;
21                  }
22                  break;
```

Assim, o estado inicial aceita quatro eventos, x_2 (3), x_{2N} (5), y_4 (31) e y_4N (33). Na ocorrência de qualquer um desses eventos, uma função particular de cada evento é executada, e no seu retorno a variável *current_state* é atualizada para um novo valor, representando um novo estado do autômato.

Esta estrutura de declaração de estados também é de grande relevância para analisar a maneira em que o supervisor distingue os eventos rotulados dos não rotulados, ou seja, de que forma um evento de carga ou descarga e por consequência, a alteração do valor das variáveis, altera a semântica de transição entre os estados. Por exemplo, considera-se que o autômato esteja no estado 37, que representa o robô $G_{\mathscr{X}}$ no setor 3 e o robô $G_{\mathscr{Y}}$ no setor 5, com ambos descarregados. O *case* do estado 37 se apresenta da seguinte forma:

```
1        case(37):
2                if(occur_event == 1) //Evento X1
3                {
4                    event_1();
5                    current_state = 6;
6                }
7                if(occur_event == 11) //Evento X4
8                {
9                    event_11();
10                   current_state = 44;
11               }
12               if(occur_event == 43) //Evento LoadX3
13               {
14                   event_43();
15                   current_state = 115;
16               }
17               if(occur_event == 39) //Evento Y6N
18               {
19                   event_39();
20                   current_state = 32;
21               }
22               break;
```

Neste estado, como ambos robôs estão descarregados, é permitido o compartilhamento do setor 3 pelos robôs. Entretanto, o evento que representa a inserção do robô $G_{\mathscr{Y}}$ no setor, é o evento rotulado 39 (evento y6.LoadY==N do arquivo no formato XML) e não o evento sem rotulo 37 (evento y6 do arquivo no formato XML). Esse comportamento ocorre pois o Supervisor utiliza-se do eventos rotulados para tomar a ação de controle quando os robôs estão descarregados, e faz uso dos eventos sem rótulos quando pelo menos um dos robôs está carregado, sendo que este chaveamento de contexto é baseado no estado atual das variáveis $v^{\mathscr{X}}$ e $v^{\mathscr{Y}}$. Como demonstrado anteriormente, este comportamento é determinístico.

No mesmo estado 37, suponha-se que ocorra o evento 43, representando que o robô $G_{\mathscr{X}}$ está carregado e por consequência, $v^{\mathscr{X}} = V$. O autômato transitará para o estado 115, cujo seu *case* se apresenta da seguinte maneira:

```
1            case(115):
2                if(occur_event == 1) //Evento X1
3                {
4                    event_1();
5                    current_state = 90;
```

```
6                   }
7                   if(occur_event == 11) //Evento X4
8                   {
9                       event_11();
10                      current_state = 122;
11                  }
12                  if(occur_event == 53) //Evento UnloadX
13                  {
14                      event_53();
15                      current_state = 37;
16                  }
17                  break;
```

Como agora os robôs não podem compartilham os setores, o evento que transita o robô $G_{\mathscr{Y}}$ para o setor 3 (seja ele o rotulado ou não rotulado) está agora desabilitado pela ação de controle. Caso ocorra o evento 53, representando a descarga do robô $G_{\mathscr{X}}$, o autômato retorna ao estado 37 apresentando anteriormente. Caso, por exemplo, ocorra o evento 1, $G_{\mathscr{X}}$ transita do setor 3 para o setor 1 e o autômato transitará para o estado 90, apresentado da seguinte forma:

```
1                   case(90):
2                   if(occur_event == 57) //Evento X7
3                   {
4                       event_57();
5                       current_state = 122;
6                   }
7                   if(occur_event == 3) //Evento X2
8                   {
9                       event_3();
10                      current_state = 100;
11                  }
12                  if(occur_event == 37) //Evento Y6
13                  {
14                      event_37();
15                      current_state = 87;
16                  }
17                  break;
```

Como agora o robô $G_{\mathscr{X}}$ não se encontra mais no setor 3, o robô $G_{\mathscr{Y}}$ pode adentrar neste setor. Entretanto, diferentemente do estado 37, em que esta transição ocorria pelo evento 39 (rotulado), agora devido a um dos robôs estar carregado, a transição é realizada na ocorrência do evento 37 (não rotulado).

Anteriormente, foi descrito que antes de realizar a atribuição de valor a variável *current_state*, o algoritmo executava uma função atribuída particularmente a cada evento. Nestas funções, é tratado o contexto do ambiente na ocorrência do evento, ou seja, alterações do setor presente de cada robô e a situação quanto a carga e descarga dos mesmos. Também nas mesmas funções, é realizado o envio dessas informações para a interface gráfica, para a mesma

demonstrar visualmente o contexto atual do ambiente.

Assim, declara-se um vetor de quatro posições, contendo informações quanto ao setor presente de $G_{\mathscr{X}}$ (valor de 0 a 5), setor presente de $G_{\mathscr{Y}}$ (valor de 0 a 5), *status* de carga de $G_{\mathscr{X}}$ (informação binária, 0 ou 1) e *status* de carga de $G_{\mathscr{Y}}$ (informação binária, 0 ou 1). O vetor é declarado no código em C da seguinte forma:

```
enum{
    Robo_X_Sec,
    Robo_Y_Sec,
    Robo_X_Load,
    Robo_Y_Load,
    Num_tags
};
char envio[Num_tags] = {'0','0','0','0'};
```

Nota-se que o vetor é inicializado com todos os seus valores iniciais atribuídos a zero, pois esta é a configuração inicial do ambiente, simbolizando que os robôs não se encontram presente em nenhum setor e estão descarregados. Porém, o valor 0 é atribuído as posições *Robo_X_Sec* e *Robo_Y_Sec* apenas na inicialização e a partir da execução, o valor atribuído a estas posições pode ser qualquer inteiro entre o intervalo de 1 a 5, que representa cada um dos setores.

O trecho de código seguinte apresenta a função específica de alguns eventos:

```
void event_1(void)  //Sequência operacional para o evento 1
{
    envio[Robo_X_Sec] = '1';
    UARTSendArray((unsigned char*)&envio,Num_tags);
    UARTSendArray("\n\r",2);
}
void event_25(void) //Sequência operacional para o evento 25
{
    envio[Robo_Y_Sec] = '1';
    UARTSendArray((unsigned char*)&envio,Num_tags);
    UARTSendArray("\n\r",2);
}
void event_41(void) //Sequência operacional para o evento 41
{
    envio[Robo_X_Load] = '1';
    UARTSendArray((unsigned char*)&envio,Num_tags);
    UARTSendArray("\n\r",2);
}
void event_53(void) //Sequência operacional para o evento 53
{
    envio[Robo_X_Load] = '0';
    UARTSendArray((unsigned char*)&envio,Num_tags);
    UARTSendArray("\n\r",2);
}
```

A função *void event_1(void)* executa comandos para o evento 1 (x_1). Assim, na ocorrência deste evento, o vetor *envio* na posição *Robo_X_Sec* recebe o valor 1, representando que o robô $G_{\mathscr{X}}$ se encontra no setor 1. As outras três posições permanecem inalteradas, mantendo o seu valor anterior, pois a ocorrência deste evento apenas altera o setor de $G_{\mathscr{X}}$, não interferindo no setor presente de $G_{\mathscr{Y}}$ ou no *status* de carga dos robôs. O mesmo comportamento se repete para os outros eventos, como pode ser analisado no trecho do código apresentado anteriormente.

Após a atribuição de valores ao vetor de envio de informações, cada função executa uma outra função, denominada de *UARTSendArray*, cujo escopo é demonstrado a seguir:

```
1 void UARTSendArray(unsigned char *TxVetor, unsigned char Tam){
2     while(Tam--){
3         while(!(IFG2 & UCA0TXIFG));
4         UCA0TXBUF = *TxVetor;
5         TxVetor++;
6     }
7 }
```

A função *UARTSendArray* recebe como parâmetros de entrada um vetor de caracteres, denominado de *TxVetor*, e um valor inteiro, denominado de *Tam*, que armazena o comprimento do vetor que será enviado do microcontrolador para a interface gráfica. Como *Tam* é declarado como *unsigned*, somente pode ser atribuído valores positivos a esta variável, portanto, seu valor mínimo é 0. Assim, a partir da linha 2 é executado um laço de repetição do tipo *while* que recebe o valor de *Tam* e decrementa em uma unidade a cada iteração do laço.

Na linha 3, é executado outro laço de repetição *while* que aguarda até a ocorrência de uma sinalização de uma interrupção de transmissão, sinalizada por *UCA0TXIFG*. Caso o valor de *UCA0TXIFG* seja 1, a instrução como um toda é verdadeira, e o código interrompe a execução deste laço, passando a execução das instruções seguintes.

Na linha 4, o *buffer* de recepção da UART (UCA0TXBUF) é carregado com o atributo da posição atual do vetor *TxVetor*. Na linha 5, a posição do vetor é incrementada em uma unidade, representando que o próximo dado armazenado no vetor será carregado em UCA0TXBUF para envio.

Ao final da execução desta função todos os dados presentes no vetor *TxVetor* são enviados para a interface gráfica via comunicação serial pela interface UART, através de UCA0TXBUF.

4.4.3 DESENVOLVIMENTO DA INTERFACE GRÁFICA

A interface gráfica, apresentada na Figura 32, foi desenvolvida em uma aplicação *desktop* na linguagem Java. Optou-se pela utilização desta linguagem de programação pela diversidade de opções e funções que a mesma suporta para o desenvolvimento de uma interface gráfica, além da utilização da biblioteca RXTX-2.2 [4], que permite desenvolver aplicações em Java utilizando comunicação serial ou paralela que apresentem um desempenho bastante estável;

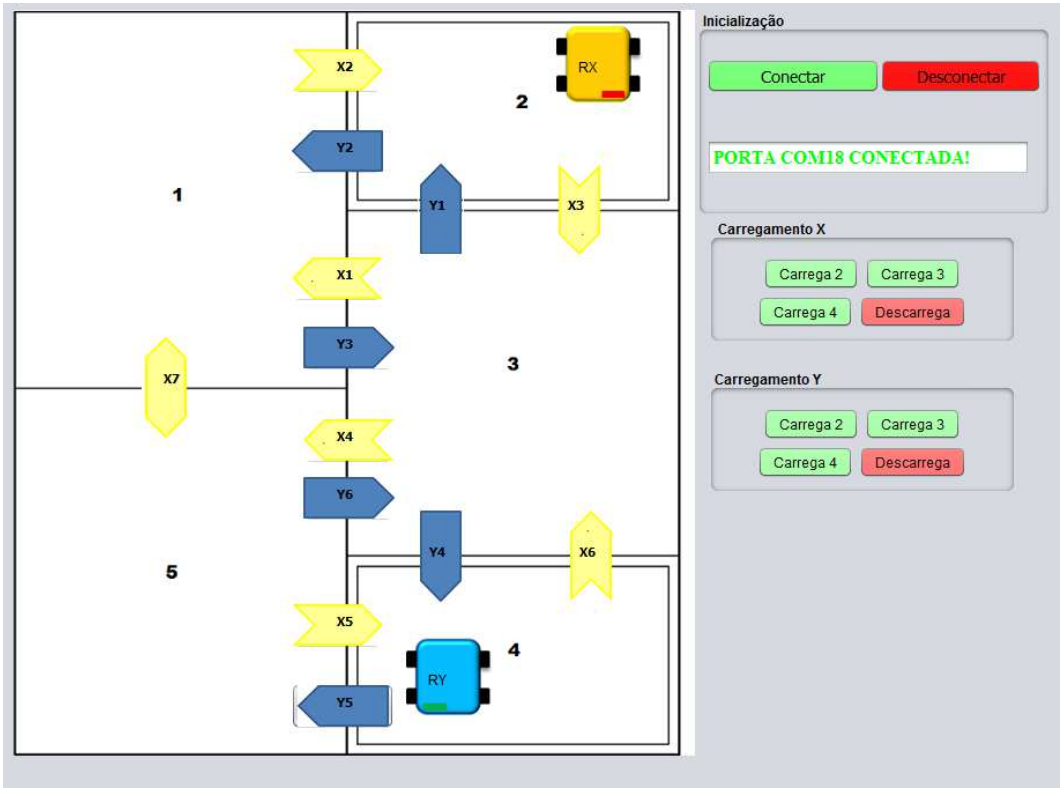


Figura 32 – Interface gráfica para visualização da ação de controle

A interface da Figura 32 foi criada em um classe denominada Tela e apresenta no seu canto superior direito, a opção de conexão da porta serial. Ao iniciar a execução da interface, a conexão serial não está em operação, sendo necessário pressionar o botão Conectar (em verde) para iniciar a conexão com a porta serial COM18, sendo esta a porta serial utilizada pelo microcontrolador adotada neste trabalho. Com a conexão operando, se o botão Desconectar (em vermelho) for pressionado, a conexão entre a interface a porta COM18 é encerrada.

O método utilizado pelos botões Conectar e Desconectar são escritos na linguagem Java da seguinte forma:

```
1     private void btnConectActionPerformed(java.awt.event.ActionEvent evt) {
```

[4] http://fizzed.com/oss/rxtx-for-java

```
2           serial.Conecta();//Executa o método Conecta do objeto serial
3           if (serial.isConectado() == true) {//Verifica se está conectado
4               if (serial.inicializa_IOStream() == true) {//Verifica se foram criados os buffers
5                           //para entrada/saída de dados
6                   serial.inicia_Listener();//Este método inicializa os listeners que sabem quando
7                           //dados estão disponíveis para serem lidos
8            txt_Conect.setFont(new Font("Times New Roman", Font.BOLD, 16));
9            txt_Conect.setText("PORTA " + serial.selectedPortIdentifier.getName()+"CONECTADA!");
10           txt_Conect.setForeground(Color.green); // altera a cor da fonte
11              }
12          }
13      }
14
15                  private void btnDesconActionPerformed(java.awt.event.ActionEvent evt) {
16          serial.desconecta();//Executa o métdo para desconectar a porta
17          txt_Conect.setFont(new Font("Times New Roman", Font.BOLD, 16));
18          txt_Conect.setText("PORTA " + serial.selectedPortIdentifier.getName()+"DESCONECTADA!");
19          txt_Conect.setForeground(Color.red); // altera a cor da fonte
20      }
```

Todos os métodos utilizados são implementados por uma outra classe, denominada de Serial, na qual é acessada pela classe Tela por meio de um objeto denominado de serial. Os métodos utilizados no trecho de código anterior são escritos da seguinte forma na classe Serial:

```
1     //Este método conecta na porta serial encontrada
2       public void Conecta(){
3           CommPort commPort = null;//Declara uma porta de comunicação, iniciando nula
4           try{
5               //Comando abaixo identifica uma porta de comunicação
6                           //TIME_OUT definido de 2000ms
7               commPort = selectedPortIdentifier.open(this.getClass().getName(),TIME_OUT);
8               serialPorta = (SerialPort)commPort;//Se identificou uma porta, converter para um
9                                   //tipo de porta serial
10              //Comando abaixo passa os paramêtros da porta Serial, com baudrate de 9600 bits/s,
11                          //8 bits de dados,1 bit de parada e sem paridade
12              serialPorta.setSerialPortParams(
13              DATA_RATE,
14              SerialPort.DATABITS_8,
15              SerialPort.STOPBITS_1,
16              SerialPort.PARITY_NONE
17                      );
18              setConectado(true);//Informa que a conexão está habilitada
19          }catch(Exception e){
20              System.err.println(e.toString());
21          }
22       }
23
24  //Este método retorna o estado da conexão. Se for true, está conectado
25  // e false não está conectado
26                  public boolean isConectado() {
27          return conectado;
28      }
29
```

```
30                      //Abre os streams de entrada e saída de dados
31      public boolean inicializa_IOStream(){
32          //Váriavel de retorno para sucesso ou não da abertura dos Streams
33          boolean sucesso = false;
34          try{
35                                      //Buffer para entrada de dados
36              input = new BufferedReader(new InputStreamReader(serialPorta.getInputStream()));
37              output = serialPorta.getOutputStream(); //Buffer para a saída de dados
38              sucesso = true;
39              return sucesso;
40          }catch(Exception e){
41              System.out.println("I/O Streams falharam na abertura (" + e.toString() + ")");
42              return sucesso;
43          }
44      }
45
46                      //Este método inicializa os listeners que sabem quando dados estão
    disponíveis para serem lidos
47      public void inicia_Listener(){
48          try{
49              serialPorta.addEventListener(this); //Adiciona um evento na lista de Listener
50              serialPorta.notifyOnDataAvailable(true);
51          }catch(TooManyListenersException e){
52              System.out.println(e.toString() + "Muitos listeners (");
53          }
54      }
55
56                      //Este método desconecta a comunicação
57      public void desconecta(){
58          try{
59              if(serialPorta != null){
60              serialPorta.removeEventListener();
61              serialPorta.close();
62              input.close();
63              output.close();
64              setConectado(false);
65          }
66          }catch(Exception e){
67             System.out.println("Falha ao fechar"
68                                          + serialPorta.getName() + "(" + e.toString() + ")");
69          }
70      }
```

Abaixo da configuração da conexão, encontram-se as opções para carga e descarga para ambos os robôs. Neste caso são três botões, representando pela cor verde, responsáveis pela carga de cada setor e um botão vermelho responsável pela descarga do respectivo robô.

Por fim, no lado esquerdo da interface, encontra-se uma representação do ambiente apresentado na Figura 26. Nesta imagem, as setas com a coloração amarela representam as transições do robô $G_{\mathscr{X}}$ e as setas com a coloração azul representam as transições do robô $G_{\mathscr{Y}}$. Os robôs também seguem este mesmo padrão de cores para identificação própria. Adicional ao

corpo dos robôs, se encontra um identificador do estado de carga, sendo este sinalizado pela cor vermelha (como no $G_{\mathscr{X}}$) quando o robô estiver descarregado e sinalizado pela cor verde (como no $G_{\mathscr{Y}}$) quando o robô estiver carregado.

As setas das transições, assim como os botões de carga e descarga, são responsáveis pelo envio de informação e interação da interface com a ação de controle implementada no microcontrolador. Assim, ao serem pressionados, executam um método que envia um caractere, equivalente ao evento correspondente, ao microcontrolador. Ao receber este caractere (pela interrupção de recebimento), o microcontrolador relaciona com o evento correspondente (função *automato*) e realiza as devidas ações, retornando para a interface as informações contidas no vetor *TxVetor* (função *UARTSendArray*). O trecho de código seguinte apresenta algum destes métodos:

```
1  private void btn_x1ActionPerformed(java.awt.event.ActionEvent evt) {
2          serial.escrever("0");//0 corresponde a X1 no microcontrolador
3      }
4  private void btn_y2ActionPerformed(java.awt.event.ActionEvent evt) {
5          serial.escrever("C");//C corresponde a Y2 no microcontrolador
6      }
7  private void btn_Load_X3ActionPerformed(java.awt.event.ActionEvent evt) {
8          serial.escrever("L");//L corresponde a Load_X3 no microcontrolador
9      }
10 private void btn_Unload_YActionPerformed(java.awt.event.ActionEvent evt) {
11         serial.escrever("R");//R corresponde a Unload_Y no microcontrolador
12     }
```

O método escrever do objeto serial, responsável pelo envio de dados para o microcontrolador, é implementado da seguinte forma:

```
1      //Método utilizdo para enviar um string de dados
2      public void escrever(String dado){
3        try{
4            output.write(dado.getBytes());//Convete a String em um vetor de Bytes
5                                          //para escrever no OutputStream
6            output.flush();//Limpa o buffer de saída do OutputStream
7        }catch(Exception e){
8           System.err.println("Não pode escrever na porta serial");
9        }
10     }
```

O recebimento de dados é tratado por meio de um evento na porta serial, no qual foi criado pelo método *inicia_Listener()* apresentado anteriormente. Assim, ao enviar um dado para o microcontrolador, a aplicação fica aguardando a comunicação serial até o recebimento de algum dado. Ao receber, notifica a ocorrência de um evento que executa o seguinte método:

```
1      //Evento Serial, responsábel pelo recebimento de dados
2      public void serialEvent(SerialPortEvent evt){
3         String entrada = null;//String para recebimento dos dados
```

```
4         //Se o evento identificado corresponde a dados disponíveis na porta
5         if(evt.getEventType() == SerialPortEvent.DATA_AVAILABLE){
6             try{
7                 //A informação recebida é uma linha do buffer de entrada
8                 //Essa linha é armazenada na String entrada
9                  entrada = input.readLine();
10                 desenha = new Desenha(janela);
11                 desenha.Recebe(entrada);
12               }catch(Exception e){
13                   System.err.println(e.toString());
14                       }
15        }
16    }
```

Neste método é criado uma *string* denominado de entrada que armazena os dados recebidos pela interface. Ao receber estes dados o método do evento declara um novo objeto referente a classe Desenha, denominado de desenha, e faz uso de um método denominado Recebe que recebe como parâmetro a string entrada.

A classe Desenha é responsável pela visualização e atualização das informações na interface gráfica. Por exemplo, na Figura 32, o robô $G_{\mathscr{X}}$ se encontra no setor 2 e o robô $G_{\mathscr{Y}}$ se encontra no setor 4. Caso o ocorra uma transição entre setores de qualquer robô, a classe Desenha se encarrega de modificar o desenho referente ao robô de um setor para outro.

Para isso, utiliza as informações passadas como parâmetro para o método Recebe. Sendo de conhecimento que a string contém quatro posições, equivalente as posições Robo_X_Sec, Robo_Y_Sec, Robo_X_Load e Robo_Y_Load do vetor *TxtVetor*, assim, cada caractere da string corresponde a uma respectiva informação do ambiente. Por exemplo, se a string entrada apresentar o valor 4510, corresponde ao robô $G_{\mathscr{X}}$ no setor 4 e carregado e o robô $G_{\mathscr{Y}}$ no setor 5 descarregado. Para isso, o método Recebe declara quatro caracteres para armazenar a informação de cada índice da string.

```
1         char Robo_X_Sec = entrada.charAt(0);
2         char Robo_Y_Sec = entrada.charAt(1);
3         char Robo_X_Load = entrada.charAt(2);
4         char Robo_Y_Load = entrada.charAt(3);
```

Finalizando, a interface gráfica é responsável pela conexão com a porta serial do microcontrolador e pelo o envio de dados que representem a ocorrência de eventos, seja do deslocamento entre setores ou de carregamento de um robô. Ao enviar a informação, a interface fica aguardando um evento de recebimento de dados, e ao receber, realiza o tratamento da informação recebida e apresenta o resultado graficamente.

5 CONCLUSÕES

Este trabalho apresentou uma abordagem para a modelagem e a síntese de controladores que podem ser utilizados em ambientes dinâmicos e concorrentes para coordenação de múltiplos robôs. Nesse tipo de ambiente, é desejável que cada robô se adapte automaticamente ao contexto imposto pelo sistema. Assim, é desejável que o controlador altere a ação de controle em tempo de execução.

Para que isso se torne possível, esse trabalho propôs que a planta do sistema seja modelada por AFEs, formalismo que permite que o modelo de um SED possa ser enriquecido com variáveis para o armazenamento do contexto do sistema. Assim, fazendo a utilização de fórmulas associadas às transições, se torna possível tanto atualizar as variáveis quanto implementar restrições ao sistema, com base em condições lógicas implementada sobre os valores das variáveis.

A abordagem foi ilustrada por meio de um exemplo em que dois robôs coletores de material circulam através de um ambiente dividido por setores, cuja ação de controle imposta se altera dependendo se os robôs estão carregando algum material ou não, sendo que a troca de contexto é alterada dinamicamente pelo controlador em tempo de execução.

Os resultados apresentados por este trabalho sugerem que a proposta apresentada é tecnicamente válida para a modelagem e controle de ambientes que requerem e necessitem um comportamento dinâmico em execução. Para reforçar os resultados obtidos, poderia ser agregado ao trabalho, exemplos de maior complexidade ou até mesmo aplicações práticas utilizando a abordagem proposta.

5.1 DIFICULDADES ENCONTRADAS

Por se tratar de uma abordagem relativamente nova, o uso de AFEs para modelar problemas reais de controle esbarrou, nesse trabalho, na necessidade da construção de uma estrutura teórica substancialmente complexa, a fim de sustentar a implementação prática. Algumas propriedades e operações envolvendo AFE foram definidas, outras foram herdadas

de trabalhos anteriores, como mostra a Sessão 4.1.4, mas, em ambos os casos, a aplicação ao contexto do trabalho requereu uma ampla análise de compreensão.

Em relação ao desenvolvimento prático, no inicio do trabalho foi proposto que a interface gráfica seria desenvolvida por meio de um software específico para ambientes supervisórios. Em princípio foi cogitado o emprego do ScadaBR, que em tese realizaria tarefas similares às da atual interface, de visualização do ambiente e comunicação com o microcontrolador, que neste caso seria via protocolo ModBus.

Entretanto, foram reportadas dificuldades relacionadas com a estabilidade do software, inicialmente relativas às versões do ScadaBR e da máquina virtual Java necessária para executar o ScadaBR. Tratado esse problema, foi realizado todo o desenvolvimento da interface gráfica no ScadaBR com sucesso, porém, observou-se que a comunicação Modbus entre o microcontrolador e o ScadaBR era demasiadamente instável, pois era necessário utilizar um conversor USB-Serial-RS-232, no caso foi utilizado um modelo da marca Prolific, que apresentou muitos problemas de versão de driver. Ao definir o driver correto, o ScadaBR e o microcontrolador estavam comunicando entre si, entretanto os dados recebidos pelo ScadaBR não eram os desejados, resultando assim em uma inconsistência de informação.

Para tratar este problema, primeiramente foi tentado a substituição do ambiente supervisório, do ScadaBR para o software supervisório Elipse Scada, porém o mesmo não obteve sucesso na comunicação. Como um plano alternativo a estes problemas, foi pensando em desenvolver uma interface gráfica *desktop* na linguagem Java utilizando comunicação serial para envio e recebimento de dados por parte da aplicação gráfica e a interface UART para envio e recebimento de dados por parte do MSP430.

Tal aplicação apresentou um desempenho satisfatório, tanto na questão de visualização das informações graficamente quanto na comunicação. Sendo assim, a alternativa foi escolhida para a execução prática do trabalho, de modo que a extensão disso para um ambiente SCADA é sugerido como trabalhos futuros.

A única limitação encontrada na implementação foi o tamanho do código do microcontrolador, que é de aproximadamente 16 Kilobytes, tamanho máximo permitido para gravação utilizando o compilador do *Code Composer* gratuito. Assim, informações adicionais que seriam interessante de passar para a interface gráfica, como transições permitidas ou não naquele exato instante de tempo, não puderam serem implementadas.

5.2 TRABALHOS FUTUROS

Como citado, pouco ainda é documentado quanto a abordagem prática de AFEs. Este trabalho visa abrir as portas para trabalhos futuros, realizando uma breve introdução a esta área do conhecimento e suas aplicações, se utilizando de um exemplo bastante intuitivo.

Entretanto, o mesmo exemplo não demonstra com grande força o poder que as variáveis podem agregar, tanto nas etapas de modelagem, quanto de síntese, pois para a solução do mesmo, foram utilizadas apenas duas variáveis booleanas com tomadas de decisões bastante simples em relação a outros tipos de conjunto de dados. Assim, muito se deve estudar em relação aos AFEs modelando problemas mais complexos, que envolvam a utilização de variáveis de domínio numérico, ou até mesmo um conjunto de variáveis de naturezas distintas, como uma estrutura.

Além disso, para uma melhor visualização da ação de controle e, principalmente, para o tratamento dos dados do sistema sob controle (banco de dados, alarmes, mensagens, etc.), duas medidas podem ser tomadas na continuidade deste trabalho. Primeiramente, o desenvolvimento de um ambiente de supervisão mais consistente e amplo e utilizando um protocolo de comunicação mais robusto, que possa apresentar melhor o ambiente e suportar uma melhor estrutura de envio e recebimento de dados. Uma segunda medida seria a aplicação prática do próprio ambiente proposto, levando em consideração o desenvolvimento real dos robôs.

Para finalizar, pretende-se focar na implementação de ferramenta relacionada ao trabalho. Embora a literatura forneça ferramentas automatizadas para a geração de códigos a partir de autômatos, tais ferramentas são, em geral, estabelecidas com base em autômatos convencionais, o que dificulta a manipulação das estruturas logicas dos AFEs.

REFERÊNCIAS

AKERS, S. B. Binary decision diagrams. **Computers, IEEE Transactions on**, C-27, n. 6, p. 509–516, June 1978. ISSN 0018-9340.

AKESSON, K. et al. Supremicaâa tool for verification and synthesis of discrete event supervisors. In: **Proc. of the 11th Mediterranean Conference on Control and Automation, Rhodos, Greece**. [S.l.: s.n.], 2003. p. 1.

AKESSON, K. et al. Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems. In: **Discrete Event Systems, 2006 8th International Workshop on**. [S.l.: s.n.], 2006. p. 384–385.

BANG-JENSEN, J.; GUTIN, G. **Digraphs Theory, Algorithms and Applications**. 1. ed. New York: Springer-Verlag, 2007.

BOLCH, G. et al. **Queueing networks and markov chains**. [S.l.]: John Wiley & Sons, Inc, 2000.

BOUZON, G. **Sensores em sistemas a eventos discretos**. Dissertação (Mestrado) — Universidade Federal de Santa Catarina - UFSC, Florianópolis, 2004. Dissertação (Mestrado em Engenharia Elérica) Programa de Pós-Graduação em Engenharia Elétrica.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to Discrete Event Systems**. 2. ed. New York: Springer Science+Bussines Media, LLC, 2008.

CHEN, Y.-L.; LIN, F. Modeling of discrete event systems using finite state machines with parameters. In: **Control Applications, 2000. Proceedings of the 2000 IEEE International Conference on**. [S.l.: s.n.], 2000. p. 941–946.

CHEN, Y.-L.; LIN, F. Safety control of discrete event systems using finite state machines with parameters. In: **American Control Conference, 2001. Proceedings of the 2001**. [S.l.: s.n.], 2001. v. 2, p. 975–980 vol.2. ISSN 0743-1619.

CHENG, K.-T.; KRISHNAKUMAR, A. Automatic generation of functional vectors using the extended finite state machine model. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, v. 1, n. 1, p. 57–79, Janeiro 1996.

CURY, J. **Teoria de Controle Supervisório de Sistemas a Eventos Discretos**. nov 2001. Acessado: 2015-08-30. Disponível em: <http://user.das.ufsc.br/~cury/cursos/apostila.pdf>.

CURY, J. E. R. et al. Supervisory control of discrete event systems with distinguishers. **Automatica**, v. 56, p. 93–104, 2015.

DESROCHERS, A. A.; AL-JAAR, R. Y. **Applications of Petri nets in manufacturing systems: modeling, control, and performance analysis**. [S.l.]: IEEE, 1995.

DORF, R. C.; BISHOP, R. H. **Sistemas de Controle Modernos**. 11. ed. Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora Ltda., 2009.

FEI Z., S. M. A. K.; LENNARTSON, B. **Efficient Supervisory Synthesis to Large-Scale Discrete Event Systems Modeled as Extended Finite Automata**. [S.l.], 2012.

HOPCROFT, R. M. J. E.; ULLMAN, J. D. **Introduction to Automata Theory, Languages and Computation**. 2. ed. [S.l.]: ser. Series in Computer Science. Addison-Wesley, 2001.

LEWIS, H. R.; PAPADIMITRIOU, C. H. **Elements of the Theory of Computation**. 2. ed. New Jersey: Prentice-Hall, 1998.

MURATA, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, IEEE, v. 77, n. 4, p. 541–580, 1989.

OGATA, K. **Discrete-Time Control Systems**. 2. ed. New Jersey: Pearson Prentice Hall, 1994.

OGATA, K. **Engenharia de Controle Moderno**. 5. ed. São Paulo: Pearson Prentice Hall, 2010.

OUEDRAOGO, L. et al. Nonblocking and safe control of discrete-event systems modeled as extended finite automata. **Automation Science and Engineering, IEEE Transactions on**, v. 8, n. 3, p. 560–569, Julho 2011. ISSN 1545-5955.

QUEIROZ, M. H. D.; CURY, J. E. R. Modular supervisory control of large scale discrete event systems. In: **Discrete Event Systems: Analysis and Control. Proc. WODES'00**. [S.l.]: Kluwer Academic, 2000. p. 103–110.

RAMADGE, P. J.; WONHAM, W. M. On the supremal controllable sublanguage of a given language. **The 23rd IEEE Conference on**, v. 23, p. 1073–1080, 1984.

RAMADGE, P. J.; WONHAM, W. M. Modular feedback logic for discrete event systems. **SIAM Journal of Control and Optimization**, v. 25, n. 5, p. 1202–1218, 1987.

RAMADGE, P. J.; WONHAM, W. M. The control of discrete event systems. **Proceedings of the IEEE**, IEEE, v. 77, n. 1, p. 81–98, 1989.

ROSA, J. L. G. **Linguagens Formais e Autômatos**. 1. ed. São Paulo: LTC - Livros Técnicos e Científicos Editora Ltda., 2010.

SHOAEI, M.; FENG, L.; LENNARTSON, B. Abstractions for nonblocking supervisory control of extended finite automata. In: **Automation Science and Engineering (CASE), 2012 IEEE International Conference on**. [S.l.: s.n.], 2012. p. 364–370. ISSN 2161-8070.

SKOLDSTAM, M.; AKESSON, K.; FABIAN, M. Modeling of discrete event systems using finite automata with variables. In: **Decision and Control, 2007 46th IEEE Conference on**. [S.l.: s.n.], 2007. p. 3387–3392. ISSN 0191-2216.

TEIXEIRA, M. **Explorando o uso de distinguidores e de autômatos finitos estendidos na teoria do controle supervisório de sistemas a eventos discretos**. Tese (Doutorado) — Universidade Federal de Santa Catarina - UFSC, Florianópolis, 2013. Tese de Doutorado (Doutorado em Engenharia de Automação e Sistemas) Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

TEIXEIRA, M. et al. Variable abstraction and approximations in supervisory control synthesis. In: **American Control Conference (ACC), 2013**. [S.l.: s.n.], 2013. p. 132–137. ISSN 0743-1619.

TEIXEIRA, M. et al. Supervisory control of des with extended finite-state machines and variable abstraction. **Automatic Control, IEEE Transactions on**, v. 60, n. 1, p. 118–129, Janeiro 2015. ISSN 0018-9286.

TORRICO, C. **Deslab 3.6**. [S.l.], 2016. Disponível em: <https://sites.google.com/site/controlediscreto9/instaladores>.

VORONOV, A.; AKESSON, K. **Verification of Supervisory Control Properties of Finite Automata Extended with Variables**. [S.l.], 2009. (R - Department of Signals and Systems, Chalmers University of Technology, no: 003/2009, 03).

YANG, Y.; MANNANI, A.; GOHARI, P. Implementation of supervisory control using extended finite-state machines. **International Journal of Systems Science**, v. 39, n. 12, p. 1115–1125, 2008.

ZHAO, J. et al. Modeling and control of discrete event systems using finite state machines with variables and their applications in power grids. **Systems & Control Letters**, Elsevier, v. 61, n. 1, p. 212–222, 2012.