MARCELO CALDAS SANTOS

MODELAGEM E PROJETO DE UM SISTEMA DE CONTROLE SUPERVISÓRIO PARA UM PROCESSO DE LAVAGEM DE MICROPLACAS

Belo Horizonte
17 de junho de 2008

Universidade Federal de Minas Gerais
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

MODELAGEM E PROJETO DE UM SISTEMA DE CONTROLE SUPERVISÓRIO PARA UM PROCESSO DE LAVAGEM DE MICROPLACAS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica.

Autor: Marcelo Caldas Santos
Orientador: Carlos Andrey Maia
Co-orientador: Denílson Laudares Rodrigues

Belo Horizonte
17 de junho de 2008

Resumo

Essa dissertação apresenta a utilização de sistemas a eventos discretos para a modelagem e o projeto de um sistema de controle supervisório para um processo de lavagem de microplacas utilizado em análises laboratoriais. O projeto é baseado em uma abordagem modular local que explora a modularidade da planta e das especificações de controle, evitando a explosão combinatorial de estados. O sistema de controle projetado garante o não-bloqueio e o cumprimento das especificações de forma minimamente restritiva, e sua arquitetura modular proporciona flexibilidade e facilidade de implementação. Apresentam-se, também, resultados de simulação que ilustram o comportamento dinâmico do sistema em malha fechada.

Palavras-chave: sistemas a eventos discretos, controle supervisório, automação laboratorial, controle supervisório modular local.

Abstract

This thesis presents the utilization of discrete events systems to the modeling and the project of a supervisory control system for a microplate washing process used in laboratory analysis. The project is based in a local modular approach which exploits the modularity of the plant and the control's specification, avoiding the explosion due to the state combination. The projected control system guarantees that the supervision is nonblocking and that the specifications are met in a minimal restrictive way. The modular architecture provides flexibility and easy implementation. Furthermore, results of simulation are presented, which illustrates the dynamic behavior of the closed loop system.

Keywords: discrete events systems, supervisory control, laboratorial automation, local modular supervisory control.

Agradecimentos

Agradeço primeiramente a Deus, que me capacitou e me deu paciência para finalizar esse trabalho, e "porque dele e por ele e para ele são todas as coisas"[1].

Agradeço à minha esposa Suellen pela compreensão e pelo apoio durante todo o período do mestrado.

Ao meu orientador Carlos Andrey Maia pelo incentivo, ajuda e grande disponibilidade. Também ao meu co-orientador Denílson Laudares Rodrigues pela idéia e grande apoio iniciais.

Agradeço a todos da CELER, que me apoiaram e permitiram que esse trabalho fosse realizado.

À minha família pelo encorajamento e por tudo que me ensinaram, que permitiu que eu chegasse até aqui.

Agradeço a todos os colegas de mestrado, faculdade e trabalho, por todo incentivo e convivência.

Aos amigos de longa data, Alexandre Leal Maia e Fabiano Franco da Rocha, pela companhia e amizade.

[1] Carta de Paulo aos Romanos, capítulo 11, verso 36.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de símbolos

$\exists$	existe
$\forall$	para todo
$\wedge$	operador "e"
:	tal que
2^X	conjunto potência de X: todos os subconjuntos de X
$f(x)!$	a função f é definida para x
G, H	*automaton* ou gerador
Q	conjunto de estados
Q_0	estado inicial
Q_m	conjunto de estados marcados
Σ	conjunto de eventos ou alfabeto
Σ^*	conjunto de todas as cadeias finitas formadas por Σ, incluindo ε
f	função de transição
Γ	função de eventos factíveis
Σ_c	conjunto de eventos controláveis
Σ_u	conjunto de eventos não-controláveis
ε	cadeia vazia
$\varnothing$	conjunto vazio

Lista de símbolos

L	linguagem
$\overline{L}$	prefixo fechamento de L
$\mathcal{L}(G)$	linguagem gerada por G
$\mathcal{L}_m(G)$	linguagem marcada por G
$Ac(G)$	operação que elimina todos os estados não-acessíveis de G
$Trim(G)$	operação que elimina todos os estados não-acessíveis e não-coacessíveis de G
$\|$	operador da composição paralela
$P_i : \Sigma^* \rightarrow \Sigma_i^*$	projeção natural de Σ^* para Σ_i^*
S	supervisor
$S(s)$	ação de controle de S para a cadeia s
S/G	sistema em malha fechada com S controlando G
$C(M, G)$	conjunto de sub-linguagens de M controláveis em relação a G
$SupC(M, G)$	máxima linguagem controlável de M em relação a G

Capítulo 1

Introdução

Nas últimas décadas, a indústria se desenvolve aceleradamente, impulsionada pelo grande desenvolvimento tecnológico e pela grande competitividade do mercado global. Os processos de produção tornaram-se complexos, buscando alcançar maior eficiência e atender a requisitos ambientais cada vez mais exigentes. Deseja-se que os sistemas produtivos sejam flexíveis e modulares, capazes de absorverem mudanças na demanda e serem facilmente aperfeiçoados. Tudo isso precisa ser feito de forma otimizada, segura e confiável. Esses requisitos demandam a utilização de ferramentas específicas, capazes de projetar sistemas de controle ótimos de forma eficaz.

Grande parte dos processos produtivos é dirigida a eventos, sendo chamados de sistemas a eventos discretos (SED's). Isso significa que não possuem uma dinâmica regida pela evolução do tempo, mas sim pela ocorrência de eventos. Apesar de ser possível representá-los por equações, chamadas *equações características*, não são representados por equações diferenciais ou de diferenças. Por exemplo, a atuação de um sensor, o início ou final de operação de uma bomba ou a chegada de uma peça, podem ser considerados eventos. Esses sistemas normalmente são controlados por uma "lógica de controle", que lida com os eventos por meio de diretivas condicionais.

1 Introdução

Os sistemas contínuos foram amplamente estudados ao longo dos anos, e diversas obras foram publicadas, como Newton, Gould & Kaiser (1957), Chen (1970), e os mais recentes Haykin & Veen (1999) e Chen (1999). Vários métodos de controle também foram estudados e desenvolvidos, e podem ser vistos em Seborg, Edgar & Mellichamp (1989), Ogata (1997) e Dorf & Bishop (2001). Da mesma forma, a teoria de Sistemas a Eventos Discretos permite a modelagem adequada dos sistemas dirigidos a eventos, e provê ferramentas que possibilitam a síntese formal de sistemas de controle minimamente restritivos e não-bloqueantes.

Os sistemas a eventos discretos são normalmente representados por *automata*[1] ou redes de Petri. A representação por *automata* tem sua origem na área de ciência da computação, o que pode ser visto em Hopcroft & Ullman (1979). Mais recentemente foram publicados trabalhos como Hopcroft, Motwani & Ullman (2000) e Cassandras & Lafortune (1999). A abordagem por redes de Petri tem sua origem no trabalho de Petri (1962), e trabalhos posteriores como Murata (1989). Outra representação possível é por meio de sistemas híbridos, que possuem dinâmica tanto contínua como discreta, podendo ser vista em Antsaklis & Koutsoukos (2002). Um estudo sobre a simulação de SED's pode ser encontrado em Banks, Carson, Nelson & Nicol (2000). Entre outras aplicações recentes, encontramos, por exemplo, a utilização de SED's no projeto de interface homem máquina (Masakazu Adachi 2006) e o projeto de um sistema de diagnóstico para SED's (Sampath, Lafortune & Teneketzis 1998).

A Teoria do Controle Supervisório inicialmente introduzida por Ramadge & Wonham (1987) baseia-se na modelagem via *automata*. Seu objetivo é projetar um supervisor que, em malha fechada com a planta, gere um comportamento que respeite determinadas restrições, garantindo que o sistema funcione conforme desejado. A atuação do supervisor consiste em

[1] A palavra *"automata"* é o plural de *"automaton"*.

desabilitar eventos controláveis da planta, quando os mesmos levarem-na a estados que violem as restrições. O sistema final projetado é minimamente restritivo, isto é, desabilita eventos controláveis o mínimo necessário para atender às especificações dadas.

Com o passar dos anos, a Teoria do Controle Supervisório tem sido aperfeiçoada, buscando, por exemplo, aproveitar a modularidade das especificações de controle (Ramadge & Wonham 1988, Ramadge & Wonham 1989). Outra abordagem, o controle modular local, foi explorado por Queiroz & Cury (2002), e o controle modular de sistemas a eventos discretos multitarefas foi explorado em Queiroz (2004) e Queiroz, Cury & Wonham (2005). Outros trabalhos sobre controle supervisório são encontrados em Wonham & Rogers (2007), Cunha & Cury (2007), Gaudin & Merchand (2007), Vahidi, Fabian & Lennartson (2006) e Cury (2001).

A síntese do controle baseado em redes de Petri pode ser feita de diferentes maneiras, como em Holloway, Krogh & Giua (1997), sendo possível para algumas classes de problemas, como o controle de redes não temporizadas baseado em invariantes lugar utilizado em Yamalidou, Moody, Lemmon & Antsaklis (1996) e Lima II (2002) e o controle via modelo de referência para redes p-temporizadas utilizado em Maia, Lüders, Mendes & Hardouin (2005).

Outros estudos diversos, lidando com implementação automática de lógica de controle, são encontrados em Frey (2000), que faz implementação automática em PLC de controle via redes de Petri, e Lee, Ang & Lee (2006) e Ljungkrantz, Akesson, Richardsson & Andersson (2007), que lidam com geração automática de lógica de controle para sistemas de manufatura em geral.

O presente trabalho utiliza o controle supervisório modular local baseado nos trabalhos Queiroz & Cury (2002) e Queiroz (2004) para projetar um sistema de controle para um equipamento de automação laboratorial, uma lavadora de microplacas. Essa abordagem, baseada na teoria do controle su-

pervisório, permite explorar a modularidade da planta, evitando a explosão combinatorial do número de estados. Por permitir a implementação em módulos, oferece grande flexibilidade e facilidade de modificação do sistema de controle.

Nesse projeto, a planta é dividida em sub-plantas, cada uma modelada por um *automaton*. Os objetivos de controle são traduzidos em especificações genéricas, representadas por *automata*. O sistema de controle projetado é formado por supervisores modulares, que controlam apenas uma parte específica da planta.

O controle supervisório modular local projetado é implementado por meio da linguagem $C++$, e seu funcionamento testado por meio de uma simulação da lavadora. A simulação foi implementada utilizando-se a teoria de sistemas a eventos discretos temporizados (Cassandras & Lafortune 1999).

Esse documento está dividido da seguinte maneira: o capítulo 2 apresenta um resumo da teoria de linguagens e *automata*, utilizada na representação dos SED's. Também apresenta a Teoria do Controle Supervisório, explicando as diferentes abordagens. O capítulo 3 explica o projeto do sistema de controle para a lavadora de microplacas, e como sua representação foi feita por meio de *automata*. Em seguida o capítulo 4 mostra a implementação do sistema de controle, e a simulação construída, apresentando alguns resultados obtidos. Por fim, as conclusões são apresentadas no capítulo 5.

Capítulo 2

Sistemas a Eventos Discretos e o Controle Supervisório

Sistemas a eventos discretos (SED's) são sistemas dinâmicos dirigidos à ocorrência de eventos. Diferentemente dos *sistemas contínuos*, as mudanças de estado ocorrem em saltos, e em níveis discretos. Sua evolução não é dirigida por uma dinâmica temporal, mas sim pela ocorrência de *eventos* em intervalos irregulares e, em geral, desconhecidos à priori (Queiroz 2004, Cassandras & Lafortune 1999).

A Figura 2.1 mostra um gráfico representando a evolução temporal do estado de um sistema a eventos discretos genérico. Nesse sistema os eventos ocorrem em intervalos irregulares, e o conjunto de estados é definido em níveis discretos, $\{s_1, s_2, s_3, s_4, s_5, s_6\}$. As transições entre estados ocorrem em saltos, e são determinadas por uma dinâmica interna inerente ao sistema.

Como a dinâmica dos SED's é regida pela ocorrência de eventos, é conveniente que seu comportamento seja representado por *linguagens*. As *linguagens* são conjuntos de *cadeias*, ou *palavras*, que são seqüências de eventos. Uma maneira de representar graficamente os SED's é por meio de *automata* (Hopcroft & Ullman 1979). Essa é a forma adotada nesse trabalho.

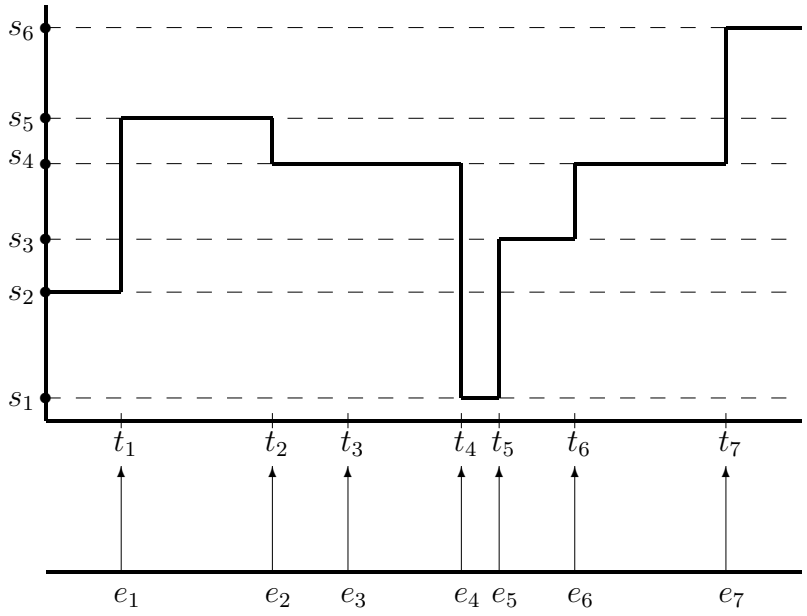


Figura 2.1: Evolução temporal do estado de um SED (Cassandras & Lafortune 1999).

Como foi dito anteriormente, a estratégia de controle adotada neste trabalho foi baseada na abordagem proposta por Ramadge & Wonham (1987), Ramadge & Wonham (1988) e Ramadge & Wonham (1989), chamada de Teoria do Controle Supervisório. Assim, nesse trabalho define-se o termo "*controle supervisório*" como o controle por meio de *automata* utilizando supervisores, como proposto por Ramadge & Wonham (1987), também conhecido como abordagem "RW". Faz-se aqui a distinção entre "*controle supervisório*" e "*sistema supervisório*", que são sistemas para automação industrial que provêm interface homem-máquina e diversas funções para controle de processos e aquisição de dados, também conhecidos como "*SCADA*" (*Supervisory Control And Data Acquisition*).

2.1 Linguagens e *automata*

Um ***automaton***, ou **gerador**, é uma representação gráfica do comportamento dinâmico de um SED. É formado por estados, que são ligados uns aos outros por setas, que representam os eventos. Um dos estados é o inicial, e um ou mais são estados marcados, representando os objetivos de controle, ou tarefas que foram completas.

A Figura 2.2 é um exemplo de *automaton* representando uma fila de clientes, com capacidade dois. O estado inicial, indicado por uma seta, representa a fila vazia. Esse estado também é o estado marcado, indicado pelos dois círculos concêntricos. Existem dois eventos, c e s, representando a chegada e a saída de clientes na fila, respectivamente. Caso a fila esteja cheia no estado 2, ocorrendo uma nova chegada, o cliente é desperdiçado e o sistema permanece no mesmo estado. O estado 2 possui uma seta que sai e entra nesse mesmo estado, o que é chamado de *auto-laço*.

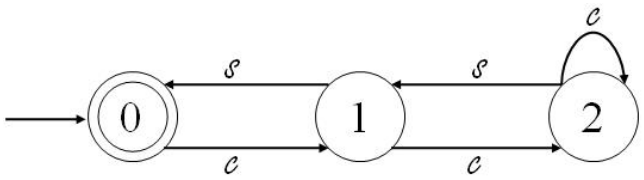


Figura 2.2: Exemplo de *automaton* representando uma fila de capacidade dois.

Um *automaton* pode ser representado pela sêxtupla $G = (Q, \Sigma, f, \Gamma, q_0, Q_m)$ (Cassandras & Lafortune 1999), sendo que:

- Q é o conjunto de estados;
- Σ é o conjunto de eventos, ou alfabeto;
- $f : Q \times \Sigma \rightarrow Q$, é a função de transição, definida parcialmente em cada

estado de Q, para um subconjunto de Σ. Ex: $f(q_1, \sigma) = q_2$ significa que a ocorrência do evento σ no estado q_1 leva ao estado q_2;

- $\Gamma : Q \rightarrow 2^{\Sigma}$, é a função de eventos factíveis. É o conjunto de todos os eventos que podem ocorrer em determinado estado, ou seja, todos os eventos para os quais a função f está definida. É representada por $\Gamma(q) = \{\sigma : \sigma \in \Sigma \wedge f(q, \sigma)!\}$[1];
- q_0 é o estado inicial;
- Q_m é o conjunto dos estados marcados.

No caso do exemplo dado na Figura 2.2:

◇ $Q = \{0, 1, 2\}$;

◇ $\Sigma = \{c, s\}$;

◇ A função de transição f é dada por:

$$\begin{array}{ll} f(0,c) = 1, & f(1,c) = 2, \\ f(1,s) = 0, & f(2,c) = 2, \\ f(2,s) = 1; & \end{array}$$

◇ A função de eventos factíveis Γ é dada por:

$$\Gamma(0) = \{c\}, \quad \Gamma(1) = \{c, s\}, \quad \Gamma(2) = \{c, s\};$$

◇ $q_0 = 0$;

◇ $Q_m = \{0\}$.

[1] $f(q, \sigma)!$ significa que a função $f(q, \sigma)$ é definida.

Eventos controláveis e não-controláveis

Na Teoria do Controle Supervisório proposta por Ramadge & Wonham (1987) considera-se que ocorrência dos eventos ocorre de forma espontânea na planta, e que alguns desses eventos podem ser impedidos de ocorrerem. Assim, para um *automaton* G, o conjunto Σ é dividido em dois sub-conjuntos:

Σ_c: Eventos controláveis, cuja ocorrência pode ser desabilitada pelo sistema de controle;

Σ_u: Eventos não-controláveis, cuja ocorrência não pode ser impedida, estando sempre habilitados.

Neste trabalho, os eventos não-controláveis são representados por setas tracejadas, e os controláveis por setas com traço cheio. Também se assume que a cadeia vazia ε faz parte do conjunto Σ.

Linguagens

Seja Σ um conjunto de eventos, também chamado **alfabeto**. Uma **linguagem** L, definida em Σ, é um conjunto de cadeias formadas por eventos de Σ. Ex:

$$\begin{array}{rcl} \Sigma & = & \{a, b, g\}, \\ L_1 & = & \{\varepsilon, a, abb\}, \\ L_2 & = & \{\text{Todas as possíveis cadeias com três eventos} \\ & & \text{iniciadas com o evento } a\}. \end{array} \tag{2.1}$$

Neste trabalho, todas as linguagens consideradas são regulares. As *linguagens regulares* podem ser representadas por meio de *automata* determinísticos de estados finitos, e todo *automaton* determinístico de estados finitos pode ser representado por uma linguagem regular.

2.1.1 Operações sobre cadeias e linguagens

Dado um alfabeto Σ, define-se Σ^* como o conjunto de todas as cadeias finitas formadas por elementos de Σ, incluindo a cadeia vazia ε (Cassandras & Lafortune 1999).

Sejam as cadeias $u = abb$ e $v = gb$. A **concatenação** dessas cadeias é dada por $uv = abbgb$.

Se $s = tuv$, sendo que t, u e $v \in \Sigma^*$, então:

- ε, t, tu e tuv são **prefixos** de s;
- u é uma **sub-cadeia** de s;
- v é um **sufixo** de s.

Sejam as linguagens L_1 e L_2. A **concatenação** L_1L_2 é definida como:

$$L_1L_2 = \{w : w = uv, u \in L_1, v \in L_2\}. \tag{2.2}$$

A concatenação entre duas linguagens L_1 e L_2 é uma linguagem L_3 cujas palavras são todas concatenações de palavras de L_1 e L_2.

Seja $L \in \Sigma^*$, então o **prefixo fechamento** de L é dado por:

$$\overline{L} := \{s \in \Sigma^* : \exists t \in \Sigma^* \wedge (st \in L)\}. \tag{2.3}$$

O prefixo fechamento é a linguagem que contém todos os prefixos de todas as palavras de L. Uma linguagem L é dita **prefixo-fechada** se $L = \overline{L}$.

O **fechamento de Kleene** de L é definido por (Cassandras & Lafortune 1999):

$$L^* := \{\varepsilon\} \cup L \cup LL \cup LLL \cup \ldots \tag{2.4}$$

Exemplos:

Sejam

- $E = \{a, b, g\}$;
- $L_1 = \{\varepsilon, a, abb\}$;
- $L_2 = \{g\}$.

Então:

- $L_1 L_2 = \{g, ag, abbg\}$;
- $\overline{L_1} = \{\varepsilon, a, ab, abb\}$;
- $\overline{L_2} = \{\varepsilon, g\}$;
- $L_1^* = \{\varepsilon, a, abb, aa, aabb, abba, abbabb, \ldots\}$;
- $L_2^* = \{\varepsilon, g, gg, ggg, \ldots\}$;
- $L_1 \cup L_2 = L_1 + L_2 = \{\varepsilon, a, abb, g\}$. ♦

O comportamento de um gerador G é definido por dois subconjuntos de Σ^*:

Linguagem gerada: formada por todas as seqüências de eventos que o gerador pode gerar, definida formalmente como:

$$\mathcal{L}(G) = \{s : s \in \Sigma^* \wedge f(q_0, s)!^2\}.$$

No caso do exemplo da Figura 2.2 a linguagem gerada é:

$$\mathcal{L}(G) = c(sc + cc^*s)^*(s + cc^* + \varepsilon) + \varepsilon.$$

[2]Lembrando que $f(q_0, s)!$ significa que a função $f(q_0, s)$ é definida.

Linguagem marcada: formada pelas seqüências de eventos que levam a estados marcados, representando as tarefas completas, definida formalmente como:

$$\mathcal{L}_m(G) = \{s : s \in \mathcal{L}(G) \wedge f(q_0, s) \in Q_m\}.$$

No caso do exemplo da Figura 2.2 a linguagem marcada é:

$$\mathcal{L}_m(G) = c(sc + cc^*s)^*s + \varepsilon.$$

Notar que:

- $\mathcal{L}(G)$ é prefixo-fechada;
- $\mathcal{L}_m(G) \subseteq \mathcal{L}(G)$.

Equivalência de *automata*

Dois *automata* G_1 e G_2 são ditos **equivalentes** se ambas suas linguagens geradas e marcadas forem iguais (Cassandras & Lafortune 1999):

$$\begin{aligned} \mathcal{L}(G_1) &= \mathcal{L}(G_2); \\ \mathcal{L}_m(G_1) &= \mathcal{L}_m(G_2). \end{aligned} \tag{2.5}$$

Assim, dois *automata* podem ter funções de transições e quantidades de estados diferentes, mas serão equivalentes se suas linguagens geradas e marcadas forem iguais.

2.1.2 Operações sobre *automata*

Um estado $q \in Q$ é chamado de *acessível* se $\exists s \in \Sigma^*$ tal que $f(q_0, s) = q$, ou seja, se existe uma cadeia s que, partindo de q_0, leve a q. É chamado de *coacessível* se $\exists s \in \Sigma^*$ tal que $f(q, s) \in Q_m$, ou seja, se existe uma cadeia s que, partindo de q, leve a um estado marcado.

Define-se a operação **Ac**(**G**) como a eliminação de todos os estados não-acessíveis de G. A operação **Trim**(**G**) elimina todos os estados não-acessíveis e os não-coacessíveis, e o resultado é um *automaton* chamado *aparado* ou *trim*.

Um *automaton* é dito ser **bloqueante** se possui pelo menos um estado não-coacessível. Se todos os estados de G forem coacessíveis, então G é **não-bloqueante**. Para que G seja não-bloqueante é necessário que $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$ (Cassandras & Lafortune 1999).

Composição paralela

Sejam os geradores $G_i, i = 1, \ldots, n$. A **composição paralela** $G = \|_{i=1}^{n} G_i$ é obtida fazendo-se a evolução em paralelo dos n geradores G_i, na qual um evento comum a múltiplos geradores só é executado se todos os geradores que contiverem este evento o executarem simultaneamente (Queiroz 2004).

Sejam os geradores G_1 e G_2. A composição paralela entre G_1 e G_2 é definida formalmente por:

$$G_1 \parallel G_2 := Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f, \Gamma, (q_{01}, q_{02}), Q_{m1} \times Q_{m2}),$$

sendo

$$f((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)), \text{ se } \sigma \in \Gamma_1(q_1) \cap \Gamma_2(q_2), \\ (f_1(q_1, \sigma), q_2), \text{ se } \sigma \in \Gamma_1(q_1) - \Sigma_2, \\ (q_1, f_2(q_2, \sigma)), \text{ se } \sigma \in \Gamma_2(q_2) - \Sigma_1, \\ \text{indefinida, senão.} \end{cases} \tag{2.6}$$

Um evento só será executado se estiver ativo em G_1 e G_2, ou se estiver ativo em G_1 e não pertencer a Σ_2, ou se estiver ativo em G_2 e não pertencer a Σ_1. A função de eventos ativos Γ é dada por:

$$\begin{aligned} \Gamma((q_1, q_2)) &= [\Gamma_1(q_1) \cup (\Sigma_2 - \Sigma_1)] \cap [\Gamma_2(q_2) \cup (\Sigma_1 - \Sigma_2)], \\ &= [\Gamma_1(q_1) \cap \Gamma_2(q_2)] \cup [\Gamma_1(q_1) - \Sigma_2] \cup [\Gamma_2(q_2) - \Sigma_1]. \end{aligned} \tag{2.7}$$

Exemplos:

Sejam os dois *automata* G_1 e G_2 da Figura 2.3:

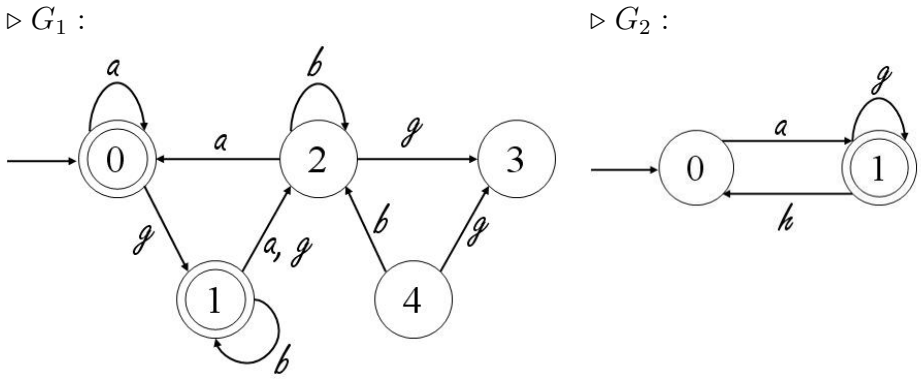
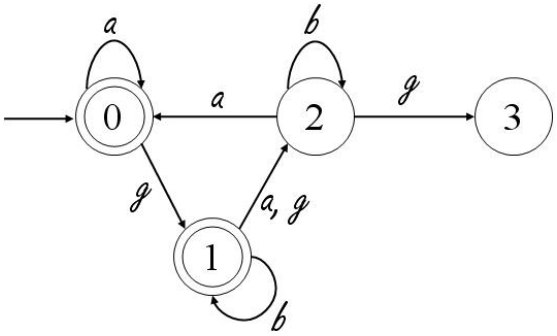
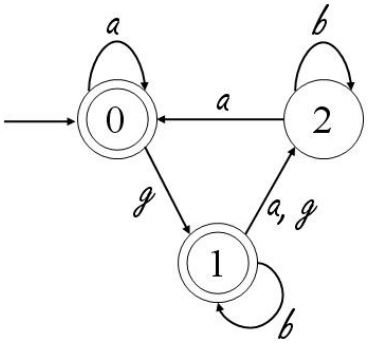


Figura 2.3: *Automata* usados como exemplos para várias operações.

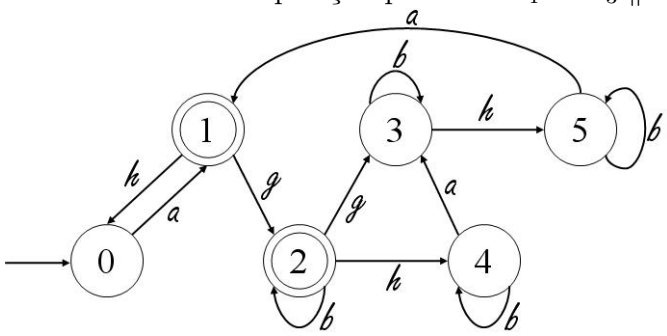
- ▷ O *automaton* G_1 é bloqueante, pois uma vez no estado 3, ele não consegue alcançar nenhum estado marcado. Esse estado é, portanto, não-coacessível.
- ▷ O estado 4 do *automaton* G_1 é não-acessível.
- ▷ A operação $Ac(G_1)$ resulta em:



- ▷ O *automaton* G_3, resultante de $G_3 = Trim(G_1)$ é:



▷ O resultado da composição paralela $G_4 = G_3 \parallel G_2$ é:



♦

Projeção natural

Sejam Σ e Σ_i conjuntos de eventos tal que $\Sigma_i \subset \Sigma$. A **projeção natural** de Σ^* para Σ_i^*, $P_i : \Sigma^* \to \Sigma_i^*$, é definida como (Queiroz 2004):

$$\begin{aligned} &P_i(\varepsilon) = \varepsilon, \\ &P_i(e) = \begin{cases} \varepsilon \text{ se } e \notin \Sigma_i, \\ e \text{ se } \in \Sigma_i; \end{cases} \\ &P_i(ue) = P_i(u)P_i(e), \text{ sendo que } u \in \Sigma^*; e \in \Sigma. \end{aligned} \tag{2.8}$$

Também se define o conceito de projeção natural para linguagens:

$$P_i(L) = \{u_i \in \Sigma_i^* : u_i = P_i(u) \text{ para algum } u \in L\}. \tag{2.9}$$

A projeção inversa é definida como:

$$P_i^{-1}(L_i) = \{u \in \Sigma^* : P_i(u) \in L_i\}. \tag{2.10}$$

O conceito de projeção natural para linguagens é utilizado para definir formalmente as linguagens gerada e marcada pela composição paralela entre dois *automata*. Sejam G_1 e G_2, então:

$$\begin{aligned} \mathcal{L}(G_1 \parallel G_2) &= P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)], \\ \mathcal{L}_m(G_1 \parallel G_2) &= P_1^{-1}[\mathcal{L}_m(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)]. \end{aligned} \tag{2.11}$$

2.2 Modelagem do sistema

O passo inicial para o projeto de um controle supervisório é construir uma representação do sistema por meio de *automata*. Primeiro, define-se o conjunto de eventos do sistema, formado por eventos controláveis e não-controláveis. Em seguida, cria-se *automata* de forma que as transições entre os estados correspondam às mudanças de estado na planta, e a linguagem gerada represente o funcionamento do sistema.

Modelagem da planta

Como a maioria dos sistemas são compostos, isto é, formados por subsistemas concorrentes interagindo entre si (Ramadge & Wonham 1989, Queiroz 2004), a planta pode ser modelada por várias subplantas. Assim, modela-se por meio de *automata* o comportamento de cada subplanta separadamente, obtendo um conjunto de geradores $G_i, i = 1, \ldots, n$. O comportamento da planta global pode ser obtido pela composição paralela $G = \|_{i=1}^{n} G_i$. Entretanto, essa composição tem complexidade exponencial em relação a "n", o que é uma desvantagem que será discutida a seguir.

Modelagem das especificações

O próximo passo para o projeto de um controle supervisório é definir quais são os objetivos de controle do sistema. Esses objetivos podem ser descritos

na forma de requisitos de controle. Da mesma forma que para a planta, os objetivos de controle devem ser representados por *automata*, e podem-se construir vários geradores, chamados especificações genéricas de controle: $E_{gen,j}, j = 1, \ldots, m$. A evolução desses geradores gerará uma linguagem que restringe o comportamento da planta, evitando estados indesejados ou proibidos, de forma a obedecer aos objetivos de controle.

Como será melhor explicado a seguir, a composição das especificações locais fornece uma especificação genérica monolítica:

$$E_{gen} = \|_{j=1}^{m} E_{gen,j};$$

Então, o comportamento desejado para o sistema é expresso por uma única linguagem chamada especificação global $K_{global} \subseteq \mathcal{L}_m(G)$, que é obtida por:

$$K_{global} = E_{gen} \parallel \mathcal{L}_m(G).$$

A especificação global é uma restrição feita à linguagem marcada da planta que reflete os requisitos de controle.

2.3 Controle Monolítico

O objetivo do controle supervisório monolítico é projetar um único supervisor S cujo objetivo é fazer com que o sistema em malha fechada não transgrida o comportamento desejado, descrito pela especificação global K_{global} (Ramadge & Wonham 1987). O supervisor atua desabilitando a ocorrência de eventos controláveis na planta.

A Figura 2.4 mostra a estrutura de controle em malha fechada de um supervisor monolítico. Para cada cadeia de eventos "s" sinalizada pela planta G, o supervisor S gera uma ação de controle $S(s)$, que consiste em um conjunto de eventos habilitados. Como o supervisor apenas habilita ou desabilita eventos, diz-se que ele tem uma natureza *permissiva* (Queiroz 2004).

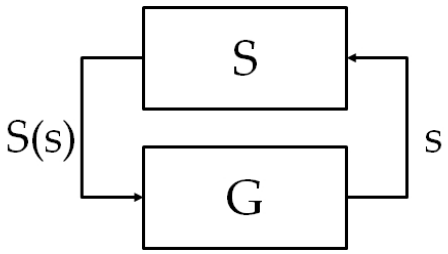


Figura 2.4: Representação do controle monolítico.

O supervisor é definido formalmente como uma função da linguagem gerada por G no conjunto dos subconjuntos de Σ:

$$S : \mathcal{L}(G) \rightarrow 2^{\Sigma}. \tag{2.12}$$

Ou seja, para cada cadeia da linguagem gerada por G, o supervisor gera um conjunto de eventos habilitados, que é um subconjunto de Σ.

Os eventos desabilitados pelo supervisor após a ocorrência da cadeia "s" são obtidos por $\Gamma(f(q_0, s)) - S(s)$, sendo Γ a função de eventos factíveis da planta. Ou seja, a ação de controle após a ocorrência da cadeia "s" é obtida subtraindo-se do conjunto de eventos ativos da planta os eventos desabilitados pelo supervisor S.

Para que seja possível realizar a ação de controle gerada, ela precisa incluir sempre os eventos não-controláveis habilitados na planta. Ou seja, o supervisor só pode desabilitar eventos não-controláveis que também estejam desabilitados na planta. Caso isso ocorra, o supervisor é dito *admissível*, o que é definido formalmente por:

$$\forall s \in \mathcal{L}(G), \Sigma_u \cap \Gamma(f(q_0, s)) \subseteq S(s). \tag{2.13}$$

A estrutura do supervisor e da planta em malha fechada é representada por S/G, e é definida recursivamente por:

$$\begin{aligned} &\varepsilon \in \mathcal{L}(S/G), \\ &s\sigma \in \mathcal{L}(S/G) \Leftrightarrow (s \in \mathcal{L}(S/G)) \wedge (s\sigma \in \mathcal{L}(G)) \wedge (\sigma \in S(s)); \end{aligned} \tag{2.14}$$

O comportamento marcado do sistema fechado é dado por $\mathcal{L}_m(S/G) = \mathcal{L}(S/G) \cap \mathcal{L}_m(G)$.

Um supervisor S é chamado *próprio* quando for não-bloqueante, ou seja, quando o comportamento em malha fechada do sistema for não-bloqueante, como definido na Seção 2.1.2. Isto é assegurado se $\overline{\mathcal{L}_m(S/G)} = \mathcal{L}(S/G)$.

Um supervisor S pode ser representado por um *automaton* H, cujo conjunto de eventos esteja contido em Σ. O *automaton* H é construído de forma que os eventos desabilitados por S também sejam desabilitados por H. Ou seja, após a ocorrência da cadeia s, os eventos factíveis em H são os eventos habilitados por S. Assim, se S for admissível, o comportamento do sistema S/G é representado por H/G, que, por sua vez, é equivalente à composição paralela $H \parallel G$. As linguagens geradas e marcadas de H/G são dadas por:

$$\begin{aligned} \mathcal{L}(H/G) &= \mathcal{L}(H \parallel G), \\ \mathcal{L}_m(H/G) &= \mathcal{L}_m(H \parallel G). \end{aligned} \tag{2.15}$$

Para que todos os estados marcados na planta permaneçam marcados após a utilização do supervisor, todos os estados deste precisam ser também marcados. Isso é necessário para que a composição paralela da equação 2.15 conserve os estados marcados da planta.

Entretanto, pode fazer sentido, para determinados problemas, que o supervisor seja capaz de desmarcar estados da planta. Isso pode acontecer, por exemplo, quando certas combinações de estados marcados em diferentes sub-plantas levarem à um estado na planta global que não corresponda a uma tarefa completa, o que requereria uma intervenção do supervisor. Nesses casos nem todos os estados do supervisor são marcados, e ele é chamado de *desmarcador*.

2.3.1 Existência de Supervisores

Uma linguagem $K \subseteq \mathcal{L}_m(G)$ é *controlável* em relação a G, ou é uma sublinguagem controlável de $\mathcal{L}(G) \subseteq \Sigma^*$ se:

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}. \tag{2.16}$$

Isso significa que qualquer evento não-controlável, que ocorra após qualquer cadeia de $\overline{K}$, ainda mantém a seqüência no conjunto $\overline{K}$.

Uma linguagem K é dita $\mathcal{L}_m(G)$ *fechada* se $K = \overline{K} \cap \mathcal{L}_m(G)$, ou seja, se todos os prefixos de K que são palavras de $\mathcal{L}_m(G)$ forem também palavras de K.

A existência de um supervisor não-bloqueante S, que atenda uma especificação global $K_{global} \subseteq \mathcal{L}_m(G)$, ou seja $\mathcal{L}_m(S/G) = K_{global}$, está condicionada à controlabilidade e ao $\mathcal{L}_m(G)$-fechamento de K_{global} (Ramadge & Wonham 1987).

Uma especificação K_{global} é controlável se não requerer a desabilitação direta de nenhum evento não-controlável; e é $\mathcal{L}_m(G)$-fechada se não desmarcar nenhum estado da planta sendo controlada. Entretanto, para um supervisor desmarcador, apenas a controlabilidade é necessária (Queiroz 2004).

Seja a linguagem M tal que $M \subseteq \mathcal{L}(G)$. O conjunto de sub-linguagens de M controláveis em relação a G é denotado por $C(M,G) := \{K : K \subseteq M$ e K é controlável em relação a $G\}$, e, por ser fechado sobre união e não-vazio ($\varnothing \in C(M,G)$), contém um único elemento supremo, chamado $SupC(M,G)$ (Ramadge & Wonham 1989, Queiroz 2004). Isso quer dizer que existe um elemento de $C(M,G)$ que representa a máxima linguagem controlável para a especificação M, e esse elemento é único.

A Figura 2.5 é um diagrama que mostra a relação entre a linguagem da planta $\mathcal{L}(G)$, a linguagem $M \subseteq \mathcal{L}(G)$, o conjunto $C(M,G)$ e a máxima linguagem controlável $SupC(M,G)$.

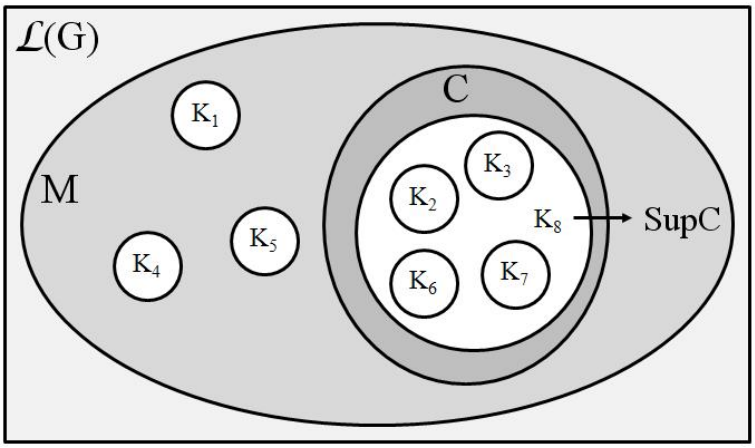


Figura 2.5: Diagrama representando a máxima linguagem controlável.

Muitas vezes um comportamento especificado para a planta não é controlável. Nesses casos pode-se projetar um supervisor que atenda a especificação de forma minimamente restritiva, ou seja, que se aproxime o máximo da especificação K e que seja controlável. Para isso calcula-se, utilizando algoritmos específicos, a máxima linguagem controlável para aquela especificação. O objetivo é, então, sintetizar um supervisor S para uma linguagem $K \subseteq \mathcal{L}(G)$, tal que $\mathcal{L}_m(S/G) = SupC(K, G)$. Se $Sup(K, G)$ não atender aos requisitos de controle de forma satisfatória, então o problema de controle não tem solução.

A obtenção de um supervisor monolítico S é uma tarefa computacionalmente complexa. Isso porque plantas com muitos estados demandam, em geral, supervisores com muitos estados. É possível obter um supervisor pela composição paralela de diversas especificações, entretanto isso pode ocasionar explosão do número de estados, dependendo da quantidade das especificações. Além disso, a implementação de um supervisor único e com grande quantidade de estados é trabalhosa, e uma pequena alteração na estratégia de controle demandaria modificá-lo completamente.

Tendo em vista essas desvantagens, o controle modular é uma alternativa ao controle monolítico, e seu objetivo é aproveitar a modularidade das

especificações.

2.4 Controle Modular

Normalmente, em um projeto de um controle supervisório para uma planta, são necessárias muitas especificações para expressar o comportamento desejado, que é a especificação global. Assim, o controle modular apresenta uma alternativa para um projeto mais simples do supervisor. O objetivo é projetar um supervisor para cada especificação, de modo que a atuação deles em conjunto satisfaça a especificação global.

A Figura 2.6 representa o funcionamento da malha fechada usando o controle modular. Essa figura mostra dois supervisores S_1 e S_2 que, para cada cadeia "s" gerada pela planta, geram as ações de controle $S_1(s)$ e $S_2(s)$. A ação de controle final é a interseção $S_1 \cap S_2$.

Assim, no controle modular cada supervisor S_j gera uma ação de controle $S_j(s), j = 1, \ldots, m$, cada um objetivando alcançar um objetivo de controle. A ação de controle final é formada pela interseção das habilitações:

$$S(s) = \bigcap_{j=1}^{m} S_j(s).$$

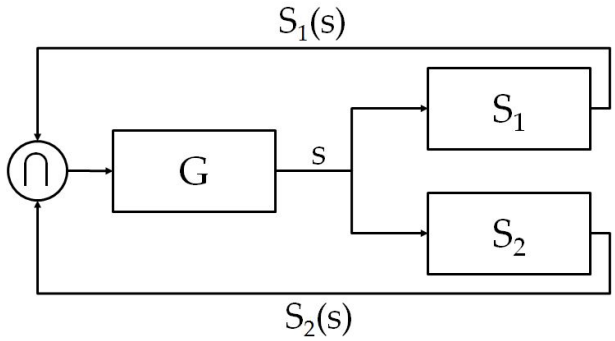


Figura 2.6: Representação do controle modular.

Cada supervisor tem sua ação de controle baseada em uma versão parcial da planta, correspondente aos eventos que por ele são habilitados ou não. Assim, esse tipo de controle pode gerar conflito quando os supervisores estiverem operando em conjunto. Para contornar esse problema, utiliza-se a definição de modularidade a seguir.

Modularidade

Sejam as linguagens $L_i \subseteq \Sigma^*$, $i = 1, \ldots, n$. Esse conjunto é não conflitante, ou modular, se $\cap_{i=1}^{n} \overline{L_i} = \overline{\cap_{i=1}^{n} L_i}$ (Ramadge & Wonham 1989). Isso quer dizer que qualquer palavra que for prefixo de todas as linguagens simultaneamente, será um prefixo do conjunto como um todo, e, portanto, permitirá que o sistema global alcance um estado marcado, não provocando bloqueio.

Para que o controle modular não seja conflitante, é necessário que as linguagens marcadas pelos supervisores sejam modulares (Ramadge & Wonham 1989). Caso isso seja verificado, o controle modular não implicará em perda de desempenho e poderá ser usado com vantagens em relação ao controle monolítico.

O controle modular apresenta maior flexibilidade, já que o projeto e a implementação dos supervisores modulares são mais simples, pois eles são menores. Além disso, a modificação do sistema de controle é muito mais fácil, por permitir a modificação de apenas uma especificação. Entretanto, uma desvantagem do controle monolítico permanece na abordagem modular, a explosão do número de estados na composição da planta global. Dessa forma, para um sistema composto, formado por várias sub-plantas, o controle modular não é tão vantajoso, e o controle modular local apresenta uma alternativa mais adequada.

2.5 Controle Modular Local

A abordagem do controle modular local explora, além da modularidade das especificações, a modularidade da planta. Assim, a planta é dividida em subsistemas, representados por modelos de menor tamanho, o que diminui a complexidade computacional da síntese dos supervisores.

A Figura 2.7 mostra o funcionamento da malha fechada do controle modular local. Os dois supervisores S_1 e S_2 geram as ações de controle $S_1(s_1)$ e $S_2(s_2)$, e controlam apenas partes da planta, chamadas plantas locais G_1 e G_2. Cada supervisor "enxerga" apenas uma parte dos eventos ocorridos na planta, que são as cadeias s_1 e s_2.

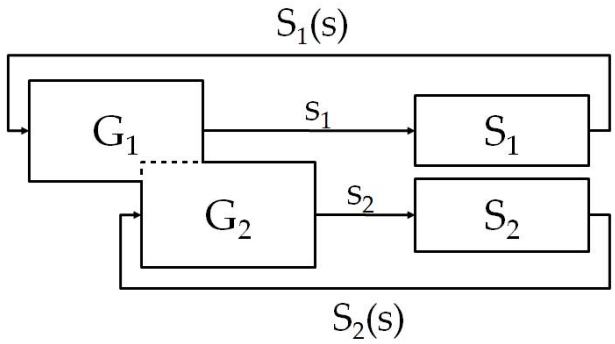


Figura 2.7: Representação do controle modular local.

Assim, no controle modular local assume-se que a planta seja formada por subsistemas, modelados por vários *automata*, $G_i = (Q_i, \Sigma_i, f_i, \Gamma_i, q_{0i}, Q_{mi})$, $i \in I = \{1, \ldots, n\}$. Constróem-se, então, especificações genéricas locais $E_{gen,j}$, $j = 1, \ldots m$, definidas em subconjuntos de eventos $\Sigma_{gen,j} \subseteq \Sigma$, de forma que expressem os objetivos de controle.

O comportamento desejado do sistema é, então, expresso localmente em relação apenas às plantas afetadas por cada especificação genérica local. Assim, definem-se *plantas locais* para cada especificação genérica:

$$\begin{array}{l} G_{loc,j} = \|_{i \in I_{loc,j}} G_i, \\ \text{sendo} \\ I_{loc,j} = \{k \in I | \Sigma_k \cap \Sigma_{gen,j} \neq \varnothing\}. \end{array} \tag{2.17}$$

Cada planta local $G_{loc,j}$ é uma composição paralela das sub-plantas afetadas direta e indiretamente pela especificação local $E_{gen,j}$. Ou seja, a planta local é a composição paralela de toda planta cujo conjunto de eventos Σ_i possua algum evento comum ao conjunto de eventos da especificação genérica $\Sigma_{gen,j}$.

O comportamento desejado para o sistema, é, então, representado por um conjunto de especificações locais, calculadas por:

$$K_{loc,j} = E_{gen,j} \parallel \mathcal{L}_m(G_{loc,j}). \tag{2.18}$$

A planta que engloba apenas os subsistemas afetados por alguma especificação local é chamada de *planta enxuta*, e é definida como $G_e = \|_{j=1}^{m} G_{loc,j}$ (Queiroz 2004). Nesse trabalho assume-se que a planta global G é equivalente à planta enxuta G_e.

Modularidade local

Um conjunto de linguagens $\{L_j, j = 1, \ldots, n\}$ é localmente modular se $\|_{j=1}^{n} \overline{L_j} = \overline{\|_{j=1}^{n} L_j}$. Se as linguagens $\{L_j, j = 1, \ldots, n\}$ forem todas marcadas por geradores aparados H_j, pode-se provar que o conjunto é localmente modular se e somente se $H = \|_{j=1}^{n} H_j$ for um gerador aparado.

Sejam $E_{gen,j}, j = 1, \ldots, m$, especificações genéricas sobre um sistema a eventos discretos, e seja este representado na forma de um sistema-produto, isto é, formado por várias sub-plantas $\{G_i, i = 1, \ldots, n\}$. De acordo com Queiroz & Cury (2002), se o conjunto $\{SupC(K_{loc,j}, G_{loc,j}), j = 1, \ldots, m\}$ for

localmente modular, então:

$$SupC(K_{global}, G) = \|_{j=1}^{m} SupC(K_{loc,j}, G_{loc,j}), \tag{2.19}$$

Isso quer dizer que a solução modular local, utilizando as máximas linguagens controláveis, será equivalente à solução monolítica.

Portanto, para que a solução modular local não implique em perda de desempenho em relação ao controle monolítico, é necessário que os supervisores sejam localmente modulares. Dado um conjunto de supervisores $\{S_j, j = 1, \ldots, n\}$, a modularidade local é verificada se, e somente se, $S = \|_{i=1}^{n} S_i$ for um gerador aparado.

Assim, se uma especificação local for controlável em relação a sua respectiva planta local $G_{loc,j}$, o gerador $K_{loc,j}$ aparado pode ser tomado como um supervisor não-bloqueante $S_{loc,j}$. Caso a controlabilidade não seja alcançada, calcula-se a máxima linguagem controlável $SupC(K_{loc,j}, G_{loc,j})$, e o gerador que marca essa linguagem é tomado como supervisor não bloqueante ótimo[3] $S_{loc,j}$. Esse procedimento é repetido para cada supervisor local, e se a modularidade local for verificada, não haverá perda de desempenho em relação ao controle monolítico.

Observa-se que a solução modular local é vantajosa em relação às outras abordagens, por aproveitar tanto a modularidade das especificações quanto da planta. Isso torna o projeto do sistema de controle mais flexível, pois permite a alteração de um supervisor local separadamente. Além disso, sua implementação é mais simples, pois não há necessidade de fazer a composição paralela de todos os supervisores.

Entretanto, a desvantagem do método é que a verificação da modularidade local exige a composição paralela de todas as especificações locais. Dependendo da planta a ser controlada essa composição pode ser inviável computacionalmente, pois o tamanho do sistema composto cresce exponenci-

[3]Nesse trabalho, "ótimo" quer dizer minimamente restritivo.

almente com a quantidade de especificações e subsistemas envolvidos. Além disso, a verificação da modularidade local será necessária sempre que uma modificação for feita no sistema de controle. Portanto, conclui-se que novos métodos devem ser desenvolvidos para que a modularidade possa ser verificada sem a necessidade da composição das especificações.

Capítulo 3

Projeto do controle supervisório

A planta para a qual o controle supervisório foi projetado é uma lavadora de microplacas, equipamento de automação laboratorial utilizado para lavar microplacas utilizadas em exames do tipo ELISA.

Exames ELISA (*Enzyme Linked Immuno Sorbent Assay*) se baseiam no princípio de que antígenos ou anticorpos que se fixam à fase sólida, como a parede da microplaca, podem ser detectados por um anticorpo ou antígeno complementar, chamado de conjugado. O teste ELISA é usado no diagnóstico de várias doenças que induzem o sistema imunológico humano a produzir anticorpos. A principal aplicação é para infecciosas, mas também pode ser usado no diagnóstico de doenças auto-imunes ou alergias.

Normalmente uma placa de superfície inerte é utilizada, a microplaca. Primeiro os poços são preenchidos com as amostras, contendo as proteínas de interesse. Depois é realizada uma lavagem com uma proteína inespecífica, para que esta ocupe os espaços livres dos poços. A superfície é, então, tratada com uma solução de um anticorpo primário específico para aquela proteína de interesse, e que se ligará a ela. A superfície é, então, lavada no-

vamente para retirar os anticorpos primários que não foram incorporados em nenhuma proteína. Em seguida, o produto é tratado com anticorpos secundários, que possuem uma enzima acoplada capaz de produzir uma substância corada, e que é um anticorpo do anticorpo primário. A superfície é lavada novamente para a retirada do excesso do anticorpo secundário. Adiciona-se o substrato de ligação para que a enzima produza a substância corada e, assim, medindo-se a intensidade de cor da superfície, pode-se quantificar ou verificar a presença de alguma substância de interesse (*ORTHO, HIV-1/HIV-2 ab-Capture ELISA Test System* n.d.). A lavadora é utilizada em todas as etapas onde há necessidade de lavar a superfície da microplaca.

As microplacas são formadas por micropoços, e podem ter vários tamanhos, sendo que a utilizada neste trabalho possui noventa e seis poços, sendo doze tiras com oito poços em cada. A Figura 3.1 mostra a microplaca utilizada nesse trabalho. Suas dimensões típicas são: 85mm × 127mm × 14mm.

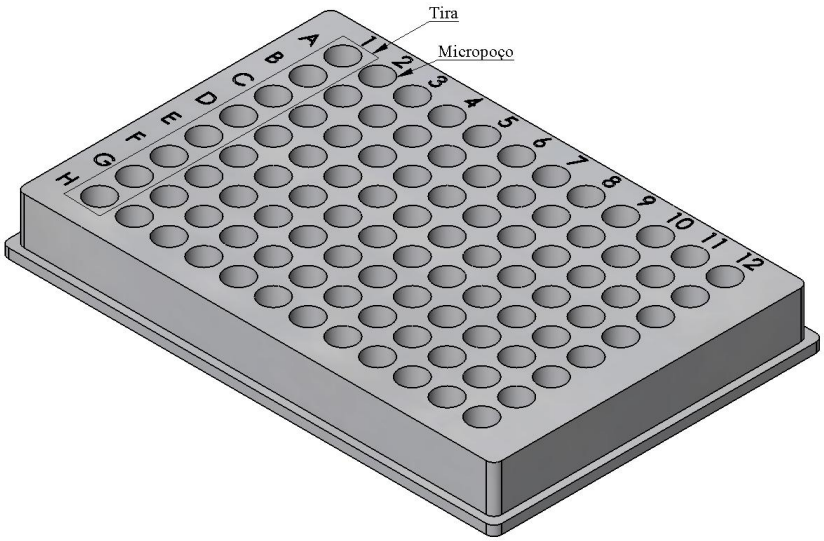


Figura 3.1: Ilustração da microplaca de noventa e seis poços utilizada nesse trabalho.

3.1 Descrição da planta

A lavadora é um equipamento capaz de injetar um volume controlado de uma solução de lavagem em cada poço de uma microplaca, um número de vezes especificado pelo usuário. Antes de injetar a solução de lavagem, o conteúdo do poço é aspirado por uma bomba. Vários parâmetros são configurados pelo usuário, que variam de acordo com o tipo de ensaio que está sendo realizado na placa.

Algumas características do equipamento são importantes:

- ▷ Baixo tempo de lavagem;
- ▷ Baixo volume residual: volume de solução deixado na microplaca depois de ser completamente aspirada pelo cabeçote;
- ▷ Alta precisão no controle do volume de solução injetado.

O projeto do sistema de controle de uma lavadora deve visar otimizar seu funcionamento, para que as características anteriormente citadas sejam alcançadas.

3.1.1 Sistemas da lavadora

A lavadora possui vários mecanismos, acionados por motores de passo, uma bomba de vácuo e diversos dispositivos eletrônicos. A Figura 3.2 mostra suas principais partes, que são descritas a seguir:

- ▶ **Reservatório de solução de lavagem**: armazena a solução que é usada para lavar a microplaca. Possui uma conexão que vai para a válvula 2;
- ▶ **Reservatório de água destilada**: armazena a água destilada, que é usada para limpar a tubulação ao final da operação. Isso evita que as

agulhas de injeção entupam devido à cristalização dos sais da solução de lavagem. Possui uma conexão que vai para a outra entrada da válvula 2;

- **Bomba de vácuo**: aspira ar de dentro do esgoto e o expira para o exterior da lavadora;

- **Esgoto**: recipiente fechado hermeticamente que recebe o que é sugado pelo cabeçote de agulhas. Possui duas conexões: uma saída de ar que vai para a bomba de vácuo e uma entrada de líquido que vem diretamente das agulhas de sucção do cabeçote de agulhas. A bomba de vácuo suga ar de dentro do esgoto criando vácuo, fazendo com que o cabeçote sugue os líquidos pelas agulhas de sucção. O esgoto possui um sensor de nível que detecta quando o nível está muito alto e evita que a bomba de vácuo aspire algum líquido, o que a danificaria;

- **Válvula 1**: válvula de três vias, sendo que a primeira vai para a válvula 2, a segunda para a bomba de seringa e a terceira para as agulhas de injeção. Possui apenas duas posições válidas, 0^o e 90^o. Quando em 0^o, liga a bomba de seringa às agulhas de injeção, permitindo que a seringa injete seu conteúdo na microplaca. Quando em 90^o, liga a bomba de seringa à válvula 2, permitindo que a bomba de seringa se encha de água destilada ou solução de lavagem, de acordo com a posição da válvula 2.

- **Válvula 2**: válvula de três vias, sendo que a primeira está ligada ao reservatório de solução de lavagem, a segunda ao reservatório de água destilada e a terceira está ligada à válvula 1. Existem duas posições válidas, 0^o e 90^o. Quando em 0^o, liga a válvula 1 ao reservatório de água destilada. Quando em 90^o, a válvula 1 é ligada ao reservatório de solução de lavagem.

- **Cabeçote de agulhas**: peça que possui oito agulhas de injeção e oito agulhas de sucção. Realiza a lavagem e a sucção do conteúdo de cada tira. As oito agulhas entram cada uma simultaneamente em cada poço de uma tira da microplaca.

  As oito agulhas de injeção estão ligadas em um duto comum conectado a uma mangueira que vai para a válvula 1. A distribuição de líquido entre as agulhas de injeção deve ser a mais homogênea possível. Para isso, a diferença de diâmetro entre as agulhas e o duto comum que as liga é a maior possível, reduzindo a perda de carga.

  As oito agulhas de sucção estão ligadas em outro duto de onde sai uma mangueira que vai diretamente para o esgoto. O cabeçote de agulhas é acionado por dois motores de passo que fazem sua movimentação nos sentidos x (horizontal) e y (vertical).

- **Bomba de seringa**: dispositivo formado por uma seringa de vidro, cujo êmbolo é acionado por um motor de passo por meio de um fuso. A bomba de seringa serve para controlar de forma precisa o volume que é injetado através das agulhas de injeção. Seu funcionamento é controlado pela válvula 1: quando a válvula 1 está posicionada em 90^o, a bomba se enche; quando a válvula 1 está em 0^o, a bomba esvazia seu conteúdo pelas agulhas de injeção.

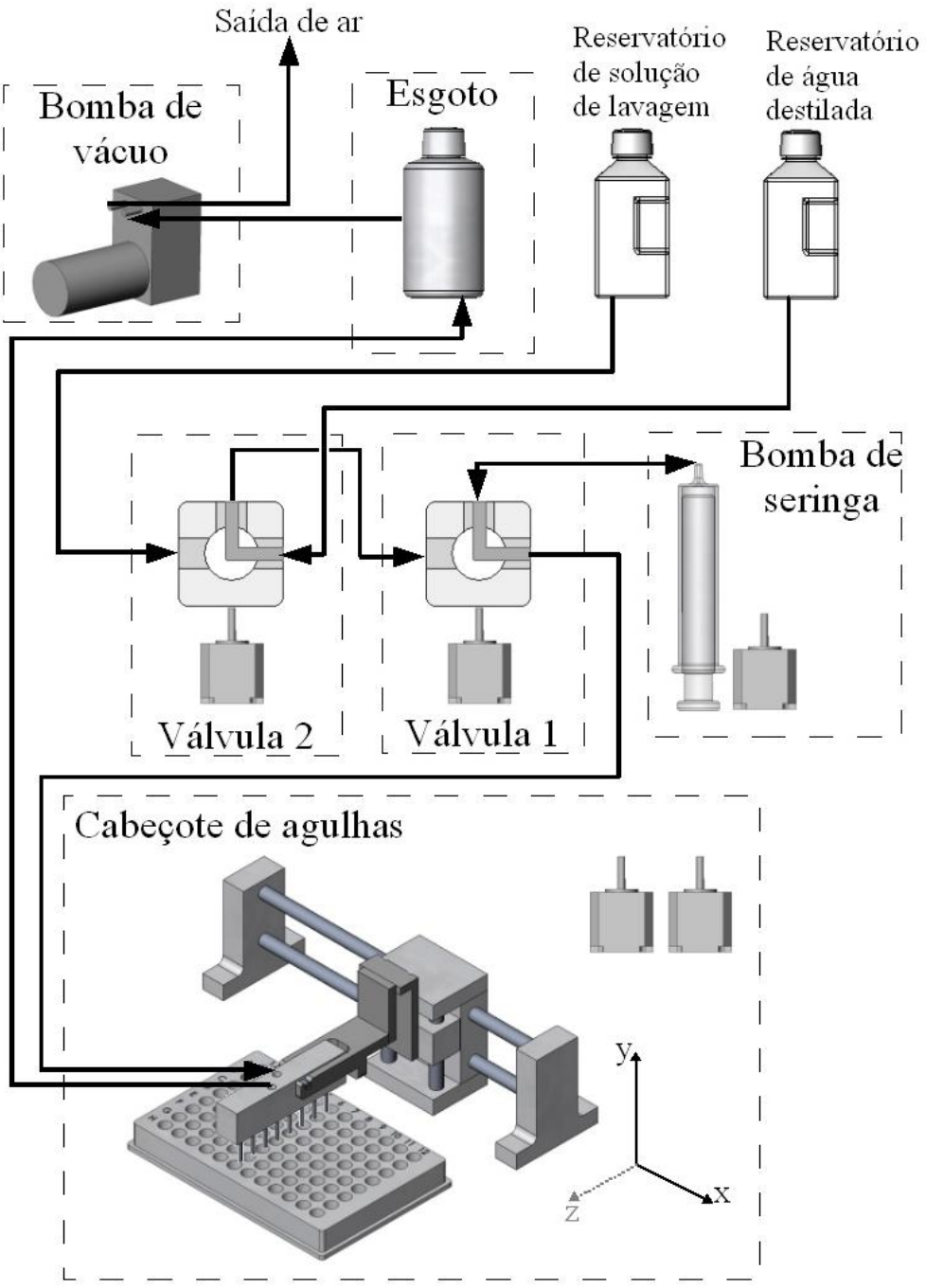


Figura 3.2: Esquema representando os subsistemas da lavadora.

3.1.2 Parâmetros de programação

Diversos parâmetros da lavadora são configurados pelo usuário, influenciando diretamente a operação de lavagem. São eles:

- **Volume:** define o volume de solução de lavagem a ser injetado em cada micropoço. O volume pode, em geral, variar de 50 a 400 μL;
- **Tiras:** define quantas tiras da microplaca serão lavadas. Ex: se o valor desse parâmetro for "n", apenas as tiras de 1 a "n" serão lavadas;
- **Ciclos:** define o número de vezes que as tiras serão lavadas;
- **Poço:** define qual o tipo de poço da microplaca a ser lavada. Fundo em U, fundo em V ou fundo plano;
- **Molho:** define o tempo de molho, que é o tempo, em segundos, durante o qual cada tira ficará cheia com a solução de lavagem antes de ser aspirada novamente;
- **Modo:** define o modo da operação de lavagem: tira ou placa, como explicado a seguir.

Lavagem modo placa

A operação no modo placa é a seguinte:

1. O cabeçote vai para a primeira tira e aspira seu conteúdo;
2. Após subir, injeta a solução de lavagem e anda para a próxima tira, deixando a tira anterior de molho;
3. Em cada tira repete-se o ciclo: desce, aspira, sobe, injeta, vai para a próxima tira.

4. Após lavar as "n" tiras especificadas, volta para a primeira tira e repete o procedimento até encher as "n" tiras o número "v" de ciclos especificado.

5. Após completar os "v" ciclos, o cabeçote volta uma última vez esvaziando as "n" tiras.

Antes que o cabeçote esvazie uma tira, verifica-se se essa tira já ficou de molho o tempo suficiente. Caso contrário, espera-se o tempo necessário antes de aspirá-la.

Lavagem modo tira

1. O cabeçote vai para a primeira tira e aspira seu conteúdo;

2. Sobe, injeta a solução de lavagem e espera o tempo de molho especificado;

3. Desce, aspira a tira novamente e repete o procedimento até lavar a tira o número "v" de ciclos especificado;

4. Após lavar a tira o número "v" de ciclos, aspira o conteúdo da tira e vai para a próxima, até lavar as "n" tiras especificadas.

3.1.3 Requisitos de controle

Para o sistema de controle supervisório desse trabalho foi escolhido arbitrariamente o modo placa. Os objetivos de controle para o sistema são expressos por requisitos de controle. Alguns requisitos dizem respeito ao funcionamento dos mecanismos da lavadora, como intertravamentos e seqüências de operações básicas, como injeção e aspiração. Outros estão relacionados ao seqüenciamento da operação de lavagem no modo placa. Assim, o funcionamento da lavadora, para atender à operação de lavagem no modo placa, deve obedecer aos seguintes requisitos:

1. A bomba de seringa não pode se movimentar enquanto alguma das duas válvulas está se movimentando. Entretanto, as válvulas podem se movimentar simultaneamente;

2. A bomba de seringa só pode injetar se seu volume for suficiente para encher uma tira com o volume especificado pelo usuário. E só irá se encher depois que o volume restante não for suficiente para encher uma tira;

3. Para encher a bomba de seringa, a válvula 1 deve estar posicionada na posição de 90º. Para esvaziá-la, a válvula 1 deve estar em 0º;

4. O cabeçote só pode se movimentar horizontalmente se estiver posicionado verticalmente em uma altura segura, tal que as agulhas estejam para fora dos poços da microplaca;

5. O cabeçote não pode se movimentar verticalmente e horizontalmente ao mesmo tempo. Esse requisito foi adotado para simplificar a movimentação do cabeçote. Entretanto, em algumas situações é, na verdade, desejável que ele se movimente nas duas direções simultaneamente;

6. A bomba de seringa só pode injetar se o cabeçote não estiver se movimentando horizontalmente;

7. Durante a operação de injeção, o cabeçote deve estar posicionado em uma certa altura, de modo que as agulhas de sucção estejam niveladas com a borda dos micropoços, para evitar transbordamento;

8. Para aspirar a tira, o cabeçote deve se posicionar verticalmente de modo que as agulhas de sucção encostem-se no fundo dos micropoços;

9. O cabeçote só pode aspirar uma tira se esta estiver de molho o tempo suficiente como especificado pelo usuário;

10. Depois que o cabeçote desce para aspirar a tira, ele deve esperar um certo tempo, que é chamado tempo de aspiração. Esse tempo é, em alguns casos, configurável pelo usuário;

11. Como a operação de lavagem utiliza apenas solução de lavagem, e não água-destilada, a válvula 2 deve estar sempre posicionada em 90º;

12. A operação de lavagem modelada é o modo placa, e deve acontecer como descrita anteriormente, sendo que o número de tiras e o número de ciclos são definidos pelo usuário.

Os doze requisitos citados anteriormente expressam os objetivos de controle para a lavadora, que serão expressos por meio de especificações genéricas representadas por geradores. Pode-se perceber que o funcionamento da lavadora envolve a interação entre vários subsistemas, sendo alguns mais complexos do que os outros. Em suma, diversos intertravamentos e seqüenciamentos são necessários para controlar a lavadora de modo a executar corretamente a operação de lavagem.

3.2 Representação do sistema

Para representar a lavadora por meio de *automata*, uma decisão importante teve que ser tomada em relação ao nível de abstração a ser considerado. Na prática, os subsistemas essenciais da lavadora são definidos pelos motores, que são a parte exterior ao *firmware* do equipamento. Todas as funções para operar esses motores foram construídas no *firmware*. Assim, qualquer nível de abstração podia ser escolhido. Podia-se assumir, por exemplo, que o cabeçote de agulhas era um subsistema, e que um evento possível era andar para uma determinada tira. Outra possibilidade era considerar cada motor do cabeçote um subsistema, e que um evento do motor horizontal era andar

apenas horizontalmente para a próxima tira. Dessa forma o motor vertical teria que ser modelado separadamente.

Assim, a solução escolhida foi representar a lavadora por meio de cinco subplantas, que representam os cinco principais mecanismos:

- ▷ Bomba de seringa;
- ▷ Motor vertical do cabeçote;
- ▷ Motor horizontal do cabeçote;
- ▷ Válvula 1;
- ▷ Válvula 2.

3.2.1 Definição dos eventos da planta

A estratégia adotada durante a modelagem foi modelar especialmente a parte fisicamente não-controlável do sistema, que é o tempo gasto na movimentação dos mecanismos após receberem qualquer comando. A Tabela 3.1 apresenta os eventos das cinco subplantas e suas descrições. No presente trabalho adotou-se como simbologia para os eventos não-controláveis a letra "δ", e para os eventos controláveis as letras α, β, h e v, diferentemente de Cunha & Cury (2007) e Queiroz & Cury (2002), que utilizam α para eventos controláveis e β para eventos não-controláveis.

Evento	Descrição
$\mathbf{E_c}$	
α_1	Mover a válvula 1 para 90º.
β_1	Mover a válvula 1 para 0º.
α_2	Mover a válvula 2 para 90º.
β_2	Mover a válvula 2 para 0º.
α_3	Encher a bomba de seringa.
β_3	Fazer a bomba de seringa injetar.
h_1	Mover o motor horizontal para a primeira tira.
h_2	Mover o motor horizontal para a próxima tira.
v_1	Mover o motor vertical para a posição de deslocamento.
v_2	Mover o motor vertical para a posição de injeção.
v_3	Mover o motor vertical para a posição de sucção.
$\mathbf{E_u}$	
δ_1	Fim da movimentação da válvula 1.
δ_2	Fim da movimentação da válvula 2.
δ_3	Fim da movimentação da bomba de seringa.
δ_4	Fim da movimentação do motor horizontal.
δ_5	Fim da movimentação do motor vertical.

Tabela 3.1: Descrição dos eventos para a lavadora de microplacas.

3.2.2 *Automata* criados

Para a modelagem das válvulas foram criados quatro estados, que representam as duas posições válidas de 0º e de 90º e as condições movimentando e parada. Os *automata* das válvulas são mostrados nas Figuras 3.3 e 3.4, sendo que os eventos não-controláveis foram representados por setas pontilhadas. As descrições dos estados dos dois *automata* estão na Tabela 3.2.

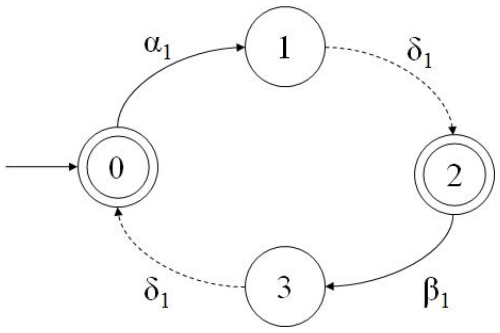


Figura 3.3: *Automaton* G_1 representando a válvula 1.

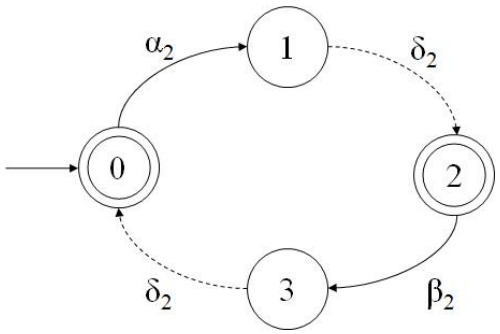


Figura 3.4: *Automaton* G_2 representando a válvula 2.

Estado	$\mathbf{G_1}$	$\mathbf{G_2}$
0	Válvula 1 parada em 0º.	Válvula 2 parada em 0º.
1	Válvula 1 indo de 0º para 90º.	Válvula 2 indo de 0º para 90º.
2	Válvula 1 parada em 90º.	Válvula 2 parada em 90º.
3	Válvula 1 indo de 90º para 0º.	Válvula 2 indo de 90º para 0º.

Tabela 3.2: Descrições dos estados de G_1 e G_2.

Os demais subsistemas, que são a bomba de seringa e os motores vertical e horizontal, possuem movimento linear, apesar de possuírem atuadores rotativos. Assim, possuem um estado contínuo: sua posição linear. Apesar disso, pode-se considerar que os motores horizontal e vertical possuem posições discretas, que são as posições de altura utilizadas e as posições horizontais de

cada tira. No caso da bomba de seringa existe uma complexidade maior devido volume de lavagem, que é escolhido pelo usuário. Assim, de acordo com o volume a ser injetado em cada tira, o deslocamento da bomba de seringa assume diferentes posições discretas.

A estratégia adotada para os três subsistemas foi modelá-los como possuindo estados discretos, abstraindo funções que representassem esses deslocamentos discretos. As Figuras 3.5 a 3.7 ilustram essas três subplantas. A Tabela 3.3 descreve os estados dos *automata* G_3, G_4 e G_5.

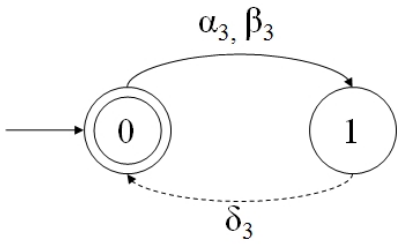


Figura 3.5: *Automaton* G_3 representando a bomba de seringa.

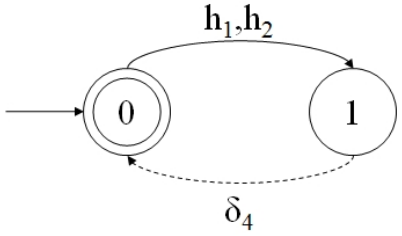


Figura 3.6: *automaton* representando o motor horizontal do cabeçote de agulhas.

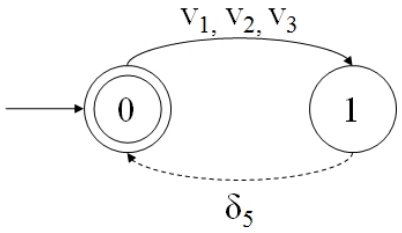


Figura 3.7: *automaton* representando o motor vertical do cabeçote de agulhas.

Estado	**G_3**	**G_4**	**G_5**
0	Bomba de seringa parada.	Motor horizontal do cabeçote parado.	Motor vertical do cabeçote parado.
1	Bomba de seringa em movimento.	Cabeçote em movimento horizontal.	Cabeçote em movimento vertical.

Tabela 3.3: Descrições dos estados de G_3, G_4 e G_5.

3.3 Construção das Especificações

A lavadora de microplacas possui diversos parâmetros de programação, citados anteriormente, que são escolhidos pelo usuário. O número de tiras e de ciclos de lavagem, o volume de solução a ser injetado e os tempos de molho e de aspiração influenciam diretamente na operação da lavadora. Assim, para o projeto do controle supervisório, assumiram-se três opções:

1. Criar especificações que contemplassem apenas uma programação normal fixa. Nessa abordagem seria projetado um único sistema de controle específico para apenas uma configuração da planta;

2. Criar especificações diferentes para cada possibilidade de programação. Essa opção era inviável, pois, apesar no número máximo de tiras ser

12, o número de ciclos e as opções de volume são praticamente ilimitadas, e isso requereria projetar infinitos sistemas de controle para cada possibilidade;

3. Criar especificações flexíveis que pudessem ser utilizadas para todas as possibilidades de programação, de forma que os parâmetros fossem variáveis e os processos relativos a eles fossem abstraídos.

A opção adotada foi a terceira. Adotou-se a estratégia de definir novos eventos que representassem processos internos da lavadora. Esses processos foram implementados por funções construídas no programa da lavadora de forma a lidar com os parâmetros variáveis: o número de tiras a serem lavadas, o número de ciclos, o volume a ser injetado e os tempos de aspiração e de molho. São, na verdade, processos simples, implementados por meio de contadores e timers. Portanto, essas funções não foram modeladas e nem controlados pelo sistema de controle.

3.3.1 Modificação dos subsistemas

Para que o projeto do controle supervisório seja possível, é necessário que $K_{global} \subseteq \mathcal{L}_m(G)$. Ou seja, a especificação global deve ser definida dentro da própria linguagem marcada pela planta. Para que isso ocorra é necessário que os novos eventos sejam incluídos em Σ^*. Assim, as representações da planta foram modificadas, de modo a incluir os novos eventos criados. A Tabela 3.4 mostra os novos eventos e suas descrições.

E_u	
x_0	O volume atual da bomba de seringa passou a ser menor do que o volume necessário.
x_1	O tempo de aspiração já passou.
x_2	O tempo de molho da tira atual já é suficiente, e ela já pode ser aspirada.
n_0	Evento gerado quando o motor horizontal inicia algum movimento e $n < tiras$ e $v < vezes$. Indica que ainda faltam tiras a serem lavadas.
n_1	Evento gerado quando o motor horizontal inicia algum movimento e $n = tiras$ e $v < vezes$. Indica que todas as tiras foram lavadas, mas ainda não o número de vezes especificado.
n_2	Evento gerado quando o motor horizontal inicia algum movimento e $n = tiras$ e $v = vezes$. Indica que todas as tiras foram lavadas o número de vezes especificado.

Tabela 3.4: Eventos não-controláveis adicionais definidos para o projeto do sistema de controle da lavadora.

A subplanta G_3, que representa a bomba de seringa, foi modificada para incluir o evento x_0, Figura 3.8. Esse evento é uma sinalização que ocorre após a operação de injeção, logo antes do evento δ_3, apenas quando o volume da seringa é insuficiente.

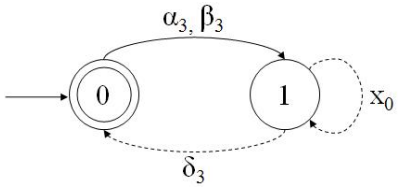


Figura 3.8: *Automaton* G_3 modificado representando a bomba de seringa.

A subplanta G_4, representando o motor horizontal, foi modificada como na Figura 3.9 para incluir os eventos de controle n_0, n_1, n_2 e x_2. Os três primeiros estão relacionados com a contagem de tiras e de ciclos de lavagem, e o último está relacionado ao tempo de molho da tira atual. A subplanta foi

modelada para que os eventos seguissem sua ordem de ocorrência de acordo com o que foi implementado na simulação construída para a lavadora. Na Figura 3.9 percebe-se que o estado 2 pode levar ao estado 0 passando pelos estados 3 ou 4, dependendo da ordem de ocorrências dos eventos x_2 e δ_4. Isso ocorre porque o motor horizontal pode chegar em determinada tira antes ou depois que o tempo de molho dessa tira tenha passado.

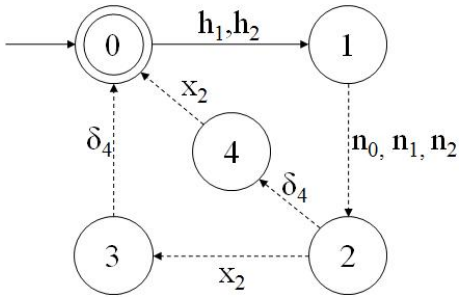


Figura 3.9: *Automaton* G_5 modificado representando o motor horizontal do cabeçote de agulhas.

A subplanta G_5, que representa do motor vertical, foi modificada para incluir o evento x_1, como na Figura 3.10. Esse evento está relacionado com o tempo de aspiração da tira.

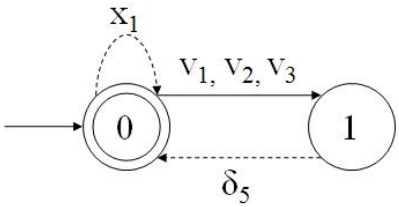


Figura 3.10: *automaton* modificado representando o motor vertical do cabeçote de agulhas.

3.3.2 Especificações genéricas

Utilizando os novos eventos, foram criadas onze especificações genéricas, de acordo com o comportamento desejado para a lavadora. Essas restringem o comportamento da lavadora, de modo que ela execute corretamente a operação de lavagem no modo placa.

Durante a construção das especificações genéricas, duas dificuldades foram encontradas. A primeira foi em relação aos eventos não-controláveis. Quando um evento não-controlável está presente no alfabeto de uma especificação, suas transições devem ser construídas de forma que esse evento não-controlável não seja desabilitado pela especificação. Mais precisamente, o evento não-controlável só pode ser desabilitado pela especificação quando todas as sub-plantas cujos alfabetos contenham esse evento também o desabilitem. Assim, a estrutura de transição da especificação deve seguir as estruturas das sub-plantas, e alguns auto-laços devem ser acrescentados para que isso ocorra.

Se um evento não-controlável for desabilitado pela especificação, sua especificação local será não-controlável em relação a sua respectiva planta local. Uma alternativa então será calcular a máxima linguagem controlável, como na Seção 2.5, e usar o gerador dessa linguagem como supervisor ótimo não-bloqueante. Entretanto, o procedimento usado nesse trabalho foi de observar a estrutura de transição das especificações genéricas para que as especificações locais fossem controláveis.

A segunda dificuldade encontrada foi com relação aos estados marcados das especificações genéricas. Como o supervisor projetado é um supervisor desmarcador, as especificações também designam tarefas cumpridas, que modificam as tarefas cumpridas da planta global. Assim, é possível que um supervisor marque um determinado estado, mas outro supervisor o impeça de alcançar esse estado. Nesse caso, o cálculo da modularidade local indicará

que esses supervisores não são localmente modulares. Isso pode ser resolvido mudando a marcação das especificações, ou alterando suas estruturas de transição. Essa alteração deve ser feita observando quais as especificações cujas plantas locais englobam subsistemas comuns. Essas especificações podem conter seqüências incompatíveis envolvendo os mesmos eventos. Na maioria dos casos, o acréscimo de auto-laços resolve o problema.

As onze especificações genéricas construídas são mostradas nas Figuras 3.11 a 3.21, e fazem referência aos requisitos de controle da Seção 3.1.3.

$\mathbf{E_{g,vl1}}$: Atende ao requisito 1. Especifica o comportamento da válvula 1 em relação à bomba de seringa. Impede a válvula 1 e a bomba de seringa de moverem ao mesmo tempo.

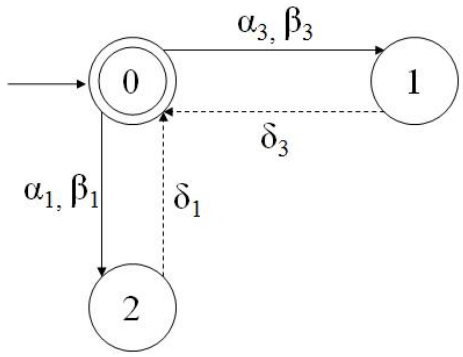


Figura 3.11: Especificação de controle $E_{g,vl1}$.

$\mathbf{E_{g,vl2}}$: Atende ao requisito 1. Especifica o comportamento da válvula 2 em relação à bomba de seringa. Impede a válvula 2 e a bomba de seringa de moverem ao mesmo tempo.

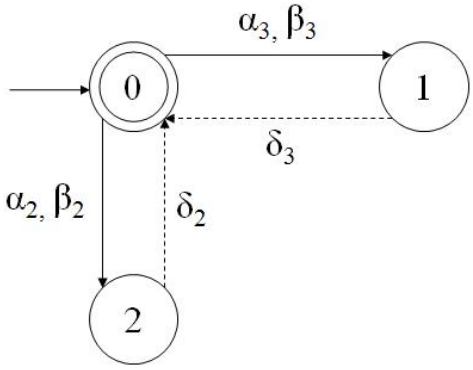


Figura 3.12: Especificação de controle $E_{g,vl2}$.

$\mathbf{E_{g,bmb}}$: Atende aos requisitos 2, 3 e 11. Especifica o comportamento da bomba de seringa em relação ao volume de solução, à válvula 1 e à 2. Faz com que a bomba de seringa só comece a encher se o volume atual não for suficiente para encher uma tira, se a válvula 1 estiver em 90º e se a válvula 2 estiver em 90º. Faz com que ela injete só se houver volume suficiente para encher uma tira e se a válvula 1 estiver em 0º.

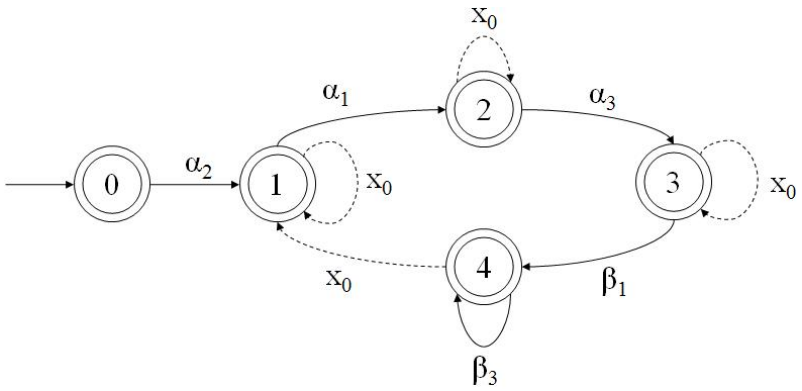


Figura 3.13: Especificação de controle $E_{g,bmb}$.

$\mathbf{E_{g,bv2}}$: Essa especificação está relacionada ao requisito 11 e serve apenas para fazer a válvula 2 voltar para a posição de 0º depois da operação

de lavagem.

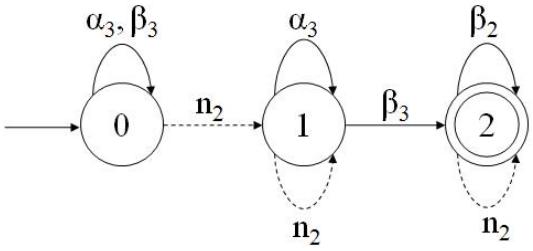


Figura 3.14: Especificação de controle $E_{g,bv2}$.

$E_{g,hv1}$: Atende ao requisito 4. Especifica o comportamento do motor horizontal em relação ao motor vertical. Faz com que o motor horizontal só se mova se o motor vertical estiver na posição de deslocamento, o que ocorre com o evento v_1.

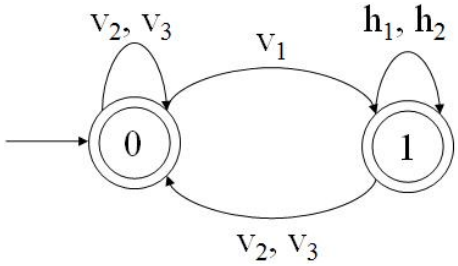


Figura 3.15: Especificação de controle $E_{g,hv1}$.

$E_{g,hv2}$: Atende ao requisito 5. Especifica o comportamento do motor horizontal em relação ao motor vertical. Impede os dois motores de moverem simultaneamente.

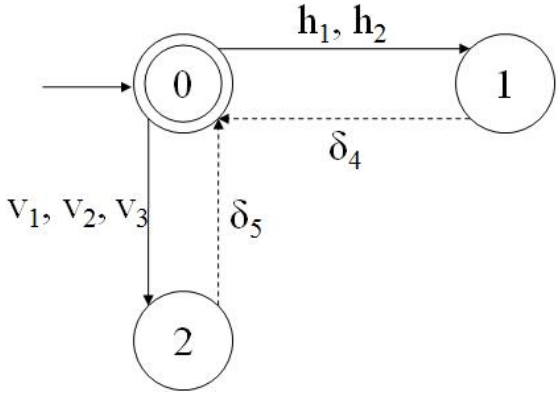


Figura 3.16: Especificação de controle $E_{g,hv2}$.

$E_{g,hbm}$: Atende ao requisito 6. Especifica o comportamento da bomba de seringa em relação ao motor horizontal. Impede a bomba de seringa de injetar se o cabeçote estiver se movimentando horizontalmente.

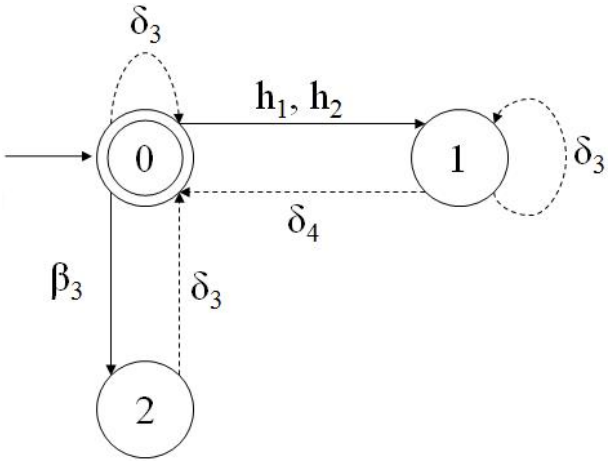


Figura 3.17: Especificação de controle $E_{g,hbm}$.

$E_{g,inj}$: Atende ao requisito 7. Especifica a operação de injeção. Ao mandar o motor vertical para a posição de injeção com o evento v_2, faz com que a operação de injeção se inicie com o evento β_3.

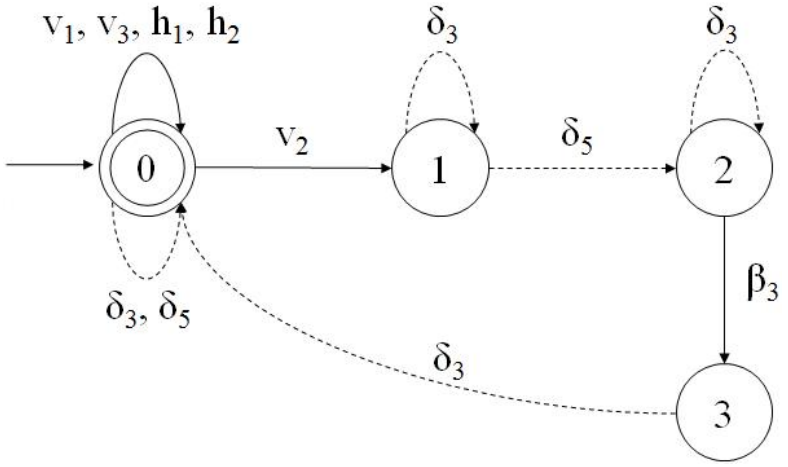


Figura 3.18: Especificação de controle $E_{g,inj}$.

$E_{g,asp}$: Atende aos requisitos 8 e 10. Especifica a operação de aspiração. Faz com que o motor vertical espere o evento x_1, que representa o tempo de aspiração, depois de chegar ao fundo do poço com o evento v_3.

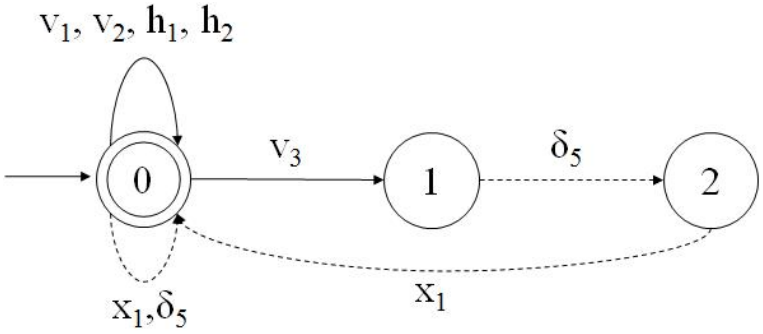


Figura 3.19: Especificação de controle $E_{g,asp}$.

$E_{g,lp1}$: Atende ao requisito 12. Primeira especificação da operação de lavagem no modo placa. Controla o posicionamento do motor horizontal de acordo com a ocorrência dos eventos especiais n_0, n_1 e n_2, fazendo com que ele se desloque de uma tira para a outra de acordo com a lavagem

modo placa.

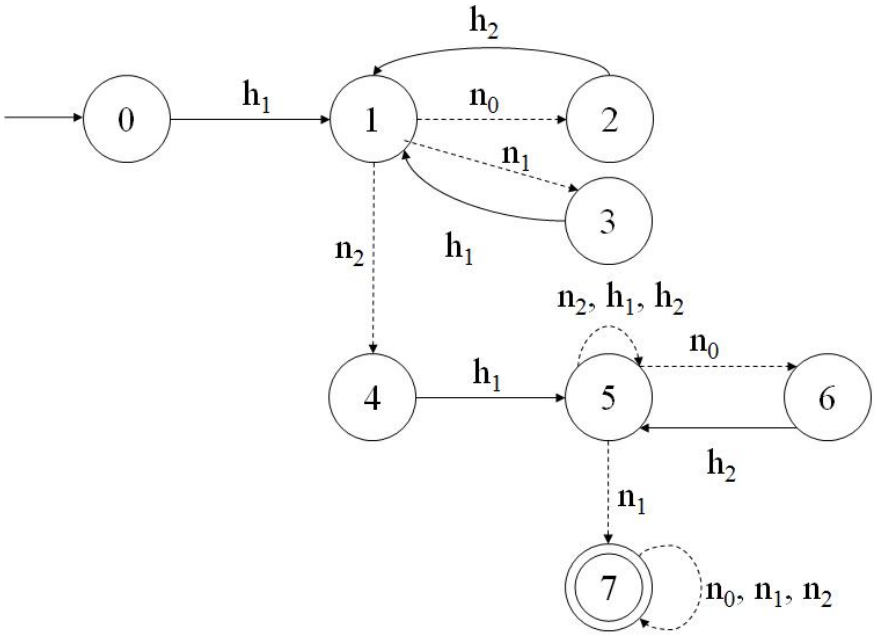


Figura 3.20: Especificação de controle $E_{g,lp1}$.

$E_{g,lp2}$: Atende aos requisitos 9 e 12. Segunda especificação da operação de lavagem no modo placa. Controla o posicionamento dos motores vertical e horizontal de acordo com a ocorrência dos eventos especiais n_0, n_1 e n_2, fazendo com que eles se desloquem de modo sincronizado e de acordo com a lavagem modo placa.

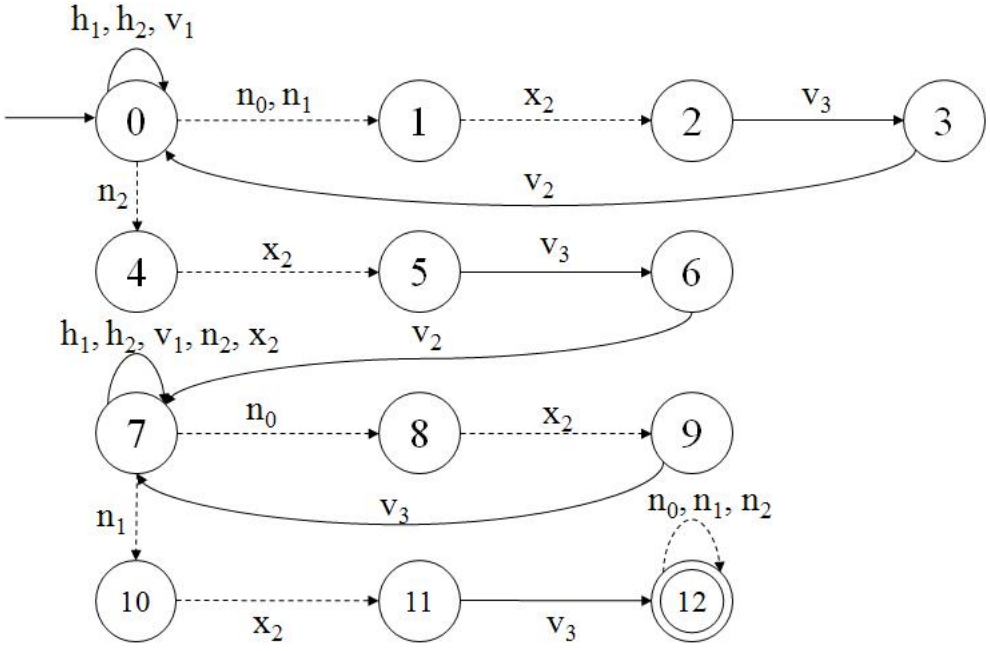


Figura 3.21: Especificação de controle $E_{g,lp2}$.

3.4 Obtenção dos supervisores locais

Construídas as especificações genéricas, as plantas locais $G_{loc,j}$ foram calculadas como na equação 2.17, da seguinte forma: para cada especificação genérica $E_{gen,j}$, a planta local correspondente $G_{loc,j}$ é obtida pela composição paralela de qualquer sub-planta G_i que contiver algum evento presente também na especificação genérica $E_{gen,j}$. Exemplo: a especificação genérica $E_{gen,vl1}$ envolve os eventos α_1, β_1, δ_1, α_3, β_3 e δ_3. As sub-plantas que contém esses eventos são: G_1, que possui os eventos α_1, β_1 e δ_1; e G_3, que possui os eventos α_3, β_3 e δ_3. Portanto, a planta local $G_{loc,vl1}$ é formada pela composição paralela das sub-plantas G_1 e G_3.

A Tabela 3.5 lista as subplantas calculadas, suas composições, número de

estados e de transições.

Planta	Composição	Estados	Transições
$G_{loc,vl1}$	$G_1 \parallel G_3$	8	24
$G_{loc,vl2}$	$G_2 \parallel G_3$	8	24
$G_{loc,bmb}$	$G_1 \parallel G_2 \parallel G_3$	32	128
$G_{loc,bv2}$	$G_2 \parallel G_3 \parallel G_4$	40	192
$G_{loc,hv1}$	$G_4 \parallel G_5$	10	43
$G_{loc,hv2}$	$G_4 \parallel G_5$	10	43
$G_{loc,hbm}$	$G_3 \parallel G_4$	10	38
$G_{loc,inj}$	$G_3 \parallel G_4 \parallel G_5$	20	126
$G_{loc,asp}$	$G_4 \parallel G_5$	10	43
$G_{loc,lp1}$	G_4	5	9
$G_{loc,lp2}$	$G_4 \parallel G_5$	10	43

Tabela 3.5: Composições das plantas locais.

O número de estados de cada planta local é o produto do número de estados das subplantas envolvidas, e isso aconteceu porque as subplantas do sistema têm alfabetos distintos entre si, o que é chamado de um *sistema produto*.

As especificações locais foram obtidas pela composição paralela da especificação genérica com sua planta local correspondente, como na equação 2.18. A Tabela 3.6 apresenta o número de estados e de transições de cada especificação local obtida.

Especificação	Composição	Estados	Transições
$K_{loc,vl1}$	$E_{gen,vl1} \parallel G_{loc,vl1}$	6	12
$K_{loc,vl2}$	$E_{gen,vl1} \parallel G_{loc,vl2}$	6	12
$K_{loc,bmb}$	$E_{gen,bmb} \parallel G_{loc,bmb}$	65	177
$K_{loc,bv2}$	$E_{gen,bv2} \parallel G_{loc,bv2}$	100	420
$K_{loc,hv1}$	$E_{gen,hv1} \parallel G_{loc,hv1}$	20	77
$K_{loc,hv2}$	$E_{gen,hv2} \parallel G_{loc,hv2}$	7	23
$K_{loc,hbm}$	$E_{gen,hbm} \parallel G_{loc,hbm}$	12	40
$K_{loc,inj}$	$E_{gen,inj} \parallel G_{loc,inj}$	45	221
$K_{loc,asp}$	$E_{gen,asp} \parallel G_{loc,asp}$	16	52
$K_{loc,lp1}$	$E_{gen,lp1} \parallel G_{loc,lp1}$	27	37
$K_{loc,lp2}$	$E_{gen,lp2} \parallel G_{loc,lp2}$	60	148

Tabela 3.6: Especificações locais obtidas a partir da composição das plantas locais com as respectivas especificações genéricas.

Para verificar a controlabilidade das especificações locais $K_{loc,j}$, foi utilizado o *XPTCT* (2007). O *TCT* (*Toy Control Theory*) é uma ferramenta computacional para a síntese de controle supervisório para sistemas a eventos discretos, disponibilizado gratuitamente pela *University Of Toronto*, na página do professor W.M. Wonham[1], e documentado em Wonham & Rogers (2007). O *XPTCT* é a versão do *TCT* para o *Windows XP*. Essa ferramenta fornece várias funções relacionadas à síntese de controle supervisório, incluindo verificação da controlabilidade, cálculo da máxima linguagem controlável, composição paralela, etc.

Utilizando-se o *XPTCT*, verificou-se que todas as especificações locais $K_{loc,j}$ obtidas são controláveis em relação às suas respectivas plantas locais $G_{loc,j}$. Assim, os geradores aparados que marcam essas especificações foram tomados como supervisores não-bloqueantes $S_{loc,j}$. Caso alguma especificação $K_{loc,j}$ não fosse controlável em relação à sua planta local, seria calculada a máxima linguagem controlável $SupC(K_{loc,j}, G_{loc,j})$, e o gerador dessa linguagem seria tomado como supervisor não bloqueante ótimo $S_{loc,j}$ (ver Se-

[1] *www.control.utoronto.ca/people/profs/wonham/wonham.html*

ção 2.5). Portanto, faz-se essa distinção entre $K_{loc,j}$, que é a especificação local, controlável ou não, e $S_{loc,j}$, que é o supervisor controlável obtido. Entretanto, nesse trabalho, todos os supervisores $S_{loc,j}$ são iguais às respectivas especificações locais $K_{loc,j}$.

Para verificar a condição de modularidade local, calcula-se a composição paralela de todos os supervisores locais:

$$\begin{aligned} S = S_{loc,vl1} \parallel S_{loc,vl2} \parallel S_{loc,bmb} \parallel S_{loc,hv2} \parallel S_{loc,hv1} \parallel S_{loc,hv2} \parallel \\ S_{loc,hbm} \parallel S_{loc,inj} \parallel S_{loc,asp} \parallel S_{loc,lp1} \parallel S_{loc,lp2}. \end{aligned} \tag{3.1}$$

O supervisor S calculado possui 677 estados e 2030 transições. Como é aparado, conclui-se que a modularidade local é verdadeira e que o controle modular local não implica em perda de desempenho, como explicado anteriormente na Seção 2.5.

A Tabela 3.6 mostra as especificações locais $K_{loc,j}$, que são iguais aos supervisores locais $S_{loc,j}$, e apresenta o número de estados de cada um. Percebe-se que, apesar de ter sido projetado um controle modular, alguns supervisores possuem um número de estados bem elevado, como o $S_{loc,bv2}$, que tem 100 estados. Isso torna sua implementação pouco prática e, portanto, faz sentido tentar reduzir os supervisores obtidos. Um algoritmo formal para minimização de supervisores é apresentado em Vaz & Wonham (1986). Entretanto, como todas as especificações locais obtidas são controláveis, a obtenção de supervisores reduzidos pode ser feita de modo direto: uma vez que $S_{loc,j}/G_{loc,j} = H_{gen,j} \parallel G_{loc,j}$, sendo $H_{gen,j}$ um gerador aparado para a especificação $E_{gen,j}$, o gerador $H_{gen,j}$ pode ser tomado como supervisor reduzido $S_{red,j}$. Assim, os supervisores modulares locais finais são implementados pelos geradores $H_{gen,j} = Trim(E_{gen,j})$.

Caso alguma das especificações locais $K_{loc,j}$ fosse não-controlável em relação à sua respectiva planta local $G_{loc,j}$, e o supervisor fosse obtido pelo cálculo da máxima linguagem controlável, o supervisor reduzido poderia ser calculado pelo algoritmo proposto por Vaz & Wonham (1986). Esse algoritmo foi

implementado na ferramenta computacional *XPTCT*.

Como foi dito na seção 2.3, a ação de controle dos supervisores consiste no conjunto de eventos por eles habilitados em cada estado. Entretanto, na implementação desse controle supervisório, considera-se que a ação de controle é, na verdade, os eventos desabilitados pelo supervisor em determinado estado. Os eventos desabilitados pelo supervisor são os que fazem parte de seu conjunto de eventos e não estão habilitados no estado atual. As Figuras 3.22 e 3.23 mostram como exemplo dois supervisores com as ações de controle (desabilitações) indicadas por setas duplas. Na Figura 3.22, por exemplo, o estado 0 está desabilitando os eventos α_1, α_3 e β_3, pois fazem parte do conjunto de eventos desse supervisor, mas não estão ativos no estado 0.

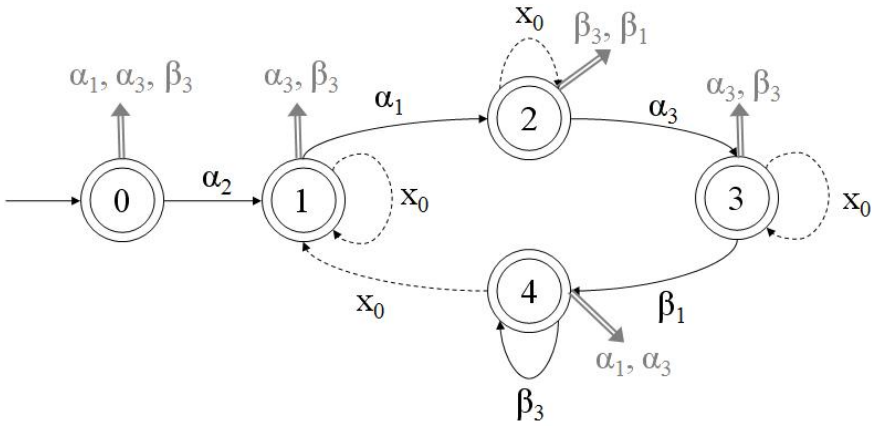


Figura 3.22: Supervisor reduzido $S_{red,bmb}$ com as desabilitações indicadas por setas duplas.

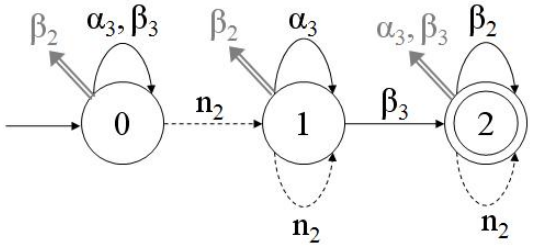


Figura 3.23: Supervisor reduzido $S_{red,bv2}$ com as desabilitações indicadas por setas duplas.

Capítulo 4

Implementação do sistema de controle

A implementação do controle supervisório foi feita utilizando uma simulação da lavadora de microplacas, construída na linguagem *C++*. Para simular a ocorrência dos eventos, foi utilizada uma estrutura de eventos temporizados (Cassandras & Lafortune 1999). Cada evento possui um tempo de vida associado, que indica quando este ocorrerá, e é obtido simulando o comportamento dos sistemas reais da lavadora. Um calendário de eventos é utilizado para escalonar os eventos, indicando qual o próximo evento a ocorrer. O resultado da simulação é um arquivo com a seqüência de eventos gerados, contendo seu tempo de ocorrência, suas descrições e os estados de cada subsistema e cada supervisor.

4.1 *Automata* temporizados

No projeto do sistema de controle não se considerou a temporização dos eventos. Entretanto, para simular o funcionamento do sistema em malha fechada, fez-se necessária a temporização dos eventos do sistema. Assim, a estrutura utilizada na simulação foi baseada na estrutura de eventos temporizados pro-

posta por Cassandras & Lafortune (1999), que é resumida a seguir.

Dado um *automaton* $G = (Q, \Sigma, f, \Gamma, q_0, Q_m)$, define-se uma **estrutura de relógio**, dada por:

$$\begin{aligned} &\mathbf{V} = \{\mathbf{v_i} : i \in \Sigma\}, \\ &\text{sendo} \\ &\mathbf{v_i} = \{v_{i,1}, v_{i,2}, v_{i,3}, \ldots\}, \quad i \in \Sigma, \quad v_{i,k} \in R^+, \quad k = 1, 2, 3, \ldots \end{aligned} \tag{4.1}$$

Na definição anterior, $\mathbf{v_i}$ é uma **seqüência de relógio**, ou seqüência de **tempos de vida**. O elemento $v_{i,k}$ determina o intervalo entre a k-ésima e a $(k-1)$-ésima ocorrência do evento i, a menos que ele seja desabilitado.

Define-se o **valor de relógio** como o tempo restante para que o tempo de vida do evento se passe, definido apenas para os eventos habilitados em determinado estado. O próximo evento a ocorrer é sempre aquele que tiver o menor valor de relógio.

O procedimento para temporizar um *automaton* via sua estrutura de relógio é sintetizado nas seguintes regras:

1. O próximo evento a ocorrer é aquele que apresentar menor valor de relógio dentre os eventos habilitados no estado atual;

2. Um evento "α" é dito ativado quando:

   a) α foi o último evento a ocorrer e permaneceu habilitado após a transição de estado;

   b) um evento $\beta \neq \alpha$ foi o último a ocorrer, e α, que não estava habilitado, passou a estar após a ocorrência de β;

3. Um evento α é desativado quando qualquer evento ocorre causando uma transição para um estado no qual α não está habilitado;

4. A cada ativação, um evento toma como valor de relógio o próximo elemento de sua seqüência de relógio.

A Figura 4.1 mostra um exemplo do funcionamento de um *automaton* temporizado possuindo dois eventos, α e β, sendo que o evento α não está habilitado nos estados x_1 e x_3. Percebe-se que no estado x_0, os dois eventos estão habilitados, mas o primeiro evento a ocorrer é o α, pois tem o menor tempo de vida. Ao ir para o estado x_1, o valor de relógio de β continua contando, e ele é o próximo a ocorrer, sendo que α está desabilitado. Ao ir para o estado x_2, os dois eventos são ativados, e seus tempos de vida assumem os segundos valores, $v_{\alpha,2}$ e $v_{\beta,2}$. Porém, o menor tempo de vida é o de β, que ocorre, e leva o *automaton* para o estado x_3, onde o evento α é desabilitado e β é ativado. Por fim, β ocorre novamente em x_3 e os dois eventos são ativados no estado x_4.

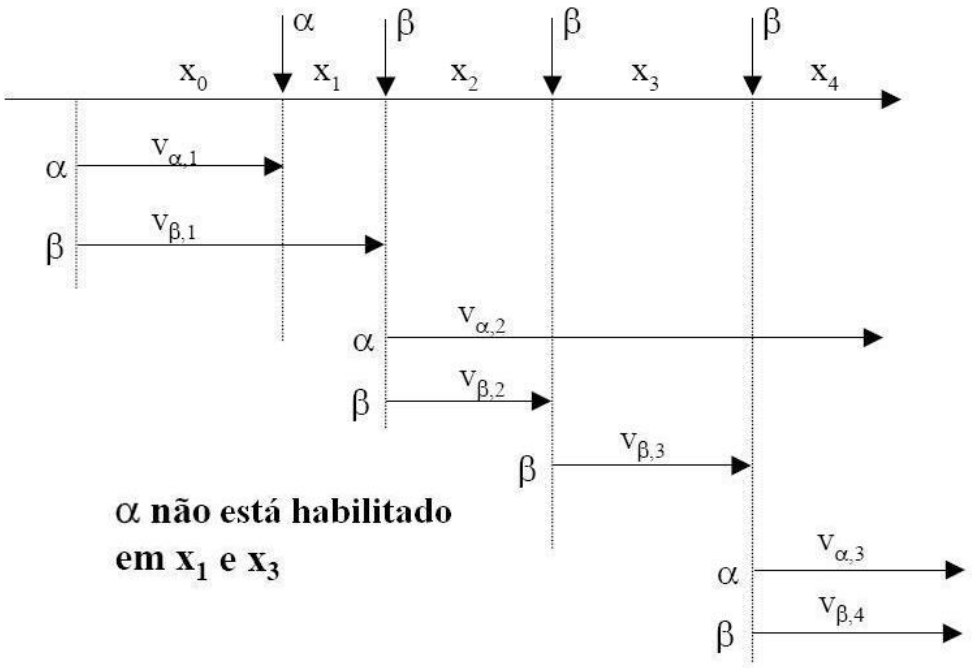


Figura 4.1: Exemplo do funcionamento de um *automaton* temporizado.

Para simular o funcionamento de um SED temporizado, faz-se necessária a utilização de uma estrutura apropriada, de escalonamento de eventos. A

Figura 4.2 mostra tal estrutura, sugerida por Cassandras & Lafortune (1999). Um calendário de eventos é utilizado para escalonar os eventos. Utilizando-se os tempos de vida, que são gerados pela estrutura de relógio, calculam-se os tempos de ocorrência dos eventos e estes são colocados no calendário em ordem de ocorrência. A primeira linha do calendário informa o próximo evento a ocorrer. Após a ocorrência de algum evento, realiza-se a evolução de estado do *automaton*, retiram-se do calendário os eventos não-habilitados no novo estado e acrescentam-se os novos eventos habilitados.

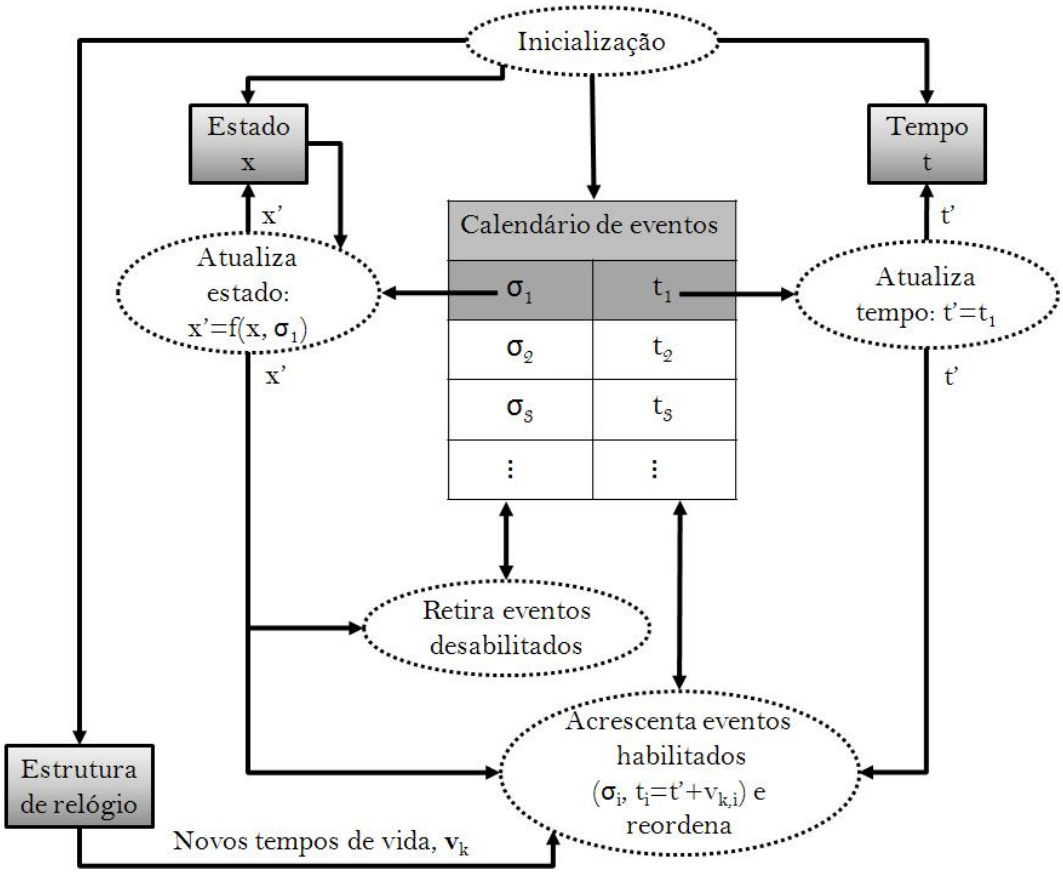


Figura 4.2: Estrutura de escalonamento de eventos.

A estrutura utilizada para a simulação do sistema baseou-se na estru-

tura de eventos temporizados explicada anteriormente, sendo adaptada para incluir o sistema de controle.

4.2 Estrutura de implementação

A Figura 4.3 mostra a estrutura de implementação do controle supervisório modular local sugerida por Queiroz (2004). O sistema de controle é composto por três camadas:

Supervisores modulares: são implementados por meio de *automata*. Recebem os eventos sinalizados pelo sistema produto, e, depois de atualizar os estados dos supervisores, gera um conjunto de desabilitações.

Sistema produto: é formado pelas subplantas, implementadas de forma concorrente por meio de *automata*. Recebe os eventos sinalizados pelas seqüências operacionais, atualiza o estado dos geradores das subplantas, e gera um conjunto de desabilitações de acordo com o estado das subplantas.

Seqüências operacionais: esse nível representa uma interface entre o sistema de controle e a planta. É responsável por receber os comandos do sistema produto, que são eventos habilitados, e traduzi-los em seqüências operacionais relacionadas com a interface de comando da planta. Também recebe as sinalizações vindas da planta e as traduz em eventos que serão sinalizados para o sistema produto.

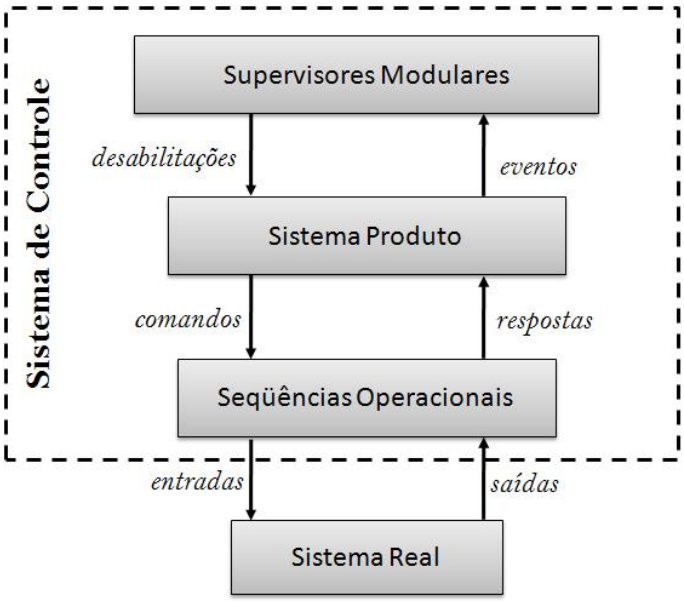


Figura 4.3: Estrutura para implementação do controle supervisório modular local.

A estrutura de implementação utilizada nesse trabalho foi baseada na estrutura da Figura 4.3 e na estrutura de eventos temporizados da Figura 4.2. A Figura 4.4 mostra a estrutura final utilizada, que é dividida em: *supervisores modulares*, *sistema produto*, *simulação da lavadora* e *calendário de eventos*. A seguir cada bloco da estrutura é explicado separadamente.

4.2.1 Supervisores modulares

Os supervisores modulares locais reduzidos $S_{red,j}$ são implementados nesse nível por meio de *automata*. Para isso uma classe *automaton* foi criada contendo funções para adicionar transições, sinalizar eventos, retornar as desabilitações, etc.

Os supervisores recebem uma entrada vinda do calendário de eventos, que são os eventos que estão sendo gerados pelo sistema. Depois de atualizar os

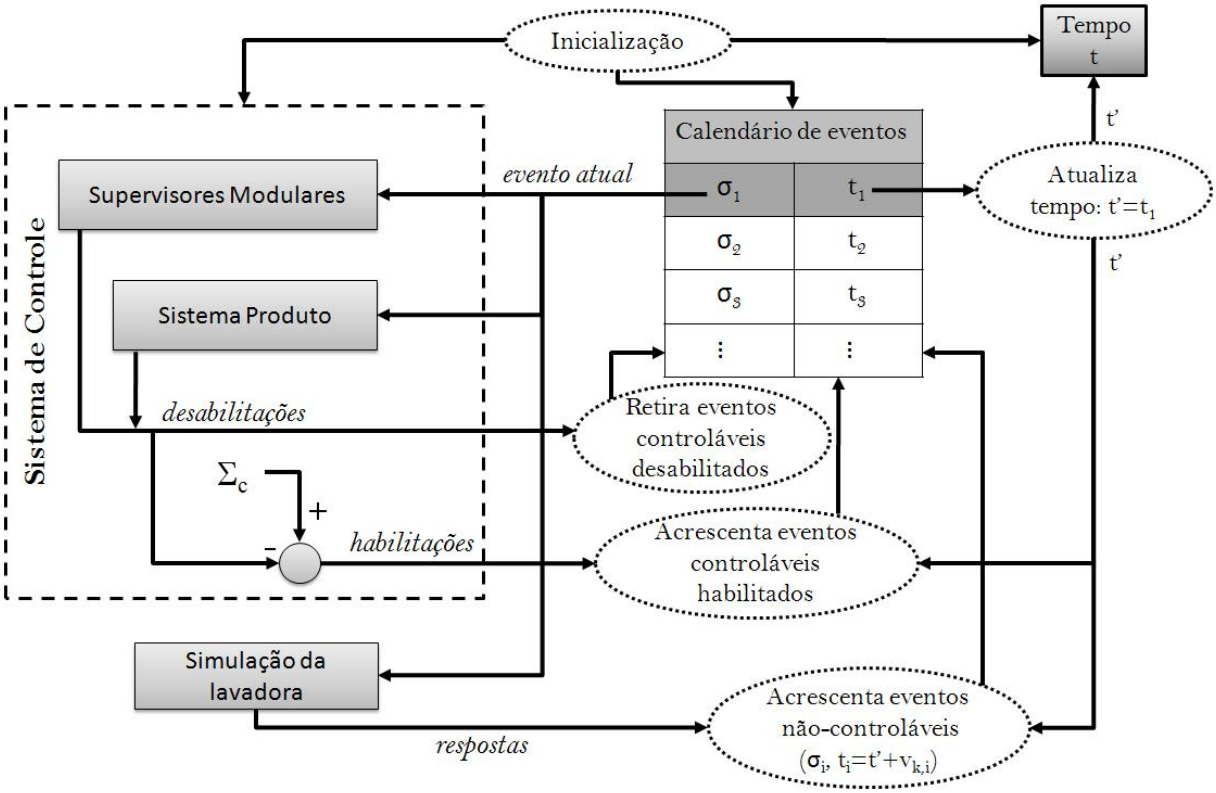


Figura 4.4: Estrutura utilizada para simulação da estrutura de controle.

estados dos supervisores, um conjunto de desabilitações é gerado, formado pela união das desabilitações de todos os supervisores modulares.

4.2.2 Sistema produto

O sistema produto é formado pelas subplantas, implementadas por meio de *automata* utilizando a mesma classe *automaton* citada anteriormente. Esse nível é necessário, pois os supervisores modulares reduzidos não possuem em si mesmos todas as transições existentes nas subplantas. É necessário que o sistema produto atue juntamente com os supervisores de modo que os eventos controláveis só sejam executados se não estiverem desabilitados pelos supervisores e estiverem ativos no estado atual do sistema produto.

O sistema produto recebe os eventos sinalizados pelo calendário de eventos, atualiza o estado dos geradores, e gera um conjunto de desabilitações de acordo com o estado das subplantas.

4.2.3 Simulação da lavadora

A simulação da lavadora foi construída baseada no seu funcionamento real. A função desse nível é receber os comandos vindos do sistema de controle, representados por eventos controláveis, e gerar as respostas, representadas pelos eventos não-controláveis.

Cada vez que um evento controlável é sinalizado à simulação, uma função responsável por simular o deslocamento dos motores é chamada. A posição de destino do motor depende de qual evento ocorreu e qual sua posição atual. As respostas da simulação são calculadas imediatamente, gerando eventos não-controláveis. Cada evento possui um tempo de vida, que indica quando ele irá ocorrer e é obtido pelo cálculo dos tempos de deslocamento dos motores. Para calcular esses tempos, utiliza-se a curva de velocidade real dos motores da lavadora, mostrada nas Figuras 4.5 e 4.6.

Para todos os deslocamentos, os motores partem de uma velocidade mínima, acelerando com aceleração constante até uma velocidade máxima, onde permanecem até começarem a desacelerar, de modo que ao chegarem à posição desejada, a velocidade seja a mínima novamente. O fato dos motores partirem e voltarem para uma velocidade mínima, e não para velocidade nula, foi necessário pelo fato dos motores serem de passo. Isso reduz o ruído gerado pelo motor nos momentos de aceleração e desaceleração.

Para o caso em que o motor consegue atingir a velocidade máxima, o tempo de deslocamento é calculado da seguinte maneira:

$$t_d = \frac{x_d - V_{min} \cdot t_a}{V_{max}} + t_a, \tag{4.2}$$

sendo:
t_d: tempo de deslocamento,
x_d: posição final desejada,
V_{min}: velocidade mínima,
V_{max}: velocidade máxima,
t_{da}: tempo da desaceleração,
t_a: tempo de aceleração.

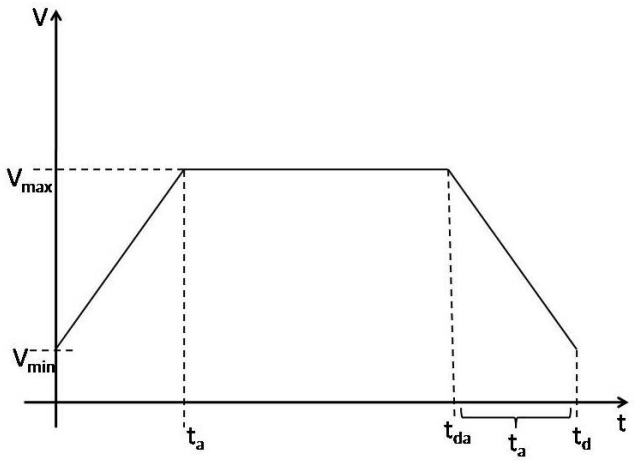


Figura 4.5: Curva normal de velocidade dos motores da lavadora.

Para o caso em que a distância a ser percorrida é pequena, de tal forma que o motor começa a desacelerar antes de alcançar a velocidade máxima, o cálculo é feito da seguinte maneira:

$$t_d = 2 \cdot \frac{x_d}{V_{min} + V_{max}}. \tag{4.3}$$

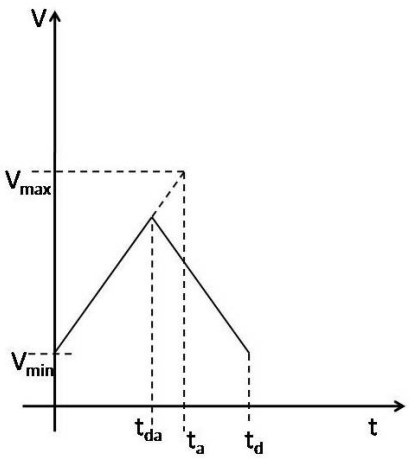


Figura 4.6: Curva de velocidade quando a velocidade máxima não é atingida.

A Tabela 4.1 apresenta os valores de velocidade e aceleração para cada motor da lavadora. A distância considerada entre as tiras foi de 9.06 mm; a primeira tira da lavadora foi posicionada em 128.1 mm, e a última tira em 28.44 mm; a posição do motor vertical na posição de sucção é 14.2 mm, na posição de injeção é 3.3 mm e a posição de deslocamento é 2.0 mm.

Motor	**Válvula 1**	**Válvula 2**	**Bomba de seringa**	**Cabeçote horizontal**	**Cabeçote vertical**
$\mathbf{V_{min}}$	150 °/s	150 °/s	1.7 mL/s	5 cm/s	5 mm/s
$\mathbf{V_{max}}$	500 °/s	500 °/s	3.2 mL/s	14 cm/s	13 mm/s
$\mathbf{t_a}$	200 ms	200 ms	170 ms	150 ms	200 ms

Tabela 4.1: Valores de velocidade e aceleração para cada motor da lavadora.

4.2.4 Calendário de eventos

O calendário de eventos é uma matriz que contém os eventos que irão acontecer futuramente no sistema e seus tempos de ocorrência. Ele informa ao sistema de controle qual o próximo evento a ocorrer, seja ele controlável ou

não. Também recebe eventos não-controláveis sinalizados nas respostas da simulação e eventos controláveis gerados nos comandos do sistema de controle. Sempre que um evento é acrescentado ao calendário, seu tempo de ocorrência é obtido somando-se o tempo global atual com o tempo de vida do evento, e sua posição no calendário é definida de acordo com seu tempo de ocorrência.

4.3 Estruturas de dados criadas

As seguintes estruturas de dados foram criadas como classes em *C++*:

Classe *Myvector*:

Essa classe herda a classe *vector<string>*, e é utilizada para auxiliar o tratamento dos eventos. Inclui operações para interseção, união e subtração de conjuntos de eventos.

```
1  class Myvector: public vector<string> {
2  public:
3    Myvector(void){};                          //construtor
4    void imprime(void);                        //imprime o vetor de
         strings
5    Myvector operator&(const Myvector & a);    //operador de
       çãointerseo
6    Myvector operator+(const Myvector & a);    //operador ãunio
7    Myvector operator-(const Myvector & a);    //operador de
       çãsubtrao
8    int achar(string a);                       //localiza um evento
9    int colocar(string a);                     //acrescenta um
       evento
10 };
```

Classe *Trans*:

Essa classe representa uma transição. É utilizada nas listas encadeadas que indicam as transições para cada estado na classe *Automaton*. Possui um

estado de destino, um apontador para a próxima transição e o nome do evento.

```
1  class Trans {
2    int dest;                  //estado de destino
3    string evt;                //evento
4    int posevt;                //indice do evento no conjunto de
         eventos do automaton
5    Trans* prox;               //apontador para a óprxima transicao
6    friend class Automaton;
7  };
```

Classe *Est*:

Estrutura de dados que representa um estado e é utilizada na matriz de transições da classe *Automaton*. Indica o número do estado e possui apontadores para o próximo estado e para a lista de transições.

```
1  class Est {
2    int num;                     //numero do estado
3    Est * prox;                  //apontador para o proximo estado
4    Trans * lista;               //apontador para a lista de
         transicoes
5    friend class Automaton;
6  };
```

Classe *Automaton*:

Principal classe utilizada no programa, usada nos supervisores modulares e no sistema produto. Possui funções para criar novas transições, sinalizar a ocorrência de algum evento, obter eventos habilitados e desabilitados no estado atual. A implementação da função de transição foi feita por meio de uma matriz de transição esparsa, que indica, para cada estado, as transições presentes. Os elementos dessa matriz são das classes *Est* e *Trans*. A Figura 4.7 mostra como exemplo a matriz esparsa do *automaton* $E_{g,hbm}$ da Figura 3.17. Entretanto, a matriz de transição poderia ser facilmente repre-

sentada por uma matriz comum, como mostra a Figura 4.8. Nessa matriz os eventos não-habilitados são representados pelo número −1.

```
1  class Automaton {
2    int Q;            //conjunto de estados
3    Myvector E;       //conjunto de eventos
4    Est * f;          //matriz de transicao, usando lista encadeada
5    int q0;           //estado inicial
6    int qx;           //estado atual
7    vector<int> Qm;   //conjunto dos estados marcados
8    string nome;      //nome do automaton
9
10 public:
11   static Myvector E_c;         //conjunto de eventos controlaveis
12   Automaton(int nestados, string name="", int estadoi=0);//
         construtor
13   void marcar(int estado);     //marca um estado
14   void trans(int est_ori, int est_des, string evt);
15                                //acrescenta uma transicao ao
                                      automaton
16   Myvector habs(void);         //retorna os eventos habilitados
         no estado atual
17   Myvector desabs(void);       //retorna os eventos desabilitados
          no estado atual
18   bool evt(string);            //sinaliza um evento no automaton.
19   int Qx(void) {return qx;};   //retorna o estado atual
20 };
```

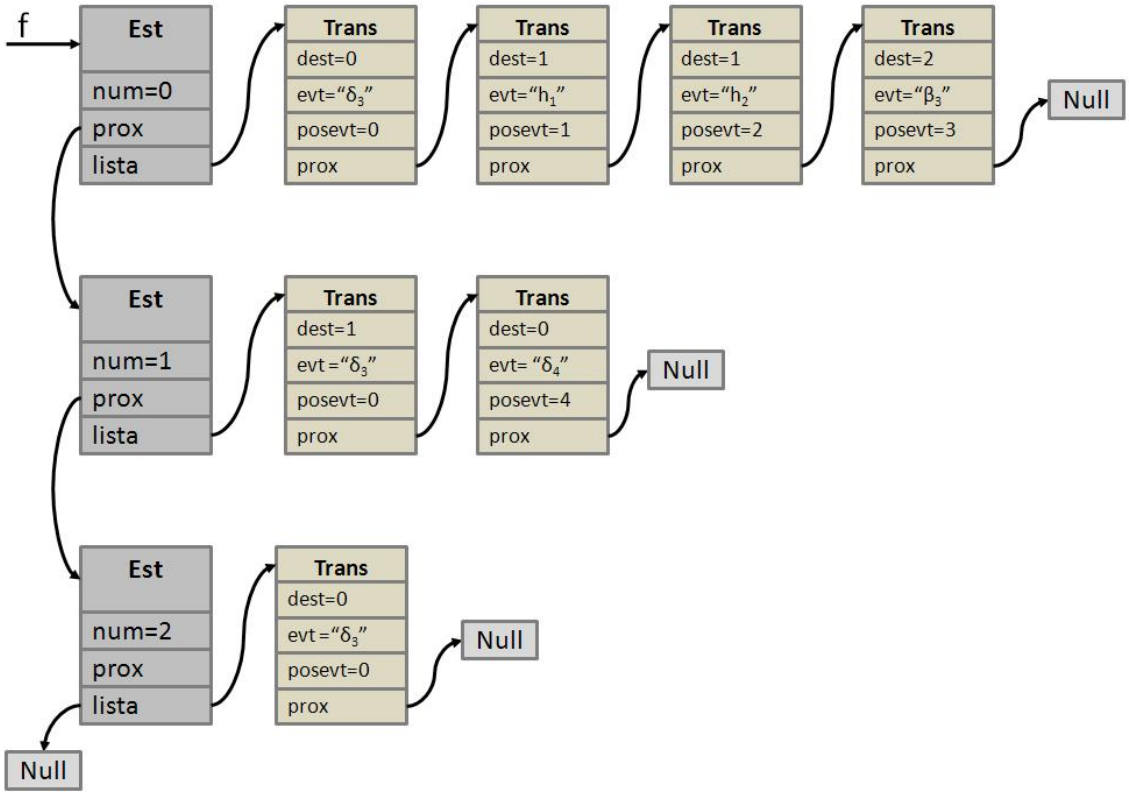


Figura 4.7: Representação da matriz de transição do *automaton* da Figura 3.17 por meio de uma matriz esparsa.

	δ_3	$\mathbf{h_1}$	$\mathbf{h_2}$	β_3	δ_4
0	0	1	1	3	-1
1	1	-1	-1	-1	0
2	0	-1	-1	-1	-1

Figura 4.8: Representação da matriz de transição do *automaton* da Figura 3.17 por meio de uma matriz simples.

Classe *Motor*:

Essa classe faz a simulação do funcionamento dos motores, calculando os tempos de deslocamento utilizando as variáveis de velocidade, aceleração e tempo.

```
1  class Motor {
2    string nome;                    //informa o nome do motor
3    unsigned int velocidade_max;    //velocidade maxima
4    unsigned int velocidade_min;    //velocidade minima
5    unsigned int tempo_a;           //tempo de aceleracao/
         desaceleracao
6    long dist_a;
7    long  posicao_atual;            //valor da posicao do motor
8    unsigned int posicao_desejada;//valor da posicao desejada
9    long posicao_max;               //posicao maxima do motor
10   int dist_desacelera;            //distancia de desaceleraracao
11
12 public:
13   Motor(unsigned int v_max, unsigned int v_min, unsigned int t_a
         , long p_max, string); //construtor
14   double mover(long  posicao);//move o motor para determinada
         posicao
15   void chegou(void);           //seta a posicao atual igual a
         desejada
16   friend class Lavadora;
17 };
```

Classe *Lavadora*:

Classe que simula o funcionamento da lavadora. Essa classe recebe as sinalizações de eventos vindas do calendário e chama funções dos motores para movê-los para as posições correspondentes aos eventos. A função "mover" dos motores retorna os tempos de deslocamento. A classe lavadora armazena esses tempos e retorna para o programa um vetor com todos os eventos não-controláveis gerados e seus respectivos tempos de vida.

```
1  class Lavadora {
2    Myvector evtu;                    //buffer que armazena os
         ultimos eventos nao-controlaveis ocorridos na planta
```

```
3    vector<double> tevtu;                //buffer que armazena os
        tempos dos ultimos eventos nao-controlaveis ocorridos na
        planta
4    Motor Val1, Val2, Vert, Hori, Bmbs; //motores
5    const long pos_inf;                  //constantes relacionadas
        a operacao da lavadora
6    const long pos_sup;
7    const long pos_supnivmax;
8    const long pos_tira0;
9    const long dist_tiras;               //distancia entre as
        tiras
10   double tempo, tempo_tira[12];        //variaveis utilizadas
        para controllar o tempo de molho de cada tira
11
12 public:
13   int tira_atual;
14   int ciclos_completos;
15   Lavadora(void);                      //construtor
16   vector<eventoT> respostas(void);//retorna os eventos nao
        controlaveis habilitados pela lavadora
17   void comando(string a, double); //sinaliza algum evento,
        controlavel ou nao, para a lavadora
18   void geraevtesp(double);             //funcao que gera os eventos
        x2, n0, n1 e n2.
19   int volume;                          //variaveis de configuracao da
        lavadora
20   int ntiras;
21   int modo;
22   int nvezes;
23   int poco;
24   int soak;
25 };
```

4.4 Funcionamento do sistema de controle

O loop principal do programa construído é mostrado e descrito a seguir:

Código

```
1 while (1){
2   //verifica o calendario
```

```
3    if (calendario.size()>0){
4      evt = calendario[0];
5      calendario.erase(calendario.begin());
6      EvtSC(evt.nome);              //seta o evento no sistema de
           controle
7      lav.comando(evt.nome, tempo);//seta o evento na simulacao da
            planta
8      tempo = evt.tempo;            //seta o tempo global como
           sendo o do evento
9      SalvaArquivo();
10   }
11
12   //simulacao da planta
13   resp = lav.respostas();
14   for (j=0;j<resp.size();j++) {  //coloca os eventos nao-
         controlaveis sinalizados pela lavadora no calendario
15     AcrescentaCal(resp[j]);
16   }
17
18   //acha os eventos habilitados pelo sistema de controle
19   desabs = EvtsDesabs() & Ec;     //retira dos eventos
         desabilitados os eventos nao-controlaveis
20   habs = Ec - desabs;
21   for (j=0;j<desabs.size();++j)
22     RetiraCal(desabs[j]);         //retira os eventos
           desabilitados do calendario
23
24   //coloca os eventos habilitados no calendario
25   for (j=0;j<habs.size();j++) {
26     evt.nome = habs[j];
27     evt.tempo = Myrand();
28     AcrescentaCal(evt);
29   }
30 }
```

Descrição

Linhas 3, 4 e 5: Verifica a primeira linha do calendário, que informa o próximo evento a ocorrer, seja ele controlável ou não. Essa linha é removida do calendário e o evento sinalizado aos sistemas;

Linha 6: Sinaliza a ocorrência do evento para o sistema de controle, incluindo o sistema produto e os supervisores modulares locais reduzidos.

Os estados dos supervisores e do sistema produto são atualizados de acordo com esse evento;

Linha 7: Sinaliza a ocorrência do evento para a simulação da lavadora. A simulação calcula os tempos de deslocamento dos motores e gera as respostas que são eventos não-controláveis;

Linha 8 e 9: Atualiza o tempo global com o tempo do evento que ocorreu e registra a ocorrência do evento em um arquivo;

Linhas 13 a 16: Verifica as respostas da planta e acrescenta os eventos não-controláveis no calendário de eventos, na posição correta. Os tempos de vida dos eventos são obtidos de acordo com a curva de velocidade dos motores, e a eles é somada uma parcela aleatória. O tempo de ocorrência é obtido somando-se o tempo de vida ao tempo global;

Linhas 18 a 22: Retira do calendário os eventos controláveis não-habilitados pelo sistema de controle;

Linhas 24 a 29: Acrescenta ao calendário os eventos controláveis habilitados pelo sistema de controle que ainda não estejam presentes. Considera-se que o tempo de vida dos eventos controláveis é um tempo aleatório muito pequeno, praticamente instantâneo, uma vez que eles são gerados pelo próprio sistema de controle exatamente depois que uma nova transição ocorre.

Dessa forma, o programa criado simula o funcionamento da malha fechada para o controle modular local projetado para a lavadora de microplacas. À medida que a simulação acontece, os eventos ocorridos são armazenados seqüencialmente em um arquivo texto, permitindo a verificação posterior do funcionamento do sistema.

4.5 Resultados obtidos

O resultado da simulação implementada é um arquivo contendo a lista dos eventos ocorridos, com o tempo de ocorrência, as evoluções dos estados de todos os *automata* do sistema e os valores das variáveis internas da simulação da lavadora. Alguns resultados obtidos, gerados em quatro simulações realizadas com diferentes parâmetros de programação, são mostrados a seguir. Alguns comentários foram feitos em relação a cada simulação, e os eventos relacionados estão sublinhados com objetivo de facilitar sua localização.

Simulação nº1.
Volume de lavagem: $200\mu L$;
Número de tiras: 3;
Número de ciclos: 1;
Tempo de molho: 30s;

i	**Tempo (m:s.ms)**	**Evento**	**Ciclo**	**Tira**	**Descrição**
0	00:00.010	v_1	0	1	Mover o motor vertical para a posição de deslocamento.
1	00:00.012	α_2	0	1	Mover a válvula 2 para 90º.
2	00:00.024	α_1	0	1	Mover a válvula 1 para 90º.
3	00:00.251	δ_5	0	1	Fim da movimentação do motor vertical.
4	00:00.262	h_1	0	1	Mover o motor horizontal para a primeira tira.
5	00:00.280	n_0	0	1	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
6	00:00.281	x_3	0	1	O tempo de molho da tira atual já é suficiente.
7	00:00.303	δ_2	0	1	Fim da movimentação da válvula 2.
8	00:00.317	δ_1	0	1	Fim da movimentação da válvula 1.
9	00:00.335	α_3	0	1	Encher a bomba de seringa.
10	00:01.339	δ_4	0	1	Fim da movimentação do motor horizontal.
11	00:01.354	v_3	0	1	Mover o motor vertical para a posição de sucção.
12	00:02.433	δ_5	0	1	Fim da movimentação do motor vertical.
13	00:02.933	x_1	0	1	O tempo de aspiração já passou.
14	00:02.944	v_2	0	1	Mover o motor vertical para a posição de injeção.
15	00:03.557	δ_3	0	1	Fim da movimentação da bomba de seringa.
16	00:03.569	β_1	0	1	Mover a válvula 1 para 0º.
17	00:03.861	δ_1	0	1	Fim da movimentação da válvula 1.
18	00:03.916	δ_5	0	1	Fim da movimentação do motor vertical.
19	00:03.929	β_3	0	1	Fazer a bomba de seringa injetar.

20	00:04.527	δ_3	0	1	Fim da movimentação da bomba de seringa.
21	00:04.538	v_1	0	1	Mover o motor vertical para a posição de deslocamento.
22	00:04.699	δ_5	0	1	Fim da movimentação do motor vertical.
23	00:04.717	h_2	0	2	Mover o motor horizontal para a próxima tira.
24	00:04.733	n_0	0	2	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
25	00:04.734	x_3	0	2	O tempo de molho da tira atual já é suficiente.
26	00:04.857	δ_4	0	2	Fim da movimentação do motor horizontal.
27	00:04.871	v_3	0	2	Mover o motor vertical para a posição de sucção.
28	00:05.943	δ_5	0	2	Fim da movimentação do motor vertical.
29	00:06.443	x_1	0	2	O tempo de aspiração já passou.
30	00:06.458	v_2	0	2	Mover o motor vertical para a posição de injeção.
31	00:07.429	δ_5	0	2	Fim da movimentação do motor vertical.
32	00:07.448	β_3	0	2	Fazer a bomba de seringa injetar.
33	00:08.040	δ_3	0	2	Fim da movimentação da bomba de seringa.
34	00:08.053	v_1	0	2	Mover o motor vertical para a posição de deslocamento.
35	00:08.214	δ_5	0	2	Fim da movimentação do motor vertical.
36	00:08.224	h_2	1	3	Mover o motor horizontal para a próxima tira.
37	00:08.241	n_2	1	3	n=tiras e v=vezes: só falta a aspiração final.
38	00:08.242	x_3	1	3	O tempo de molho da tira atual já é suficiente.
39	00:08.365	δ_4	1	3	Fim da movimentação do motor horizontal.
40	00:08.379	v_3	1	3	Mover o motor vertical para a posição de sucção.
41	00:09.456	δ_5	1	3	Fim da movimentação do motor vertical.
42	00:09.956	x_1	1	3	O tempo de aspiração já passou.
43	00:09.975	v_2	1	3	Mover o motor vertical para a posição de injeção.
44	00:10.952	δ_5	1	3	Fim da movimentação do motor vertical.
45	00:10.964	β_3	1	3	Fazer a bomba de seringa injetar.
46	00:11.555	δ_3	1	3	Fim da movimentação da bomba de seringa.
47	00:11.569	v_1	1	3	Mover o motor vertical para a posição de deslocamento.
48	00:11.570	β_2	1	3	Mover a válvula 2 para 0º.
49	00:11.731	δ_5	1	3	Fim da movimentação do motor vertical.
50	00:11.742	h_1	1	1	Mover o motor horizontal para a primeira tira.
51	00:11.755	n_0	1	1	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
52	00:11.864	δ_2	1	1	Fim da movimentação da válvula 2.
53	00:12.004	δ_4	1	1	Fim da movimentação do motor horizontal.
54	00:33.940	x_3	1	1	O tempo de molho da tira atual já é suficiente.
55	00:33.958	v_3	1	1	Mover o motor vertical para a posição de sucção.
56	00:35.034	δ_5	1	1	Fim da movimentação do motor vertical.
57	00:35.534	x_1	1	1	O tempo de aspiração já passou.
58	00:35.551	v_1	1	1	Mover o motor vertical para a posição de deslocamento.
59	00:36.622	δ_5	1	1	Fim da movimentação do motor vertical.
60	00:36.639	h_2	1	2	Mover o motor horizontal para a próxima tira.
61	00:36.651	n_0	1	2	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
62	00:36.775	δ_4	1	2	Fim da movimentação do motor horizontal.
63	00:37.458	x_3	1	2	O tempo de molho da tira atual já é suficiente.
64	00:37.468	v_3	1	2	Mover o motor vertical para a posição de sucção.
65	00:38.545	δ_5	1	2	Fim da movimentação do motor vertical.
66	00:39.045	x_1	1	2	O tempo de aspiração já passou.

67	00:39.055	v_1	1	2	Mover o motor vertical para a posição de deslocamento.
68	00:40.132	δ_5	1	2	Fim da movimentação do motor vertical.
69	00:40.144	h_2	2	3	Mover o motor horizontal para a próxima tira.
70	00:40.158	n_1	2	3	n=tiras e v<vezes: final do ciclo. Faltam mais ciclos.
71	00:40.282	δ_4	2	3	Fim da movimentação do motor horizontal.
72	00:40.978	$\underline{x_3}$	2	3	O tempo de molho da tira atual já é suficiente.
73	00:40.993	v_3	2	3	Mover o motor vertical para a posição de sucção.
74	00:42.072	δ_5	2	3	Fim da movimentação do motor vertical.
75	00:42.572	x_1	2	3	O tempo de aspiração já passou.

Essa simulação envolveu 75 eventos em 42.572 segundos. O tempo de molho foi muito grande, de 30s. Como apenas três tiras foram lavadas, o tempo gasto pelo cabeçote de agulhas para percorrer as três tiras é inferior a 30s. Assim, nota-se no evento 53 que o cabeçote termina o movimento, iniciado no evento 50, chegando à primeira tira no tempo 12,004s. Entretanto, o tempo de molho da primeira tira ainda não passou, e o cabeçote espera até que isso aconteça, no evento 54, que ocorre em 33.940 segundos, exatamente 30,011s após a bomba de seringa ter começado a operação de injeção nessa tira, no evento 19. Após lavar a tira 1, o motor horizontal não precisa esperar muito para lavar as tiras 2 e 3 (eventos 63 e 72). Nessa simulação também nota-se a ocorrência do evento β_2, número 48, que representa a válvula 2 indo para 0º. Esse evento ocorre somente uma vez, depois que a operação de injeção da última tira no último ciclo foi realizada, de acordo com a especificação $E_{g,bv2}$.

Simulação nº2.
Volume de lavagem: $400\mu L$;
Número de tiras: 3;
Número de ciclos: 2;
Tempo de molho: 5s;

i	**Tempo (m:s.ms)**	**Evento**	**Ciclo**	**Tira**	**Descrição**
0	00:00.014	v_1	0	1	Mover o motor vertical para a posição de deslocamento.
1	00:00.015	α_2	0	1	Mover a válvula 2 para 90º.
2	00:00.030	α_1	0	1	Mover a válvula 1 para 90º.
3	00:00.250	δ_5	0	1	Fim da movimentação do motor vertical.
4	00:00.265	h_1	0	1	Mover o motor horizontal para a primeira tira.
5	00:00.275	n_0	0	1	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
6	00:00.276	x_3	0	1	O tempo de molho da tira atual já é suficiente.
7	00:00.301	δ_2	0	1	Fim da movimentação da válvula 2.
8	00:00.325	δ_1	0	1	Fim da movimentação da válvula 1.
9	00:00.336	α_3	0	1	Encher a bomba de seringa.
10	00:01.334	δ_4	0	1	Fim da movimentação do motor horizontal.
11	00:01.352	v_3	0	1	Mover o motor vertical para a posição de sucção.
12	00:02.432	δ_5	0	1	Fim da movimentação do motor vertical.
13	00:02.932	x_1	0	1	O tempo de aspiração já passou.
14	00:02.949	v_2	0	1	Mover o motor vertical para a posição de injeção.
15	00:03.553	δ_3	0	1	Fim da movimentação da bomba de seringa.
16	00:03.563	β_1	0	1	Mover a válvula 1 para 0º.
17	00:03.852	δ_1	0	1	Fim da movimentação da válvula 1.
18	00:03.929	δ_5	0	1	Fim da movimentação do motor vertical.
19	00:03.948	β_3	0	1	Fazer a bomba de seringa injetar.
20	00:05.040	δ_3	0	1	Fim da movimentação da bomba de seringa.
21	00:05.056	v_1	0	1	Mover o motor vertical para a posição de deslocamento.
22	00:05.211	δ_5	0	1	Fim da movimentação do motor vertical.
23	00:05.223	h_2	0	2	Mover o motor horizontal para a próxima tira.
24	00:05.234	n_0	0	2	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
25	00:05.235	x_3	0	2	O tempo de molho da tira atual já é suficiente.
26	00:05.358	δ_4	0	2	Fim da movimentação do motor horizontal.
27	00:05.369	v_3	0	2	Mover o motor vertical para a posição de sucção.
28	00:06.443	δ_5	0	2	Fim da movimentação do motor vertical.
29	00:06.943	x_1	0	2	O tempo de aspiração já passou.
30	00:06.958	v_2	0	2	Mover o motor vertical para a posição de injeção.
31	00:07.933	δ_5	0	2	Fim da movimentação do motor vertical.
32	00:07.947	β_3	0	2	Fazer a bomba de seringa injetar.

33	00:09.045	δ_3	0	2	Fim da movimentação da bomba de seringa.
34	00:09.064	v_1	0	2	Mover o motor vertical para a posição de deslocamento.
35	00:09.225	δ_5	0	2	Fim da movimentação do motor vertical.
36	00:09.237	h_2	1	3	Mover o motor horizontal para a próxima tira.
37	00:09.250	n_1	1	3	n=tiras e v<vezes: final do ciclo. Faltam mais ciclos.
38	00:09.251	x_3	1	3	O tempo de molho da tira atual já é suficiente.
39	00:09.374	δ_4	1	3	Fim da movimentação do motor horizontal.
40	00:09.392	v_3	1	3	Mover o motor vertical para a posição de sucção.
41	00:10.466	δ_5	1	3	Fim da movimentação do motor vertical.
42	00:10.966	x_1	1	3	O tempo de aspiração já passou.
43	00:10.983	v_2	1	3	Mover o motor vertical para a posição de injeção.
44	00:11.959	δ_5	1	3	Fim da movimentação do motor vertical.
45	00:11.975	β_3	1	3	Fazer a bomba de seringa injetar.
46	00:11.986	x_0	1	3	Volume da bomba de seringa é menor do que o necessário.
47	00:13.065	δ_3	1	3	Fim da movimentação da bomba de seringa.
48	00:13.077	v_1	1	3	Mover o motor vertical para a posição de deslocamento.
49	00:13.078	α_1	1	3	Mover a válvula 1 para 90º.
50	00:13.240	δ_5	1	3	Fim da movimentação do motor vertical.
51	00:13.251	h_1	1	1	Mover o motor horizontal para a primeira tira.
52	00:13.266	n_0	1	1	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
53	00:13.267	x_3	1	1	O tempo de molho da tira atual já é suficiente.
54	00:13.371	δ_1	1	1	Fim da movimentação da válvula 1.
55	00:13.388	α_3	1	1	Encher a bomba de seringa.
56	00:13.515	δ_4	1	1	Fim da movimentação do motor horizontal.
57	00:13.527	v_3	1	1	Mover o motor vertical para a posição de sucção.
58	00:14.605	δ_5	1	1	Fim da movimentação do motor vertical.
59	00:15.105	x_1	1	1	O tempo de aspiração já passou.
60	00:15.117	v_2	1	1	Mover o motor vertical para a posição de injeção.
61	00:16.091	δ_5	1	1	Fim da movimentação do motor vertical.
62	00:16.482	δ_3	1	1	Fim da movimentação da bomba de seringa.
63	00:16.492	β_1	1	1	Mover a válvula 1 para 0º.
64	00:16.784	δ_1	1	1	Fim da movimentação da válvula 1.
65	00:16.803	β_3	1	1	Fazer a bomba de seringa injetar.
66	00:17.898	δ_3	1	1	Fim da movimentação da bomba de seringa.
67	00:17.916	v_1	1	1	Mover o motor vertical para a posição de deslocamento.
68	00:18.078	δ_5	1	1	Fim da movimentação do motor vertical.
69	00:18.090	h_2	1	2	Mover o motor horizontal para a próxima tira.
70	00:18.109	n_0	1	2	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
71	00:18.110	x_3	1	2	O tempo de molho da tira atual já é suficiente.
72	00:18.233	δ_4	1	2	Fim da movimentação do motor horizontal.
73	00:18.244	v_3	1	2	Mover o motor vertical para a posição de sucção.
74	00:19.315	δ_5	1	2	Fim da movimentação do motor vertical.
75	00:19.815	x_1	1	2	O tempo de aspiração já passou.
76	00:19.829	v_2	1	2	Mover o motor vertical para a posição de injeção.
77	00:20.803	δ_5	1	2	Fim da movimentação do motor vertical.

78	00:20.814	β_3	1	2	Fazer a bomba de seringa injetar.
79	00:21.908	δ_3	1	2	Fim da movimentação da bomba de seringa.
80	00:21.923	v_1	1	2	Mover o motor vertical para a posição de deslocamento.
81	00:22.079	δ_5	1	2	Fim da movimentação do motor vertical.
82	00:22.089	h_2	2	3	Mover o motor horizontal para a próxima tira.
83	00:22.100	n_2	2	3	n=tiras e v=vezes: só falta a aspiração final.
84	00:22.101	x_3	2	3	O tempo de molho da tira atual já é suficiente.
85	00:22.224	δ_4	2	3	Fim da movimentação do motor horizontal.
86	00:22.236	v_3	2	3	Mover o motor vertical para a posição de sucção.
87	00:23.307	δ_5	2	3	Fim da movimentação do motor vertical.
88	00:23.807	x_1	2	3	O tempo de aspiração já passou.
89	00:23.822	v_2	2	3	Mover o motor vertical para a posição de injeção.
90	00:24.800	δ_5	2	3	Fim da movimentação do motor vertical.
91	00:24.811	β_3	2	3	Fazer a bomba de seringa injetar.
92	00:24.827	x_0	2	3	Volume da bomba de seringa é menor do que o necessário.
93	00:25.906	δ_3	2	3	Fim da movimentação da bomba de seringa.
94	00:25.917	v_1	2	3	Mover o motor vertical para a posição de deslocamento.
95	00:25.917	β_2	2	3	Mover a válvula 2 para 0º.
96	00:25.917	α_1	2	3	Mover a válvula 1 para 90º.
97	00:26.076	δ_5	2	3	Fim da movimentação do motor vertical.
98	00:26.092	h_1	2	1	Mover o motor horizontal para a primeira tira.
99	00:26.107	n_0	2	1	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
100	00:26.108	x_3	2	1	O tempo de molho da tira atual já é suficiente.
101	00:26.203	δ_2	2	1	Fim da movimentação da válvula 2.
102	00:26.205	δ_1	2	1	Fim da movimentação da válvula 1.
103	00:26.356	δ_4	2	1	Fim da movimentação do motor horizontal.
104	00:26.371	v_3	2	1	Mover o motor vertical para a posição de sucção.
105	00:27.451	δ_5	2	1	Fim da movimentação do motor vertical.
106	00:27.951	x_1	2	1	O tempo de aspiração já passou.
107	00:27.966	v_1	2	1	Mover o motor vertical para a posição de deslocamento.
108	00:29.045	δ_5	2	1	Fim da movimentação do motor vertical.
109	00:29.056	h_2	2	2	Mover o motor horizontal para a próxima tira.
110	00:29.066	n_0	2	2	n<tiras e v<vezes: Ainda faltam tiras a serem lavadas.
111	00:29.067	x_3	2	2	O tempo de molho da tira atual já é suficiente.
112	00:29.190	δ_4	2	2	Fim da movimentação do motor horizontal.
113	00:29.204	v_3	2	2	Mover o motor vertical para a posição de sucção.
114	00:30.275	δ_5	2	2	Fim da movimentação do motor vertical.
115	00:30.775	x_1	2	2	O tempo de aspiração já passou.
116	00:30.790	v_1	2	2	Mover o motor vertical para a posição de deslocamento.
117	00:31.864	δ_5	2	2	Fim da movimentação do motor vertical.
118	00:31.876	h_2	3	3	Mover o motor horizontal para a próxima tira.
119	00:31.892	n_1	3	3	n=tiras e v<vezes: final do ciclo. Faltam mais ciclos.
120	00:31.893	x_3	3	3	O tempo de molho da tira atual já é suficiente.

121	00:32.016	δ_4	3	3	Fim da movimentação do motor horizontal.
122	00:32.034	v_3	3	3	Mover o motor vertical para a posição de sucção.
123	00:33.109	δ_5	3	3	Fim da movimentação do motor vertical.
124	00:33.609	x_1	3	3	O tempo de aspiração já passou.

Nessa simulação, apenas três tiras são lavadas, mas, diferente da primeira simulação, são feitos dois ciclos de lavagem, e são injetados $400\mu L$ em cada micropoço. Apesar disso, o tempo dessa simulação foi menor, 33,609s, contra os 42,572s da simulação anterior. Isso aconteceu porque o tempo de molho dessa simulação foi seis vezes menor em relação ao da primeira, apenas 5s. Entretanto, o número de eventos dessa simulação foi maior, 124, contra os 75 da segunda. Isso aconteceu porque foram realizados dois ciclos de lavagem, contra apenas um na simulação anterior.

Simulação nº3.
Volume de lavagem: $400\mu L$;
Número de tiras: 3;
Número de ciclos: 2;
Tempo de molho: 5s;

Essa simulação foi feita com os mesmos parâmetros de programação da simulação 2. A diferença é que dessa vez foram mostrados os estados de todos os *automata*, e não as descrições dos evento.

i	$\mathbf{t_i}$ (s.ms)	$\mathbf{E_i}$	**v**	**n**	$\mathbf{G_1}$	$\mathbf{G_2}$	$\mathbf{G_3}$	$\mathbf{G_4}$	$\mathbf{G_5}$	E_g										
										vl1	**vl2**	**bmb**	**bv2**	**hv1**	**hv2**	**hbm**	**inj**	**asp**	**lp1**	**lp2**
0	00.011	α_2	0	1	0	1	0	0	0	0	2	1	0	0	0	0	0	0	0	0
1	00.012	v_1	0	1	0	1	0	0	1	0	2	1	0	1	2	0	0	0	0	0
2	00.022	α_1	0	1	1	1	0	0	1	2	2	2	0	1	2	0	0	0	0	0
3	00.250	δ_5	0	1	1	1	0	0	0	2	2	2	0	1	0	0	0	0	0	0
4	00.267	h_1	0	1	1	1	0	1	0	2	2	2	0	1	1	1	0	0	1	0
5	00.285	n_0	0	1	1	1	0	2	0	2	2	2	0	1	1	1	0	0	2	1
6	00.286	x_2	0	1	1	1	0	3	0	2	2	2	0	1	1	1	0	0	2	2
7	00.299	δ_2	0	1	1	2	0	3	0	2	0	2	0	1	1	1	0	0	2	2
8	00.313	δ_1	0	1	2	2	0	3	0	0	0	2	0	1	1	1	0	0	2	2
9	00.327	α_3	0	1	2	2	1	3	0	1	1	3	0	1	1	1	0	0	2	2
10	01.344	δ_4	0	1	2	2	1	0	0	1	1	3	0	1	0	0	0	0	2	2
11	01.358	v_3	0	1	2	2	1	0	1	1	1	3	0	0	2	0	0	1	2	3
12	02.438	δ_5	0	1	2	2	1	0	0	1	1	3	0	0	0	0	0	2	2	3
13	02.938	x_1	0	1	2	2	1	0	0	1	1	3	0	0	0	0	0	0	2	3
14	02.952	v_2	0	1	2	2	1	0	1	1	1	3	0	0	2	0	1	0	2	0
15	03.541	δ_3	0	1	2	2	0	0	1	0	0	3	0	0	2	0	1	0	2	0
16	03.554	β_1	0	1	3	2	0	0	1	2	0	4	0	0	2	0	1	0	2	0
17	03.841	δ_1	0	1	0	2	0	0	1	0	0	4	0	0	2	0	1	0	2	0
18	03.931	δ_5	0	1	0	2	0	0	0	0	0	4	0	0	0	0	2	0	2	0
19	03.941	β_3	0	1	0	2	1	0	0	1	1	4	0	0	0	2	3	0	2	0
20	05.034	δ_3	0	1	0	2	0	0	0	0	0	4	0	0	0	0	0	0	2	0
21	05.048	v_1	0	1	0	2	0	0	1	0	0	4	0	1	2	0	0	0	2	0
22	05.208	δ_5	0	1	0	2	0	0	0	0	0	4	0	1	0	0	0	0	2	0
23	05.223	h_2	0	2	0	2	0	1	0	0	0	4	0	1	1	1	0	0	1	0
24	05.234	n_0	0	2	0	2	0	2	0	0	0	4	0	1	1	1	0	0	2	1
25	05.235	x_2	0	2	0	2	0	3	0	0	0	4	0	1	1	1	0	0	2	2
26	05.358	δ_4	0	2	0	2	0	0	0	0	0	4	0	1	0	0	0	0	2	2
27	05.369	v_3	0	2	0	2	0	0	1	0	0	4	0	0	2	0	0	1	2	3
28	06.442	δ_5	0	2	0	2	0	0	0	0	0	4	0	0	0	0	0	2	2	3
29	06.942	x_1	0	2	0	2	0	0	0	0	0	4	0	0	0	0	0	0	2	3
30	06.957	v_2	0	2	0	2	0	0	1	0	0	4	0	0	2	0	1	0	2	0
31	07.933	δ_5	0	2	0	2	0	0	0	0	0	4	0	0	0	0	2	0	2	0
32	07.947	β_3	0	2	0	2	1	0	0	1	1	4	0	0	0	2	3	0	2	0
33	09.036	δ_3	0	2	0	2	0	0	0	0	0	4	0	0	0	0	0	0	2	0
34	09.054	v_1	0	2	0	2	0	0	1	0	0	4	0	1	2	0	0	0	2	0
35	09.215	δ_5	0	2	0	2	0	0	0	0	0	4	0	1	0	0	0	0	2	0
36	09.233	h_2	1	3	0	2	0	1	0	0	0	4	0	1	1	1	0	0	1	0

37	09.248	n_1	1	3	0	2	0	2	0	0	0	4	0	1	1	1	0	0	3	1
38	09.249	x_2	1	3	0	2	0	3	0	0	0	4	0	1	1	1	0	0	3	2
39	09.372	δ_4	1	3	0	2	0	0	0	0	0	4	0	1	0	0	0	0	3	2
40	09.391	v_3	1	3	0	2	0	0	1	0	0	4	0	0	2	0	0	1	3	3
41	10.468	δ_5	1	3	0	2	0	0	0	0	0	4	0	0	0	0	0	2	3	3
42	10.968	x_1	1	3	0	2	0	0	0	0	0	4	0	0	0	0	0	0	3	3
43	10.978	v_2	1	3	0	2	0	0	1	0	0	4	0	0	2	0	1	0	3	0
44	11.956	δ_5	1	3	0	2	0	0	0	0	0	4	0	0	0	0	2	0	3	0
45	11.971	β_3	1	3	0	2	1	0	0	1	1	4	0	0	0	2	3	0	3	0
46	11.984	x_0	1	3	0	2	1	0	0	1	1	1	0	0	0	2	3	0	3	0
47	13.063	δ_3	1	3	0	2	0	0	0	0	0	1	0	0	0	0	0	0	3	0
48	13.075	v_1	1	3	0	2	0	0	1	0	0	1	0	1	2	0	0	0	3	0
49	13.076	α_1	1	3	1	2	0	0	1	2	0	2	0	1	2	0	0	0	3	0
50	13.231	δ_5	1	3	1	2	0	0	0	2	0	2	0	1	0	0	0	0	3	0
51	13.250	h_1	1	1	1	2	0	1	0	2	0	2	0	1	1	1	0	0	1	0
52	13.268	n_0	1	1	1	2	0	2	0	2	0	2	0	1	1	1	0	0	2	1
53	13.269	x_2	1	1	1	2	0	3	0	2	0	2	0	1	1	1	0	0	2	2
54	13.365	δ_1	1	1	2	2	0	3	0	0	0	2	0	1	1	1	0	0	2	2
55	13.382	α_3	1	1	2	2	1	3	0	1	1	3	0	1	1	1	0	0	2	2
56	13.517	δ_4	1	1	2	2	1	0	0	1	1	3	0	1	0	0	0	0	2	2
57	13.532	v_3	1	1	2	2	1	0	1	1	1	3	0	0	2	0	0	1	2	3
58	14.610	δ_5	1	1	2	2	1	0	0	1	1	3	0	0	0	0	0	2	2	3
59	15.110	x_1	1	1	2	2	1	0	0	1	1	3	0	0	0	0	0	0	2	3
60	15.126	v_2	1	1	2	2	1	0	1	1	1	3	0	0	2	0	1	0	2	0
61	16.100	δ_5	1	1	2	2	1	0	0	1	1	3	0	0	0	0	2	0	2	0
62	16.480	δ_3	1	1	2	2	0	0	0	0	0	3	0	0	0	0	2	0	2	0
63	16.496	β_1	1	1	3	2	0	0	0	2	0	4	0	0	0	0	2	0	2	0
64	16.784	δ_1	1	1	0	2	0	0	0	0	0	4	0	0	0	0	2	0	2	0
65	16.795	β_3	1	1	0	2	1	0	0	1	1	4	0	0	0	2	3	0	2	0
66	17.892	δ_3	1	1	0	2	0	0	0	0	0	4	0	0	0	0	0	0	2	0
67	17.902	v_1	1	1	0	2	0	0	1	0	0	4	0	1	2	0	0	0	2	0
68	18.060	δ_5	1	1	0	2	0	0	0	0	0	4	0	1	0	0	0	0	2	0
69	18.075	h_2	1	2	0	2	0	1	0	0	0	4	0	1	1	1	0	0	1	0
70	18.085	n_0	1	2	0	2	0	2	0	0	0	4	0	1	1	1	0	0	2	1
71	18.086	x_2	1	2	0	2	0	3	0	0	0	4	0	1	1	1	0	0	2	2
72	18.209	δ_4	1	2	0	2	0	0	0	0	0	4	0	1	0	0	0	0	2	2
73	18.223	v_3	1	2	0	2	0	0	1	0	0	4	0	0	2	0	0	1	2	3
74	19.297	δ_5	1	2	0	2	0	0	0	0	0	4	0	0	0	0	0	2	2	3
75	19.797	x_1	1	2	0	2	0	0	0	0	0	4	0	0	0	0	0	0	2	3
76	19.807	v_2	1	2	0	2	0	0	1	0	0	4	0	0	2	0	1	0	2	0
77	20.787	δ_5	1	2	0	2	0	0	0	0	0	4	0	0	0	0	2	0	2	0
78	20.802	β_3	1	2	0	2	1	0	0	1	1	4	0	0	0	2	3	0	2	0
79	21.891	δ_3	1	2	0	2	0	0	0	0	0	4	0	0	0	0	0	0	2	0
80	21.906	v_1	1	2	0	2	0	0	1	0	0	4	0	1	2	0	0	0	2	0
81	22.063	δ_5	1	2	0	2	0	0	0	0	0	4	0	1	0	0	0	0	2	0

82	22.079	h_2	2	3	0	2	0	1	0	0	0	4	0	1	1	1	0	0	1	0
83	22.092	n_2	2	3	0	2	0	2	0	0	0	4	1	1	1	1	0	0	4	4
84	22.093	x_2	2	3	0	2	0	3	0	0	0	4	1	1	1	1	0	0	4	5
85	22.216	δ_4	2	3	0	2	0	0	0	0	0	4	1	1	0	0	0	0	4	5
86	22.230	v_3	2	3	0	2	0	0	1	0	0	4	1	0	2	0	0	1	4	6
87	23.303	δ_5	2	3	0	2	0	0	0	0	0	4	1	0	0	0	0	2	4	6
88	23.803	x_1	2	3	0	2	0	0	0	0	0	4	1	0	0	0	0	0	4	6
89	23.813	v_2	2	3	0	2	0	0	1	0	0	4	1	0	2	0	1	0	4	7
90	24.784	δ_5	2	3	0	2	0	0	0	0	0	4	1	0	0	0	2	0	4	7
91	24.798	β_3	2	3	0	2	1	0	0	1	1	4	2	0	0	2	3	0	4	7
92	24.814	x_0	2	3	0	2	1	0	0	1	1	1	2	0	0	2	3	0	4	7
93	25.893	δ_3	2	3	0	2	0	0	0	0	0	1	2	0	0	0	0	0	4	7
94	25.904	β_2	2	3	0	3	0	0	0	0	2	1	2	0	0	0	0	0	4	7
95	25.905	α_1	2	3	1	3	0	0	0	2	2	2	2	0	0	0	0	0	4	7
96	25.909	v_1	2	3	1	3	0	0	1	2	2	2	2	1	2	0	0	0	4	7
97	26.065	δ_5	2	3	1	3	0	0	0	2	2	2	2	1	0	0	0	0	4	7
98	26.079	h_1	2	1	1	3	0	1	0	2	2	2	2	1	1	1	0	0	5	7
99	26.098	n_0	2	1	1	3	0	2	0	2	2	2	2	1	1	1	0	0	6	8
100	26.099	x_2	2	1	1	3	0	3	0	2	2	2	2	1	1	1	0	0	6	9
101	26.196	δ_1	2	1	2	3	0	3	0	0	2	2	2	1	1	1	0	0	6	9
102	26.197	δ_2	2	1	2	0	0	3	0	0	0	2	2	1	1	1	0	0	6	9
103	26.347	δ_4	2	1	2	0	0	0	0	0	0	2	2	1	0	0	0	0	6	9
104	26.365	v_3	2	1	2	0	0	0	1	0	0	2	2	0	2	0	0	1	6	7
105	27.437	δ_5	2	1	2	0	0	0	0	0	0	2	2	0	0	0	0	2	6	7
106	27.937	x_1	2	1	2	0	0	0	0	0	0	2	2	0	0	0	0	0	6	7
107	27.953	v_1	2	1	2	0	0	0	1	0	0	2	2	1	2	0	0	0	6	7
108	29.030	δ_5	2	1	2	0	0	0	0	0	0	2	2	1	0	0	0	0	6	7
109	29.046	h_2	2	2	2	0	0	1	0	0	0	2	2	1	1	1	0	0	5	7
110	29.061	n_0	2	2	2	0	0	2	0	0	0	2	2	1	1	1	0	0	6	8
111	29.062	x_2	2	2	2	0	0	3	0	0	0	2	2	1	1	1	0	0	6	9
112	29.185	δ_4	2	2	2	0	0	0	0	0	0	2	2	1	0	0	0	0	6	9
113	29.200	v_3	2	2	2	0	0	0	1	0	0	2	2	0	2	0	0	1	6	7
114	30.276	δ_5	2	2	2	0	0	0	0	0	0	2	2	0	0	0	0	2	6	7
115	30.776	x_1	2	2	2	0	0	0	0	0	0	2	2	0	0	0	0	0	6	7
116	30.789	v_1	2	2	2	0	0	0	1	0	0	2	2	1	2	0	0	0	6	7
117	31.865	δ_5	2	2	2	0	0	0	0	0	0	2	2	1	0	0	0	0	6	7
118	31.881	h_2	3	3	2	0	0	1	0	0	0	2	2	1	1	1	0	0	5	7
119	31.892	n_1	3	3	2	0	0	2	0	0	0	2	2	1	1	1	0	0	7	10
120	31.893	x_2	3	3	2	0	0	3	0	0	0	2	2	1	1	1	0	0	7	11
121	32.016	δ_4	3	3	2	0	0	0	0	0	0	2	2	1	0	0	0	0	7	11
122	32.034	v_3	3	3	2	0	0	0	1	0	0	2	2	0	2	0	0	1	7	12
123	33.108	δ_5	3	3	2	0	0	0	0	0	0	2	2	0	0	0	0	2	7	12
124	33.608	x_1	3	3	2	0	0	0	0	0	0	2	2	0	0	0	0	0	7	12

Pode-se notar nessa simulação o intertravamento entre os motores vertical e horizontal, e entre a bomba de seringa e as válvulas. O motor horizontal não fica nos estados 1, 2, 3 ou 4 enquanto o motor vertical está no estado 1. Também se percebe que a bomba de seringa não fica no estado 1 enquanto alguma das válvulas está no estado 1 ou 3.

Capítulo 5

Conclusões

Nessa dissertação foram apresentados a modelagem e o projeto de um controle supervisório modular local para uma lavadora de microplacas, contemplando a operação de lavagem modo placa. A abordagem modular local foi escolhida por ser mais flexível e evitar a composição global da planta, sendo vantajosa em relação ao controle monolítico e ao controle modular. Entretanto, verificou-se que uma desvantagem desse método é a necessidade da composição de todas as especificações locais para verificar a modularidade local. Assim, sempre que uma modificação é feita no sistema de controle, há necessidade de efetuar a composição das especificações, para verificar se o sistema continua modular e, portanto, equivalente à solução monolítica.

A lavadora de microplacas foi modelada e dividida em cinco subplantas: motores vertical e horizontal, bomba de seringa e válvulas 1 e 2. Para o projeto do controle supervisório foram criados novos eventos representando processos internos da lavadora, que foram abstraídos. Esses processos lidam com os parâmetros de programação da lavadora, que são variáveis. Os modelos das subplantas da lavadora foram modificados para incluir os novos eventos criados, e assim, onze especificações genéricas foram construídas baseadas em requisitos de controle para a lavadora.

As onze especificações genéricas deram origem a onze especificações locais, todas controláveis em relação às suas respectivas plantas locais. Então, foram obtidos onze controladores modulares locais reduzidos, e localmente modulares. Assim, o sistema de controle projetado não implica em perda de desempenho em relação à abordagem monolítica.

Os supervisores modulares locais projetados foram implementados em uma simulação construída em *C++*, permitindo testar o sistema de controle. O resultado da simulação é um arquivo contendo os eventos gerados, seus tempos de ocorrência e os estados dos geradores, que permite que diversas análises sejam feitas relacionadas à operação da lavadora. A simulação mostrou-se uma ferramenta útil para testar modificações feitas no sistema de controle antes de serem implementadas, permitindo prever qual o ganho de desempenho proporcionado pelo novo sistema. Também serve como uma base para a implementação do sistema na planta real.

Futuras modificações podem ser feitas no sistema de controle, como alterações na lógica de intertravamento dos motores para tornar o sistema mais eficiente. Uma possibilidade, por exemplo, é fazer a bomba de seringa começar a injetar solução antes que o motor vertical chegue à posição correta de injeção, aqui representada pelo evento v_2, desde que as agulhas estejam em uma altura mínima. Essa modificação diminuiria o tempo de lavagem final. Outras otimizações podem ainda ser feitas. Para adaptar o sistema para a lavagem no modo "tira", apenas algumas especificações teriam que ser modificadas, já que a maior parte do funcionamento da lavadora é semelhante para os dois modos de operação.

No controle supervisório para a lavadora de microplacas, apenas o intertravamento entre os subsistemas foi modelado. Entretanto, outros níveis de aplicação estão presentes no firmware, como a interface com o usuário, que envolve a navegação entre as telas e o aperto das teclas do teclado.

Por fim, conclui-se que a modelagem e o controle supervisório apresen-

tados possuem aplicação muito ampla, sendo possível serem utilizados em diversos sistemas a eventos discretos, e em diversos níveis de abstração.

Referências Bibliográficas

Antsaklis, P. J. & Koutsoukos, X. D. (2002), 'Hybrid systems control', *Encyclopedia of Physical Science and Technology* .

Banks, J., Carson, J. S., Nelson, B. L. & Nicol, D. M. (2000), *Discrete-Event System Simulation (3rd Edition)*, Prentice Hall.

Cassandras, C. G. & Lafortune, S. (1999), *Introduction to Discrete Event Systems*, Springer.

Chen, C. T. (1970), *Introduction to linear system theory*.

Chen, C. T. (1999), *Linear System Theory And Design*, Oxford University Press.

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

Cury, J. E. R. (2001), 'Teoria de controle supervisório de sistemas a eventos discretos', V Simpósio Brasileiro de Automação Inteligente.

Dorf, R. C. & Bishop, R. H. (2001), *Sistemas de Controle Modernos*, LTC Editora.

Frey, G. (2000), Automatic implementation of petri net based control algorithms on plc, *in* 'Proceedings of the 2000 American Control Conference', Vol. 4, pp. 2819–2823.

Gaudin, B. & Merchand, H. (2007), 'An efficient modular method for the control of concurrent discrete event systems: A language-based approach', *Discrete Event Dynamic Systems* **17**(2), 179–209.

Haykin, S. & Veen, B. V. (1999), *Signals and Systems*, Wiley.

Holloway, L. E., Krogh, B. H. & Giua, A. (1997), 'A survey of petri net methods for controlled discrete event systems', *Discrete Event Dynamic Systems: Theory And Applications* **7**, 151–190.

Hopcroft, J. E., Motwani, R. & Ullman, J. D. (2000), *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Addison Wesley.

Hopcroft, J. E. & Ullman, J. D. (1979), *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Massachusetts.

Lee, S., Ang, M. A. & Lee, J. (2006), 'Automatic generation of logic control'. A Joint Study by Loughborough University and University of Michigan for Ford Motor Company.

Lima II, E. J. (2002), 'Uma metodologia para a implementação através de clps de controle supervisório de células de manufatura utilizando redes de petri'. *Dissertação de mestrado em engenharia elétrica*, UFBA.

Ljungkrantz, O., Akesson, K., Richardsson, J. & Andersson, K. (2007), Implementing a control system framework for automatic generation of manufacturing cell controllers, *in* 'ICRA', pp. 674–679.

Maia, C. A., Lüders, R., Mendes, R. S. & Hardouin, L. (2005), 'Estratégias de controle por modelos de referência de sistemas a eventos discretos max-plus lineares', *Revista Controle & Automação* **16**(3), 263–278.

Masakazu Adachi, Toshimitsu Ushio, Y. U. (2006), 'Design of user-interface without automation surprises for discrete event systems', *Control Engineering Practice* **14**(10), 1249–1258.

Murata, T. (1989), Petri nets: Properties, analysis and applications, *in* 'Proceedings of the IEEE', Vol. 77.

Newton, G. C., Gould, L. A. & Kaiser, J. F. (1957), *Analytical Design of Linear Feedback Controls*.

Ogata, K. (1997), *Modern Control Engineering*, Prentice Hall.

ORTHO, HIV-1/HIV-2 ab-Capture ELISA Test System (n.d.).

Petri, C. A. (1962), Kommunikation mit Automaten, PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2. Second Edition: Communication with Automata, New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377, Vol.1, 1966, Pages: Suppl. 1, English translation.

Queiroz, M. H. (2004), 'Controle supervisório modular e multitarefa de sistemas compostos'. *Tese de doutorado em engenharia elétrica*, UFSC.

Queiroz, M. H. & Cury, J. E. R. (2002), 'Controle supervisório modular de sistemas de manufatura', *Revista Controle & Automação* **13**(2), 123–133.

Queiroz, M. H., Cury, J. E. R. & Wonham, W. M. (2005), 'Multitasking supervisory control of discrete-event systems', *Discrete Event Dynamic Systems: Theory and Application* **15**, 375–395.

Ramadge, P. J. G. & Wonham, W. M. (1987), 'Supervisory control of a class of discrete event processes', *SIAM Journal on Control and Optimization* **25**(1), 206–230.

Ramadge, P. J. G. & Wonham, W. M. (1988), 'Modular supervisory control of discrete event systems', *Mathematics of Control of Discrete Event Systems* **1**(1), 13–30.

Ramadge, P. J. G. & Wonham, W. M. (1989), The control of discrete event systems, *in* 'Proceedings of the IEEE', Vol. 77.

Sampath, M., Lafortune, S. & Teneketzis, D. (1998), 'Active diagnosis of discrete event systems', *IEEE Transactions On Automatic Control* **43**(7).

Seborg, D. E., Edgar, T. F. & Mellichamp, D. A. (1989), *Process Dynamics and Control*, John Wiley & Sons.

Vahidi, A., Fabian, M. & Lennartson, B. (2006), 'Efficient supervisory synthesis of large systems', *Control Engineering Practice* **14**(10), 1157–1167.

Vaz, A. & Wonham, W. (1986), 'On supervisor reduction in discrete event systems', *IJC* **44**(2), 475–491.

Wonham, W. M. & Rogers, E. S. (2007), Supervisory control of discrete-event systems. Disponível em `www.control.utoronto.ca/people/profs/wonham/wonham.html`.

XPTCT (2007). Ferramenta computacional para síntese de controle supervisório para sistemas a eventos discretos. Disponível em: `www.control.utoronto.ca/people/profs/wonham/wonham.html`.

Yamalidou, K., Moody, J., Lemmon, M. & Antsaklis, P. (1996), 'Feedback control on petri nets based on place invariants', *Automatica* **32**(1), 15–28.