UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Química

FILIPE ALVES COELHO

MODELAGEM DE SISTEMAS QUÍMICOS COMO MÁQUINAS DE ESTADOS FINITOS PARA FINS DE VERIFICAÇÃO FORMAL

CAMPINAS

2018

FILIPE ALVES COELHO

MODELAGEM DE SISTEMAS QUÍMICOS COMO MÁQUINAS DE ESTADOS FINITOS PARA FINS DE VERIFICAÇÃO FORMAL

Tese apresentada à Faculdade de Engenharia Química da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Engenharia Química.

Orientador: Prof. Dr. ROGER JOSEF ZEMP

Este exemplar corresponde à versão final da tese defendida pelo aluno Filipe Alves Coelho e orientada pelo prof. Dr. Roger Josef Zemp.

CAMPINAS

2018

Agência(s) de fomento e nº(s) de processo(s): CNPq, 140590/2014-5

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

C65m

Coelho, Filipe Alves, 1989-
Modelagem de sistemas químicos como máquinas de estados finitos para fins de verificação formal / Filipe Alves Coelho. – Campinas, SP : [s.n.], 2018.

Orientador: Roger Josef Zemp.
Tese (doutorado) – Universidade Estadual de Campinas, Faculdade de Engenharia Química.

1. Verificação formal. 2. Processos industriais. 3. Simulação de processos. I. Zemp, Roger Josef, 1962-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Química. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Modeling of chemical process systems as finite state machines for formal verification
Palavras-chave em inglês:
Formal verification
Industrial processes
Process simulation
Área de concentração: Engenharia Química
Titulação: Doutor em Engenharia Química
Banca examinadora:
Roger Josef Zemp [Orientador]
Augusto Cezar Alves Sampaio
Luís Cláudio Oliveira Lopes
Luís Fernando Novazzi
Nandamudi Lankalapalli Vijaykumar
Sávio Souza Venancio Vianna
Data de defesa: 18-06-2018
Programa de Pós-Graduação: Engenharia Química

Tese de doutorado defendida por Filipe Alves Coelho e aprovada em 18 de junho de 2018 pela banca examinadora constituída pelos doutores:

Prof. Dr. Roger Josef Zemp
FEQ/UNICAMP

Prof. Dr. Augusto Cezar Alves Sampaio
CIn/UFPE

Prof. Dr. Luís Cláudio Oliveira Lopes
FEQ/UFU

Prof. Dr. Luís Fernando Novazzi
FEI

Prof. Dr. Nandamudi Lankalapalli Vijaykumar
INPE

Prof. Dr. Sávio Souza Venancio Vianna
FEQ/UNICAMP

A ata de defesa, assinada pelos membros da Comissão Examinadora, consta no processo de vida acadêmica do aluno.

"I find your lack of faith disturbing"

Darth Vader

Agradecimentos

Inicialmente aos meus pais e minha avó pelo incentivo e apoio sem os quais não teria chegado até aqui.

Um agradecimento especial à minha namorada Juliana, com a qual dividi bons e maus momentos durante os anos de mestrado e doutorado e que sempre me apoiou incondicionalmente.

Agradeço imensamente os professores Roger Zemp e Ana Cristina por terem me apresentado e proposto o tema desafiador de aplicar a verificação de modelos na engenharia química. Nossas reuniões sempre foram muito produtivas e o aprendizado foi muito além da verificação de modelos!

Agradeço meus amigos de faculdade pelas horas de café, companhia nos bares e conversas acadêmicas. Agora no fim, sinto falta de todos esses momentos.

Gostaria também de agradecer aos membros da banca, os professores Augusto, Luís Cláudio, Luís Novazzi, Vijay e Sávio, pelas contribuições e pela discussão que se deu na defesa de tese, assim como à professora Maria Tereza, que não esteve presente, mas que também fez importantes comentários na qualificação. Sem dúvida as contribuições me fizeram refletir mais sobre o alcance do trabalho e sobre o futuros direcionamentos.

Finalmente, agradeço o suporte financeiro do CNPq através da bolsa de doutorado.

Resumo

A análise da segurança de processos industriais é feita primariamente com ferramentas como HAZOP e Análise de Árvore de Falhas. Esses métodos exigem conhecimentos de diversas áreas e contam com a participação de engenheiros e outros profissionais na elaboração de documentos que descrevem cenários de desastre e planos de contingência. A modelagem e simulação de processos poderia ser usada nesse sentido para auxiliar na determinação de condições operacionais que levariam ao desastre. Porém, explorar todo o espaço de estados do modelo de um processo em busca destes cenários é custoso, e frequentemente inviável, em termos de tempo computacional, pois os cenários a considerar são, usualmente, infinitos. Para resolver problemas exploratórios semelhantes, cientistas da computação desenvolveram métodos para deduzir se modelos de sistemas físicos atendem ou não certas propriedades (um programa nunca pode travar ou um processador nunca pode errar os cálculos, por exemplo). Esse conjunto de métodos, chamado de verificação formal, em geral exige modelos que os engenheiros químicos não estão acostumados a trabalhar, como as máquinas de estados finitos. Assim, o presente trabalho teve por objetivo o desenvolvimento de um método para modelagem de processos químicos como máquinas de estados finitos para que estes modelos pudessem ser utilizados em verificação formal. O método extrai informações sobre a dinâmica do processo e constrói automaticamente as máquinas. O método foi aplicado em um estudo de caso de um reator CSTR com múltiplos estados estacionários, onde o modelo resultante foi verificado a fim de descobrir rotas de partida do equipamento e alcançabilidade dos estados estacionários em termos das variáveis operacionais disponíveis. A técnica se mostrou eficaz e flexível o suficiente para construir a máquina a partir de diferentes fontes de informação da dinâmica do processo, inclusive, em princípio, de simuladores de processos.

Palavras-chave: Verificação formal, dinâmica de processos, detecção de falhas.

Abstract

HAZOP and Fault Tree Analysis are generally used as tools for industrial safety analysis and they usually require knowledge from several different areas. Engineers and other professionals use those tools to develop documents describing hazard scenarios and contingency plans. Process modelling and simulation could help during the elaboration of this documentation by indicating operational conditions with potential to disaster. However, exhaustive exploration of the model state space to search for these scenarios is very time consuming, and frequently impracticable, because the considered scenarios are, usually, infinite. To solve similar exploratory problems, computer scientists developed several methods to prove if models of physical systems meet some specifications (e.g. a software must not have deadlock or a CPU must not miss any operation). These methods, called formal verification, sometimes require models that chemical engineers do not usually deal with, like finite state machines. This work aimed at the development of a method to model chemical processes as finite state machines for formal verification purposes. The method extracts information about the system's dynamics and builds automatically the machines. The technique was applied on a case study of a multiple steady state CSTR, where the finite state model was verified in order to find viable startup strategies and analyze reachability of stable steady states with the available operational variables. It was demonstrated that the method is effective and flexible enough to build the machines from serveral different sources of information of the process dynamics, including, in principal, from process simulators.

Keywords: Formal verification, process dynamics, fault detection.

Nomenclatura

Variáveis

A	Área
a	Abertura da válvula
C	Concentração
c_p	Calor específico
c_v	Coeficiente da válvula
h	Nível
M_{h}	Massa da haste da válvula
n	Número macroestados de uma variável
n_*	Número de microestados da variável
Q	Vazão volumétrica
Q_h	Taxa de transferência de calor
r	Raio da partícula ou taxa de reação
T	Temperatura
t	Tempo
V	Volume
U	Coeficiente global de transferência de calor
$\mathbf{y}$	Estado do sistema (vetor)
$\Delta\tilde{y}_*$	Salto do microestado de y
ΔH_r	Entalpia de reação

Símbolos

$\mathcal{M}$	Máquina de estados
$\mathcal{M}_\delta$	Máquina de estados que modela um microestado
$\mathcal{M}_\pi$	Máquina de estados que modela um macroestado
Re	Parte real de um número complexo
$\lfloor\ \rceil$	Arredondamento para inteiro mais próximo
$\lfloor\ \rfloor$	Arredondamento para baixo
$x \bmod y$	Resto da divisão x/y
?	Recepção de mensagem
!	Envio de mensagem
ρ	Massa específica
λ	Autovalor

Sobrescritos

$\tilde{}$	Variável codificada (ou operação de codificação) para números inteiros

Subscritos

*	Microestado da variável
0	Alimentação
r	Fluido refrigerante

Abreviaturas

AAF Análise de Árvore de Falhas

CLP Controlador Lógico Programável

CSTR Continous stirred tank reactor

CTL Computation Tree Logic

EDO Equação Diferencial Ordinária

EDP Equação Diferencial Parcial

HAZOP Hazard and Operability Studies

LAC Lógica da árvore de computação

LTL Linear Temporal Logic

MEF Máquina de estados finitos

MTL Metric Temporal Logic

OBDD Ordered Binary Decision Diagrams

PCTL Probabilistic Computation Tree Logic

PRISM Probabilistic Symbolic Model Checker

SMV Symbolic Moldel Verifier

TCTL Timed Computation Tree Logic

Sumário

capítulo 1

Introdução

1.1 Motivação

A operação adequada de processos químicos depende de inúmeros fatores, como o funcionamento correto dos equipamentos, estrutura de controle eficaz (quando necessária) e operadores devidamente preparados. Entretanto, a interação entre essas diferentes partes que constituem o sistema químico pode ser consideravelmente complexa e difícil de ser prevista na etapa de projeto, principalmente ao se considerar análises de segurança ou dimensionamento de sistemas robustos.

O projeto de um sistema químico robusto leva em conta incertezas e possíveis desvios do processo em relação à operação normal(1), de forma que a compensação dos desvios permite o produto final atingir as especificações ou pelo menos manter o sistema seguro para os operadores. Tais sistemas devem considerar diferentes fontes de perturbações ou até falhas operacionais.

Em termos de segurança, algumas metodologias podem ser empregadas para avaliar os processos químicos, como a metodologia HAZOP (*Hazard and Operability Studies*) e Análise de Árvore de Falhas (AAF). Ambos têm como resultado uma lista das sequências de eventos que podem resultar em acidentes, danos ambientais ou desligamentos indesejados do processo (2). Conhecendo as sequências de eventos é possível preparar os operadores para situações adversas e elaborar procedimentos emergenciais em função de acidentes.

Os engenheiros químicos também podem lançar mão de simulações para avaliar o

funcionamento de sistemas químicos. Com um modelo do sistema, pode-se testar faixas de operação das variáveis e reproduzir possíveis condições operacionais a fim de predizer o comportamento do processo. Porém, uma vasta quantidade de condições operacionais não pode ser avaliada e um dos fatores limitantes é o tempo computacional, visto que dependendo da quantidade de variáveis de entrada no sistema, uma explosão combinatorial tornaria a avaliação de todos os casos impraticável para ser feita manualmente.

Às vezes, casos simples levantam problemas inesperados na prática, como a situação exemplificada na Figura 1.1: um sistema automático de enchimento de recipientes composto por um sensor de vazão e uma válvula solenóide. Se esse sistema for programado para encher uma quantidade de fluido nos tanques ao pressionar o botão *Quantidade* e iniciar o processo de enchimento em resposta ao pressionamento do botão *Inicia*, o que aconteceria se os dois botões fossem pressionados *simultaneamente*? Como a programação reagiria a essa situação?

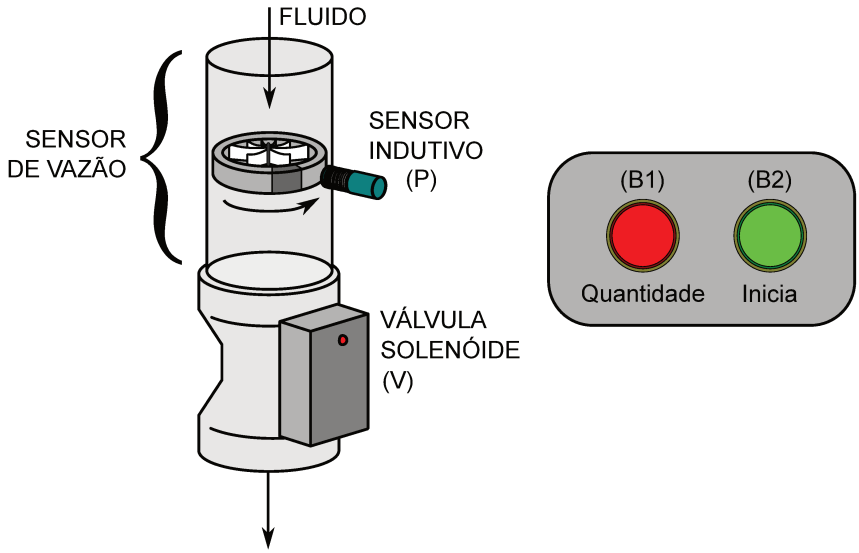


Figura 1.1: Sistema de enchimento automático de recipientes.

Problemas desse tipo são encontrados em praticamente todas as engenharias, pois, idealmente, todo projeto de engenharia deveria ser razoavelmente robusto, mas nem todo projeto pode ser simulado para todas as possíveis condições operacionais.

A ciência (e engenharia) da computação, na contramão das outras engenharias, desenvolveram métodos matemáticos sofisticados para avaliar se o projeto de um *hardware* ou um *software* satisfaz determinados comportamentos desejados (propriedades). Dentro dos métodos matemáticos, a verificação formal engloba uma série de metodologias para

provar matematicamente que um sistema viola ou não propriedades especificadas pelo projetista.

Dentre as ramificações da verificação formal existe a verificação de modelos, utilizada para fazer verificação de *software*[1] de forma automática, testando "todos" os possíveis cenários que o programa pode ser submetido durante seu uso. Quando um verificador de modelos encontra uma falha em um programa, ele encerra a verificação e avisa ao programador a sequência de eventos que levaram o programa a violar a propriedade lógica especificada, um contraexemplo. Dessa forma, o programador pode reprojetar o *software* para que ele não viole mais a propriedade.

O que a verificação de modelos proporcionou aos engenheiros da computação e programadores de uma forma geral, foi um auxílio na detecção de falhas na etapa de projeto, pois o engenheiro é alertado a modificá-lo sempre que uma falha é encontrada. Tal auxílio teria grande valor se aplicado à engenharia química, pois permitiria o projeto de sistemas químicos que não violassem propriedades especificadas pelo engenheiro como "um tanque nunca deve vazar" ou "mesmo com uma sequência de falhas de válvulas da planta, a temperatura de ignição do reator nunca deve ser atingida", por exemplo, e orientaria o projeto na direção de um sistema "livre" de falhas.

Uma inconveniência da verificação de modelos que dificulta sua aplicação na engenharia química é que, para a maioria dos verificadores, os sistemas devem ser modelados como Máquinas de Estados Finitos (MEF). A modelagem com MEFs considera que o sistema assume estados discretos e que certos eventos provocam a transição entre os estados (3). Em muitas situações, esses modelos são essencialmente qualitativos e não descrevem explicitamente o tempo.

Dado que os modelos usuais de processos químicos são normalmente quantitativos e contínuos (descritos por equações diferenciais e/ou algébricas) a mudança de paradigma de modelagem tem sido uma dificuldade para o emprego da verificação de modelos nas engenharias, especialmente a química.

Diante desta dificuldade, o presente trabalho tem por objetivo a demonstração de uma metodologia para construção automática de modelos na forma de máquinas de estados finitos, especificamente para fins de verificação formal, a partir da compilação do conhecimento da dinâmica do processo e independente da natureza da fonte de conheci-

[1] O verificador de modelos checa o **modelo matemático** de um programa (neste texto os termos *software* e programa são usados de forma indistinta).

mento (equações diferenciais, dados reais da planta, conhecimento dos operadores, etc). Se a fonte de conhecimento dispuser de informações suficientes, a máquina de estados resultante imitará a dinâmica do processo que se deseja modelar. Para contextualizar a principal contribuição deste trabalho, a Figura 1.2 destaca como essa metodologia pode facilitar o uso da verificação formal a partir, por exemplo, de equações diferenciais como fonte de conhecimento da dinâmica do processo.

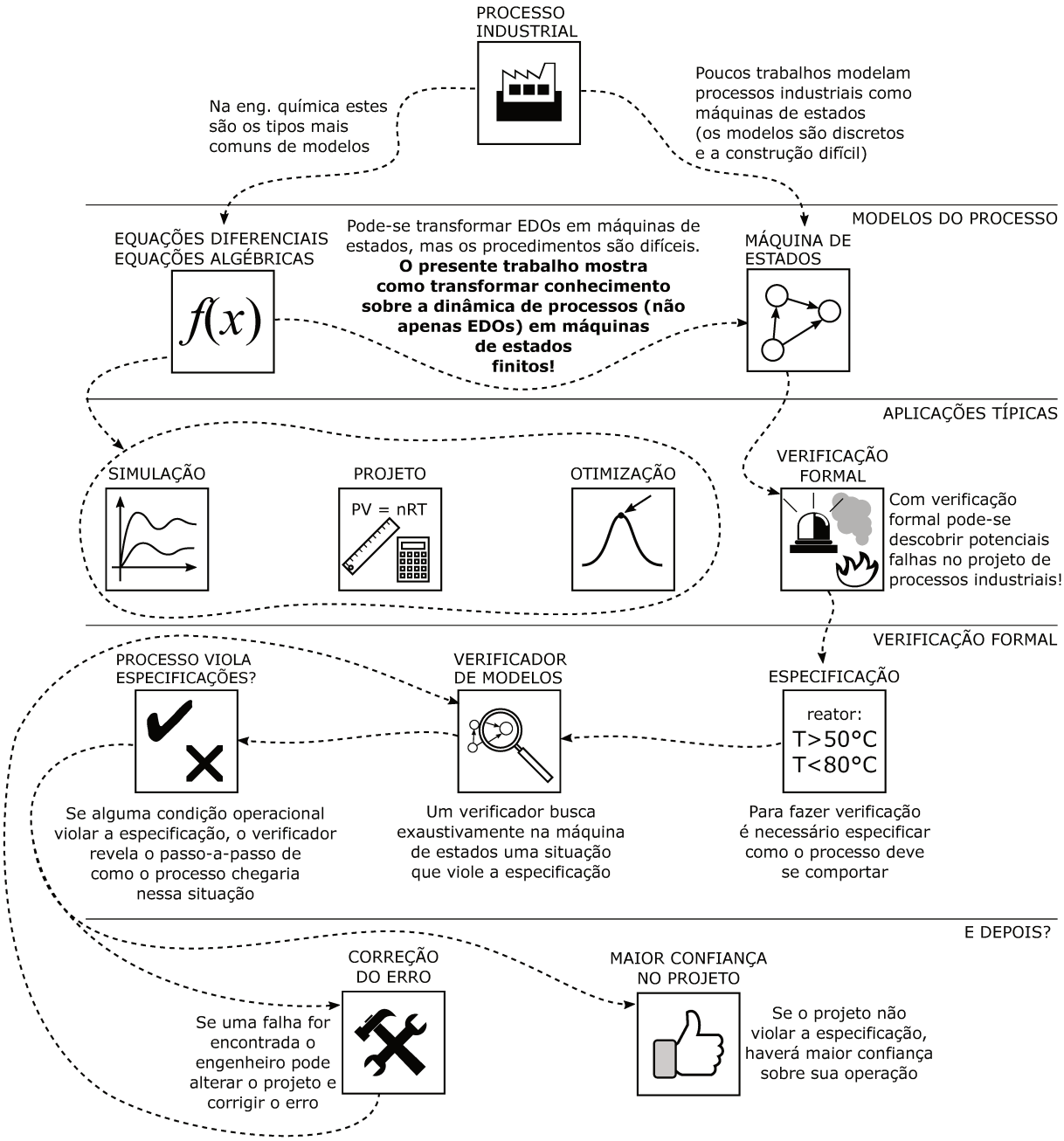


Figura 1.2: Aplicação da verificação formal na inspeção de falhas na etapa de projeto do processo.

1.2 Objetivos

Neste trabalho, teve-se como objetivo geral o desenvolvimento de uma metodologia de conversão de modelos (fontes de conhecimento) da dinâmica de processos químicos para máquinas de estados estendidos visando o uso em verificação formal.

Durante a execução do trabalho, objetivos específicos foram estabelecidos a fim de cumprir o objetivo geral. Dentre eles:

- Escolha da abordagem e da ferramenta de verificação formal adequada para a engenharia de processos
- Estabelecimento de metodologia para tradução dos modelos para máquinas de estados estendidos
- Implementação da metodologia em linguagem Matlab para construção das máquinas a partir de modelos escritos em m-files
- Aplicação da metodologia em estudos de caso para avaliar a capacidade de conversão da metodologia
- Análise de um estudo de caso em verificação formal para avaliação do desempenho dos modelos construídos

1.3 Estrutura do trabalho

O presente trabalho foi dividido em 5 capítulos cujo conteúdo é resumidamente descrito a seguir.

Capítulo 2 Apresenta a teoria por trás da verificação formal e verificação de modelos, lógica temporal, máquinas de estados e aplicações desses métodos e modelos em diversas áreas, inclusive na engenharia química.

Capítulo 3 Neste capítulo é introduzido o conceito do método $\delta\pi$ para construção automática de máquinas estendidas a partir de equações diferenciais e/ou outras fontes de conhecimento da dinâmica do processo.

Capítulo 4 O método $\delta\pi$ foi empregado para modelar um reator com múltiplos estados estacionários e diversas propriedades foram utilizadas para verificar o comportamento dinâmico do reator. Este capítulo também possui uma breve introdução à teoria de multiplicidade de estados estacionários.

Capítulo 5 Contém as conclusões do trabalho e perspectivas futuras em relação ao método e verificação formal na engenharia química.

capítulo 2

Fundamentação teórica

Para compreender o desenvolvimento do método $\delta\pi$ para síntese de máquinas de estados é necessário entender os fundamentos da verificação de modelos. Este método de verificação formal foi empregado juntamente com a lógical temporal CTL e máquinas estendidas para especificar e modelar, respectivamente, os processos industriais. Neste capítulo, são introduzidos os conceitos básicos e alguns exemplos, quando possível, direcionados para a engenharia química.

2.1 Máquinas de estados finitos

Grande parte dos verificadores de modelos utilizam Máquinas de Estados Finitos (MEF), ou autômatos finitos, para modelar o sistema que se quer provar correto. Essa classe de modelos é pouco conhecida na engenharia química, mas foi a base da modelagem deste trabalho.

Como o próprio nome já diz, máquinas de estados finitos representam os estados discretos e finitos de um determinado sistema. Apesar do autômato possuir finitos estados, sua execução pode ser infinita, ou seja, ele pode transitar entre os estados indefinidamente.

A Figura 2.1 ilustra um diagrama de estados de uma MEF: os estados discretos do sistema são representados por círculos, as transições entre os estados são representadas por setas, o estado inicial (por onde a máquina começa sua execução) é representado por uma seta sem origem chegando a esse estado e, finalmente, o estado de aceitação é representado por um círculo duplo.

A máquina deste exemplo pode receber uma cadeia de entradas (variáveis de entrada) formada por 0 e 1, que é processada e fornece uma saída do tipo “aceita” ou “rejeita” (4). Se a máquina receber uma cadeia de entrada 0101 ela se comportará da seguinte maneira:

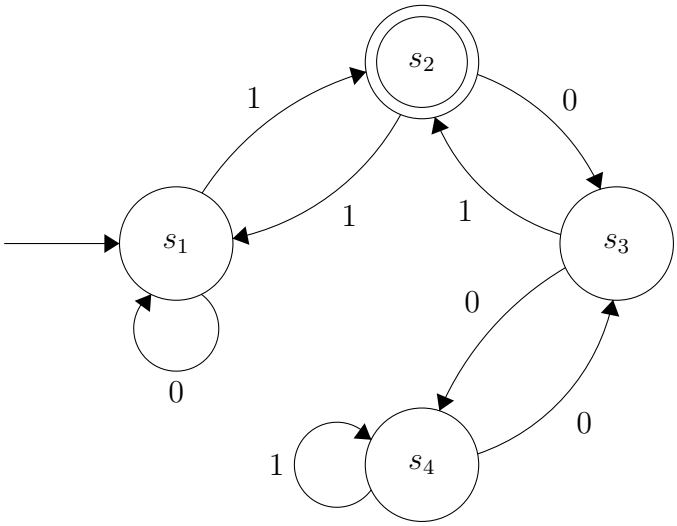


Figura 2.1: Exemplo de autômato finito com 4 estados.

1. Começa no estado s_1
2. Recebe 0 e permanece em s_1
3. Recebe 1 e efetua transição para s_2
4. Recebe 0 e efetua transição para s_3
5. Recebe 1 e efetua transição para s_2

Como o estado final da cadeia de entrada é s_2 isso significa que a máquina aceita a entrada (s_2 é o estado de aceitação). Se a cadeia terminasse em s_3 a máquina rejeitaria a cadeia de entrada. Essa é uma forma clássica de representar reconhecimento de linguagens na área da computação. A entrada seria um fragmento de linguagem e a máquina reconhece, ou não, aquela entrada como um fragmento da linguagem.

Nem toda máquina de estados é usada para reconhecimento de linguagem, portanto, o estado de aceitação pode ou não existir e dependendo da aplicação, a máquina pode incorporar outras características, como manipulação de variáveis de estado inteiras ou informações de probabilidades. De fato, neste trabalho as máquinas não possuirão estados de aceitação, mas outras informações de maior interesse na modelagem de processos químicos.

Outra observação importante acerca dos autômatos finitos é que nem sempre eles modelam o tempo de forma explícita. Por isso, não é possível extrair informações sobre o tempo que o sistema da Figura 2.1 leva para sair do estado q_1 até o estado final. Autômatos finitos mais simples focam na descrição dos estados do sistema e dos eventos que provocam as transições, portanto, são modelos essencialmente qualitativos. Entretanto, outras classes de modelos mais complexas podem incorporar a informação de tempo, como será mostrado mais à frente.

Sistemas encontrados na engenharia química podem ser modelados como autômatos finitos, porém raramente o são, visto estes sistemas são em geral descritos por meio de equações diferenciais, equações algébricas, entre outros modelos contínuos. Porém, a mudança de paradigma de modelagem é essencial para se utilizar das técnicas já estabelecidas de verificação de modelos.

2.1.1 Definição matemática

Existem diversas definições matemáticas para as máquinas de estados, dependendo do tipo de informação agregada a ela. Uma definição simples, sem estados de aceitação, de acordo com Schnoebelen *et al*(5) é que dado um conjunto de proposições atômicas $A = \{P_1,P_2,\ldots\}$[1], uma máquina de estados finitos pode ser definida como uma estrutura $\mathcal{M} = (S,E,T,L,s_0)$, assumindo-se

- S é um conjunto finito de estados;
- E é um conjunto finito de rótulos de transições;
- $T \subseteq S \times E \times S$ é um conjunto de transições;
- L é um mapeamento que associa cada estado de S ao conjunto finito de proposições atômicas, A válidas nesse estado;
- $s_0 \in S$ é o estado inicial da máquina.

O uso de proposições atômicas não é obrigatório, mas é uma forma de descrever os estados da máquina. No método para construção de máquinas de estados apresentado

[1]Proposições atômicas são sentenças declarativas que podem ser verdadeiras ou falsas e, no contexto das máquinas de estados, elas definem o que está acontecendo em um determinado estado. A seção 2.2 contém exemplos de proposições atômicas.

neste trabalho, não foi necessário o uso de proposições atômicas, visto que as informações processadas pelas máquinas são numéricas, mas elas foram usadas na construção das propriedades do sistema (como o sistema deve se comportar) e por isso foram discutidas na seção de lógica temporal.

Por fim, os rótulos das transições indicam os eventos responsáveis pela transição da máquina entre os estados.

Para exemplificar a definição matemática apresentada, retoma-se a Figura 1.1, um sistema de enchimento de recipientes composto por válvula solenoide, sensor de vazão e alguns botões de comando.

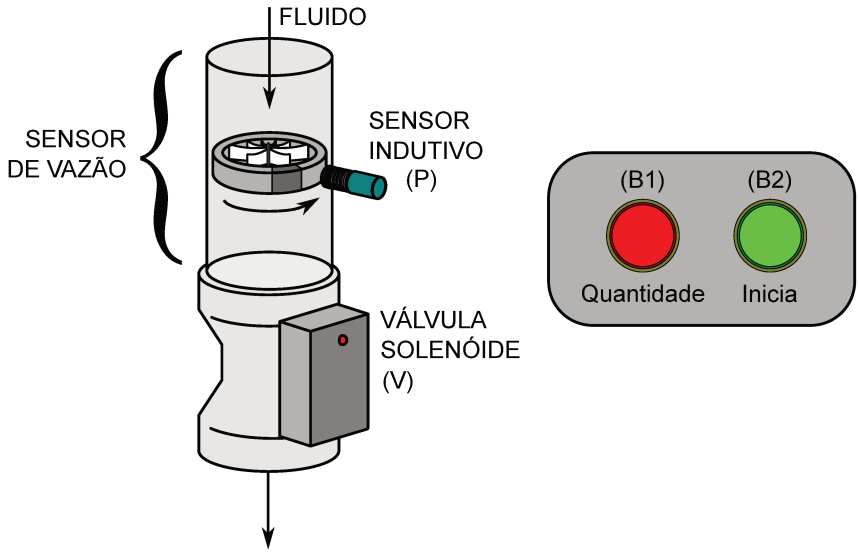


Figura 1.1: Sistema de enchimento automático de recipientes.

O sensor de vazão é composto por uma turbina, que ao completar uma volta envia um pulso para o sensor indutivo contabilizar a passagem de 0,1 ℓ. O botão B_1 é usado para programar a quantidade de fluido a ser colocada no recipiente: 100, 200 ou 300 ℓ, que equivale, respectivamente, à contagem de 1000, 2000 e 3000 pulsos do sensor indutivo. A programação é feita pressionando-se o botão B_1 uma vez para 100 ℓ, duas vezes para 200 ℓ e três vezes para 300 ℓ. Se o botão for pressionado quatro vezes o sistema reinicia e zera a quantidade de fluido a ser despejada no recipiente. Finalmente, depois de pressionar B_1 e selecionar a quantidade de fluido, o enchimento inicia ao pressionar-se o botão B_2. Um contador interno do sistema contabiliza os pulsos e fecha a válvula quando a quantidade programada de fluido passar pela turbina.

A respectiva máquina de estados está ilustrada na Figura 2.3, a qual modela um comportamento simplificado do sistema.

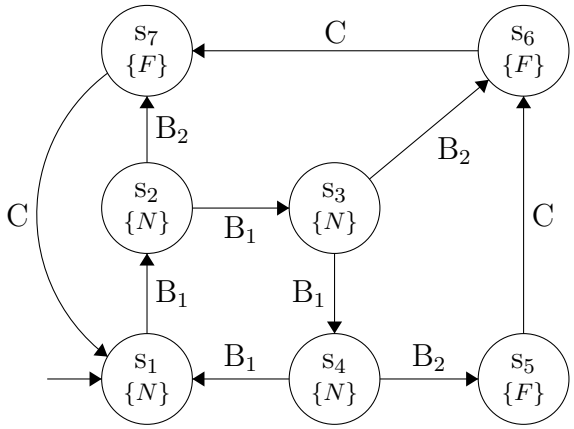


Figura 2.3: Estados discretos do sistema da Figura 1.1.

Este autômato pode ser definido formalmente como:

$$S = \{\mathrm{s}_1,\mathrm{s}_2,\mathrm{s}_3,\mathrm{s}_4,\mathrm{s}_5,\mathrm{s}_6,\mathrm{s}_7\};$$

$$E = \{\mathrm{B}_1,\mathrm{B}_2,\mathrm{C}\};$$

$$s_0 = \{\mathrm{s}_1\};$$

$$L = \{(\mathrm{s}_1,\{N\}),(\mathrm{s}_2,\{N\}),(\mathrm{s}_3,\{N\}),(\mathrm{s}_4,\{N\}),(\mathrm{s}_5,\{F\}),(\mathrm{s}_6,\{F\}),(\mathrm{s}_7,\{F\})\}$$

$$T = \{(\mathrm{s}_1,\mathrm{B}_1,\mathrm{s}_2),(\mathrm{s}_2,\mathrm{B}_1,s_3),(\mathrm{s}_2,\mathrm{B}_2,\mathrm{s}_7),(\mathrm{s}_3,\mathrm{B}_2,\mathrm{s}_6),(\mathrm{s}_3,\mathrm{B}_1,\mathrm{s}_4),(\mathrm{s}_4,\mathrm{B}_1,\mathrm{s}_1),$$
$$(\mathrm{s}_4,\mathrm{B}_2,\mathrm{s}_5),(\mathrm{s}_5,\mathrm{C},\mathrm{s}_6),(\mathrm{s}_6,\mathrm{C},\mathrm{s}_7),(\mathrm{s}_7,\mathrm{C},\mathrm{s}_1)\}.$$

onde as proposições atômicas possuem as seguintes definições:

F : "Recipiente está sendo enchido (a válvula está aberta)"

N : "Recipiente não está sendo enchido (a válvula está fechada)"

e os eventos:

B_1 : "Botão B_1 pressionado"

B_2 : "Botão B_2 pressionado"

C : "Contador contabilizou 1000 pulsos adicionais do sensor indutivo"

Nos estados s_1, s_2, s_3 e s_4 é válida a proposição N, ou seja, nesses estados a válvula está fechada e o recipiente não está sendo enchido. Esses estados fazem parte da etapa de programação do sistema. Os estados s_5, s_6 e s_7, por sua vez, possuem a proposição F, indicando enchimento do recipiente e são ativados quando o evento B_2 ocorre. Pode-se usar ainda a notação $L(\mathrm{s}_2) = \{N\}$, para indicar a proposição atômica válida para o estado s_2.

A MEF definida para este exemplo é uma abstração do comportamento real do sistema, que pode ou não representar todas as possíveis respostas do sistema. Por exemplo, o sistema real poderia interromper o enchimento ao pressionar o botão B_1 (e retornar para o estado s_1) ou permitir enchimento manual enquanto o sistema estivesse no estado s_1 e o botão B_2 fosse mantido pressionado. Esses comportamentos não são contemplados pelo modelo apresentado, mas essas situações poderiam ser de interesse de investigação para verificação formal, assim, o modelo pode ser refinado ou simplificado dependendo das necessidades.

Finalmente, um autômato finito nem sempre precisa responder a todas as entradas. Por exemplo, se o sistema está no estado s_5 a MEF não responde a uma entrada B_1 ou B_2. Ela também não responde a entradas simultâneas, como $B_1 \wedge B_2$ quando se encontra no estado s_3, apenas uma entrada por vez. Essa é uma definição que será usada neste trabalho, ou seja, **se uma dada entrada não corresponde a uma transição em um dado estado, essa entrada é ignorada**.

2.1.2 Execuções de um autômato

A execução de uma máquina de estados finitos é uma sequência de estados que descreve uma possível evolução do sistema. Portanto, formalmente, um fragmento de execução (ϱ) de uma máquina $\mathcal{M}$, como definida na seção anterior, é uma sequência alternada de estados e transições, encerrando sempre em um estado, conforme Equação 2.1, onde $s_i \in S$ e $e_i \in E$ (3).

$$\varrho = s_0 e_1 s_1 e_2 \ldots e_n s_n \tag{2.1}$$

Para a constante n dá-se o nome de comprimento da execução e ela deve ser um número inteiro tal que $n \geq 0$. No exemplo da Figura 2.3, um fragmento de execução é $\varrho = s_1\, B_1\, s_2\, B_2\, s_7\, C\, s_1$.

O fato dos autômatos abordados neste trabalho possuírem estados discretos *finitos* não limita o comprimento de sua execução também a um número finito. Dessa forma, se uma MEF permitir, uma execução pode ser infinita e esta é representada de acordo com a Equação 2.2.

$$\rho = s_0 e_1 s_1 e_2 s_2 e_3 \ldots \tag{2.2}$$

Por fim, um estado s_k é dito alcançável se para esse estado existe pelo menos uma execução partindo do estado inicial s_0 e que termine no estado s_k, ou seja, $\varrho = s_0 e_1 s_1 e_2 \dots e_k s_k$ (3). Em verificação formal é comum analisar a alcançabilidade de estados de MEFs a fim de determinar se existe uma execução que leve o autômato a alcançar um estado proibido ou indesejável.

O conjunto dos fragmentos de execuções também é de interesse no estudo dos autômatos finitos. Uma forma de visualizar esse conjunto é por meio do ordenamento das execuções de acordo com algum critério, como a ordem crescente de comprimento da execução (5), por exemplo. O ordenamento do conjunto de execuções do autômato da Figura 2.3 (omitindo as transições) pode ser observado na Figura 2.4.

s_1
$s_1 s_2$
$s_1 s_2 s_7$, $s_1 s_2 s_3$
$s_1 s_2 s_7 s_1$, $s_1 s_2 s_3 s_6$, $s_1 s_2 s_3 s_4$
$\vdots$

Figura 2.4: Ordenamento pelo comprimento da execução.

Ao invés do ordenamento pelo comprimento, o conjunto das execuções pode ser organizado também na forma de uma árvore. Essa organização é a base da lógica de árvore de computação, que será apresentada no capítulo 2. A Figura 2.5 ilustra o ordenamento da execução do autômato da Figura 2.3 na forma de uma árvore de computação.

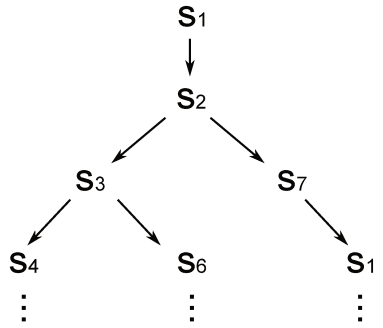


Figura 2.5: Fragmento de uma árvore de computação.

2.1.3 Não-determinismo

Ao falar da execução de autômatos é preciso falar de um tipo especial de comportamento associado às transições: o não-determinismo. Ao modelar um sistema, ao contrário do

exemplo do tanque de enchimento, nem sempre pode-se decidir quando o sistema vai tomar um caminho ou outro. Em situações em que para um dado estado de uma máquina uma entrada específica resulta em apenas uma única transição, diz-se que essa máquina é determinística.

Em uma máquina não-determinística existem várias possibilidades de transição a partir de uma única entrada para um dado estado. No diagrama de estados, a diferença entre uma máquina determinística e uma não-determinística é fácil de identificar: no autômato determinístico cada rótulo de transição corresponde a apenas uma seta de transição entre dois estados, enquanto no autômato não-determinístico um único rótulo pode ativar diversas transições para estados diferentes (4). Existem ainda as transições-ϵ que são utilizados quando a causa da transição é desconhecida. Elas representam eventos que causam mudança nos estados do sistema, mas não são observáveis. Assim, não se pode atribuir um evento específico a essa transição, mas sabe-se que ela existe e por isso ela recebe o símbolo ϵ (6).

Na Figura 2.6 observa-se que uma vez no estado s_1, o evento a pode levar o sistema para s_2 ou s_3, mas não se sabe a priori para qual estado ele vai. Em termos de computação da máquina, quando essa situação é atingida, a máquina cria cópias de si e continua a computação em todos os caminhos de forma paralela ou, simplesmente, pode-se interpretar que a máquina faz uma escolha arbitrária sobre qual caminho seguir. Se o símbolo ϵ é encontrado na máquina da Figura 2.6, a mesma coisa acontece, ou seja, sem receber nenhuma entrada, a máquina divide-se em duas cópias sendo que uma permanece no estado s_2 e a outra cópia segue a computação no estado s_4 (4).

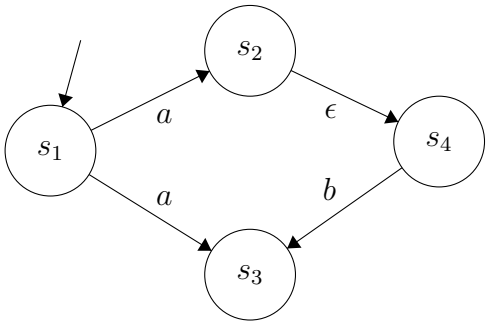


Figura 2.6: Exemplo de transições não determinísticas.

No caso do exemplo do tanque de enchimento da Figura 1.1, um operador que tivesse que interagir com o sistema poderia assumir os seguintes estados:

- Repor um novo recipiente vazio [R]
- Programar o volume do recipiente [P]
- Aguardar o sistema [A]
- Descarregar o recipiente [D]
- Encerrar o trabalho [H]

A máquina de estados que modela o comportamento do operador está ilustrada na Figura 2.7.

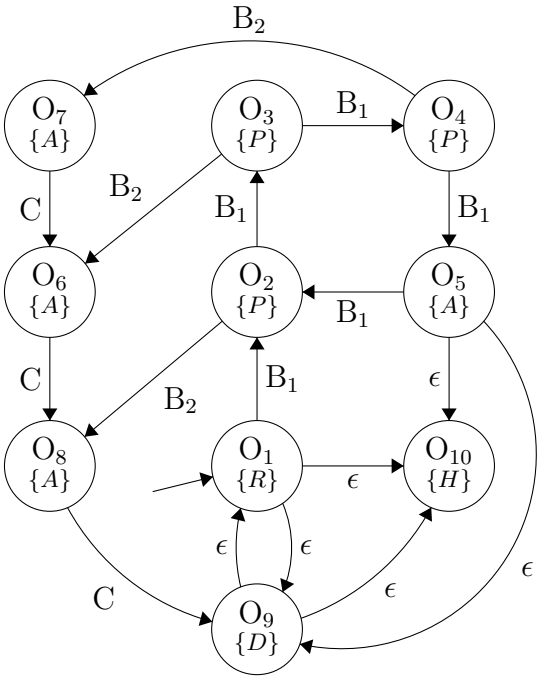


Figura 2.7: Autômato do operador com algumas transições não-determinísticas.

Esse diagrama de estados mostra que em certas ocasiões, o operador pode tomar decisões que não são determinísticas, não são previsíveis: no estado O_1, em que o operador carrega o novo tanque, ele pode pressionar B_1 a fim de programar o volume de enchimento (e transitar para o estado O_2) ou ele pode tomar a decisão de descarregar o tanque vazio (estado O_9) ou largar o tanque e encerrar o trabalho naquele dia (estado O_{10}). Não existem eventos específicos que façam o operador sair do estado O_1 para os estados O_9 ou O_{10} e portanto essas transições são não-determinísticas.

Essas transições podem ser usadas quando se desconhece o comportamento completo do processo que se quer modelar ou simplesmente não se deseja detalhar o funcionamento interno do sistema.

2.1.4 Sincronização

Uma característica importante das máquinas de estados finitos é a possibilidade de reunir diversos autômatos simples para modelar um sistema complexo ao invés de utilizar uma única máquina para esse fim. Assim como a modelagem de processos químicos é feita, dispondo dos modelos individuais dos equipamentos, que são ligados por correntes de processo, pequenos autômatos que se comunicam por meio de variáveis de sincronização formam redes que modelam processos mais complexos.

Neste trabalho serão destacadas a sincronização por passagem de mensagem e a sincronização por variáveis compartilhadas. No primeiro tipo, uma mensagem é enviada de um autômato para outro, em que a mensagem enviada é denotada por $!m$ e a mensagem recebida por $?m$. Suponha a necessidade de integrar o modelo do operador e do sistema de enchimento de modo que eles possam interagir. Então, quando o modelo do operador saísse do estado O_1 para o estado O_2 ao pressionar o botão B_1 o modelo do sistema efetuaria a transição de s_1 para s_2 por ser ativado também pelo acionamento de B1. A Figura 2.8 ilustra essa integração e como as variáveis são sincronizadas por envio de mensagem de uma máquina para a outra. Ressalta-se que as transições são efetuadas simultaneamente e que os símbolos "!" e "?" são indicativos do sentido de envio da mensagem.

A sincronização por compartilhamento de variáveis, por outro lado, supõe que os autômatos envolvidos manipulam variáveis de estado, de modo que atribuições podem ser feitas para essas variáveis. Se houvesse a necessidade de modelar o contador de voltas da turbina do sistema de enchimento a fim de detalhar seu funcionamento, haveria a necessidade de realizar a sincronização com o comportamento da máquina da Figura 2.3. Essa sincronização está ilustrada na Figura 2.9 e nela é possível observar que a máquina referente ao contador possui uma variável de estado c (que *não é* o evento C referente à contabilização de 1000 voltas) que acumula o valor do número de voltas da turbina (evento v) e a reinicia sempre que o número de voltas atinge 1000. A outra máquina realiza as transições de s_5 para s_6, s_6 para s_7 e s_7 para s_8 quando c atinge o valor de 1000.

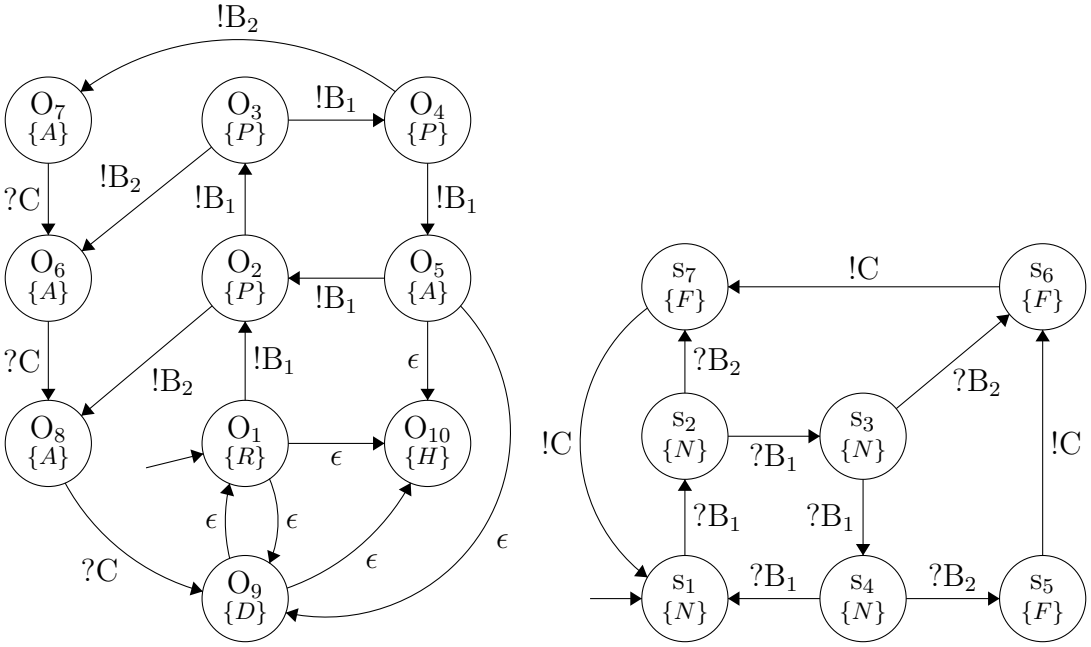


Figura 2.8: Sincronização do modelo do operador e do sistema de enchimento por passagem de mensagem.

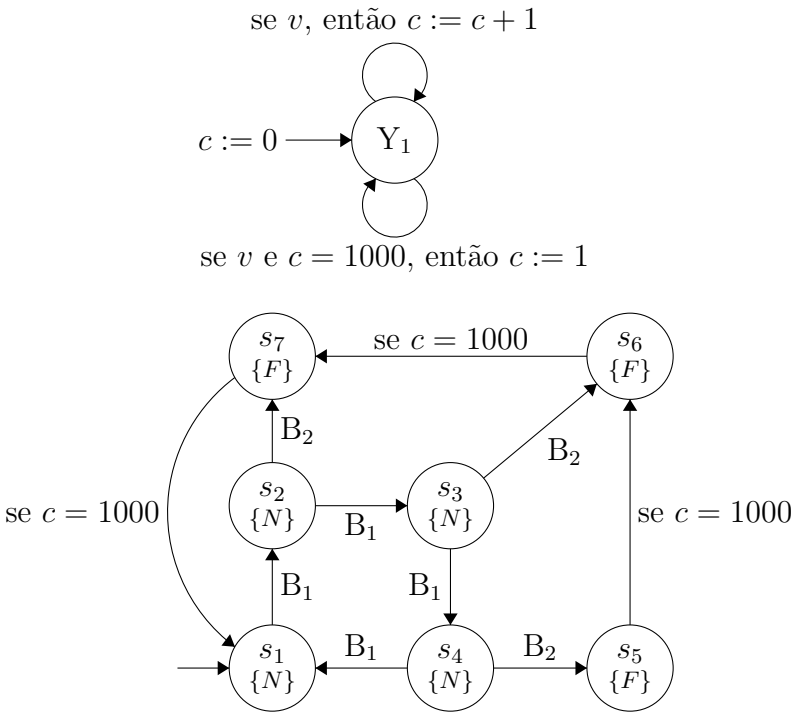


Figura 2.9: Máquinas sincronizadas por compartilhamento de variáveis (a máquina acima representa o contador de voltas da turbina).

As variáveis de estado são essenciais nas chamadas máquinas estendidas e as transições por compartilhamento de variáveis aparecem naturalmente nessas máquinas. Elas serão usadas nos capítulos seguintes para representar a dinâmica dos processos.

Ao desenvolver um *software* capaz de modelar processos químicos para fins de veri-

ficação é essencial que os modelos se comuniquem, visto que há um constante fluxo de informações entre os equipamentos e entre os sistemas de controle e o processo. Portanto, se $\mathcal{M}_i$ é o modelo de um equipamento i em um processo e $\mathcal{M}_c$ é o modelo de um controlador, o autômato formado pela rede de MEFs é dado pela composição de todos os modelos $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_c\}$. A verificação, então, é feita sobre $\mathcal{M}$.

Machado *et al* (7) discutem a importância do uso de redes de autômatos sincronizados para modelar o processo e os controladores, visto que diversos trabalhos na literatura verificam diagramas ladder de CLPs (Controladores Lógico Programáveis) sem considerar o processo, que resulta em um modelo não realístico do comportamento do sistema.

2.1.5 Máquinas estendidas

As máquinas estendidas (*extended-state machines*) incorporam mais informações ao modelo e permitem a construção de modelos mais complexos que as máquinas não-estendidas. Na notação de máquinas estendidas, os estados tradicionalmente representados nas outras máquinas por círculos são chamados de *modos* e estes modos têm sua descrição ampliada pela adição de variáveis de estado, como apresentado anteriormente durante a sincronização por compartilhamento de variáveis (8).

As transições dessas máquinas, em geral, possuem dois elementos: uma condição de guarda e expressões para atualização das variáveis de estado (representado esquematicamente como *guarda* $\rightarrow$ *atualização*). A condição de guarda é uma expressão booleana que uma vez satisfeita permite a transição entre os modos. Se essa condição sempre é satisfeita ela pode ser omitida e a transição é representada apenas pela expressão *atualização*. Se, por outro lado, a transição não atualizar nenhuma variável, então ela é omitida e a transição contará apenas com a condição de guarda (8).

Formalmente, uma máquina estendida é uma estrutura definida por $\mathcal{M} = (S,I,O,D,F,U,T)$ (9), tal que

- S é um conjunto de estados simbólicos ou modos
- I é um conjunto de entradas
- O é um conjunto de saídas
- D é o espaço n-dimensional $D_1 \times \ldots \times D_n$

- F é o conjunto de condições de guarda f_i de modo que $f_i : D \rightarrow \{0,1\}$
- U é o conjunto de atualizações u_i de modo que $u_i : D \rightarrow D$
- T é o conjunto de transições de modo que $T : S \times F \times I \rightarrow S \times U \times O$

Retomando o exemplo do sistema de enchimento automático é possível montar um novo modelo agora na forma de máquina estendida. Para isso, pode-se definir que o sistema possui dois modos: um modo de configuração (*conf*) e um modo em que ocorre efetivamente o enchimento (*enche*). Além disso, o sistema pode contar com três variáveis de estado: N, que armazena o número de pulsos do sensor e decresce conforme os pulsos são contados enquanto o sistema está enchendo; P que será igual a 1 se um pulso for detectado e 0 se nenhum pulso for detectado e V que será igual a 1 se a válvula estiver aberta e 0, fechada.

A máquina que modela o funcionamento desse sistema pode ser observada na Figura 2.10. Neste exemplo, especificamente, não existem variáveis de saída, que poderiam ser computadas por expressões em função das variáveis de estado e das entradas.

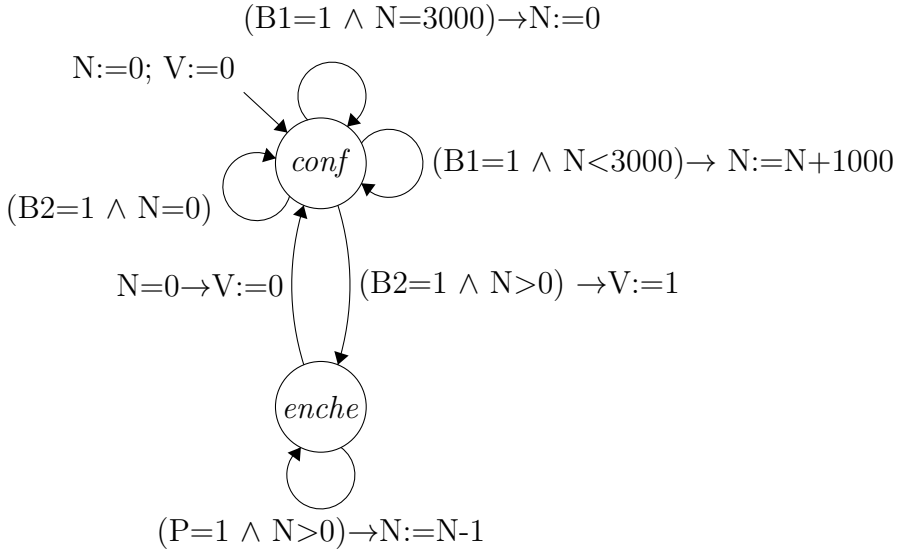


Figura 2.10: Modelo do sistema de enchimento automático como máquina estendida.

A máquina apresentada na Figura 2.10 engloba o comportamento das duas máquinas mostradas na Figura 2.9 e ainda assim consegue ser um modelo mais compacto em termos de diagrama de estados.

O fato de lidar com variáveis de estado, entrada e saída se assemelha em termos de notação à nomenclatura e estrutura usada na área de controle de processos no chamado

"espaço de estados". Como a construção de modelos dinâmicos complexos no espaço de estados a partir de sistemas de equações diferenciais é utilizada pela literatura há muitas décadas, no presente trabalho partiu-se da premissa que em função da semelhança de notação, seria possível, por analogia, construir um modelo em máquina estendida, partindo-se de simplificações e adequações das equações diferenciais.

De fato, (8) mostra que os sistemas dinâmicos em notação do espaço de estados é uma generalização para variáveis contínuas das máquinas estendidas.

As MEFs estendidas foram usadas como modelo de máquinas de estados na metodologia de síntese automática a ser apresentada no capítulo 3.

2.1.6 Outras classes de autômatos

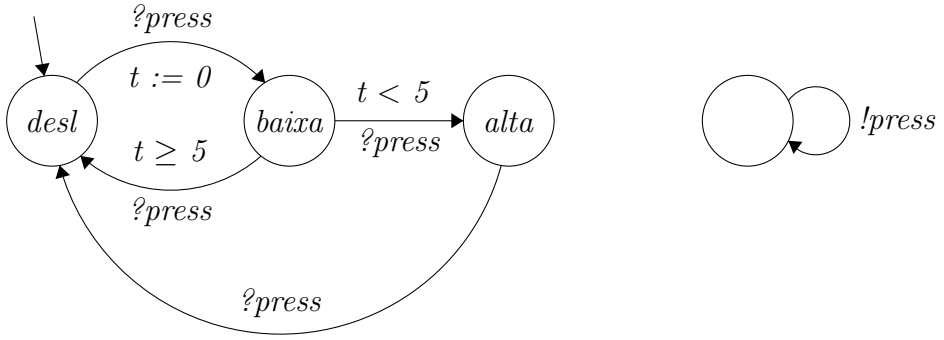
As classes de modelos apresentadas anteriormente constituem os modelos mais simples de máquinas de estados, modelando apenas variáveis inteiras ou estados discretos, mas muitos processos químicos apresentam simultaneamente características contínuas e discretas. Enquanto o comportamento contínuo é oriundo de fenômenos como a conservação de massa, energia e momento, os comportamentos discretos, ocorrem devido à descontinuidades e ações discretas de controle ou perturbações. Exemplos de controle discreto e/ou perturbações incluem mudanças operacionais planejadas, partidas e desligamentos de processos, mudanças na matéria-prima e intervenções manuais de operadores (10).

Os autômatos mais simples não conseguem lidar de maneira prática com variáveis contínuas. Assim, o tempo não pode ser modelado de maneira contínua. Alguns outros processos apresentam ainda características probabilísticas, como a probabilidade de um instrumento falhar ou um acidente ocorrer e essas informações também não são facilmente incorporadas ao modelo para verificação. Para contornar essas limitações, outras classes de autômatos foram criadas.

Os autômatos temporizados (*timed automata*) são sistemas de transição que levam em consideração relógios (*clocks*) com valores reais. As transições entre os estados contêm as chamadas condições de guarda, que de forma semelhante ao que foi apresentada para máquinas estendidas, são condições de restrições para a ocorrência da transição, mas baseada nos valores dos relógios. Os estados também podem conter invariantes (*invariants*) que indicam o tempo máximo que o sistema deve ficar no estado antes de uma transição ocorrer (11). Portanto, se o tempo é uma variável relevante, pode-se lançar mão de

autômatos temporizados.

A Figura 2.11 ilustra dois autômatos temporizados sincronizados que modelam um botão e uma lâmpada (12). A lâmpada (Figura 2.11a) possui três estados: *desl* (desligada), *baixa* (baixa intensidade luminosa) e *alta* (alta intensidade luminosa). Se o usuário pressiona o botão (Figura 2.11b) (ativando a sincronização com *?press*), então a lâmpada acende. Se o usuário pressionar o botão novamente, a lâmpada é desligada. Entretanto, se o usuário for rápido e pressionar o botão duas vezes, a lâmpada acende com a intensidade mais alta (entra no estado *alta*). O usuário pode pressionar o botão aleatoriamente (transição não-determinística) a qualquer momento ou não pressionar. A variável t representa o relógio e é usado para detectar se o usuário foi rápido ($t < 5$) ou lento ($t \geq 5$).



(a) Lâmpada (b) Botão

Figura 2.11: Modelo de uma lâmpada.

Outra importante classe de máquinas de estados finitos é a de autômatos probabilísticos. O modelo probabilístico mais simples é chamado de Cadeia de Markov em Tempo Discreto (*Discrete-time Markov Chain*), que especifica diretamente a probabilidade $\mathbf{P}(s,s')$ de um autômato efetuar uma transição de um estado s para o estado s' (13). A Figura 2.12 mostra um exemplo de uma cadeia de Markov utilizada para modelar uma moeda (14). Os valores associados a cada seta são as probabilidades de transição.

Processos de Decisão de Markov (*Markov Decision Processes*) são uma extensão das cadeias de Markov que permitem modelar também comportamentos não-determinísticos com os autômatos. Outra extensão são as Cadeias de Markov em Tempo Contínuo (*Continuos-time Markov Chains*) para as quais são atribuídas taxas $\mathbf{R}(s,s')$ de ocorrência de transições, em que a probabilidade de transição de s para s' em um tempo $t \in \mathbb{R}^{>0}$ é $1 - e^{\mathbf{R}(s,s')t}$ (13), ou seja, além de probabilidades também são incluídas informações contínuas sobre o tempo.

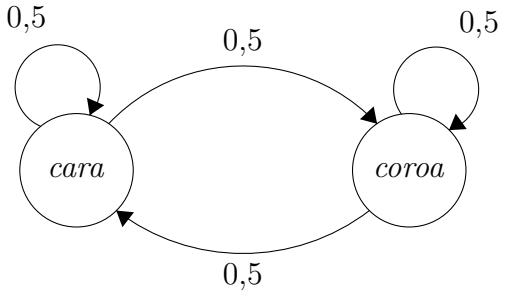


Figura 2.12: Autômato probabilístico de uma moeda. Adaptado com permissão a partir de (14) © 1986 IEEE.

Finalmente, as máquinas híbridas unem os estados discretos dos autômatos tradicionais com modelos contínuos baseados principalmente em equações diferenciais (15). Cada estado discreto possui um sistema de equações contínuas (algébricas ou diferenciais) que descreve a evolução das variáveis do sistema com o tempo.

A Figura 2.13 ilustra um autômato híbrido que modela um termostato. A variável x representa a temperatura. No estado Desligado, o aquecedor é desligado e a temperatura cai de acordo com a equação $\dot{x} = -0{,}1x$. No estado Ligado, o aquecedor é ligado e a temperatura aumenta conforme a equação $\dot{x} = 5 - 0{,}1x$. A condição de guarda $x < 19$ indica que o sistema pode passar para o estado ligado assim que a temperatura cai abaixo de 19 °C. A condição invariante $x \geq 18$ indica que o aquecedor será ligado obrigatoriamente se $x < 18$ (16).

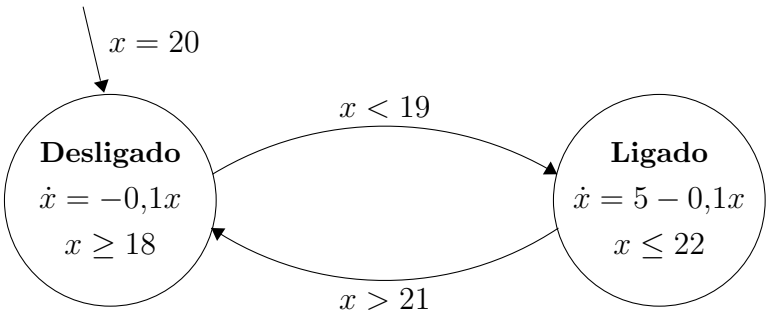


Figura 2.13: Autômato híbrido de um termostato. Adaptado com permissão a partir de (16).

A vantagem dos autômatos híbridos é a facilidade para expressar os modelos de equipamentos típicos da engenharia química. Ao invés de discretizar as variáveis, a discretização pode ser aplicada a cada equipamento, ou seja, as transições seriam equivalentes a passar os reagentes/produtos para outro equipamento, mantendo as equações contínuas que

descrevem o funcionamento dos equipamentos.

Especialmente processos em batelada podem se beneficiar da verificação com esse tipo de modelo. Um exemplo dessa aplicação é apresentado por Koutsoukos e Riley (17) em um sistema de produção de biodiesel. O autômato está representado na Figura 2.14, sem as condições de guarda.

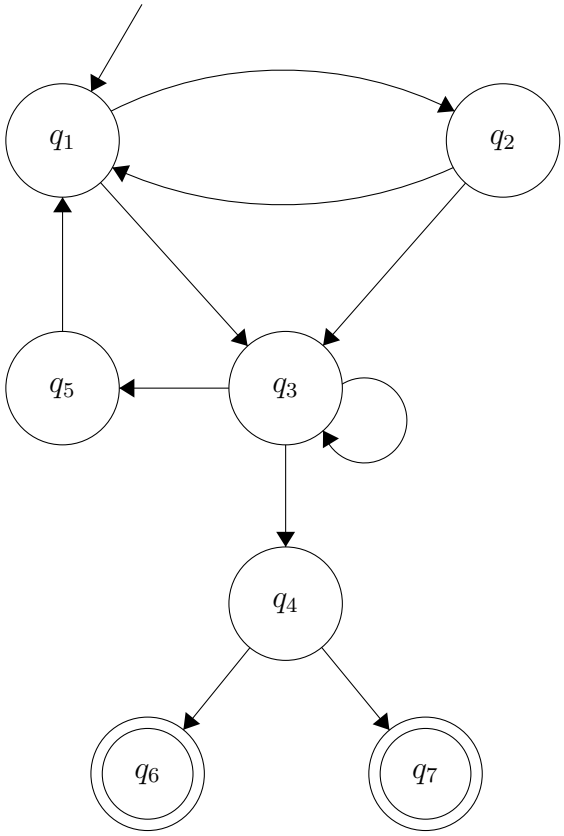


Figura 2.14: Modelo híbrido de um sistema de produção de biodiesel. Os estados representam q_1: aquecimento, q_2: resfriamento, q_3: decantação, q_4: análise, q_5: adição de metanol, q_6: falha na produção e q_7: produção bem sucedida. Adaptado de (17).

Nesse sistema, cada estado q_i representa uma etapa da produção do biodiesel com os respectivos sistemas de equações diferenciais que refletem a evolução da temperatura e concentrações de reagentes e produtos.

Autômatos híbridos podem ser usados, portanto, para modelar muitos processos industriais. No presente trabalho, essa classe de modelo de estados também foi considerada para modelagem dos processos químicos, mas dada as limitações atuais em termos de verificação formal (apenas um conjunto limitado de propriedades pode ser provada sobre esses modelos), optou-se por manter a modelagem com máquinas estendidas.

Uma vez que se tenha um modelo do processo é imprescindível para sua verificação especificar como ele deve (ou não) se comportar. Essa especificação normalmente é feita

em uma lógica temporal, assunto a ser tratado a seguir.

2.2 Lógica proposicional e temporal

A lógica é usada para representar propriedades do mundo real e para isso ela possui uma semântica e um mecanismo de raciocínio muito bem definidos. No entanto, ela é uma abstração da realidade, de modo que se queremos representar um ponto de vista mais complexo da realidade, um sistema lógico mais complexo deve ser usado (18).

A lógica proposicional é uma lógica muito simples, mas empregada largamente em engenharia (especialmente a elétrica em análise de circuitos digitais). Ela faz parte da lógica matemática que analisa as relações entre proposições, dando atenção à sua forma e não ao seu significado (19). Pode-se dizer que uma proposição é qualquer sentença declarativa que assuma valor verdadeiro ou falso (19). Por exemplo, considere as sentenças a seguir:

- A bomba de alimentação do reator está ligada
- Temperatura no tanque *flash* é de 90°C
- O nível do fundo da coluna de destilação está baixo e composição de topo está alta
- Fujam da nuvem de cloro!

As três primeiras são sentenças que podem assumir valores verdadeiro ou falso e por isso são consideradas proposições, mas a última, uma sentença imperativa, não. Como as duas primeiras fazem apenas uma afirmação, elas são chamadas de proposições simples ou atômicas, enquanto a terceira, que faz duas afirmações em uma única proposição (e poderia ser dividida em duas proposições atômicas) é classificada como proposição composta (19).

As proposições podem ser substituídas por símbolos (que neste trabalho são representados por P_1, P_2, ..., P_i) para abreviar as sentenças e esses símbolos podem ser unidos por conectivos para formar proposições compostas. O resumo dos conectivos e os respectivos significados está na Tabela 2.1

Assim, se P_1 = "nível do fundo da coluna de destilação está baixo" e P_2 = "composição de topo está alta", a proposição "O nível do fundo da coluna de destilação está baixo e composição de topo está alta" poderia ser reescrita como $P_1 \wedge P_2$.

Tabela 2.1: Conectivos utilizados em lógica proposicional.

Conectivo	**Significado**
$\neg$	Negação
$\wedge$	e (conjunção)
$\vee$	ou (disjunção)
$\Rightarrow$	Se... então...
$\Longleftrightarrow$	Se e somente se

Para avaliar se uma fórmula é verdadeira, utiliza-se uma tabela verdade. Cada conectivo da Tabela 2.1 possui sua própria tabela verdade. Por exemplo, o conectivo E tem sua tabela verdade apresentada na Tabela 2.2. Nota-se que a expressão final $P_1 \wedge P_2$ será verdade apenas se P_1 for verdadeiro e P_2 também. Expressões mais complexas podem ser avaliadas de maneira semelhante, analisando-se cada uma das partes que constituem a expressão em uma tabela verdade.

Tabela 2.2: Tabela verdade do conectivo E (conjunção).

P_1	P_2	$P_1 \wedge P_2$
V	V	V
V	F	F
F	V	F
F	F	F

A lógica, de uma forma geral, permite expressar problemas, propor teoremas e desenvolver provas, mas uma vez que este último pode se tornar um processo longo e tedioso, muito esforço tem sido aplicado para automatizar, na medida do possível, o processo de prova para várias lógicas (18).

Para muitas delas, o mecanismo de dedução pode ser inteiramente automatizado (a lógica proposicional é um exemplo). Em tais lógicas, pode-se formular uma hipótese (uma pergunta) e automatizar completamente o processo de chegar em uma resposta do tipo verdadeiro ou falso para a hipótese. Lógicas desse tipo são chamadas decidíveis. Em outras lógicas, mais expressivas, o processo pode nunca terminar e elas são chamadas indecidíveis (18).

A lógica da árvore de computação, diferentemente da lógica proposicional, é capaz de agregar informações sobre a dinâmica de um certo sistema. Por ser uma lógica decidível, a verificação de modelos utilizando a lógica de árvore de computação tem o potencial de ser um valiosa ferramenta ao provar propriedades sobre modelos de processos da engenharia química.

2.2.1 Lógica da árvore de computação

A lógica mais utilizada para expressar propriedades de sistemas em verificação de modelos é a Lógica da Árvore de Computação (LAC) ou em inglês *Computation Tree Logic* (será utilizada a abreviatura CTL neste trabalho, ao invés de LAC, pela popularidade da sigla mesmo em trabalhos escritos em português). Uma vez obtida a propriedade, um verificador de CTL automaticamente determina se essa propriedade é verdadeira ou não para o modelo do sistema.

Essa lógica se baseia na árvore de computação mostrada na seção 2.1.2, quando as execuções da máquina de estados finitos são explicitadas na forma de uma árvore. Um ramo (caminho) na árvore representa um certo comportamento do sistema.

As fórmulas mais simples em CTL consistem de apenas proposições atômicas. Assim, se P é uma proposição atômica, então a fórmula P será verdadeira para um estado s se $P \in L(s)$, ou seja, se P é uma fórmula válida para esse estado. Expressões mais complexas podem ser construídas com os conectivos da Tabela 2.1.

O que diferencia a lógica CTL da lógica proposicional clássica é o uso dos operadores **AX**, **EX**, **AU** e **EU**, onde **A** significa "para todos os caminhos na árvore de computação"; **E**, "existe algum caminho na árvore de computação"; **X** significa "próximo estado" e **U**, "até". **A** e **E** são quantificadores de caminhos e **X** e **U** são quantificadores de estados (2).

Com esses quantificadores é possível escrever fórmulas que englobam não apenas proposições atômicas em um estado, mas também de outros estados, expressando dessa maneira a evolução do sistema. A Tabela 2.3, resume o significado dos operadores.

Tabela 2.3: Operadores temporais e os respectivos significados.

Operador	Significado
AX f	Verdadeiro para o estado s se todos os estados sucessores imediatos satisfazem f
EX f	Verdadeiro para o estado s se f for verdadeiro em pelo menos um dos próximos estados
A$[f_1$ **U** $f_2]$	Verdadeiro para o estado s se a propriedade f_1 é verdadeira até o estado s e todos os caminhos posteriores a s a propriedade f_2 é verdadeira
E$[f_1$ **U** $f_2]$	Verdadeiro para o estado s se a propriedade f_1 é verdadeira até o estado s e algum caminho posterior a s a propriedade f_2 é verdadeira

A partir dos quantificadores de caminhos e estados apresentados, pode-se construir

outros quantificadores (2) (Tabela 2.4), cuja demonstração foi omitida neste trabalho.

Tabela 2.4: Outros operadores temporais.

Operador	Significado
EF f	Verdadeiro para o estado s se existe um estado em algum caminho começando em s que satisfaça f
EG f	Verdadeiro para o estado s se existe um caminho começando em s onde todos os estado desse caminho satisfazem f
AF f	Verdadeiro para o estado s se existe algum estado em todos os caminhos começando em s que satisfaçam f
AG f	Verdadeiro para o estado s se todos os estados de todos os caminhos satisfazem f

A Figura 2.15 ilustra algumas árvores de computação com a interpretação das expressões de alguns operadores temporais. Deve-se salientar que essas expressões são válidas para o estado s no "topo" das árvores ilustradas.

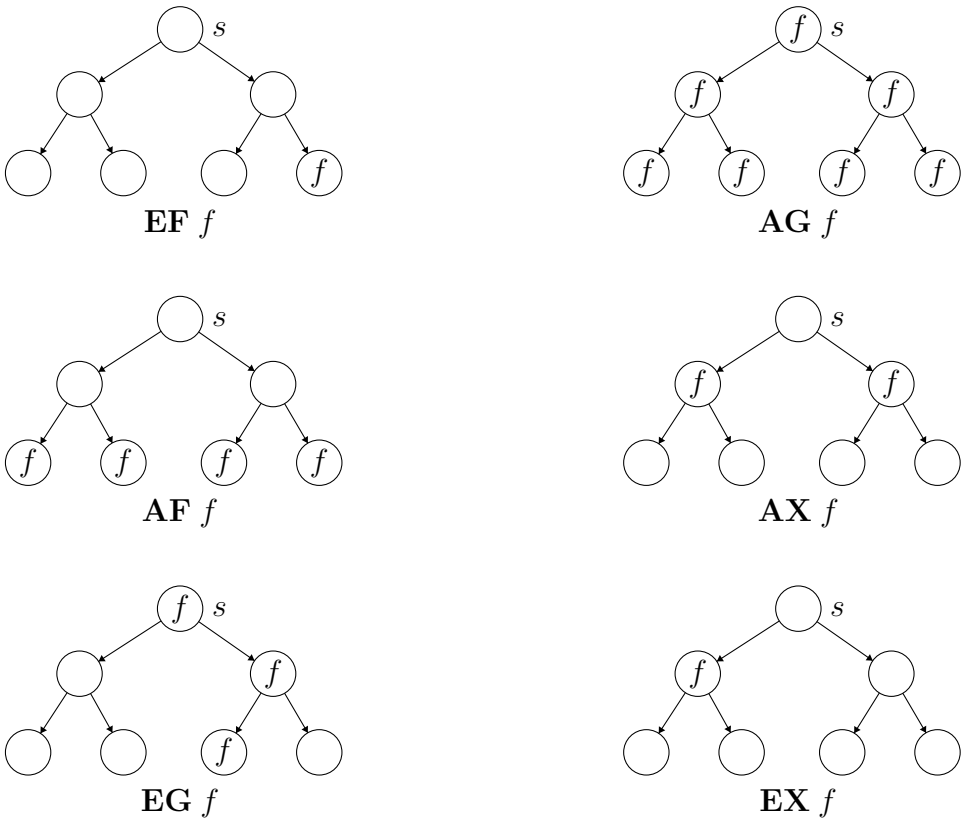


Figura 2.15: Árvores de computação ilustrando algumas fórmulas CTL. Adaptado com permissão a partir de (2).

Para exemplificar, se P_1 = "abertura da válvula de admissão de reagente" e P_2 = "tanque vazando", a expressão $P_1 \Rightarrow$ **EF** P_2 significa "se ocorrer a abertura da válvula

de admissão de reagente então existe pelo menos um estado em um caminho da árvore de computação que levará o tanque a vazar".

Fórmulas ainda mais complexas podem ser geradas pela combinação de operadores CTL. Por exemplo, **AG AF** f significa que para todos os estados s, todos os caminhos começando em s contém pelo menos um estado onde f é válido. Outra expressão utilizada é **EF EG** f, que significa que para algum estado no futuro, existe um caminho no qual f é verdadeiro para todos os estados desse caminho (2).

O verificador de modelos automaticamente determina se uma expressão CTL é satisfeita pelo modelo do sistema. O algoritmo processa a fórmula "de baixo para cima", ou seja, testa primeiro as subfórmulas menores antes das subfórmulas que as contém. Para cada operador CTL existe um algoritmo para determinar se a fórmula é válida e o verificador de modelos é resultado da combinação desses algoritmos junto com um algoritmo para produzir um contraexemplo em resposta a uma fórmula falsa.

Um contraexemplo é uma sequência de estados que demonstra que a fórmula é falsa. Essa característica do verificador é muito útil para identificar a causa de erros no sistema (2).

2.2.2 Outras lógicas temporais

Além da lógica CTL existem outras lógicas temporais específicas para autômatos temporizados e autômatos probabilísticos. Neste sentido, destacam-se a Lógica Temporizada da Árvore de Computação (TCTL, do inglês *Timed Computation Tree Logic*) e a Lógica Probabilística da Árvore de Computação (PCTL, do inglês *Probabilistic Computation Tree Logic*).

No TCTL os operadores da lógica CTL recebem informações quantitativas sobre o fluxo de tempo. Por exemplo, a fórmula $f_1 \mathbf{U}_{<2} f_2$ afirma que f_1 será verdadeiro até que f_2 seja verdadeiro e que isso ocorrerá dentro de duas unidades de tempo.

A notação usada para os operadores em TCTL é: $\mathbf{EF}_{\sim k} f$, $\mathbf{EG}_{\sim k} f$, $\mathbf{E}\ f_1\ \mathbf{U}_{\sim k}\ f_2$, $\mathbf{AF}_{\sim k} f$, $\mathbf{AG}_{\sim k} f$ e $\mathbf{A}\ f_1\ \mathbf{U}_{\sim k}\ f_2$. O símbolo $\sim$ representa um símbolo de comparação ($<\ ,\ \leq, =\ ,\ \geq, >$) e k é um número racional ($k \in \mathbb{Q}$) em unidades de tempo (5). Um exemplo de propriedade em TCTL é dado por

$$\mathbf{AG}\ (\mathit{problema} \Rightarrow \mathbf{AG}_{\leq 5}\ \mathit{alarme})$$

cujo significado é "se um problema ocorrer, então o alarme deverá soar imediatamente e o fará por pelo menos 5 unidades de tempo".

A lógica PCTL estende a lógica CTL ao acrescentar um novo quantificador de caminho **P**, que é um operador probabilístico. Com esse operador é possível analisar probabilidades sobre a validade de um expressão. Por exemplo, a expressão

$$enviar \Rightarrow \mathbf{P}_{\geq 0,95}[\mathbf{F}^{\leq 10} entrega]$$

significa "se uma mensagem for enviada, então a probabilidade de ser entregue dentro dos próximos 10 passos de tempo é de pelo menos 0,95"(20). Outro exemplo de uma propriedade em PCTL é

$$\mathbf{P}_{<0,4}[\neg falha_A \mathbf{U} falha_B]$$

cujo significado é "a probabilidade do componente B falhar antes do componente A é menor que 0,4".

2.2.3 Padrões de propriedades

Em geral, escrever propriedades adequadas ao sistema que se quer verificar não é tarefa trivial e sem uma propriedade que expresse corretamente uma especificação do sistema a verificação não terá sucesso em seu objetivo.

Alguns trabalhos tentam facilitar o problema da síntese da propriedade com *software* específicos que geram-na a partir de modelos (*templates*), assim, o usuário que queira testar o funcionamento do seu programa (lembrando que a verificação de modelos foi criada para verificação de programas) só precisa escolher algumas características da propriedade e ela é gerada automaticamente (21)(22).

Esses modelos de propriedades podem ser reaproveitados ou adaptados para diferentes classes de problemas. Esses modelos são chamados de padrões de propriedades e foram propostos originalmente por Dwyer *et al* (23), após coletar cerca de 500 exemplos de propriedades e classificá-los adequadamente. De acordo com esses autores, os padrões podem ser divididos em duas grandes classes: propriedades que dizem respeito à ocorrência ou não de certos eventos ou estados e propriedades em que a preocupação é com a ordem que os eventos ou estados ocorrem. Os padrões de ocorrência podem ser do tipo:

ausência, universalidade, existência ou existência limitada. Os padrões de ordem podem ser: precedência, resposta, precedência em cadeia e resposta em cadeia.

Cada padrão tem um escopo, que é, de forma simplificada, "o período de tempo" que a propriedade é válida. Existem cinco escopos conforme classificado por (23): **global** (execução completa do sistema), **antes** (execução da máquina até um certo evento ou estado), **entre** (trecho de execução entre dois eventos/estados específicos) e o **após/até** (semelhante ao **entre**, mas o último estado/evento pode ou não ocorrer).

Os padrões podem ser escritos para diversas lógicas temporais, mas como no presente trabalho será utilizada a lógica CTL, os padrões para CTL estão listados a seguir com uma breve explicação do seu significado (para a listagem completa e com mais exemplos, consultar (24)). Alguns dos padrões utilizam o operador **até fraco** (**weak until**) que compacta a notação de alguns padrões:

$\mathbf{A}[x\ \mathbf{W}\ y] = \neg\mathbf{E}[\neg y\ \mathbf{U}\ (\neg x \wedge \neg y)]$

Ausência: Um dado estado/evento P não ocorre dentro do respectivo escopo:

Escopo	**Propriedade**
Global	$\mathbf{AG}\ \neg P$
Antes do estado/evento R	$\mathbf{A}[(\neg P \vee \mathbf{AG}\ \neg R)\ \mathbf{W}\ R]$
Depois do estado/evento Q	$\mathbf{AG}\ (Q \rightarrow \mathbf{AG}\ \neg P)$
Entre Q e R	$\mathbf{AG}\ [Q \wedge \neg R \rightarrow \mathbf{A}((\neg P \vee \mathbf{AG}\ \neg R)\ \mathbf{W}\ R)]$
Após Q até R	$\mathbf{AG}\ [Q \wedge \neg R \rightarrow \mathbf{A}(\neg P\ \mathbf{W}\ R)]$

Existência: Um dado estado/evento P deve ocorrer dentro de um escopo:

Escopo	**Propriedade**
Global	**AF** P
Antes do estado/evento R	**A**[¬R **W** (P ∧ ¬R)]
Depois do estado/evento Q	**A**[¬Q **W** (Q ∧ **AF** P)]
Entre Q e R	**AG** (Q∧¬R → **A**[¬R **W** (P ∧¬R)])
Após Q até R	**AG** (Q∧¬R → **A**[¬R **U** (P ∧¬R)])

Existência limitada: Um dado estado/evento P deve ocorrer k vezes dentro do escopo (neste exemplo $k = 2$):

Escopo	**Propriedade**
Global	¬**EF** (¬P ∧ **EX** (P ∧ **EF** (¬P ∧ **EX** (P ∧ **EF** (¬P ∧ **EX** P)))))
Antes do estado/evento R	¬**E**[¬R **U** (¬P ∧ ¬R ∧ **EX**(P ∧ **E**[¬R **U** (¬P ∧ ¬R ∧ **EX**(P ∧ **E**[¬R **U** (¬P ∧ ¬R ∧ **EX**(P ∧ ¬R))])])])]
Depois do estado/evento Q	¬**E**[¬Q **U** (Q ∧ **EF** (¬P ∧ **EX** (P ∧ **EF** (¬P ∧ **EX** (P ∧ **EF** (¬P ∧ **EX** P))))))]
Entre Q e R	**AG** (Q → ¬**E**[¬R **U** (¬P ∧ ¬R ∧ **EX** (P ∧ **E**[¬R **U** (¬P ∧ ¬R ∧ **EX** (P ∧ **E**[¬R **U** (¬P ∧ ¬R ∧ **EX** (P ∧ ¬R ∧ **EF** R))])])])])
Após Q até R	**AG** (Q → ¬**E**[¬R **U** (¬P ∧ ¬R ∧ **EX** (P ∧ **E**[¬R **U** (¬P ∧ ¬R ∧ **EX** (P ∧ **E**[¬R **U** (¬P ∧ ¬R ∧ **EX** (P ∧ ¬R))])])])])

Universalidade: Um dado estado/evento P deve sempre ocorrer dentro do escopo:

Escopo	**Propriedade**
Global	**AG** P
Antes do estado/evento R	**A**[(P ∨ **AG** (¬R)) **W** R]
Depois do estado/evento Q	**AG** (Q → **AG** P)
Entre Q e R	**AG** (Q ∧ ¬R → **A**[(P ∨ **AG** ¬R) **W** R])
Após Q até R	**AG** (Q ∧ ¬R → **A**[P **W** R])

Precedência: Um estado/evento S deve sempre ser precedido por um estado/evento P dentro do escopo.

Escopo	**Propriedade**
Global	**A**[¬P **W** S]
Antes do estado/evento R	**A**[(¬P ∨ **AG** ¬R) **W** (S ∨ R)]
Depois do estado/evento Q	**A**[¬Q **W** (Q ∧ **A**[¬P **W** S])]
Entre Q e R	**AG** (Q ∧ ¬R → **A**[(¬P ∨ **AG** ¬R) **W** (S ∨ R)])
Após Q até R	**AG** (Q ∧ ¬R → **A**[¬P **W** (S ∨ R)])

Resposta: Um dado estado/evento S sempre deve ser seguido de um estado/evento P dentro do escopo (neste exemplo, S responde ao estado/evento P).

Escopo	**Propriedade**
Global	**AG** (P → **AF** S)
Antes do estado/evento R	**A**[((P → **A**[¬R **U** (S ∧ ¬R)]) ∨ **AG** ¬R) **W** R]
Depois do estado/evento Q	**A**[¬Q **W** (Q ∧ **AG** (P → **AF** S)]
Entre Q e R	**AG** (Q ∧ ¬R → **A**[((P → **A**[¬R **U** (S ∧ ¬R)]) ∨ **AG** ¬R) **W** R])
Após Q até R	**AG** (Q ∧ ¬R → **A**[(P → **A**[¬R **U** (S ∧ ¬R)]) **W** R])

Precedência em cadeia: Uma sequência de estados/eventos P_1, ...,P_n deve sempre preceder uma sequência de estados Q_1, ..., Q_n (neste exemplo, P precede S e T).

Escopo	**Propriedade**
Global	$\neg\mathbf{E}[\neg P\ \mathbf{U}\ (S \wedge \neg P \wedge \mathbf{EX}\ (\mathbf{EF}\ T))]$
Antes do estado/evento R	$\neg\mathbf{E}[(\neg P \wedge \neg R)\ \mathbf{U}\ (S \wedge \neg P \wedge \neg R \wedge \mathbf{EX}\ (\mathbf{E}[\neg R\ \mathbf{U}\ (T \wedge \neg R)]))]$
Depois do estado/evento Q	$\neg\mathbf{E}[\neg Q\ \mathbf{U}\ (Q \wedge \mathbf{E}[\neg P\ \mathbf{U}\ (S \wedge \neg P \wedge \mathbf{EX}\ (\mathbf{EF}\ T))])]$
Entre Q e R	$\mathbf{AG}\ (Q \rightarrow \neg\mathbf{E}[(\neg P \wedge \neg R)\ \mathbf{U}\ (S \wedge \neg P \wedge \neg R \wedge \mathbf{EX}\ (\mathbf{E}[\neg R\ \mathbf{U}\ (T \wedge \neg R \wedge \mathbf{EF}\ R)]))])$
Após Q até R	$\mathbf{AG}\ (Q \rightarrow \neg\mathbf{E}[(\neg P \wedge \neg R)\ \mathbf{U}\ (S \wedge \neg P \wedge \neg R \wedge \mathbf{EX}\ (\mathbf{E}[\neg R\ \mathbf{U}\ (T \wedge \neg R)]))])$

Resposta em cadeia: Uma sequência de estados/eventos P_1, ..., P_n deve sempre ser seguida de uma sequência de estados/eventos Q_1, ..., Q_n (neste exemplo, S e T respondem ao estado/evento P).

Escopo	**Propriedade**
Global	$\mathbf{AG}\ (P \rightarrow \mathbf{AF}\ (S \wedge \mathbf{AX}\ (\mathbf{AF}\ T)))$
Antes do estado/evento R	$\neg\mathbf{E}[\neg R\ \mathbf{U}\ (P \wedge \neg R \wedge (\mathbf{E}[\neg S\ \mathbf{U}\ R] \vee \mathbf{E}[\neg R\ \mathbf{U}\ (S \wedge \neg R \wedge \mathbf{EX}\ (\mathbf{E}[\neg T\ \mathbf{U}\ R]))]))]$
Depois do estado/evento Q	$\neg\mathbf{E}[\neg Q\ \mathbf{U}\ (Q \wedge \mathbf{EF}\ (P \wedge (\mathbf{EG}\ (\neg S) \vee \mathbf{EF}\ (S \wedge \mathbf{EX}\ (\mathbf{EG}\ \neg T)))))]$
Entre Q e R	$\mathbf{AG}\ (Q \rightarrow \neg\mathbf{E}[\neg R\ \mathbf{U}\ (P \wedge \neg R \wedge (\mathbf{E}[\neg S\ \mathbf{U}\ R] \vee \mathbf{E}[\neg R\ \mathbf{U}\ (S \wedge \neg R \wedge \mathbf{EX}\ (E[\neg T\ \mathbf{U}\ R]))]))])$
Após Q até R	$\mathbf{AG}\ (Q \rightarrow \neg\mathbf{E}[\neg R\ \mathbf{U}\ (P \wedge \neg R \wedge (\mathbf{E}[\neg S\ \mathbf{U}\ R] \vee \mathbf{EG}\ (\neg S \wedge \neg R) \vee \mathbf{E}[\neg R\ \mathbf{U}\ (S \wedge \neg R \wedge \mathbf{EX}\ (\mathbf{E}[\neg T\ \mathbf{U}\ R] \vee \mathbf{EG}\ (\neg T \wedge \neg R)))]))])$

Schnoebelen (5) classifica ainda algumas propriedades em termos de aplicações:

Alcançabilidade: são propriedades que indicam que certas situações **podem ser alcançadas**

***Safety*:** expressam que sob certas condições, **algo ruim nunca ocorre**

***Liveness*:** expressam que sob certas condições, **algo bom ocorrerá em algum momento**

***Fairness*:** indicam que sob certas condições, algo irá (ou não) **ocorrer de forma infinitamente frequente**

Nesta classificação as propriedades de alcançabilidade são as de maior interesse no presente trabalho visto que foram empregadas no estudo de caso do capítulo 4. Em termos computacionais, esse tipo de propriedade é a mais fácil de verificar (5) e se caracteriza por apresentar o operador **EF** P, em que P é a proposição ou estado. Uma forma interessante de expressar alcançabilidade é aninhando-se os operadores **AG** e **EF**, ou seja propriedades expressas como **AG EF** P indicam a possibilidade de partindo-se de qualquer estado alcançável do sistema, chegar em um estado onde P é válido. Esse tipo de informação pode ser usada na engenharia química para determinar se um estado seguro de um equipamento sempre é alcançável, independente da forma que ele está sendo operado.

Propriedades de *safety* (ou segurança), podem ser usadas na engenharia para localizar situações em que um processo possa se encontrar em condições perigosas, como combinações de pressão e temperatura altas, composições explosivas em um reator, etc. Propriedades de *safety* são caracterizadas pelo uso de operadores **AG** $\neg P$.

Para mais informações sobre as propriedades de *fairness* e *liveness*, recomenda-se consultar (5) e (25).

Alguns exemplos de uso dos padrões apresentados, dentro da engenharia química, são ilustrados a seguir para máquinas estendidas (as propriedades são construídas em função das variáveis de estado):

Exemplo 1: Um conjunto de estados do sistema nunca pode ocorrer simultaneamente. Por exemplo, isto poderia levar a uma situação de perigo, como uma temperatura (T) de 500 K com pressão (P) de 10 bar em um reator.

Propriedade: **AG** $\neg(\mathrm{T} = 500 \wedge \mathrm{P} = 10)$

Exemplo 2: Um sistema de controle sempre conseguirá levar a variável manipulada para o novo *setpoint*. Por exemplo, no controle de nível de um tanque, pode-se querer

saber se sempre que o *setpoint* for estabelecido em 50 cm se o sistema de controle será capaz de chegar nesse nível.

Propriedade: **AG** (SP = 50 → **AF** (L = 50))

Exemplo 3: Se um CLP, com programação em Ladder, comanda o acionamento de válvulas e bombas no enchimento e esvaziamento automático de um tanque, é possível verificar se o tanque será esvaziado (L = 0) apenas depois que o nível atingir o valor máximo (L = 100).

Propriedade: **A**[¬(L=0) **W** (L=100)]

Estabelecer as propriedades adequadas para a verificação de um modelo é tarefa difícil até mesmo para os cientistas da computação. Assim, a prática do uso da verificação de modelos e padrões de propriedades são boas formas de identificar propriedades interessantes para classes específicas de problemas dentro da engenharia.

A partir do momento que se tenha um modelo do processo e uma propriedade do sistema, tem-se os requisitos necessários para efetuar a verificação do modelo, conforme explicado na próxima seção.

2.3 Verificação formal

A dependência cada vez maior de computadores e *software* para armazenar e processar dados importantes trouxe a preocupação acerca das garantias de funcionamento desses sistemas. Quem pode garantir que ao fazer uma transferência bancária, o *software* do banco não vai retirar o dinheiro de uma conta e depositar na conta errada? Como garantir que um processador vai computar corretamente todas as operações matemáticas? Ou até mesmo, como garantir que o programa que controla o resfriamento de um reator não vai se comportar de maneira inesperada?

Questionamentos desse tipo surgem naturalmente para os engenheiros(as) da computação e de *software* porque falhas como essas podem afetar diretamente a vida de milhares de consumidores. Na verdade, algumas falhas costumam causar prejuízos financeiros (e até humanos) às empresas de tecnologia e por isso, antes de liberar um novo produto, o *hardware* e/ou o *software* é testado.

Erros sempre existirão em projetos de *hardware* ou *software*, porém a gravidade dessas falhas é que dará notoriedade e definirá o tamanho do impacto causado por *recalls* ou atualizações. Um exemplo clássico em engenharia de *hardware* é o caso do *bug* de ponto flutuante do processador Pentium II, que rendeu um prejuízo de aproximadamente US$475 milhões com *recalls* para a Intel (3).

A Figura 2.16 evidencia em que etapas do ciclo de vida de um *software* os erros são introduzidos e detectados e os respectivos custos para correção. Após o lançamento do *software* os custos de correção sobem rapidamente, por isso, quanto mais cedo um erro for encontrado, melhor.

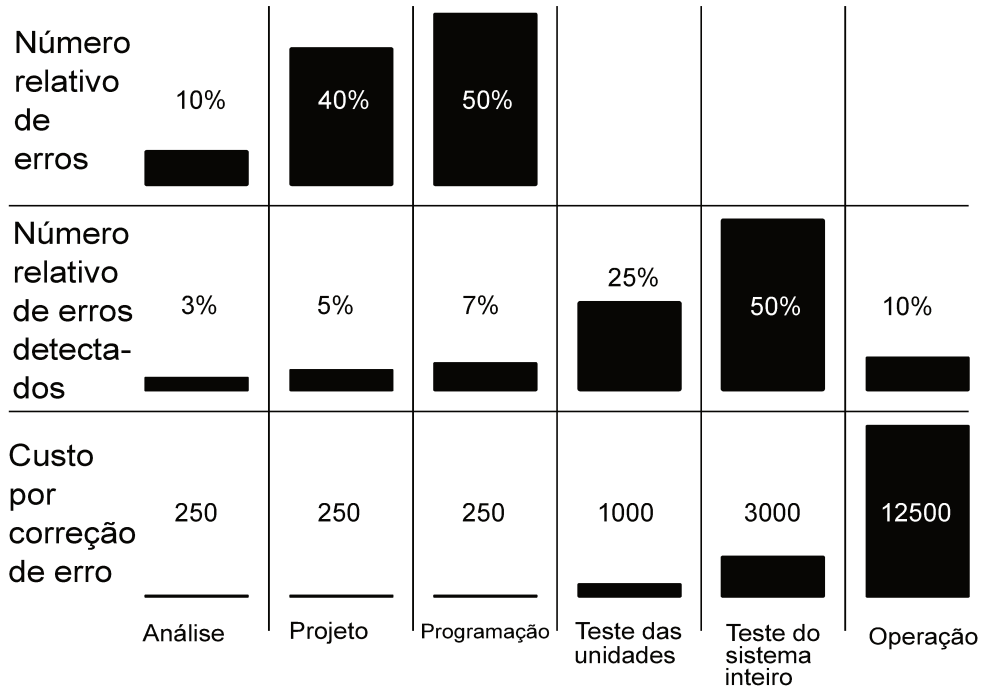


Figura 2.16: Ciclo de vida de um *software* e introdução de erros, detecção e custos de reparo. Adaptado com permissão de (26).

Surge então um problema: não é possível eliminar todos os erros, mas as falhas que restam no projeto podem causar grandes prejuízos. O que fazer então?

As duas formas mais comuns de avaliar o funcionamento de *software* é através de revisão por pares e por meio de testes. Na revisão por pares a inspeção do *software* é realizada por um time de engenheiros que preferencialmente não esteve envolvida no desenvolvimento. O *software* não é compilado e o código é analisado diretamente de forma estática. Os erros encontrados são imediatamente corrigidos no código.

O teste, ao contrário da revisão por pares, consiste em compilar e/ou rodar uma parte do código e forçar o programa a executar uma série de diferentes entradas que represen-

tariam o seu funcionamento normal. As saídas são analisadas com a documentação do projeto para saber se estão corretas. Porém, nenhuma das duas formas *garante* o correto funcionamento do sistema, pois ainda podem deixar passar erros de forma despercebida (3).

A verificação formal surge então como forma de provar *matematicamente* que um *software* ou *hardware* funciona de acordo com o seu projeto, ou seja, que esse *software* ou *hardware* atende algumas propriedades desejadas. A verificação formal fornece uma confiança extra sobre o funcionamento correto do sistema e complementa as análises feitas na revisão por pares e por teste, especialmente em sistemas mais críticos.

Hoje, diversas empresas de tecnologia e até órgãos públicos empregam a verificação formal como meio de avaliar a existência de falhas em projetos de *hardware* ou *software*. Dentre estas, estão a Microsoft, Intel, Cisco, IBM, NASA, NSA entre outras (27).

A verificação formal pode ser dividida em algumas técnicas, cujas abordagens são muito distintas. Destacam-se a prova de teoremas (*theorem proving*) e a verificação de modelos (*model checking*).

Na prova de teoremas, um programa (chamado de provador de teoremas) é usado para auxiliar deduções em lógica formal. Uma lógica formal compreende os seguintes componentes (28):

- Uma linguagem formal para expressar fórmulas
- Um conjunto de fórmulas válidas chamadas de axiomas
- Um conjunto de regras de inferência para derivar novas fórmulas a partir das fórmulas já existentes

Os axiomas formam um conjunto inicial de fórmulas válidas e com as regras de inferência é possível gerar, a partir dos axiomas, novas fórmulas também válidas. Repetindo esse procedimento, pode-se gerar uma sequência de fórmulas que é chamada de derivação ou dedução.

No contexto da prova de teoremas, "verificar uma fórmula" significa demonstrar que existe uma dedução para tal fórmula sob a lógica de um provador de teoremas. Uma desvantagem dessa abordagem é que um provador de teoremas, em geral, não consegue fazer a dedução sozinho, ou seja, ele exige que um usuário experiente em prova de teoremas

guie o provador. Portanto, um provador de teoremas *auxilia*, mas não *substitui* o ser humano ao fazer as deduções.

Ainda assim, o provador de teoremas é muito útil visto que consegue mecanicamente checar uma prova, ou seja, verificar se uma sequência de fórmulas de fato corresponde a uma derivação válida na prova do sistema (28). Essa capacidade é utilizada para verificar o funcionamento tanto de *software* quanto de *hardware*.

A verificação de modelos, por outro lado, é uma técnica que permite verificar um conjunto de propriedades acerca do comportamento de um sistema (com base em um modelo adequado) através da inspeção sistemática de todos os estados do sistema. A grande vantagem da verificação de modelos é que ela é uma técnica completamente automática (3), de modo que fornecidos um modelo, uma propriedade e um verificador, resta ao usuário "o apertar de um botão" para verificar o sistema. Assim, não é exigido do usuário um conhecimento tão aprofundado de verificação para utilizar a técnica, como acontece com a prova de teoremas.

Quando uma propriedade é violada, o verificador de modelos fornece um contraexemplo, que é a sequência de estados que levou o sistema a violar a propriedade. Essa informação é muito útil para identificar e corrigir as falhas.

Para utilizar a verificação de modelos são necessários os seguintes elementos:

- Um modelo do sistema
- Uma propriedade, em geral, definida em uma lógica
- Uma técnica de verificação, baseada no sistema de provas

Para fins de verificação, a maioria dos *model checkers* utiliza como modelo uma máquina de estados finitos, que é uma representação discreta dos estados que o sistema pode assumir. A propriedade, por sua vez, pode assumir várias formas: linguagens regulares, lógica clássica, lógica modal, autômatos (como os autômatos de Büchi), etc. Neste trabalho, foi usada lógica temporal para expressar as propriedades, que dá a liberdade para o usuário descrever uma característica do comportamento dinâmico. Finalmente, o algoritmo de verificação dependerá da lógica utilizada e deverá ser capaz de responder a seguinte pergunta:

Dado um modelo do sistema $\mathcal{M}$ e uma propriedade ϕ em uma lógical temporal, $\mathcal{M}$ satisfaz ϕ ($\mathcal{M} \models \phi$) ?

Exemplos de propriedades lógicas de um *software* são: "um programa nunca deve chegar numa situação de *deadlock* (travamento)" ou "uma requisição eventualmente deve ser atendida". Portanto, uma propriedade representa um comportamento desejado do sistema.

A motivação do presente trabalho é que o modelo $\mathcal{M}$ e a propriedade ϕ não precisam necessariamente estar relacionadas com um *software*, de modo que se o modelo representar um processo químico e a propriedade for, por exemplo, "se o nível do tanque estiver baixo, a resistência deve estar desligada", também pode-se fazer verificação desse processo.

De fato, a verificação de modelos tem sido aplicada nas últimas décadas para verificar o código Ladder de CLPs (Controlador Lógico Programável) conforme comprovam as referências (2), (29), (30), (31), (32), (33) e (34) (uma busca pelos termos "ladder logic" e "model checking" retorna 306 resultados no Google Acadêmico a partir de 1994, apesar da referência (2) mostrar que em 1992 já haviam trabalhos nessa área). O código é exaustivamente avaliado para todas as possíveis trajetórias que a execução do programa pode tomar a fim de descobrir se existe alguma sequência de ações que violem a propriedade. Essa rotina de avaliações substitui o teste manual do programa.

Deve-se destacar que, da mesma forma que o projeto, a otimização e a simulação de sistemas químicos fornecem resultados compatíveis com a qualidade dos modelos dados, as provas geradas pela verificação formal dependem da qualidade do *modelo* do sistema e da propriedade fornecidos. Assim, a verificação não prova uma propriedade do sistema físico se o modelo não corresponder à realidade. A verificação formal pode ser colocada ao lado das outras análises clássicas baseadas em modelos e que são utilizadas na engenharia química (Figura 2.17).

De posse de modelos de controladores, do processo e até do comportamento dos operadores, em teoria é possível para um engenheiro checar uma propriedade com um verificador. Porém, visto que na engenharia química dificilmente encontra-se alguém com os conhecimentos específicos, o presente trabalho teve por finalidade apresentar uma metodologia automática de conversão de modelos matemáticos para uso em verificação.

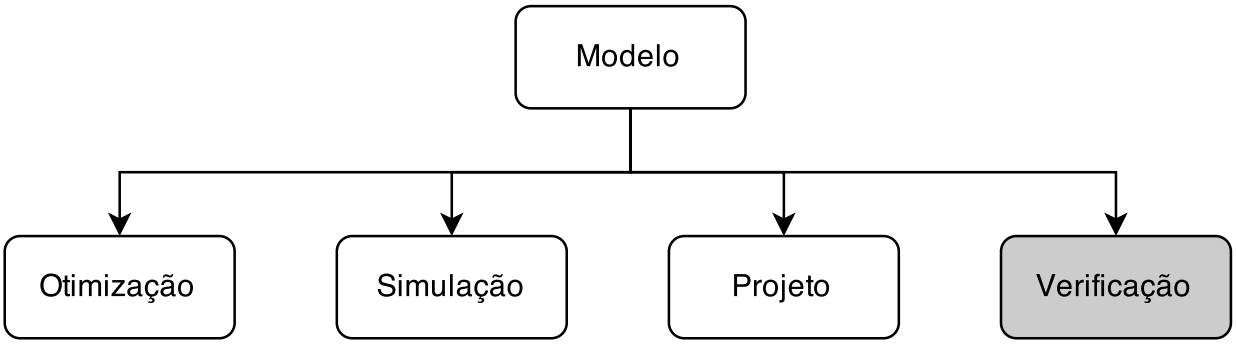


Figura 2.17: Possíveis aplicações para um modelo matemático na engenharia de química.

2.3.1 Uma visão geral da verificação de modelos

Em capítulos anteriores a verificação de modelos foi apresentada como uma abordagem automática de verificação com grande potencial de aplicação na engenharia química. Aqui a discussão sobre essa abordagem será aprofundada, com destaque para os principais *software* utilizados e as atuais aplicações da técnica em engenharia.

A verificação de modelos é uma técnica que pode ser aplicada a uma grande quantidade de problemas que possam ser modelados por autômatos finitos. Ela consiste de 3 partes: representação de um sistema por um autômato, representação de uma propriedade por uma fórmula lógica e um algoritmo de verificação de modelos (5).

Cada lógica temporal possui seus próprios algoritmos de verificação de modelos. O algoritmo para CTL, por exemplo, avalia a fórmula em CTL contra o comportamento infinito da máquina de estados para determinar se a propriedade temporal é satisfeita. O algoritmo possui garantias de encerrar em um tempo finito porque o sistema possui finitos estados, apesar de um caminho poder ser uma sequência infinita de transições. Se a resposta do verificador for "sim", o sistema está de acordo com a propriedade. Se a resposta é "não", o sistema viola a propriedade (32).

O processo de verificação é iterativo, ou seja, sempre que uma falha é encontrada em um modelo, ela deve ser corrigida e o sistema deve ser verificado novamente, pois não é possível garantir que a correção não tenha introduzido novas falhas no sistema (Figura 2.18). O sistema é considerado formalmente correto quando nenhuma outra falha (para uma dada propriedade) é encontrada.

A grande vantagem da verificação de modelos é a capacidade de *automaticamente* inspecionar todos os estados do modelo a fim de determinar se algum estado viola a

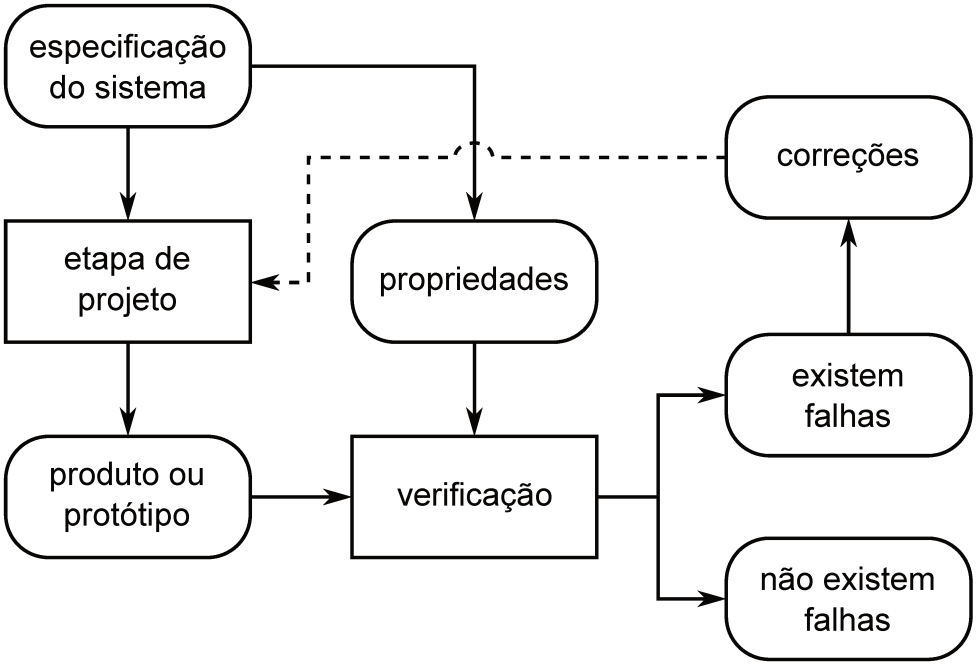


Figura 2.18: Esquema de verificação a posteriori. Adaptado de (3).

especificação. Isso inclui a capacidade de verificar modelos com mais de 10^{20} estados (32). Há relatos de verificação de modelos com um número de estados de até 10^{150} (35). Entretanto, esse número só é possível de ser tratado graças às técnicas que lidam com o problema de explosão de estados.

A explosão de estados pode ser mitigada ao codificar a máquina de estados de forma simbólica através de Diagramas de Decisão Binária Ordenado (*Ordered Binary Decision Diagrams*, OBDD). O OBDD reduz o número de estados explorando a redundância na árvore de computação ao combinar nós duplicados. Quando o modelo é tratado como um OBDD, a verificação de modelos passa a ser chamada de simbólica (32).

Além da verificação simbólica de modelos existem outras classes de verificação, como a verificação quantitativa, que como sugere o nome, engloba algoritmos que verificam propriedades quantitativas sobre o sistema (11). A verificação quantitativa é aplicada para autômatos temporizados e probabilísticos, cujos algoritmos são diferentes dos utilizados para lógica CTL.

Para todas essas classes de verificação de modelos existem *software* consolidados e disponíveis gratuitamente para a comunidade acadêmica. Neste sentido, quatro se destacam: SMV/NuSMV(36), UPPAAL (37), PRISM(38) e CheckMate (39).

O SMV (e a nova versão NuSMV) foram desenvolvidos para a lógica CTL e LTL, permitindo a construção e verificação de modelos de MEFs. O *software* é simples de

utilizar e graças ao tempo que vem sendo atualizado, possui muitas funcionalidades. A desvantagem é a ausência de interface gráfica, limitando o uso à linha de comando. Esse programa foi escolhido para ser usado no presente trabalho, em vista da linguagem simples e possibilidade de exportar resultados que podem ser lidos por outros programas como o Matlab.

O UPPAAL, por sua vez, é voltado para autômatos temporizados. Essa ferramenta permite a modelagem, verificação e simulação dos modelos, tanto por linha de comando, como através de uma interface gráfica. O algoritmo utilizado emprega verificação simbólica para reduzir a quantidade de estados e vários *software* foram desenvolvidos para fazer comunicação com o UPPAAL (37).

Como verificador probabilístico, o PRISM pode ser usado para modelar, analisar e verificar muitos tipos de modelos estocásticos como cadeias de Markov discretas ou contínuas e processos de decisão de Markov. Há suporte ainda a sistemas probabilísticos de tempo real, modelados como autômatos temporizados probabilísticos (11).

Por fim, o CheckMate é uma ferramenta utilizada para verificar autômatos híbridos criados no Stateflow (Matlab). A grande vantagem desse verificador é a possibilidade de utilizar a interface gráfica já existente do Stateflow e alguns poucos comandos do Matlab para executar a verificação. A desvantagem é que a lógica temporal utilizada é ACTL, um subconjunto da lógica CTL que usa como quantificador de caminhos apenas o operador **A**, o que restringe as propriedades que podem ser utilizadas (39).

Apesar de ferramentas adequadas serem necessárias para que o analista construa e modifique os modelos rapidamente, ainda são necessários *software* mais acessíveis para engenheiros químicos.

2.3.2 Aplicações em engenharia

Uma das principais aplicações da verificação de modelos em engenharia é na verificação de diagramas ladder, que são utilizados para programação de CLPs (Controladores Lógico Programáveis), graças à característica discreta e a possibilidade de representar os diagramas como autômatos.

A Figura 2.19 ilustra, como exemplo, um sistema em que os alarmes de nível alto e temperatura alta do tanque de armazenamento são ativados por um CLP e a sirene é silenciada (confirmada) pelo operador. Um diagrama ladder para esse processo está

apresentado na Figura 2.20 e o respectivo autômato é mostrado na Figura 2.21.

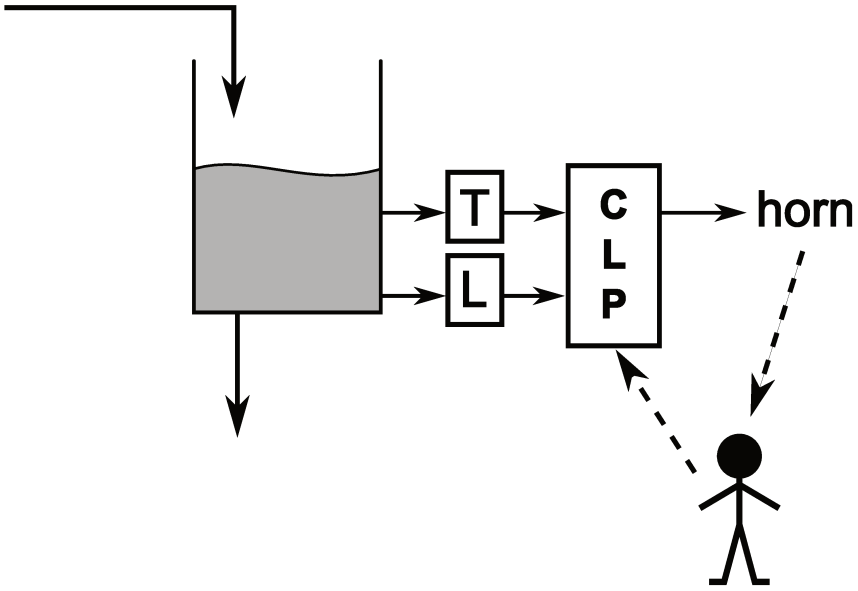


Figura 2.19: Sistema de confirmação de alarme. Adaptado com permissão de (2).

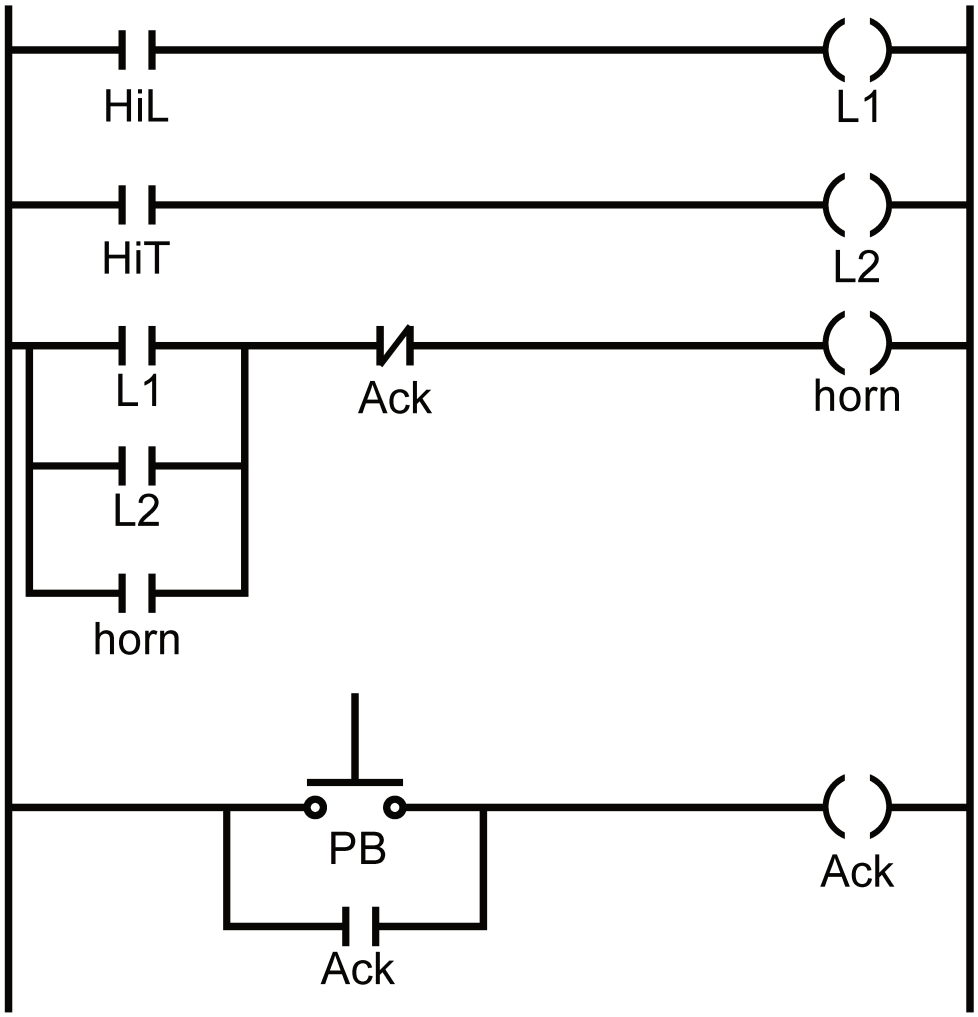


Figura 2.20: Exemplo de um diagrama ladder. Adaptado com permissão de (2).

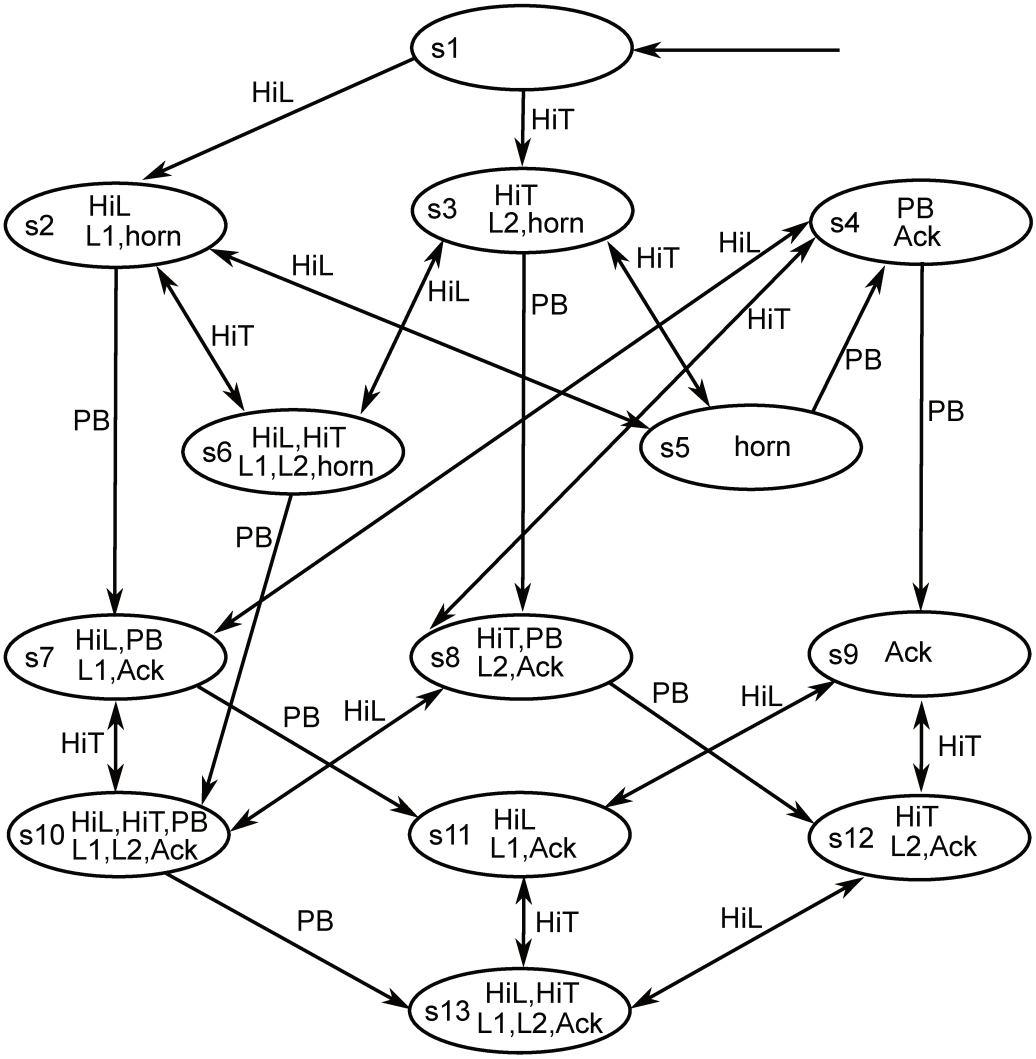


Figura 2.21: Autômato finito que representa o diagrama ladder da Figura 2.20. Adaptado com permissão de (2).

No diagrama ladder da Figura 2.20, os trilhos verticais (*power rails*) indicam os trilhos de energia, enquanto as linhas horizontais, representam possíveis fluxos de energia ou sinal. Vários símbolos para contatos e relés podem ser colocados nas linhas do ladder. Se os contatos apropriados são ativados, os relés são energizados e os correspondentes contatos são fechados. Por exemplo, fechando o contato de nível alto (HiL) na linha 1, provoca a ativação do relé L1 (também na linha 1) e causa o fechamento do contato L1 na linha 3 (2).

Diagramas ladder extensos são difíceis de ler e interpretar, especialmente quando são feitos por outra pessoa. Portanto, para saber se a programação feita corresponde ao comportamento desejado para o CLP são feitos testes manuais para as entradas e observa-se as respostas do ladder, ou seja, são feitas simulações.

Esse processo pode ser automatizado com a verificação de modelos. Se o engenheiro quer se certificar que a sirene sempre vai soar quando o nível alto é atingido, pode-se lançar mão da propriedade

$$\mathbf{AG}\ \mathrm{HiL} \Rightarrow \mathbf{AF}\ \mathrm{horn}$$

e com o modelo em MEF do diagrama ladder, deixar o trabalho de teste para um verificador. Se a propriedade for falsa, será retornado um contraexemplo com a sequência de estados (e consequentemente quais contatos foram acionados) que levaram o ladder a não acionar a sirene quando foi detectado nível alto.

Dois trabalhos demonstram o uso da verificação de modelos de códigos ladder no Brasil. Silva (29) e Sampaio (33) direcionaram seus trabalhos para a tradução automática de diagramas ladder para autômatos temporizados para fins de verificação com o UPPAAL. Assim, mesmo que um usuário tenha pouco conhecimento em verificação, eles demonstraram a possibilidade de verificação de diagramas ladder.

Uma outra direção que pode ser dada para a verificação de modelos é a possibilidade de "sintonia" do verificador para síntese de novos sistemas e procedimentos operacionais. O primeiro é sugerido por Barner *et al* (40), que ao invés de utilizar um verificador de modelos para encontrar falhas em *hardware* ou *software*, demonstrou que é possível utilizar contraexemplos para projetar sistemas que se comportam de acordo com uma dada propriedade. Analogamente, esse procedimento poderia ser adotado em processos químicos.

A síntese de procedimentos operacionais foi apresentada pelo trabalho de Margolis (41) para fornecer uma solução, em termos operacionais, para os operadores quando o processo se encontra em condições anormais de funcionamento. Ele desenvolveu também um linguagem (V-OPL) para reduzir o tempo e a dificuldade de síntese de modelos lógicos.

Outra área que merece destaque por ter se beneficiado da verificação de modelos foi a engenharia bioquímica e a biologia. Antoniotti *et al* (42) mostraram como a verificação de modelos híbridos pode ser utilizada para analisar rotas de reações bioquímicas. Eles desenvolveram dois *software*, o Simpathica e o XSSYS, para facilitar a verificação de propriedades como "eventualmente o sistema alcançará o estado estacionário e o nível de guanosina trifosfato será menor que um valor x". Vários outros programas para verificação de sistemas bioquímicos foram desenvolvidos desde então (43), ratificando a

técnica como uma importante ferramenta matemática na área e ilustrando a importância do desenvolvimento de *software* especializados.

Estes foram alguns exemplos de como a verificação de modelos está sendo aplicada em áreas não correlatas com a computação. Há muito a ser feito ainda para que a técnica ganhe espaço nas engenharias, mas sem dúvidas o potencial existe e deve ser explorado.

capítulo 3

Modelagem com máquinas de estados finitos

3.1 Introdução

A construção de autômatos finitos para modelar sistemas químicos não é uma tarefa trivial. Além do conhecimento do processo é preciso decidir acerca do tipo de autômato que será usado (alguns tipos estão mostrados na seção 2.1.6), propriedades e escolher um verificador adequado.

Sistemas modelados com MEFs podem ser construídos a partir de algumas informações qualitativas, mas suficientes, para descrever situações onde falhas podem ocorrer, seja em função de problemas no projeto, falta de informação, falhas dos operadores ou procedimentos mal planejados.

Assim como ocorre com outros tipos de modelos, não existe uma forma certa ou metodologia única para construir MEFs. Algumas tentativas de sistematizar esse processo estão descritas nas referências (44) e (34). Ambos os autores utilizaram autômatos que manipulam variáveis de estado para modelar o sistema físico, eventos, procedimentos operacionais e até a atuação de operadores humanos.

As variáveis de estado no modelo são discretas, normalmente inteiras ou booleanas. Assim, para modelar uma temperatura de um equipamento ou uma pressão é preciso realizar uma discretização. O domínio da variável pode ser discretizado em diversas faixas, utilizando como critério as especificações do sistema e/ou pontos de referência do sistema de controle e do sistema físico.

O exemplo mostrado na Figura 3.1 ilustra uma parte de um sistema de fundição de metal (44). Nesse sistema a prensa funciona como um elevador, que inicialmente está adjacente ao molde que contém metal líquido. Conforme a prensa desce, um líquido refrigerante é borrifado no metal, que esfria e começa a ganhar a forma do lingote.

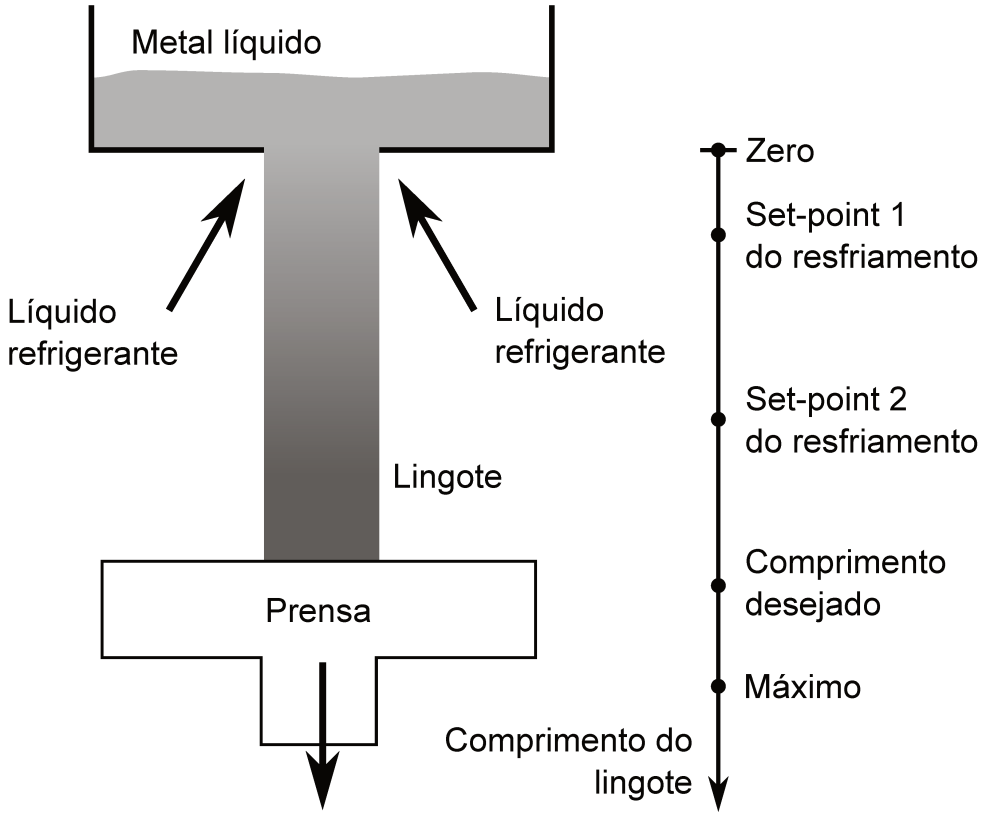


Figura 3.1: Exemplo de discretização do comprimento de um lingote (44).

Duas propriedades do lingote são:

- O lingote sai do estado de "moldagem" para "finalizado" quando o comprimento desejado é alcançado
- Iniciar a produção de um lingote implica que o comprimento desejado eventualmente será alcançado

Assim, em função das propriedades, o comprimento do lingote é uma variável importante na modelagem. O sistema se comporta de maneira diferente dependendo se o comprimento do lingote é maior ou menor que o comprimento desejado. Assim, o comprimento desejado pode ser considerado um ponto de discretização.

O sistema de controle também se comporta de maneira diferente em função do comprimento do lingote. Por exemplo, a taxa que o líquido de resfriamento é borrifado pode

variar de acordo com o comprimento. A Figura 3.1 mostra dois pontos onde o *set-point* da taxa de aspersão do líquido é alterada. Esses pontos também são considerados pontos de discretização.

O próprio lingote também pode sugerir pontos de discretização. Comprimento negativo, por exemplo, não possui sentido, por isso, o zero também foi incluído na discretização. No outro extremo, a prensa não consegue descer mais que o comprimento máximo, que, portanto, também foi escolhido como ponto de discretização.

Assim, conforme sugerido pela Figura 3.1, a variável comprimento pode ser discretizada em 6 partes:

0 : igual a zero

1 : entre o zero e o primeiro *set-point* da taxa de líquido de refrigeração

2 : entre os dois *set-points*

3 : entre o segundo *set-point* e o comprimento desejado

4 : igual ao comprimento desejado

5 : entre o comprimento desejado e o comprimento máximo

6 : igual ao comprimento máximo

Este exemplo mostra que as faixas de discretização podem ter diferentes comprimentos ou se tratar de apenas um ponto. A etapa de síntese dos modelos é muito importante e a escolha de uma discretização com muitas faixas pode levar a uma situação onde o tempo computacional para a verificação pode se tornar muito elevado ou a quantidade de memória pode se tornar insuficiente.

A discretização é apenas uma das etapas na construção do modelo para verificação. Uma proposta para a metodologia de síntese de um modelo para verificação consiste das seguintes etapas (44):

Coleta de informações: Reunião com os projetistas, operadores e engenheiros para coletar documentação sobre o sistema e sobre os comportamentos desejados e proibidos para esse sistema.

Análise do fluxograma e das especificações : Análise das informações coletadas na etapa anterior.

Seleção dos módulos: Baseado na análise da etapa anterior, deve-se selecionar os módulos apropriados a partir da biblioteca de módulos.

Criação das variáveis: Criação do esqueleto do modelo com o número e tipos corretos de variáveis dos módulos.

Conexão dos módulos: Definição das variáveis de entrada de cada módulo.

Construção do módulo principal: Nesta etapa o módulo principal é finalizado.

Definição das propriedades: As propriedades são extraídas da documentação ou de outras fontes e traduzida do português (ou inglês) para uma linguagem temporal usando as variáveis definidas no modelo.

Entretanto, a criação da biblioteca de módulos (que corresponderia aos modelos dos equipamentos, por exemplo) depende, hoje, exclusivamente do usuário, que deve ter conhecimentos em autômatos finitos e verificação. Como apontado por Milam (44), "*a prática industrial da verificação teria muito a ganhar se uma ferramenta computacional amigável voltada para engenheiros químicos fosse desenvolvida*".

3.2 Sistemas de eventos discretos

O presente trabalho tem por finalidade estabelecer uma metodologia simples para extração de uma máquina de estados finitos a partir de informações da dinâmica de processos, visando o uso em verificação formal. Entretanto, esta não é a primeira tentativa de estabelecer tal metodologia e uma "fonte" viável de informação do processo são as equações diferenciais.

Há uma razoável quantidade de trabalhos neste sentido, especialmente na área de Sistemas de Eventos Discretos (*Discrete Event Systems* ou DES). Modelos DES possuem estados discretos e pode-se dizer que eles são do tipo *event-driven*, ou seja, a evolução dos estados do modelo dependem inteiramente da ocorrência de eventos discretos assíncronos (ao contrário dos modelos sincronizados discutidos neste trabalho) ao longo do tempo (6). Em geral, a estratégia é baseada no particionamento do espaço de estados e transições são

atribuídas a fim de computar a trajetória que o sistema contínuo teria. O presente trabalho também utiliza uma estratégia de particionamento, mas a construção das transições e a existência de informação sobre o tempo de forma implícita o diferem dos trabalhos encontrados na literatura.

Um trabalho que merece destaque por estar envolvido com a engenharia química foi desenvolvido por Bloin (45), que desenvolveu um método computacional para gerar as máquinas de estados finitos (especificamente um sistema de eventos discretos) por abstração. Apesar dos estudos de caso estarem muito próximos da realidade da engenharia química (tanques de mistura), a matemática utilizada para construção das máquinas é difícil de implementar para a maioria dos engenheiros, tornando a metodologia proposta pouco prática para aplicação direta.

Mais recentemente, em sua tese de PhD, Thombre (46) também utilizou DES aplicado à análises de HAZOP e síntese de controladores discretos. A limitação de seu trabalho, assim como todo DES, é a falta de informação sobre o tempo, de modo que é possível acompanhar o que acontece com o sistema em termos de eventos, mas não é possível saber quando esses eventos ocorrem. Um exemplo extraído do trabalho de Thombre (Figura 3.2) ilustra bem esse problema: pode-se rastrear a trajetória do sistema ao sair do estado inicial até o final, mas não se sabe quanto tempo essa trajetória durou e essa informação muitas vezes é essencial em análise de sistemas de controle.

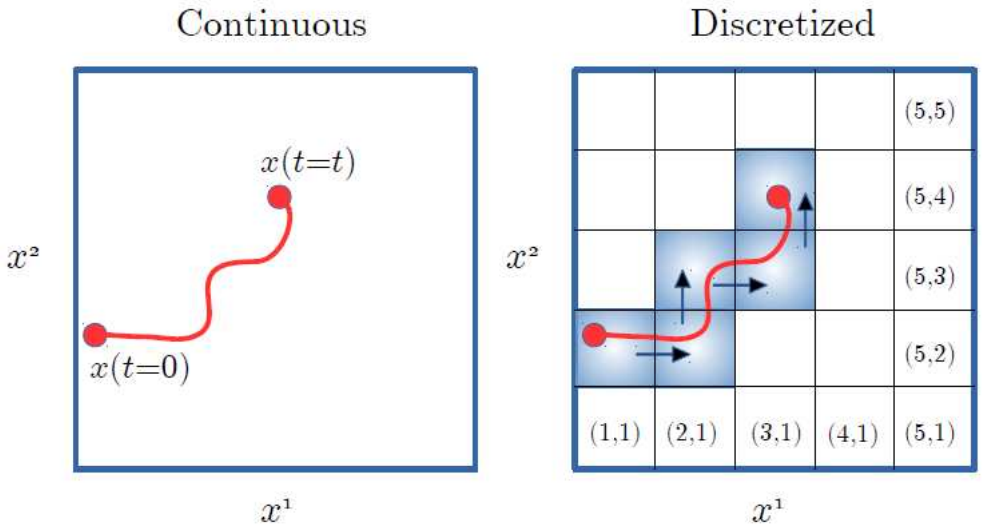


Figura 3.2: Discretização do espaço de estados do problema, com ilustração de uma trajetória no espaço contínuo e a mesma trajetória no espaço discretizado (46).

Na área de engenharia bioquímica, Radulescu *et al*(47) propõem uma aproximação

na forma de máquinas de estados finitos da dinâmica de rotas bioquímicas em diferentes escalas de tempo. Uma conclusão interessante desse trabalho é que os autores conseguiram reduzir o sistema contínuo a um autômato tão simples que este possuía um número de estados menor que o número de compostos que formavam o sistema original. Este trabalho reforça o que foi observado em diversos momentos ao longo da elaboração do presente trabalho: a engenharia bioquímica aparenta estar na vanguarda de muitos desafios na área de modelagem com máquinas de estados e em verificação formal. Entretanto, o trabalho de Radulescu *et al* não cita o uso desses modelos para fins de verificação formal.

Em um linha de trabalho um pouco diferente, Meenakshi *et al* (48) demonstraram a possibilidade de transformar modelos do Simulink diretamente para a linguagem do NuSMV. Os autores desenvolveram um programa que faz essa tradução do modelo de forma automática e aplicaram o *software* em problemas de aviônica. Este trabalho inspirou a construção da ferramenta de tradução de equações diferenciais para a linguagem do NuSMV que será apresentada no próximo capítulo.

Por fim, merece um especial destaque o desenvolvimento de uma metodologia automática para geração de máquinas de estados temporizadas a partir de EDOs (Equações Diferenciais Ordinárias) desenvolvido por Stursberg *et al* (49). Eles propuseram duas abordagens diferentes para modelagem, ambas baseadas em particionamento do espaço de estados, onde uma das abordagens utiliza integração numérica do sistema para decidir sobre as transições das máquinas de estados e os exemplos são específicos da engenharia química. O propósito dos modelos é a utilização em análise de alcançabilidade e estudo de sistemas de eventos discretos.

O uso de equações diferenciais para dar suporte à construção de máquinas de estados finitos foi utilizado até com o objetivo de auxiliar a visualização da trajetória de sistemas com dinâmica complexa, conforme relatado por Brian (50). Um exemplo de aplicação desse trabalho está mostrado na Figura 3.3 onde a máquina de estados indica todas as direções que o sistema é capaz de seguir.

Estes trabalhos são uma fração do que se observa na literatura a respeito de tentativas no sentido de sistematizar a obtenção de máquinas de estados diretamente a partir de equações diferenciais. Muitos deles, nem citam a possibilidade de verificação formal e utilizam os modelos resultantes para propósitos muito distintos.

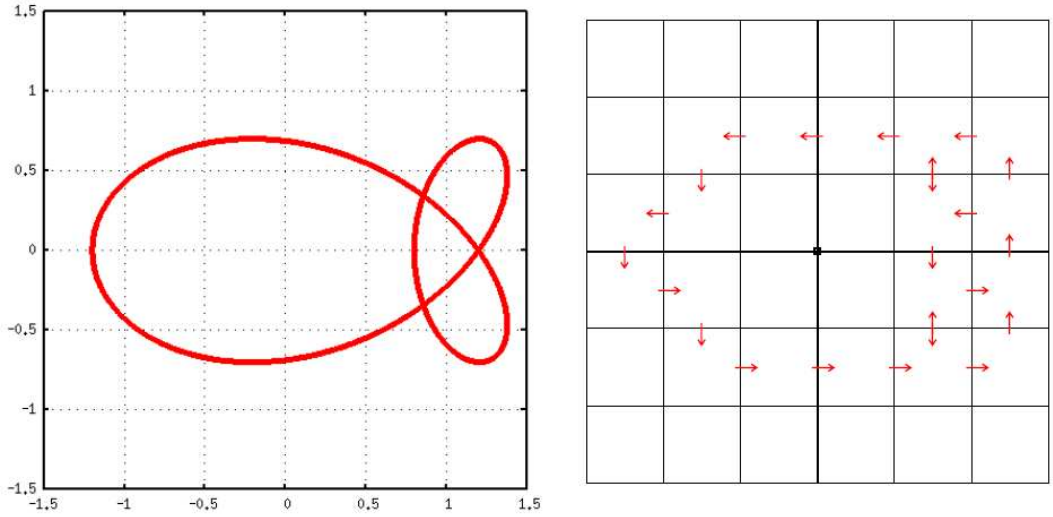


Figura 3.3: Máquina de estados (direita) auxiliando na visualização da trajetória de um sistema dinâmico complexo (esquerda). Cada quadrado representa um estado discreto do sistema.

Assim, o presente trabalho é mais um passo na tentativa de desenvolver automaticamente máquinas de estados a partir de modelos já estabelecidos de processos industriais. Apresenta-se aqui uma nova rota viável para modelagem discreta com MEFs: o método $\delta\pi$ para máquinas estendidas.

3.3 O método $\delta\pi$ para máquinas estendidas

A primeira (e mais importante) decisão tomada neste trabalho para estabelecer uma metodologia de modelagem foi a escolha da classe de modelo que seria usada, ou seja, se o modelo em máquina de estados seria na forma estendida, temporizada, probabilística ou híbrida.

À princípio, a escolha de modelos temporizados ou híbridos, por tratarem de tempo e variáveis contínuas, respectivamente, aparentou ser uma decisão sensata. Entretanto, tal escolha foi revisada ao vislumbrar a possibilidade de discretizar as variáveis contínuas e representar de modo implícito o tempo com máquinas estendidas. Assim, apesar da escolha de um modelo mais simples, com uma quantidade menor de informação explícita, o procedimento de discretização dos modelos dinâmicos (seja das variáveis ou do tempo) já é largamente consolidado na engenharia química para solução de equações diferenciais e acabou por facilitar a produção das máquinas de estados que representassem a dinâmica dos processos industriais.

O método desenvolvido e apresentado neste trabalho emprega uma estratégia de linearização por partes do espaço de estados do modelo dinâmico, onde a variável do problema é discretizada em duas variáveis: uma representa o trecho da linearização local (e isso permite acompanhar a taxa de mudança da variável) e a outra efetivamente indica o valor variável. O método se apoia ainda no uso de uma matriz onde é memorizada a dinâmica do processo e, portanto, dispensa o uso, a nível de máquina de estados, da função analítica do modelo.

As seções a seguir têm por objetivo demonstrar o método desenvolvido neste trabalho. Ao final do capítulo, uma série de exemplos demonstram a transformação de sistemas dinâmicos nas respectivas máquinas de estados.

3.4 Motivação e codificação das variáveis

A metodologia de construção das máquinas de estados estendidos a ser apresentada **foi inspirada**[1] no método de Euler para integração numérica (Equação 3.1).

[1] A inspiração serviu apenas para a estruturação das máquinas e transições. A solução das EDOs para construção das MEFs não é necessariamente feita pelo método de Euler.

$$y_{k+1} = y_k + \left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)_k \Delta t \tag{3.1}$$

O segundo termo do lado direito da igualdade é uma estimativa local da variação de y aproximada pela derivada da própria variável, ou seja, $\frac{dy}{dt}\Delta t = \Delta y$.

Portanto, para conhecer o próximo estado da variável y é necessário conhecer o estado atual e Δy. Dado que os autômatos estendidos lidam com variáveis de estado, é possível computar as variações de y atualizando seus valores por meio de atribuições nas transições do autômato. A Figura 3.4 ilustra como o cálculo do valor do novo estado de y poderia ser feito. A cada atualização da máquina de estados, uma transição seria responsável por atualizar o valor de y.

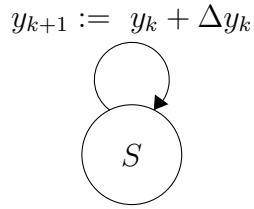


Figura 3.4: Implementação do método de Euler na transição de uma máquina de estados estendidos.

A abordagem mostrada na Figura 3.4 seria uma implementação direta do método de Euler em uma máquina de estados finitos e não seria diferente do que poderia ser feito em qualquer linguagem de programação.

Entretanto, as máquinas de estados estendidos podem não lidar com números reais e, além disso, o valor de Δy é função do estado atual da variável o que obrigaria a máquina de estados a calcular o valor da derivada em cada transição. Essas são fortes limitações, visto que essas máquinas, em geral, realizam operações muito simples nas transições.

Portanto, a estratégia precisa de adaptações para que as máquinas de estados reproduzam o comportamento de uma equação diferencial. A primeira adaptação proposta é a codificação das variáveis reais em um par de novas variáveis inteiras $\tilde{y}$ e $\tilde{y}_*$, chamadas respectivamente de **macroestado** e **microestado**. A variável $\tilde{y}$ representa o índice da faixa discretização dentro de um intervalo da variável original. Por exemplo, se um intervalo de pressão for definido entre 5 PSI e 10 PSI e esse intervalo for discretizado em 10 trechos, o macroestado $\tilde{y}$ representa o índice de cada uma dessas partes discretizadas, iniciando pelo índice 0. Na Figura 3.5 observa-se o significado de $\tilde{y}$.

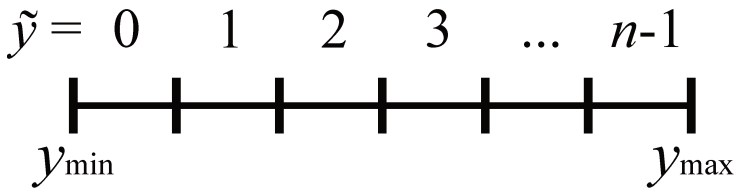


Figura 3.5: Significado do macroestado $\tilde{y}$.

Ainda no mesmo exemplo, se a pressão for codificada em $\tilde{y} = 0$ isso significa que ela possui um valor qualquer entre 5 PSI e 5,5 PSI (primeiro intervalo discretizado). Neste trabalho, para fins de simplificação na implementação da estratégia de conversão, o índice sempre assumirá o **valor inferior do intervalo real** em operações matemáticas, ou seja, $\tilde{y} = 0$ significaria 5 PSI em uma operação matemática. Todavia, não existe nenhum impedimento para o uso de outro valor representativo, como o valor médio ou o valor superior do intervalo.

O macroestado pode ser visto ainda como uma aproximação "grosseira" da variável y. Em tese é possível melhorar a qualidade da codificação aumentando o número de discretizações, mas isso tende a comprometer a verificação formal (fato observado de forma experimental durante o desenvolvimento do trabalho). A fim de reduzir o problema da complexidade que surge com o aumento do número de intervalos de discretização, foi proposto neste trabalho o uso do microestado $\tilde{y}_*$ que pode ser interpretado como uma discretização de um macroestado, ou seja, $\tilde{y}_*$ é um refinamento do valor de $\tilde{y}$. A interpretação da variável $\tilde{y}_*$ está ilustrada na Figura 3.6.

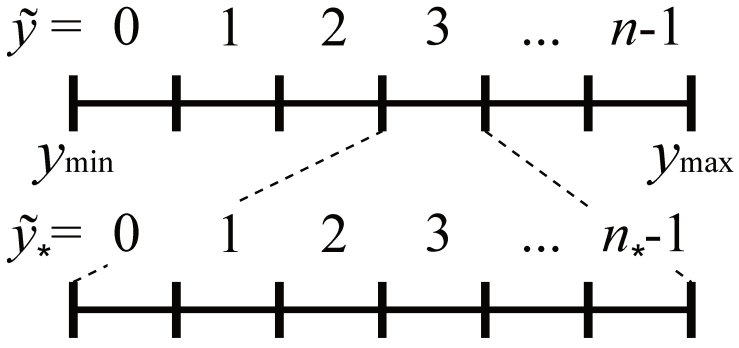


Figura 3.6: Significado do microestado $\tilde{y}_*$.

Dado que a variável real y é limitada num intervalo $[y_{\min}; y_{\max}]$ e que esse intervalo é discretizado em n partes iguais enumeradas por $[0; n-1]$, deduz-se a expressão que converte y em $\tilde{y}$ por uma interpolação. A expressão final para $\tilde{y}$ está apresentada na

Equação 3.2.

$$\tilde{y} = \left(\frac{y - y_{\min}}{y_{\max} - y_{\min}} \right) n \tag{3.2}$$

Posto que $\tilde{y}$ modela, em operações matemáticas, o valor mínimo de um intervalo discretizado de y e que $\tilde{y} \in \mathbb{Z}$, faz-se necessário acrescentar o operador $\lfloor\ \rfloor$ (menor inteiro) à Equação 3.2. Portanto, a expressão final de $\tilde{y}$ é mostrada na Equação 3.3.

$$\tilde{y} = \left\lfloor \left(\frac{y - y_{\min}}{y_{\max} - y_{\min}} \right) n \right\rfloor \tag{3.3}$$

Neste ponto deve-se fazer uma ressalva: se $y = y_{\max}$ pela Equação 3.3 teria-se $\tilde{y} = n$, que está fora do intervalo proposto inicialmente para $\tilde{y}$. Assim, conclui-se que o mapeamento de y em $\tilde{y}$ é válido apenas no intervalo aberto $[y_{\min}; y_{\max}[$.

De maneira semelhante, $\tilde{y}_*$ assume que um intervalo de $\tilde{y}$ é dividido em n_* partes iguais enumeradas por $[0; n_* - 1]$, $\tilde{y}_* \in \mathbb{Z}$ e que para operações matemáticas $\tilde{y}_*$ também usa como valor representativo o valor mínimo do intervalo da subdiscretização. A expressão para $\tilde{y}_*$ pode ser obtida por analogia: o termo entre parênteses da Equação 3.3 é a fração do intervalo $[y_{\min}; y_{\max}[$ em que y se encontra, portanto, pode-se escrever a Equação 3.4 em que o termo entre parênteses tem a mesma função para $\tilde{y}_*$ dentro de um intervalo de $\tilde{y}$.

$$\tilde{y}_* = \left\lfloor \left(\frac{y - y_{\min}}{y_{\max} - y_{\min}} n - \tilde{y} \right) n_* \right\rfloor \tag{3.4}$$

Portanto, as Equações 3.3 e 3.4 podem ser usadas para fazer a codificação inteira usada nas máquinas de estados estendidos neste trabalho. Resta ainda a necessidade de uma expressão que converta os valores codificados de volta para números reais e essa expressão pode ser vista na Equação 3.5, que computa as contribuições de $\tilde{y}$ e $\tilde{y}_*$ na composição de y. Assim como na Equação 3.2, as Equações 3.4 e 3.5 também foram deduzidas por interpolação.

$$y = \left(\frac{y_{\max} - y_{\min}}{n} \right) \left(\tilde{y} + \frac{\tilde{y}_*}{n_*} \right) + y_{\min} \tag{3.5}$$

Finalmente, a última operação de codificação é a do valor de Δy que é codificado em uma variável chamada de **salto** do microestado, $\Delta\tilde{y}_*$. A dedução da expressão resume-se ao cálculo de "quantos microestados estão contidos em Δy", conforme apresentado na Equação 3.6. O operador $\lfloor\ \rceil$ significa arredondamento para o inteiro mais próximo. O valor de $\Delta\tilde{y}_*$ é usado nas transições para atualizar os valores dos estados das máquinas.

$$\Delta\tilde{y}_* = \left\lfloor \frac{nn_*}{y_{\max} - y_{\min}} \Delta y \right\rceil \tag{3.6}$$

Para ilustrar os procedimentos de codificação, suponha que uma variável real definida dentro de um intervalo [0;4] é discretizada em 10 macroestados ($n = 10$) e 100 microestados ($n_* = 100$). Aplicando-se as Equações 3.3 e 3.4 para alguns valores dentro desse intervalo nota-se como a codificação evolui em termos de $\tilde{y}$ e $\tilde{y}_*$ (Tabela 3.1).

Tabela 3.1: Exemplo de codificação dentro do intervalo [0;4] com $n = 10$ e $n_* = 100$.

Real	$\tilde{y}$	$\tilde{y}_*$
0	0	0
0,1	0	25
0,2	0	50
0,3	0	75
0,4	1	0
0,5	1	25
⋮	⋮	⋮
3,99	9	97
3,999	9	99

A transformação de volta para números reais usando o par $\tilde{y}$ e $\tilde{y}_*$ e a Equação 3.5 resulta sempre em um valor subestimado de y. Por exemplo, para $\tilde{y} = 9$ e $\tilde{y}_* = 99$ (último item da Tabela 3.1) tem-se $y = 3{,}996$.

Uma vez que agora existe uma representação codificada das variáveis do problema físico é possível construir as máquinas de estados que efetivamente se comportarão como as equações diferenciais que se quer representar. O desenvolvimento dessas máquinas de estados estendidos está descrito na próxima seção.

3.5 Desenvolvimento e implementação das máquinas de estados

Na seção anterior foram expostos os motivos e os procedimentos para a codificação das variáveis reais em um par de números inteiros. Também foi apresentada uma máquina de estados finitos que teoricamente implementaria o método de Euler em uma transição (Figura 3.4). Nesta seção serão apresentadas as alterações do autômato da Figura 3.4 a fim de computar as atualizações das variáveis de estado $\tilde{y}$ e $\tilde{y}_*$ a partir de um método análogo ao método de Euler.

Ao longo do desenvolvimento deste trabalho surgiu a necessidade de representar as atualizações de $\tilde{y}$ e $\tilde{y}_*$ em autômatos distintos. Isso se deve à possibilidade de uma notação mais enxuta, com máquinas mais simples. Apesar disso, as máquinas ainda devem atender algumas características importantes:

1. As transições sempre devem respeitar os limites das variáveis $\tilde{y}$ e $\tilde{y}_*$.

2. Se a solução da equação diferencial levar a valores fora do intervalo especificado para uma variável, as transições forçarão a variável a ficar dentro do intervalo válido ao "travar" os valores no limite superior ou inferior.

3. Uma vez que a estratégia se baseia no método de Euler, ou seja, o tempo também existe de forma discreta, as transições não devem se limitar a estados adjacentes no *range* da variável, ou seja, se uma temperatura muda muito rápido pode ocorrer um salto diretamente de um macroestado 0 para o macroestado 5, por exemplo, sem ter passado pelos estados 1, 2, 3 e 4.

Essas regras básicas levaram ao desenvolvimento de quatro transições para a máquina responsável pela atualização do macroestado, chamada de $\mathcal{M}_\pi$ e quatro transições para atualização do microestado, com a máquina $\mathcal{M}_\delta$. Em função dessas máquinas o método foi nomeado como **método** $\delta\pi$. As transições estão descritas a seguir.

- **Transições da máquina $\mathcal{M}_\delta$**

  $\mathbf{T}_1$: Quando o próximo valor de $\tilde{y}_*$ for negativo (indicando transição de $\tilde{y}$ para um valor menor) e quando $\tilde{y}_*$ cair no primeiro microestado de algum macroestado ou quando o próximo valor calculado de $\tilde{y}$ for menor que zero (o que não é permitido), atribuir o valor $\tilde{y}_* = 0$.

  $\mathbf{T}_2$: Atribuir o valor $n_* - 1$ sempre que os próximos estados de $\tilde{y}$ e $\tilde{y}_*$ estiverem acima dos respectivos valores máximos permitidos (ou seja, o valor será "travado" no limite superior da variável).

  $\mathbf{T}_3$: Quando o próximo valor de $\tilde{y}_*$ for não-negativo e o novo valor de $\tilde{y}$ estiver dentro do intervalo codificado permitido, atribuir o valor de $\tilde{y}_*$ como sendo o resto da divisão entre $(\tilde{y}_* + \Delta\tilde{y}_*)$ e n_*.

T_4: Quando o próximo valor de $\tilde{y}_*$ for negativo e este valor não cair no primeiro microestado de algum macroestado, atribuir para $\tilde{y}_* = n_*+$ o resto da divisão entre $(\tilde{y}_* + \Delta\tilde{y}_*)$ e n_*.

- **Transições da máquina $\mathcal{M}_\pi$**

T_5: Quando o próximo valor calculado de y for maior que o intervalo codificado permitido, atribuir o valor máximo para $\tilde{y}$.

T_6: Se o novo valor de $\tilde{y}$ for menor que zero, atribuir $\tilde{y} = 0$.

T_7: Quando o próximo valor calculado de $\tilde{y}_*$ for não negativo e o novo valor calculado de $\tilde{y}$ for menor ou igual ao valor máximo permitido, calcular o novo valor de $\tilde{y}$ como sendo o valor atual + $(\tilde{y}_* + \Delta\tilde{y}_*)/n_*$.

T_8: Quando o próximo valor calculado de $\tilde{y}_*$ for negativo e o novo valor calculado de $\tilde{y}$ for menor ou igual ao valor máximo permitido, calcular o novo valor de $\tilde{y}$ como sendo o valor atual + $(\tilde{y}_* + \Delta\tilde{y}_* - n_* - 1)/n_*$.

As transições T_1, T_2, T_5 e T_6 têm o objetivo de garantir que $\tilde{y}_*$ e $\tilde{y}$ sempre fiquem dentro do intervalo codificado permitido ($[0; n_* - 1]$ e $[0; n - 1]$, respectivamente) "travando" os valores nos limites superior ou inferior quando estes são violados e as transições T_3, T_4, T_7 e T_8 atualizam $\tilde{y}_*$ e $\tilde{y}$ quando os novos valores respeitam os intervalos codificados permitidos. O diagrama de estados de $\mathcal{M}_\delta$ e $\mathcal{M}_\pi$ pode ser observado na Figura 3.7.

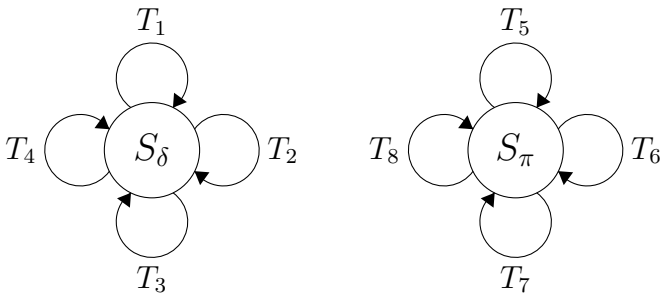


Figura 3.7: Diagrama de estados das máquinas $\mathcal{M}_\delta$ e $\mathcal{M}_\pi$.

Uma vez que as variáveis codificadas não são números no domínio dos reais, as operações matemáticas realizadas nas transições são feitas **sobre números inteiros sempre com arredondamento para baixo**. Além disso, para fins de notação, a operação a mod b refere-se à operação "resto de divisão", assim, 7 mod 3 = 1, por exemplo. Assim, matematicamente, as transições assumem os seguintes formatos:

- **Transições da máquina $\mathcal{M}_\delta$**

  $\mathbf{T}_1$: $(\tilde{y}_* + \Delta\tilde{y}_*) < 0 \wedge [(\tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_* - n_* + 1)/n_*) < 0 \vee (\tilde{y}_* + \Delta\tilde{y}_*) \bmod n_* = 0] \rightarrow \tilde{y}_* = 0$

  $\mathbf{T}_2$: $(\tilde{y}_* + \Delta\tilde{y}_*) > (n_* - 1) \wedge [\tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_*)/n_*] > (n-1) \rightarrow \tilde{y}_* = n_* - 1$

  $\mathbf{T}_3$: $(\tilde{y}_* + \Delta\tilde{y}_*) \geq 0 \wedge [\tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_*)/n_*] \leq (n-1) \rightarrow \tilde{y}_* = (\tilde{y}_* + \Delta\tilde{y}_*) \bmod n_*$

  $\mathbf{T}_4$: $(\tilde{y}_* + \Delta\tilde{y}_*) < 0 \wedge (\tilde{y}_* + \Delta\tilde{y}_*) \bmod n_* \neq 0 \rightarrow \tilde{y}_* = n_* + (\tilde{y}_* + \Delta\tilde{y}_*) \bmod n_*$

- **Transições da máquina $\mathcal{M}_\pi$**

  $\mathbf{T}_5$: $[\tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_*)/n_*] > (n-1) \rightarrow \tilde{y} = n - 1$

  $\mathbf{T}_6$: $[\tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_* - n_* + 1)/n_*] < 0 \rightarrow \tilde{y} = 0$

  $\mathbf{T}_7$: $(\tilde{y}_* + \Delta\tilde{y}_*) \geq 0 \wedge [\tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_*)/n_*] \leq (n-1) \rightarrow \tilde{y} = \tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_*)/n_*$

  $\mathbf{T}_8$: $(\tilde{y}_* + \Delta\tilde{y}_*) < 0 \wedge [\tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_* - n_* + 1)/n_*] \geq 0 \rightarrow \tilde{y} = \tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_* - n_* + 1)/n_*$

Uma vantagem dessa formulação é que a transformação de equações diferencias para máquina de estados se torna um procedimento sistemático, bastando preencher as equações listadas para as máquinas $\mathcal{M}_\delta$ e $\mathcal{M}_\pi$ com algumas informações como n, n_* e durante as transições obter os valores de $\Delta\tilde{y}_*$, para que elas já adquiram um comportamento semelhante ao da respectiva equação diferencial.

Observa-se que as transições da máquina $\mathcal{M}_\delta$ têm guarda/atribuição na forma $f_1(\tilde{y},\tilde{y}_*,\Delta\tilde{y}_*) \rightarrow \tilde{y}_* = f_2(\tilde{y},\tilde{y}_*,\Delta\tilde{y}_*)$, onde f_1 é uma fórmula composta e f_2 é uma expressão para o novo valor de $\tilde{y}_*$. De maneira análoga, a máquina $\mathcal{M}_\pi$ possui transições com a forma $f_2(\tilde{y},\tilde{y}_*,\Delta\tilde{y}_*) \rightarrow \tilde{y} = f_3(\tilde{y},\tilde{y}_*,\Delta\tilde{y}_*)$. Como f_1, f_2, f_3 e f_4 dependem de $\tilde{y}$ e $\tilde{y}_*$, as máquinas $\mathcal{M}_\delta$ e $\mathcal{M}_\pi$ são sincronizadas por compartilhamento de variáveis. Uma interpretação de como essa sincronização funciona está ilustrada na Figura 3.8.

Suponha que as máquinas de estados da Figura 3.7 foram usadas para se comportar conforme uma dada equação diferencial e para isso foram utilizados $n = 6$ e $n_* = 5$ e ao resolver a equação diferencial, conclui-se que $\Delta\tilde{y}_* = [+5, +2, +1, -1, -3, -5]$ (cada salto neste vetor corresponde a um macroestado da variável y). Suponha ainda que a condição inicial é $\tilde{y} = 0$ e $\tilde{y}_* = 3$. A esquematização da evolução dessa máquina de estados está apresentada na Figura 3.9. Nesta figura, observa-se que dentro de cada macroestado a máquina se comporta de forma linear até atingir um estado estacionário em que a máquina fica alternando entre os macroestados 2 e 3 indefinidamente.

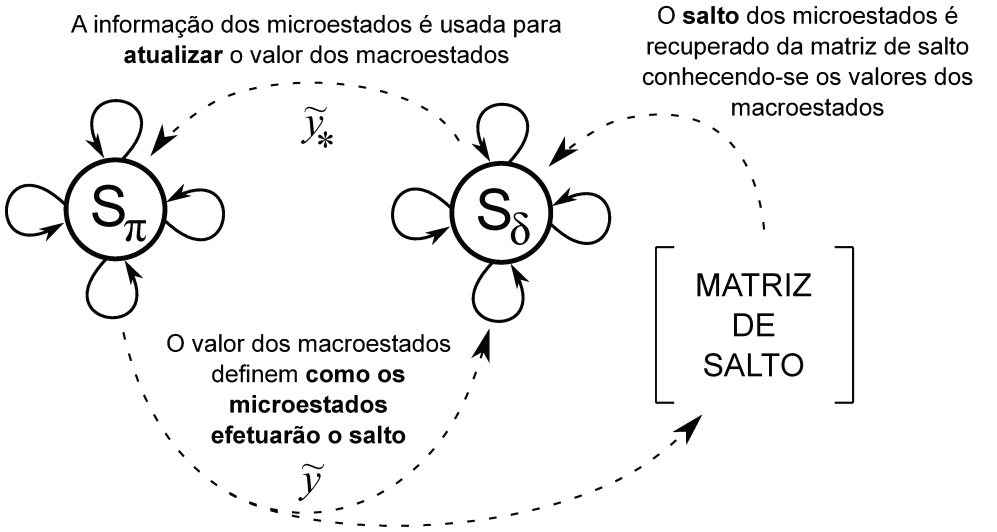


Figura 3.8: Interpretação da sincronização das máquinas $\mathcal{M}_\delta$ e $\mathcal{M}_\pi$.

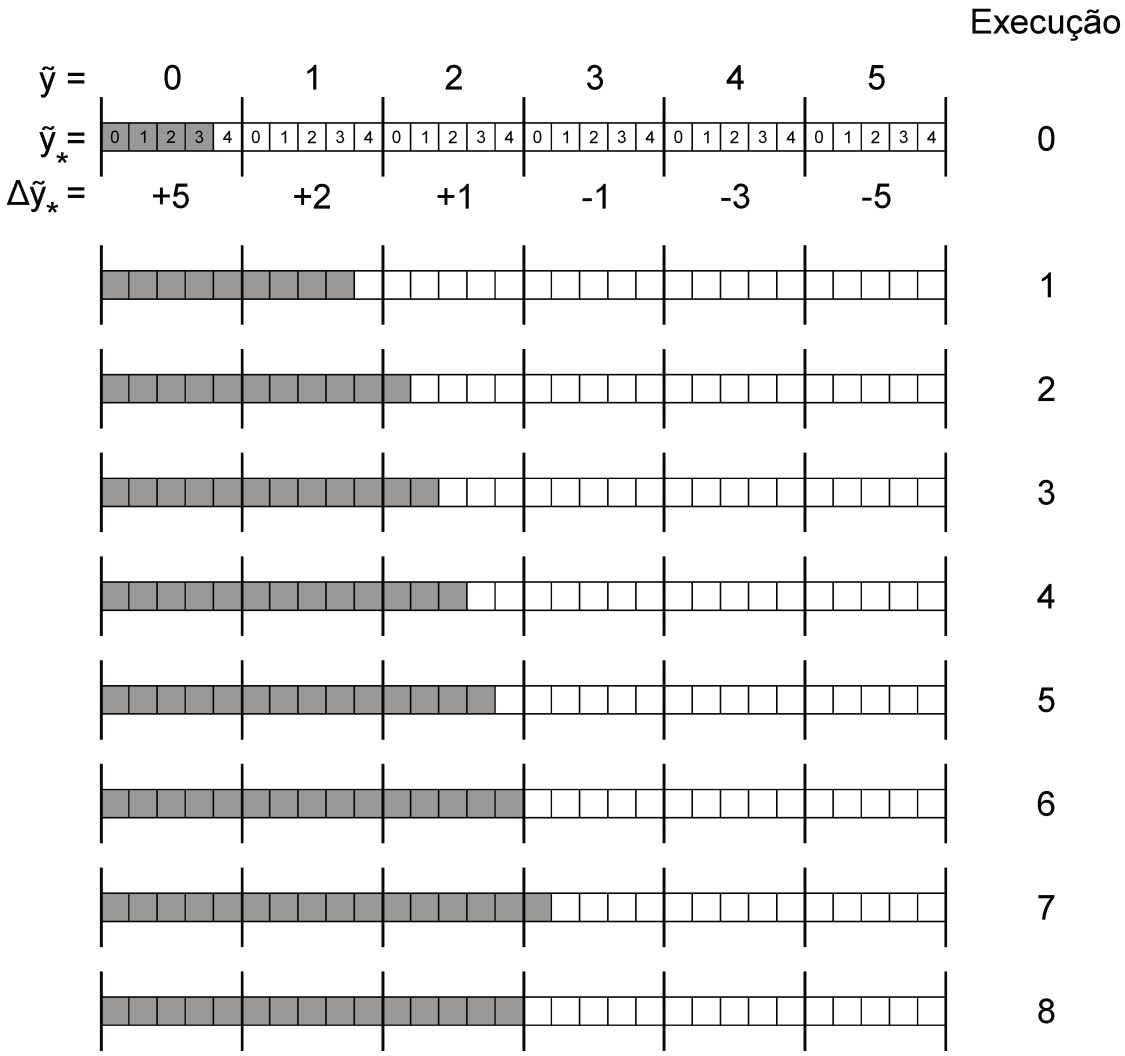


Figura 3.9: Representação esquemática da execução de uma máquina estendida construída com o método $\delta\pi$.

As equações das transições servem justamente para calcular os novos valores de $\tilde{y}$ e $\tilde{y}_*$ em cada passo da máquina. Especificamente para esse exemplo, no estado inicial o valor de $\Delta\tilde{y}_*$ é +5, portanto, a próxima transição levaria o sistema a $\tilde{y} = 1$ e $\tilde{y}_* = 3$ (como observado na Figura 3.9), porque o estado inicial satisfaz as condições de guarda das transições $\mathbf{T}_3$ e $\mathbf{T}_7$. Portanto, as transições executaram os seguintes cálculos para atualização dessas variáveis:

$$\tilde{y}_* = (\tilde{y}_* + \Delta\tilde{y}_*) \bmod n_* = (3 + 5) \bmod 5 = 3$$

$$\tilde{y} = \tilde{y} + (\tilde{y}_* + \Delta\tilde{y}_*)/n_* = 0 + (3 + 5)/5 = 1$$

A sistematização proposta para geração das máquinas facilita a implementação das equações no computador. Neste trabalho, a geração das máquinas de estados a partir de equações diferenciais[2] foi dividida em duas etapas: uma etapa de pré-processamento, onde as equações diferenciais foram resolvidas para todas as combinações de estados possíveis de todas as variáveis do problema, a fim de gerar a **matriz de salto** $\mathbf{L}(y)$ (matriz que compila todos os valores dos saltos dos microestados) e a etapa efetiva de geração das máquinas. O pré-processamento foi feito no Matlab com solução das equações pelo comando ODE15s, visto que este *solver* pode ser usado também para sistemas com rigidez. As máquinas foram implementadas na linguagem do *software* NuSMV. O fluxograma da estratégia está ilustrado na Figura 3.10.

Para facilitar a construção das máquinas, foi esquematizada uma variável do tipo "estrutura" no Matlab que coleta as informações das variáveis, como o intervalo, número de macroestados, dentre outras. Um exemplo de programa onde foi implementado um sistema de equações diferenciais para posterior transformação nas respectivas máquinas de estados está ilustrado na Figura 3.11. Neste exemplo, a variável do tipo "estrutura" é declarada como `data`.

Nesse programa, especificamente entre as linhas 2 até 5 estão as declarações do nome do programa, tempo de simulação (se o objetivo for apenas simulação do modelo da máquina de estados) e o passo de tempo. Esse passo de tempo é usado na solução das equação diferenciais para computar a matriz de salto. Esse passo de tempo pode influenciar no resultado final, especialmente se a equação for não-linear.

[2]Posteriomente será apresentada a argumentação que amplia o uso do método para outras fontes de informação e não apenas equações diferenciais.

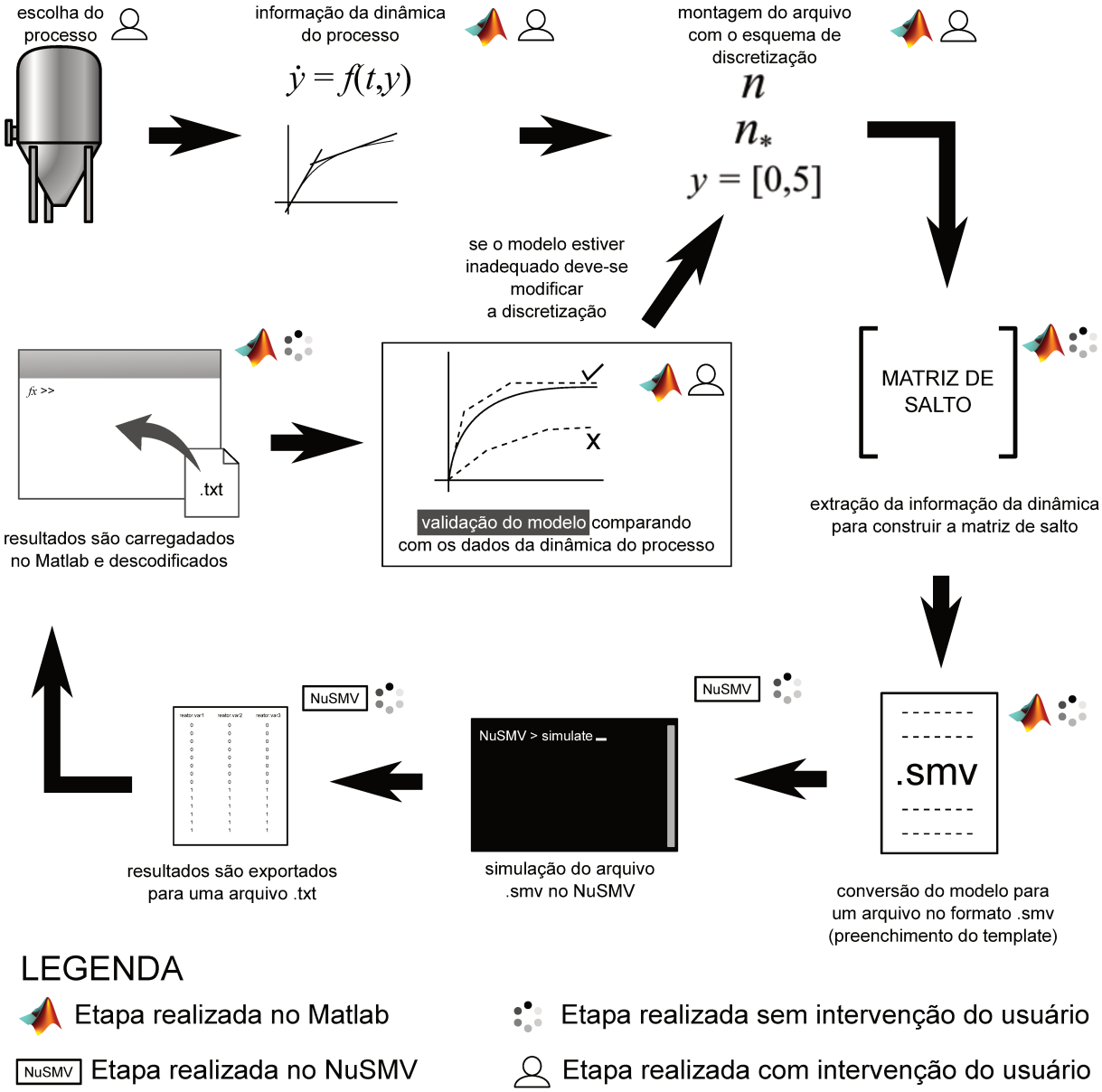


Figura 3.10: Etapas para validação do modelo na forma de máquina de estados segundo a estratégia implementada neste trabalho.

Nas linhas 7 até 14 são configuradas as variáveis por tipo. Por exemplo, no programa da Figura 3.11 existem duas variáveis que se comportam de maneira diferente e por isso as faixas de discretização são diferentes. Todas as variáveis que forem indicadas do tipo 1, serão discretizadas do mesmo jeito, ou seja, terão o mesmo número de macroestados e mesmo intervalo.

A declaração do modelo a ser transformado em máquina de estados finitos, por sua vez, foi feita entre as linhas 16 e 22. Essas linhas apresentam informações como o nome das variáveis do modelo, o tipo das variáveis do problema, a condição inicial dessas variáveis e o nome do arquivo m-file (*script* do Matlab) que contém o modelo. A formatação do

```
1  %% Configuração geral do modelo
2  data.title = 'equacoes';
3  data.endtime = 20;
4  data.timestep = 0.01;
5  data.simulation = true;
6
7  %% Informações das variáveis
8  data.vartype(1).label = 'variavel1';
9  data.vartype(1).n = 20;
10 data.vartype(1).range = [-1 5];
11
12 data.vartype(2).label = 'variavel2';
13 data.vartype(2).n = 20;
14 data.vartype(2).range = [-1 2];
15
16 %% Módulos
17 data.module(1).label = 'modelo';
18 data.module(1).class = 'ODE';
19 data.module(1).statelabels = {'y1' 'y2'};
20 data.module(1).statetype = [1 2];
21 data.module(1).stateinit = [0 0];
22 data.module(1).file = @funcao;
23
24 %% Compila e roda o modelo
25 data = buildmodules(data);
26 translate2NuSMV(data);
27 data = runmodel(data);
28
```

Figura 3.11: Exemplo de um programa do Matlab com as informações de um sistema de EDOs.

arquivo m-file é exatamente a mesma utilizada para solução com os *solvers* de EDOs do Matlab, assim, o mesmo arquivo pode ser utilizado para construção e posterior validação da máquina de estados. A implementação das Equações 3.7 e 3.8 utilizadas para este exemplo estão apresentadas na Figura 3.12.

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = 1 - 3y_2 \tag{3.7}$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = y_1 - 2 \tag{3.8}$$

O programa `buildmodules` chamado na linha 25 foi construído com a finalidade de montar a matriz de salto. Uma discussão mais aprofundada sobre a matriz de salto foi feita na seção 3.6. O programa executa análise da dependência das derivadas em relação às variáveis do modelo. Essa análise é feita computando-se a matriz Jacobiana do sistema de equações diferenciais. Cada valor nulo da matriz indica que aquela equação não depende da respectiva variável que deu origem ao valor nulo. A matriz Jacobiana foi calculada

```
1  function dy = funcao(t,y)
2  % Modelo do sistema de equacoes diferenciais
3
4  dy(1) = 1  - 3*y(2);
5  dy(2) = y(1) - 2;
6
7  dy = dy';
8
```

Figura 3.12: Arquivo m-file de um sistema de equações diferenciais.

utilizando-se diferenças finitas em pontos arbitrários dentro do intervalo de cada variável. Suponha o sistema apresentado nas Equações 3.7 e 3.8, se os valores de y_1 e y_2 forem variados em 10% em relação a um valor de referência arbitrário $y_1 = 1$ e $y_2 = 2$, é possível determinar ao analisar o Jacobiano (Tabela 3.2) que de fato a Equação 3.7 não depende da variável y_1. Os resultados obtidos nesta etapa simplificam o modelo final na forma de máquinas de estados finitos.

Tabela 3.2: Jacobiano do sistema formado pelas Equações 3.7 e 3.8.

	y_1	y_2
$\frac{\mathrm{d}y_1}{\mathrm{d}t}$	0	-3
$\frac{\mathrm{d}y_2}{\mathrm{d}t}$	1	0

O comando `translate2NuSMV` (linha 26) executa um programa desenvolvido para converter a matriz de salto e todas as informações do modelo em um arquivo com extensão .smv para ser executado no NuSMV.

Os programas escritos pelo `translate2NuSMV` na linguagem do NuSMV seguem um *template* cujas partes possuem funções específicas. O código é dividido em duas partes principais: o módulo `main` (obrigatório para todos os programas do NuSMV) e os módulos que contêm as equações dos modelos (cada modelo, que pode ser constituído por um conjunto de EDOs, tem um módulo próprio).

O módulo `main` neste trabalho foi dividido em três partes: a declaração das matrizes de salto (criadas pelo comando `buildmodules`), declaração das variáveis globais (em geral são entradas dos modelos) e inicialização dos módulos dos modelos, ou seja, nesta última parte, por exemplo, um único modelo que descreve o funcionamento de um tanque poderia ser reutilizado para modelar todos os tanques de uma simulação. O *template* usado para o NuSMV está apresentado na Figura 3.13.

A codificação das variáveis entre 0 e $n-1$ é muito conveniente a esta altura: para recuperar o valor de $\Delta\tilde{y}_*$ no NuSMV a partir das matrizes, basta utilizar o próprio valor codificado como índice da matriz de salto. Por exemplo, se a derivada de y depende das variáveis x, y e z, a matriz multidimensional pode ser declarada como `dy[x][y][z]`, onde x, y e z serão substituídos pelos respectivos macroestados dessas variáveis. O programa no Matlab já constrói a matriz para que essa indexação seja possível no NuSMV.

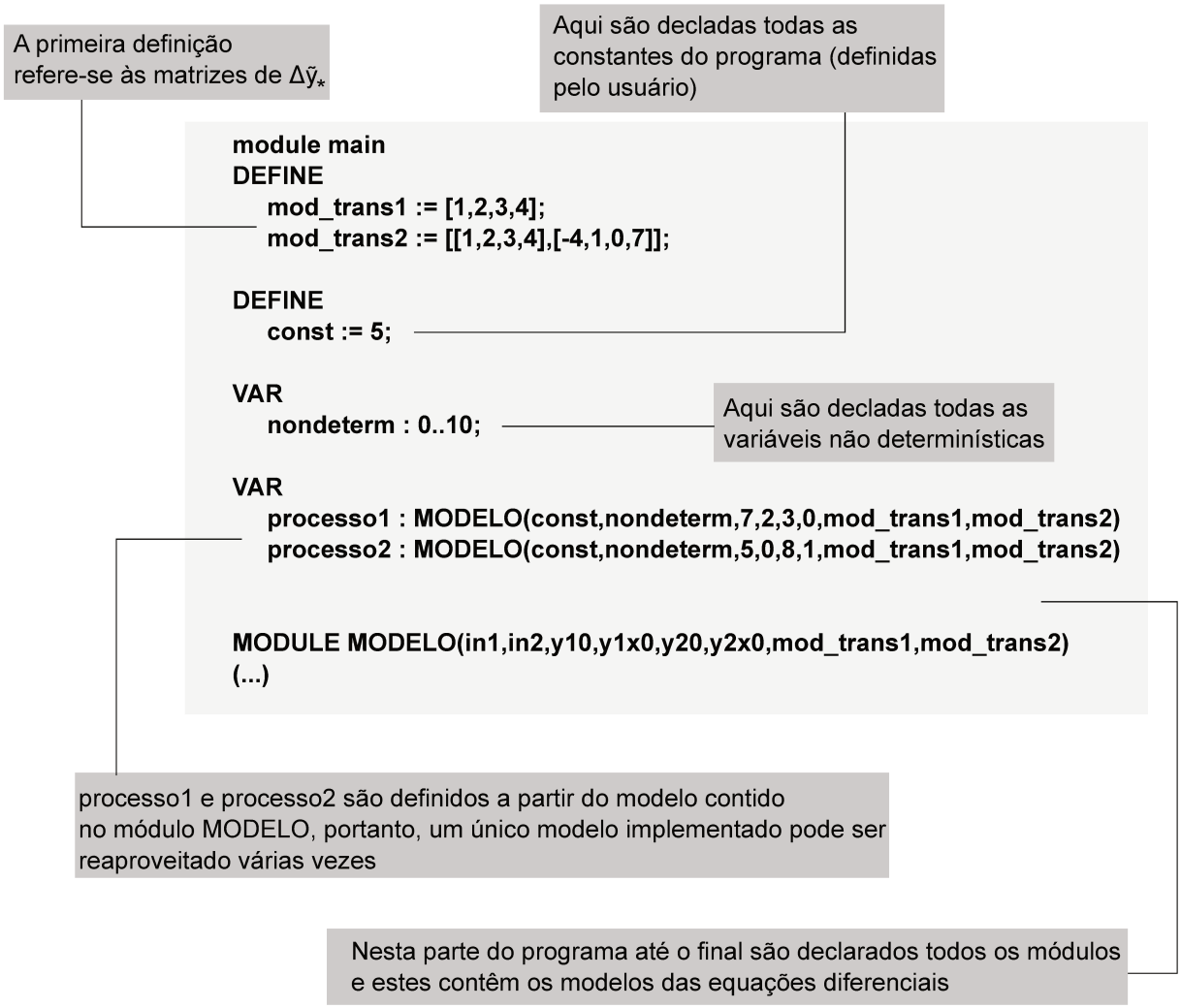


Figura 3.13: *Template* do programa main.

Por fim, o *template* dos módulos de cada modelo também tem suas especificidades. O primeiro bloco de informações é o carregamento do valor da matriz de salto para cada variável a partir do estado atual do sistema para ser usado nas transições. Depois, são declarados os macro e microestados, atribuídas as condições iniciais dessas variáveis e, finalmente, são declaradas as transições das máquinas $\mathcal{M}_\delta$ e $\mathcal{M}_\pi$ (Figura 3.14).

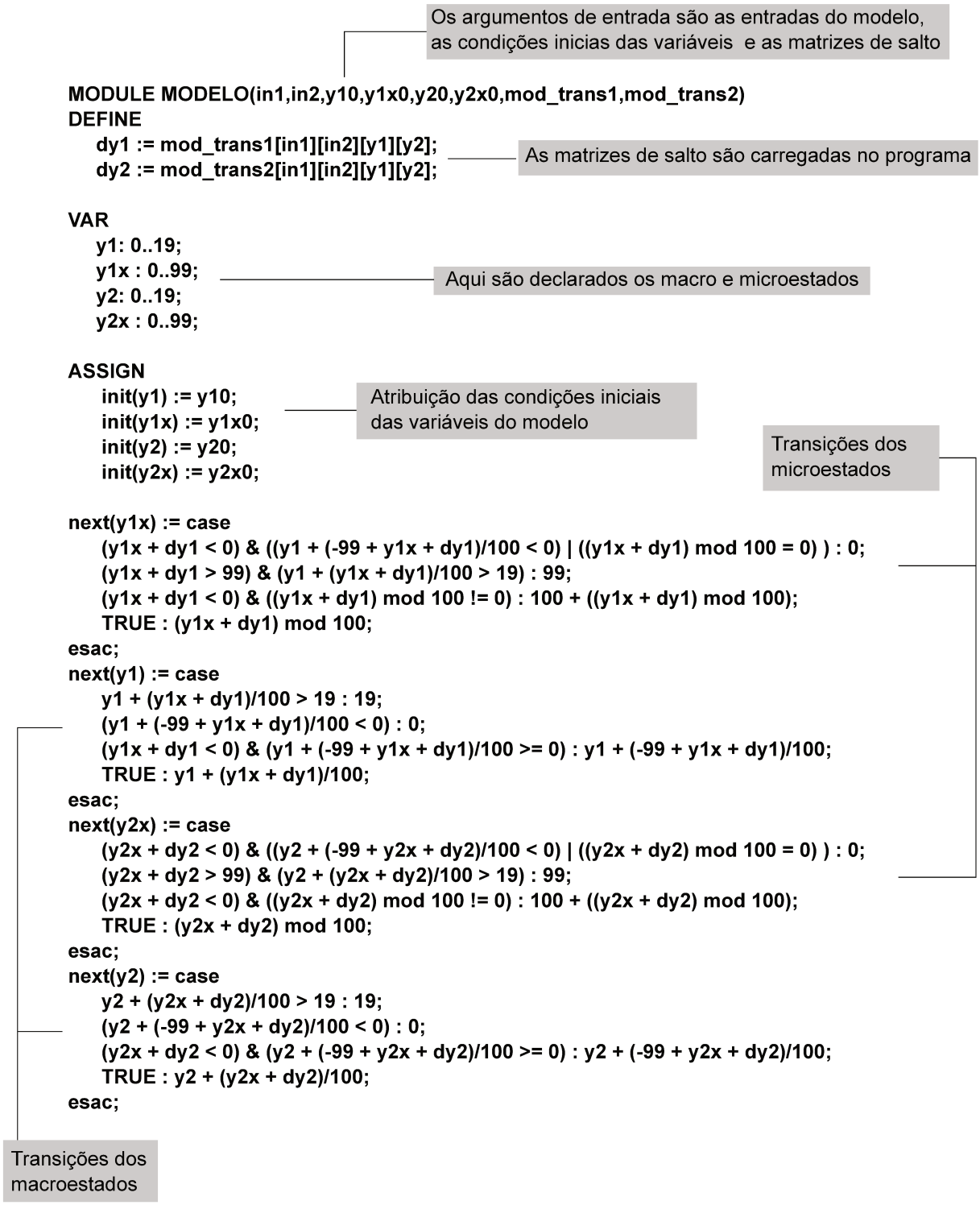


Figura 3.14: *Template* de um módulo.

3.6 Matriz de salto

Conforme estabelecido, os valores dos microestados têm de ser atualizados em cada transição efetuada pelo sistema e esses valores são a princípio calculados a partir da informação da dinâmica do sistema. Para implementação da estratégia $\delta\pi$ é primordial a compilação de todos os saltos dos microestados numa matriz, a fim de que tais valores possam ser recuperados pelas máquinas de estados. Essa matriz é a memória da dinâmica do sistema, ou seja, ela contém as informações necessárias para reproduzir o comportamento transiente do processo.

Tal matriz, chamada neste trabalho de matriz de salto (*leap matrix*), pode assumir alta dimensionalidade caso a respectiva variável de estado seja dependente de muitas variáveis. A matriz de salto pode ser definida como

$$\mathbf{L}(y) = \Delta\tilde{y}_{*,ijk\ldots q} = \left[f(y_1,y_2,\ldots,y_n)\Big|_0^{\Delta t} \right]^{\sim} \tag{3.9}$$

em que $y_1, y_2, \ldots, y_n$ são as variáveis independentes e f é a solução da equação diferencial do sistema avaliada no intervalo $[0;\Delta t]$, posteriormente codificada para valores inteiros. O símbolo $[\ ]^{\sim}$ indica codificação para números inteiros.

A matriz de salto armazena o conhecimento da dinâmica do processo e, por isso, quanto mais discretizadas forem as variáveis de estado, mais informação a matriz de salto deve armazenar, levando no limite a um problema de explosão combinatorial. Por exemplo, se um balanço de energia depender da composição de 3 componentes (C_1, C_2 e C_3), da temperatura atual do sistema (T), da vazão de alimentação (v_0) e da temperatura de alimentação (T_0), a temperatura desse sistema é função de 6 variáveis independentes, ou seja, $T(C_1,C_2,C_3,T,v_0,T_0)$. Se cada variável independente for discretizada em 10 estados, a matriz multidimensional $\mathbf{L}(T)$ teria 10^6 itens que poderiam levar um longo tempo para serem computados.

Assim, é necessário uma análise criteriosa para a escolha das variáveis independentes que deseja analisar e as que serão fixadas a fim de conter a explosão da matriz.

Quando f é uma equação diferencial, existem duas formas de calcular o valor de $\Delta\tilde{y}_*$: a forma mais simples é uma aproximação pela derivada avaliada localmente como

$$\Delta\tilde{y}_{*,ijk\ldots q} = \left[\frac{\mathrm{d}y}{\mathrm{d}t}\Big|_{\mathbf{y}} \cdot \Delta t \right]^{\sim}$$

Essa estimativa é viável quando a EDO é linear ou ao menos pouco não linear. Quando a EDO não for suficientemente linear, a melhor forma de estimar $\Delta\tilde{y}_*$ é resolvendo a equação para o intervalo $[0;\Delta t]$ e tomando-se a diferença entre o valor final e inicial.

Em geral, a solução da EDO deve ser obtida numericamente a fim de facilitar a automação do procedimento de construção da matriz de salto. Em contrapartida, modelos compostos por de sistemas de EDOs com rigidez podem dificultar a solução numérica, mas métodos numéricos implícitos podem, em tese, viabilizar a construção da matriz e consequentemente a respectiva máquina de estados.

Outro ponto a ser destacado é o fato do modelo do processo ser invariante no tempo, sob pena de inviabilizar a construção da matriz de salto, visto que a equação teria de ser resolvida para todas as combinações de valores das variáveis independentes em cada instante de tempo. **Portanto, pode-se afirmar que o método $\delta\pi$ é aplicável para equações diferenciais lineares ou não lineares e invariantes no tempo.**

Em termos de implementação em linguagem de programação, uma escolha adequada de codificação das variáveis simplifica o procedimento de recuperação dos valores da matriz em cada transição. Suponha que uma variável z dependa de y e dela mesma ($z = f(y,z)$) e que ambas foram discretizadas em 5 macroestados. A matriz de salto de z seria

$$\mathbf{L}(z) = \overbrace{\left.\begin{bmatrix} \Delta\tilde{z}_{*,11} & \Delta\tilde{z}_{*,12} & \Delta\tilde{z}_{*,13} & \Delta\tilde{z}_{*,14} & \Delta\tilde{z}_{*,15} \\ \Delta\tilde{z}_{*,21} & \Delta\tilde{z}_{*,22} & \Delta\tilde{z}_{*,23} & \Delta\tilde{z}_{*,24} & \Delta\tilde{z}_{*,25} \\ \Delta\tilde{z}_{*,31} & \Delta\tilde{z}_{*,32} & \Delta\tilde{z}_{*,33} & \Delta\tilde{z}_{*,34} & \Delta\tilde{z}_{*,35} \\ \Delta\tilde{z}_{*,41} & \Delta\tilde{z}_{*,42} & \Delta\tilde{z}_{*,43} & \Delta\tilde{z}_{*,44} & \Delta\tilde{z}_{*,45} \\ \Delta\tilde{z}_{*,51} & \Delta\tilde{z}_{*,52} & \Delta\tilde{z}_{*,53} & \Delta\tilde{z}_{*,54} & \Delta\tilde{z}_{*,55} \end{bmatrix}\right\} z}^{y}$$

com ordenamento arbitrário das variáveis (coluna ou linha), porém o ordenamento deve ser único no programa. Se z e y assumirem valores de 0 até 4, em qualquer ponto do programa a recuperação do valor de $\Delta\tilde{z}_*$ para um dado estado do sistema poderia ser feito usando os próprios valores de $\tilde{z}$ e $\tilde{y}$ como índice na matriz de salto (no NuSMV os índices de matrizes começam em zero). Portanto, se fosse necessário obter um valor da matriz de salto de z para $\tilde{y} = 2$ e $\tilde{z} = 0$ e a matriz fosse declarada no NuSMV como `lmatrix` o respectivo comando seria `lmatrix[0][2]`. Haverá tantos índices na matriz

quantas variáveis independentes.

Até o presente momento a discussão sobre a construção da matriz de salto se limitou às equações diferenciais como fonte de informação da dinâmica do processo, entretanto a estratégia proposta neste trabalho não se limita a esta classe de equações, pelo contrário, expande a construção das máquinas de estados a outras fontes de informação. Na verdade, a matriz de salto é uma sistematização de como armazenar o conhecimento do processo e não faz parte necessariamente de uma metodologia de aquisição do conhecimento. A função f da Equação 3.9 é uma função (mapeamento) da dinâmica do processo.

Logo, $\Delta\tilde{y}_*$ pode ser estimado a partir de simuladores de processos, de dados reais de plantas de processos e até de operadores, por meio de entrevistas.

Portanto, ao contrário de trabalhos anteriores ((51), (49), (46), dentre outros) que se limitavam apenas a EDOs, estende-se a abordagem $\delta\pi$ com a matriz de salto a uma metodologia de construção de máquinas de estados a partir de outras fontes de informação de um sistema dinâmico. Em termos de síntese das máquinas, esta é uma das maiores contribuições do presente trabalho.

A seguir são apresentados alguns exemplos de aplicação do método $\delta\pi$ na modelagem de sistemas físicos. Há, entretanto, uma ressalva: **todos os exemplos tratam-se de Problemas de Valor Inicial (PVI), ou seja, a condição inicial da variáveis é conhecida**. Numericamente, Problemas de Valor de Contorno (PVC), nos quais o valor das variáveis são conhecidas em outros instantes, necessitariam de uma procedimento iterativo para solução e não são cobertos pelo método $\delta\pi$.

3.7 Ex.1: Máquina de estados a partir de dados experimentais

Neste exemplo inicial, um sistema físico teve uma variável acompanhada no tempo, em dois experimentos distintos, conforme apresentado na Tabela 3.3 e nas Figuras 3.16 e 3.15. Apenas para fins de reprodutibilidade deste exemplo, esses dados foram produzidos pela Equação 3.10, mas parte-se do princípio aqui de que não há modelo disponível para explicar o processo, ou seja, a única fonte de conhecimento da dinâmica do processo são os dados experimentais.

$$\frac{\mathrm{d}y}{\mathrm{d}t} = 5 - 0{,}1y^2 \tag{3.10}$$

Tabela 3.3: Dados coletados de um experimento em duas condições iniciais diferentes.

Tempo/min	Experimento 01	Experimento 02
0	0	14
0,5	2,3955	9,81239
1	4,3025	8,30417
1,5	5,5562	7,65544
2	6,281	7,35316
2,5	6,6689	7,21061
3	6,8687	7,13926
3,5	6,9697	7,10408
4	7,0208	7,08761
4,5	7,0484	7,07944
5	7,0615	7,07472

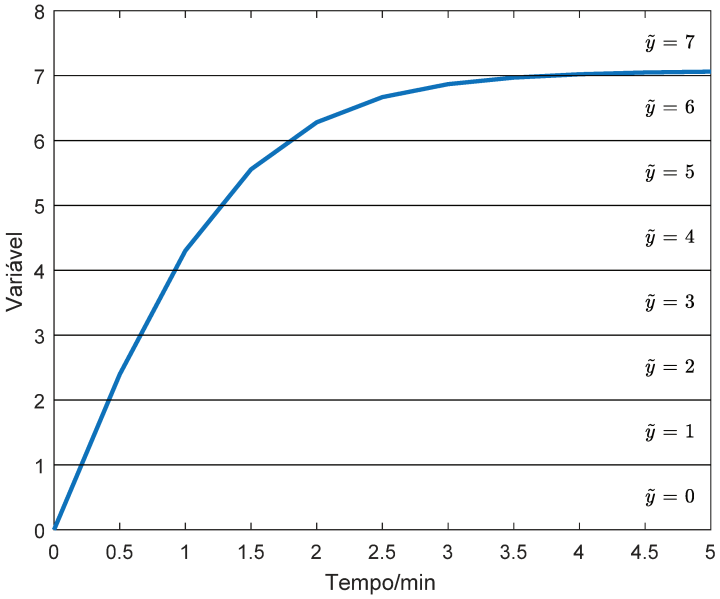


Figura 3.15: Dados do experimento 1.

Para ilustrar a construção da máquina de estados pelo método $\delta\pi$ a partir dos dados experimentais foi arbitrado o uso de 14 macroestados e 100 microestados dentro do intervalo [0;14], intervalo este da variável coberto pelo experimento. As Figuras 3.16 e 3.15 destacam os macroestados. Uma vez que o macroestado 7 aparece nos dois experimentos foi necessário escolher um ponto representativo do intervalo para o cálculo de Δy. Neste experimento foi escolhido o início do intervalo, ou seja, dentro do macroestado 7 que compreende o intervalo aberto [7;8[da variável do experimento, o valor de Δy é calculado

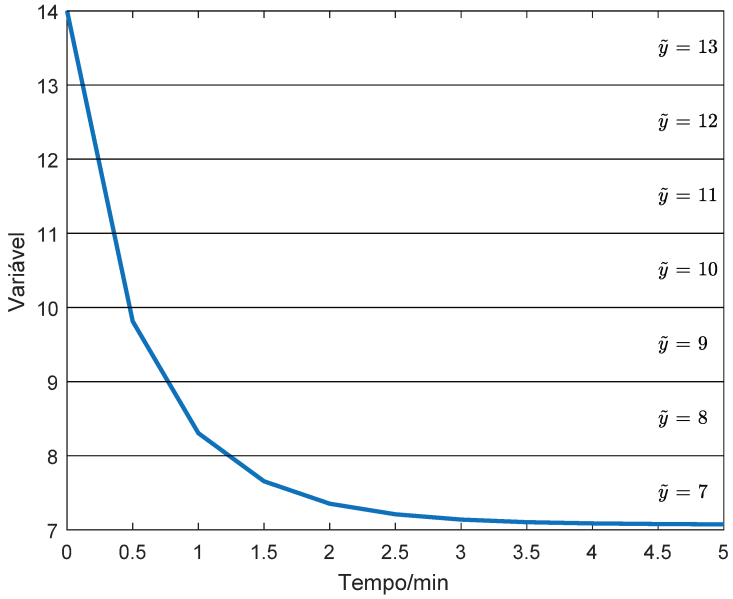


Figura 3.16: Dados do experimento 2.

tomando-se como referência o início do intervalo onde o modelo será linearizado, portanto, Δy foi calculado a partir do experimento 1.

Para a preparação da matriz de salto foi feita a suposição de que $\dot{y} = f(y)$, ou seja, a derivada de y depende apenas da própria variável e que não há nenhuma outra variável afetando o comportamento desse sistema. Portanto, Δy pode ser estimado diretamente da Tabela 3.3, ligando-se os pontos com retas, conforme esquematizado na Figura 3.17. Nesta figura Δy foi calculado como sendo a variação de y partindo-se do início do intervalo do macroestado e avançando $\Delta t = 0{,}5\,\mathrm{s}$. Este intervalo de tempo também foi arbitrário, mas escolhido na mesma ordem de grandeza da taxa de amostragem do experimento.

Os valores resultantes do cálculo de Δy a partir da linearização feita sobre as curvas está compilado na Tabela 3.4. Nota-se que Δy quando $\tilde{y}_* = 7$ é próximo de zero, refletindo a existência do estado estacionário no trecho do macroestado 7. A correspondente matriz de salto está apresentada na Tabela 3.5.

A construção e validação da máquina de estados a partir desses dados seguiu o procedimento da Figura 3.10. O resultado do modelo validado está apresentado na Figura 3.18.

É importante ter em mente que a extração da máquina com o método $\delta\pi$ a partir de dados experimentais está sujeita à qualidade dos dados, ou seja, se os dados possuem algum tipo de erro ou ruído, a qualidade da máquina de estados será comprometida.

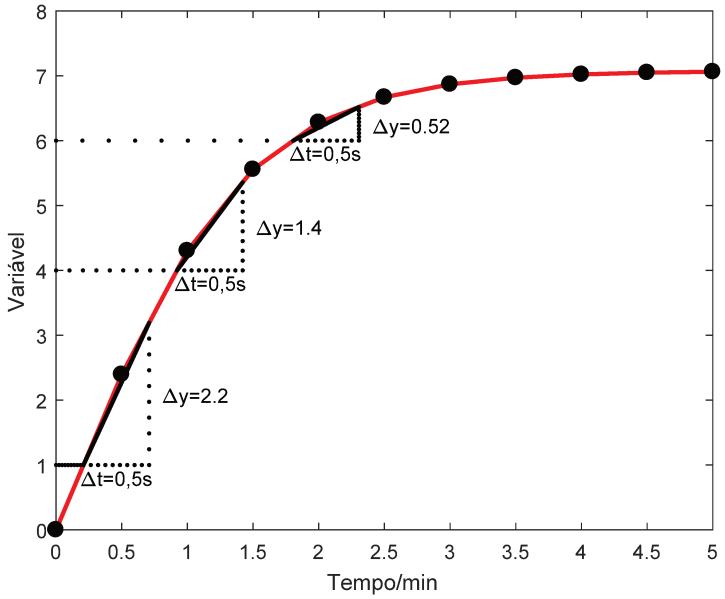


Figura 3.17: Aproximação do comportamento do sistema para $\Delta t = 0{,}5\,\mathrm{s}$ nos macroestados 1, 4 e 6.

Tabela 3.4: Cálculo de Δy a partir de dados experimentais.

$\tilde{y}$	Δy	$\tilde{y}$	Δy
0	2,3955	7	0,0372
1	2,1916	8	-0,4863
2	1,9876	9	-1,0453
3	1,6999	10	-1,6283
4	1,3573	11	-2,2681
5	0,9594	12	-2,9079
6	0,5185	13	-3,5478

Tabela 3.5: Matriz de salto da máquina ($n = 14$ e $n_* = 100$).

$\tilde{y}$	$\Delta\tilde{y}_*$	$\tilde{y}$	$\Delta\tilde{y}_*$
0	240	7	4
1	219	8	-49
2	199	9	-105
3	170	10	-163
4	136	11	-227
5	96	12	-291
6	52	13	-355

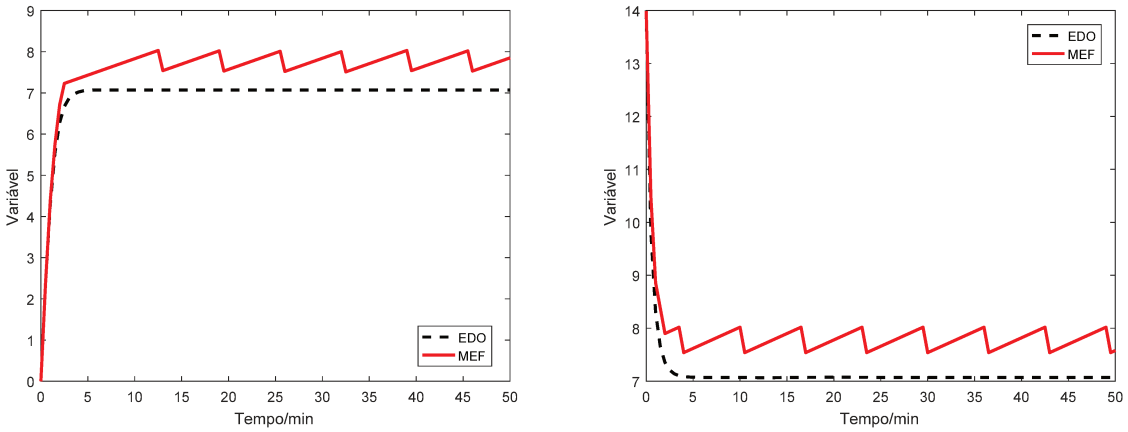


Figura 3.18: Simulação da máquina para as condições do experimento 1 e 2.

3.8 Ex.2: Equação Diferencial Ordinária

Para demonstrar o potencial da técnica de geração de máquinas de estados estendidos conforme apresentado anteriormente, neste exemplo foi feita a conversão de uma equação diferencial ordinária não linear que modela o nível de um tanque aberto para a atmosfera, conforme esquematizado na Figura 3.19.

O tanque possui ainda uma válvula de descarga cuja vazão depende do nível h do tanque.

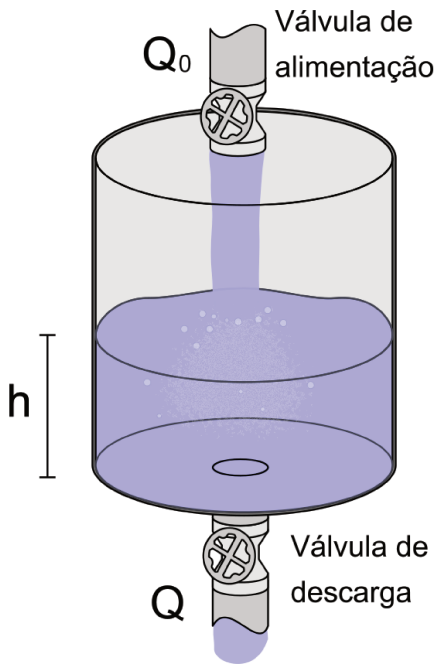


Figura 3.19: Representação esquemática do tanque com as válvulas de alimentação e descarga.

A EDO que modela o nível desse tanque pode ser vista na Equação 3.11, em que A é a área transversal do tanque, c_v é a constante característica da válvula e Q_0 é a vazão de alimentação do tanque. Para fins de exemplo, foram adotados os valores $A = 1\,\mathrm{m}^2$, $c_v = 0{,}01\,\mathrm{m}^{2{,}5}/\mathrm{s}$, $h_{\max} = 5\,\mathrm{m}$ e $Q_{0,\max} = 0{,}05\,\mathrm{m}^3/\mathrm{s}$.

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \frac{Q_0 - c_v\sqrt{h}}{A} \tag{3.11}$$

Observando a Equação 3.11 nota-se que próximo estado da variável h depende do próprio h atual e do valor de Q_0 atual. Portanto, conforme apresentado nas seções anteriores, o primeiro passo para a construção da máquina de estados é varrer os valores de Δh em função de uma faixa de valores de h e Q_0. Uma vez que os valores máximos do nível e da vazão são conhecidos, pode-se estabelecer os seguintes intervalos $h = [0; 5]$ e $Q_0 = [0; 0{,}05]$.

O passo seguinte é discretizar essas variáveis em n intervalos, cujos valores inferiores serão usados para avaliar a derivada $\mathrm{d}h/\mathrm{d}t$. Se forem adotados 10 macroestados para cada variável, totalizará 100 avaliações da derivada e como $\Delta h = \mathrm{d}h/\mathrm{d}t\Delta t$, fixando-se $\Delta t = 1\,\mathrm{s}$ é possível estimar Δh. Os valores calculados para Δh deste modelo estão reunidos na Tabela 3.6.

Tabela 3.6: Valores de $\Delta h \cdot 10^3$ calculados para um $\Delta t = 1\,\mathrm{s}$ em função de diferentes valores de h e Q_0.

		h									
		0	**0,5**	**1,0**	**1,5**	**2,0**	**2,5**	**3,0**	**3,5**	**4,0**	**4,5**
Q_0	**0**	0	-7,1	-10	-12	-14	-16	-17	-19	-20	-21
	0,005	5	-2,1	-5	-7,2	-9,1	-11	-12	-14	-15	-16
	0,01	10	2,9	0	-2,2	-4,1	-5,8	-7,3	-8,7	-10	-11
	0,015	15	7,9	5	2,8	0,86	-0,81	-2,3	-3,7	-5	-6,2
	0,02	20	13	10	7,8	5,9	4,2	2,7	1,3	0	-1,2
	0,025	25	18	15	13	11	9,2	7,7	6,3	5	3,8
	0,03	30	23	20	18	16	14	13	11	10	8,8
	0,035	35	28	25	23	21	19	18	16	15	14
	0,04	40	33	30	28	26	24	23	21	20	19
	0,045	45	38	35	33	31	29	28	26	25	24

Esses valores também poderiam ser estimados resolvendo-se a Equação 3.11 para um intervalo de tempo de 1 s tomando-se como condição inicial o valor inferior de cada macroestado de h.

Uma vez compilados os valores da variação de h em uma tabela, eles podem ser codificados para números inteiros. Como trata-se de variações de h e não de seu valor absoluto, a conversão é feita com auxílio da Equação 3.6. Neste exemplo, foi considerado $n_* = 100$, ou seja, cada macroestado de h contém 100 microestados.

Portanto, se em algum momento as variáveis de estado que representam o problema deste exemplo possuírem os estados $\tilde{h} = 3$ e $\tilde{Q_0} = 1$, isso significa que o microestado $\tilde{h}_*$ será decrementado em 2 unidades. Essa variação pode ou não fazer $\tilde{h}$ sair do macroestado 3, isso dependerá se o microestado atual for menor que 2, o que resultaria em $\tilde{h}_* < 0$ e consequentemente uma transição para $\tilde{h} = 2$. Ao implementar a técnica, a matriz apresentada na Tabela 3.7 pode ser armazenada em uma variável e os valores dos estados de $\tilde{h}$ e $\tilde{Q_0}$ podem ser usados como índices para recuperar os valores da matriz, conforme apresentado na seção 3.5.

Tabela 3.7: Valores de Δh codificados para $\Delta\tilde{h}_*$ a partir dos valores da Tabela 3.6.

		h									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Q_0	**0**	0	-1	-2	-2	-3	-3	-3	-4	-4	-4
	1	1	0	-1	-1	-2	-2	-2	-3	-3	-3
	2	2	1	0	0	-1	-1	-1	-2	-2	-2
	3	3	2	1	1	0	0	0	-1	-1	-1
	4	4	3	2	2	1	1	1	0	0	0
	5	5	4	3	3	2	2	2	1	1	1
	6	6	5	4	4	3	3	3	2	2	2
	7	7	6	5	5	4	4	4	3	3	3
	8	8	7	6	6	5	5	5	4	4	4
	9	9	8	7	7	6	6	6	5	5	5

O próximo passo é estabelecer as transições das máquinas $\mathcal{M}_\delta$ e $\mathcal{M}_\pi$:

$\mathbf{T}_1$: $(\tilde{h}_* + \Delta\tilde{h}_*) < 0 \wedge [(\tilde{h} + (\tilde{h}_* + \Delta\tilde{h}_* - 99)/100) < 0 \vee (\tilde{h}_* + \Delta\tilde{h}_*) \bmod 100 = 0]$
$\rightarrow \tilde{h}_* = 0$

$\mathbf{T}_2$: $(\tilde{h}_* + \Delta\tilde{h}_*) > 99 \wedge [\tilde{h} + (\tilde{h}_* + \Delta\tilde{h}_*)/100] > 9 \rightarrow \tilde{h}_* = 99$

$\mathbf{T}_3$: $(\tilde{h}_* + \Delta\tilde{h}_*) \geq 0 \wedge [\tilde{h} + (\tilde{h}_* + \Delta\tilde{h}_*)/100] \leq 9 \rightarrow \tilde{h}_* = (\tilde{h}_* + \Delta\tilde{h}_*) \bmod 100$

$\mathbf{T}_4$: $(\tilde{h}_* + \Delta\tilde{h}_*) < 0 \wedge (\tilde{h}_* + \Delta\tilde{h}_*) \bmod 100 \neq 0 \rightarrow \tilde{h}_* = 100 + (\tilde{h}_* + \Delta\tilde{h}_*) \bmod 100$

$\mathbf{T}_5$: $[\tilde{h} + (\tilde{h}_* + \Delta\tilde{h}_*)/100] > 9 \rightarrow \tilde{h} = 9$

$\mathbf{T_6}$: $[\tilde{h} + (\tilde{h}_* + \Delta\tilde{h}_* - 99)/100] < 0 \rightarrow \tilde{h} = 0$

$\mathbf{T_7}$: $(\tilde{h}_* + \Delta\tilde{h}_*) \geq 0 \wedge [\tilde{h} + (\tilde{h}_* + \Delta\tilde{h}_*)/100] \leq 9 \rightarrow \tilde{h} = \tilde{h} + (\tilde{h}_* + \Delta\tilde{h}_*)/100$

$\mathbf{T_8}$: $(\tilde{h}_* + \Delta\tilde{h}_*) < 0 \wedge [\tilde{h} + (\tilde{h}_* + \Delta\tilde{h}_* - 99)/100] \geq 0 \rightarrow \tilde{h} = \tilde{h} + (\tilde{h}_* + \Delta\tilde{h}_* - 99)/100$

A implementação das transições em NuSMV pode ser feita conforme a listagem mostrada na Figura 3.20.

A partir deste ponto a máquina de estados estendidos que representa a Equação 3.11 está pronta para ser submetida à verificação ou mesmo simulação para fins de comparação com a equação diferencial. Para ilustrar o comportamento da máquina, uma execução de 102 passos foi feita no *software* NuSMV considerando $Q_0 = 0{,}01\,\mathrm{m}^3/\mathrm{s}$ e $h(0) = 0\,\mathrm{m}$. O resumo das informações dos estados de $\tilde{h}$ e $\tilde{h}_*$ está contido na Tabela 3.8. A comparação gráfica dos resultados da máquina de estados com o resultado oriundo da Equação 3.11 está ilustrada na Figura 3.21.

Tabela 3.8: Execução da máquina de estados que modela o nível do tanque.

$\tilde{h}$	$\tilde{h}_*$	transições ativadas	valor descodificado	valor calculado pela EDO
0	0	7 e 3	0	0
0	1	7 e 3	0,005	0,009
0	2	7 e 3	0,010	0,018
0	3	7 e 3	0,015	0,027
0	4	7 e 3	0,020	0,035
0	5	7 e 3	0,025	0,043
⋮	⋮	⋮	⋮	⋮
0	97	7 e 3	0,485	0,479
0	98	7 e 3	0,490	0,482
0	99	7 e 3	0,495	0,485
1	0	7 e 3	0,500	0,488
1	0	7 e 3	0,500	0,491

É possível notar o comportamento linear da máquina de estados dentro do intervalo de tempo da simulação. Esse resultado expressa o fato de que dentro do estado $\tilde{h} = 0$ e considerando a vazão $Q_0 = 0{,}01\,\mathrm{m}^3/\mathrm{s}$ tem-se apenas $\Delta\tilde{h}_* = 2$, ou seja, a variação da subdiscretização é sempre a mesma até o passo 99. Para fins de comparação, quando $n = 30$ para h e Q_0 e repetindo-se a simulação (com as transições devidamente adaptadas para o novo valor de n), o resultado é uma melhoria na qualidade do modelo, conforme apresentado na Figura 3.22.

```
MODULE main

DEFINE
tanquenivel_trans1 := [[0,-1,-2,-2,-3,-3,-3,-4,-4,-4],[1,0,-1,-1,-2,-2,-2,-3,-3,-3],
  ↪ [2,1,0,0,-1,-1,-1,-2,-2,-2],[3,2,1,1,0,0,0,-1,-1,-1],[4,3,2,2,1,1,1,0,0,0],[5
  ↪ ,4,3,3,2,2,2,1,1,1],[6,5,4,4,3,3,3,2,2,2],[7,6,5,5,4,4,4,3,3,3],[8
  ↪ ,7,6,6,5,5,5,4,4,4],[9,8,7,7,6,6,6,5,5,5]];

DEFINE
Q0 := 1;

VAR
tanque : tanquenivel(Q0,0,0,tanquenivel_trans1);


MODULE tanquenivel(in1,y10,y1x0,tanquenivel_trans1)

DEFINE
dy1 := tanquenivel_trans1[in1][y1];

VAR
y1: 0..9;
y1x : 0..99;

ASSIGN
init(y1) := y10;
init(y1x) := y1x0;

next(y1x) := case
(y1x + dy1 < 0) & ((y1 + (-99 + y1x + dy1)/100 < 0) | ((y1x + dy1) mod 100 = 0) ) :
  ↪ 0;
(y1x + dy1 > 99) & (y1 + (y1x + dy1)/100 > 9) : 99;
(y1x + dy1 < 0) & ((y1x + dy1) mod 100 != 0) : 100 + ((y1x + dy1) mod 100);
TRUE : (y1x + dy1) mod 100;
esac;
next(y1) := case
y1 + (y1x + dy1)/100 > 9 : 9;
(y1 + (-99 + y1x + dy1)/100 < 0) : 0;
(y1x + dy1 < 0) & (y1 + (-99 + y1x + dy1)/100 >= 0) : y1 + (-99 + y1x + dy1)/100;
TRUE : y1 + (y1x + dy1)/100;
esac;

```

Figura 3.20: Programa gerado para o NuSMV com a implementação da máquina de estados do Exemplo 1.

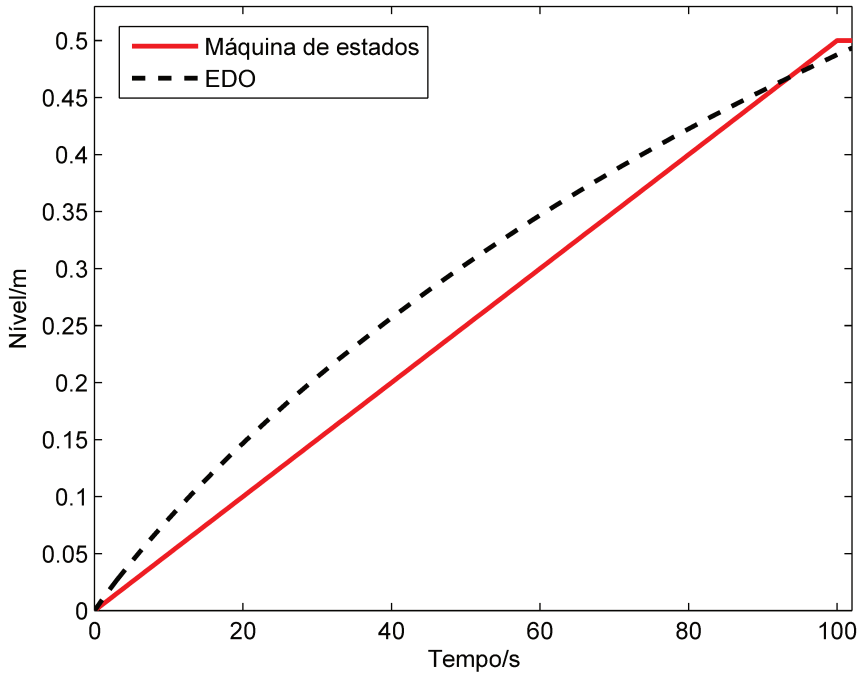


Figura 3.21: Comparação entre o resultado da máquina de estados e a respectiva equação diferencial ($n = 10$ e $n_* = 100$ para h e Q_0).

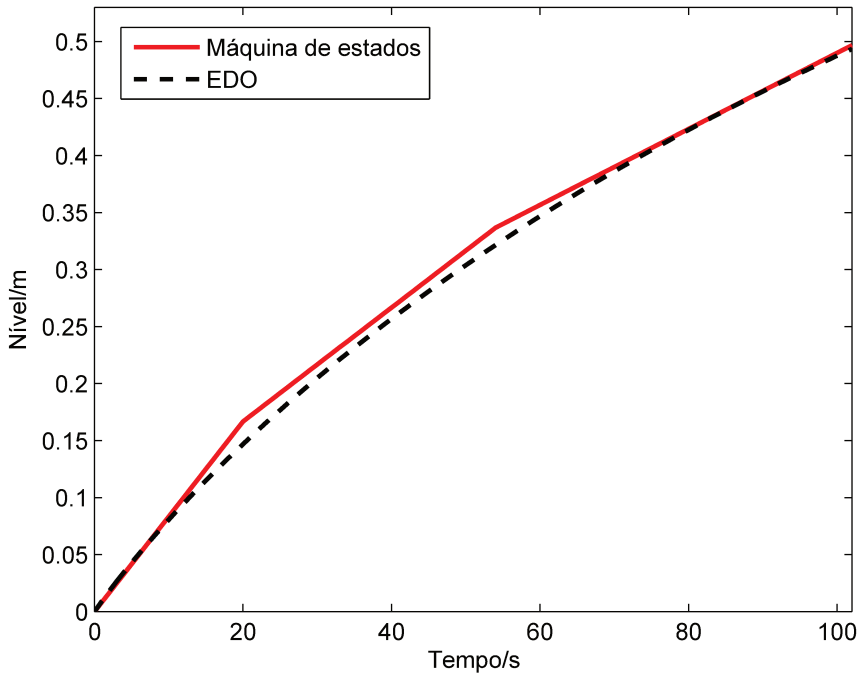


Figura 3.22: Comparação entre o resultado da máquina de estados e a respectiva equação diferencial ($n = 30$ e $n_* = 100$ para h e Q_0).

3.9 Ex.3: sistema de Equações Diferenciais Ordinárias

Uma das vantagens da técnica aqui apresentada é a geração automática das máquinas de estados não apenas para um equação diferencial mas também para sistemas de equações.

Cada variável do sistema dará origem a duas máquinas de estados ($\mathcal{M}_\delta$ e $\mathcal{M}_\pi$) cujas transições são sincronizadas. A fim de exemplificar a conversão de um sistema de EDOs para máquinas de estados, será retomado o exemplo da seção anterior, acrescentado um balanço de massa para contabilizar a concentração de um componente A (C_A) no tanque (Equação 3.12) onde também ocorre uma reação irreversível A $\rightarrow$ B com taxa de consumo de $-r_A = -0{,}05C_\mathrm{A}$.

Neste exemplo, considera-se que a composição da alimentação pode variar dentro do intervalo $C_{\mathrm{A},0} = [1{,}2; 3{,}1]$ (em mol/m^3), o nível em $h = [0{,}1; 2]$ (em m) e a vazão $Q_0 = [0; 0{,}05]$ (em m$^3/s$).

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \frac{Q_0 - c_v\sqrt{h}}{A} \tag{3.11}$$

$$\frac{\mathrm{d}C_\mathrm{A}}{\mathrm{d}t} = \left(Q_0 C_{\mathrm{A},0} - c_v\sqrt{h}C_\mathrm{A} - C_\mathrm{A}\frac{\mathrm{d}h}{\mathrm{d}t}A\right)\frac{1}{V} - 0{,}05C_\mathrm{A} \tag{3.12}$$

Assim como no exemplo da seção anterior, o primeiro passo é discretizar as variáveis do problema. Todas as variáveis (h, Q_0, C_A e $C_{\mathrm{A},0}$) foram discretizadas para $n = 10$, assim como $n_* = 100$. Os valores de Δh dependem apenas de Q_0 e h (conforme indicado pela Equação 3.11), mas os valores de ΔC_A, por outro lado, dependem de Q_0, h, C_A e $C_{\mathrm{A},0}$ o que torna a matriz de ΔC_A multidimensional. Para simplificar o problema e ilustrar o comportamento das máquinas de estados para um sistema de EDOs fixou-se ao menos o valor $Q_0 = 0{,}01\,\mathrm{m}^3/\mathrm{s}$. A composição de alimentação, por sua vez, pode alterar sensivelmente o comportamento do sistema conforme observa-se nas Tabelas 3.9 e 3.10 que contêm os valores de ΔC_A e $\Delta\tilde{C}_{\mathrm{A}*}$, respectivamente, para dois valores de $C_{\mathrm{A},0}$.

Tabela 3.9: Tabelas de valores de $\Delta C_{\mathrm{A}} \cdot 10^3$ calculados considerando $\Delta t = 1\,\mathrm{s}$, $Q_0 = 0{,}01\,\mathrm{m}^3/\mathrm{s}$ e (a) $C_{\mathrm{A},0} = 1{,}2\,\mathrm{mol/m}^3$ e (b) $C_{\mathrm{A},0} = 2{,}91\,\mathrm{mol/m}^3$.

(a)

		h									
		0,1	**0,29**	**0,48**	**0,67**	**0,86**	**1,05**	**1,24**	**1,43**	**1,62**	**1,81**
	1,2	-0,0479	-0,0539	-0,0556	-0,0564	-0,0568	-0,0571	-0,0573	-0,0575	-0,0576	-0,0577
	1,39	-0,12	-0,0907	-0,0824	-0,0785	-0,0762	-0,0748	-0,0737	-0,0729	-0,0724	-0,0719
	1,58	-0,192	-0,128	-0,109	-0,101	-0,0957	-0,0924	-0,0901	-0,0884	-0,0871	-0,0861
	1,77	-0,264	-0,164	-0,136	-0,123	-0,115	-0,11	-0,107	-0,104	-0,102	-0,1
C_{A}	**1,96**	-0,336	-0,201	-0,163	-0,145	-0,135	-0,128	-0,123	-0,119	-0,117	-0,114
	2,15	-0,408	-0,238	-0,19	-0,167	-0,154	-0,145	-0,139	-0,135	-0,131	-0,129
	2,34	-0,484	-0,275	-0,217	-0,189	-0,173	-0,163	-0,156	-0,15	-0,146	-0,143
	2,53	-0,556	-0,312	-0,243	-0,211	-0,193	-0,181	-0,172	-0,166	-0,161	-0,157
	2,72	-0,629	-0,348	-0,27	-0,234	-0,212	-0,198	-0,189	-0,181	-0,176	-0,171
	2,91	-0,702	-0,385	-0,297	-0,256	-0,232	-0,216	-0,205	-0,197	-0,191	-0,185

(b)

		h									
		0,1	**0,29**	**0,48**	**0,67**	**0,86**	**1,05**	**1,24**	**1,43**	**1,62**	**1,81**
	1,2	0,537	0,201	0,107	0,0627	0,0371	0,0204	0,0087	9,03e-06	-0,00669	-0,012
	1,39	0,464	0,164	0,0799	0,0405	0,0176	0,00278	-0,00769	-0,0155	-0,0214	-0,0262
	1,58	0,392	0,127	0,0531	0,0184	-0,00175	-0,0148	-0,0241	-0,0309	-0,0362	-0,0404
	1,77	0,319	0,0901	0,0262	-0,00378	-0,0211	-0,0325	-0,0405	-0,0464	-0,051	-0,0547
C_{A}	**1,96**	0,244	0,0533	-0,000647	-0,0259	-0,0405	-0,0502	-0,0569	-0,0619	-0,0658	-0,0689
	2,15	0,172	0,0165	-0,0274	-0,0481	-0,06	-0,0678	-0,0733	-0,0774	-0,0806	-0,0831
	2,34	0,1	-0,0203	-0,0543	-0,0702	-0,0795	-0,0855	-0,0898	-0,0929	-0,0953	-0,0973
	2,53	0,028	-0,0571	-0,0811	-0,0924	-0,0989	-0,103	-0,106	-0,108	-0,11	-0,111
	2,72	-0,0441	-0,0939	-0,108	-0,115	-0,118	-0,121	-0,123	-0,124	-0,125	-0,126
	2,91	-0,116	-0,131	-0,135	-0,137	-0,138	-0,138	-0,139	-0,139	-0,14	-0,14

Como foram escolhidos $n = 10$ e $n_* = 100$, as transições das máquinas $\mathcal{M}_\pi$ e $\mathcal{M}_\delta$ para h são idênticas às apresentadas no exemplo do tanque da seção anterior. As transições para C_{A}, por sua vez, são semelhantes e estão apresentadas na lista a seguir.

T$_1$: $(\tilde{C}_{\mathrm{A}*}+\Delta\tilde{C}_{\mathrm{A}*}) < 0 \wedge [(\tilde{C}_{\mathrm{A}}+(\tilde{C}_{\mathrm{A}*}+\Delta\tilde{C}_{\mathrm{A}*}-99)/100) < 0 \vee (\tilde{C}_{\mathrm{A}*}+\Delta\tilde{C}_{\mathrm{A}*}) \bmod 100 = 0] \rightarrow \tilde{C}_{\mathrm{A}*} = 0$

T$_2$: $(\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*}) > 99 \wedge [\tilde{C}_{\mathrm{A}} + (\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*})/100] > 9 \rightarrow \tilde{C}_{\mathrm{A}*} = 99$

T$_3$: $(\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*}) \geq 0 \wedge [\tilde{C}_{\mathrm{A}} + (\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*})/100] \leq 9 \rightarrow \tilde{C}_{\mathrm{A}*} = (\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*}) \bmod 100$

T$_4$: $(\tilde{C}_{\mathrm{A}*}+\Delta\tilde{C}_{\mathrm{A}*}) < 0 \wedge (\tilde{C}_{\mathrm{A}*}+\Delta\tilde{C}_{\mathrm{A}*}) \bmod 100 \neq 0 \rightarrow \tilde{C}_{\mathrm{A}*} = 100+(\tilde{C}_{\mathrm{A}*}+\Delta\tilde{C}_{\mathrm{A}*}) \bmod 100$

T$_5$: $[\tilde{C}_{\mathrm{A}} + (\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*})/100] > 9 \rightarrow \tilde{C}_{\mathrm{A}} = 9$

T$_6$: $[\tilde{C}_{\mathrm{A}} + (\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*} - 99)/100] < 0 \rightarrow \tilde{C}_{\mathrm{A}} = 0$

T$_7$: $(\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*}) \geq 0 \wedge [\tilde{C}_{\mathrm{A}} + (\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*})/100] \leq 9 \rightarrow \tilde{C}_{\mathrm{A}} = \tilde{C}_{\mathrm{A}} + (\tilde{C}_{\mathrm{A}*} + \Delta\tilde{C}_{\mathrm{A}*})/100$

Tabela 3.10: Tabelas de valores de $\Delta\tilde{C}_{\mathrm{A}*}$ calculados a partir da Tabela 3.9.

(a)

		h									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	0	-25	-28	-29	-30	-30	-30	-30	-30	-30	-30
	1	-63	-48	-43	-41	-40	-39	-39	-38	-38	-38
	2	-101	-67	-57	-53	-50	-49	-47	-47	-46	-45
	3	-139	-86	-72	-65	-61	-58	-56	-55	-54	-53
	4	-177	-106	-86	-76	-71	-67	-65	-63	-61	-60
C_{A}	**5**	-215	-125	-100	-88	-81	-77	-73	-71	-69	-68
	6	-255	-145	-114	-100	-91	-86	-82	-79	-77	-75
	7	-293	-164	-128	-111	-101	-95	-91	-87	-85	-83
	8	-331	-183	-142	-123	-112	-104	-99	-95	-92	-90
	9	-369	-203	-156	-135	-122	-114	-108	-104	-100	-98

(b)

		h									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	0	283	106	56	33	20	11	5	0	-4	-6
	1	244	86	42	21	9	1	-4	-8	-11	-14
	2	206	67	28	10	-1	-8	-13	-16	-19	-21
	3	168	47	14	-2	-11	-17	-21	-24	-27	-29
	4	129	28	0	-14	-21	-26	-30	-33	-35	-36
C_{A}	**5**	91	9	-14	-25	-32	-36	-39	-41	-42	-44
	6	53	-11	-29	-37	-42	-45	-47	-49	-50	-51
	7	15	-30	-43	-49	-52	-54	-56	-57	-58	-59
	8	-23	-49	-57	-60	-62	-64	-65	-65	-66	-66
	9	-61	-69	-71	-72	-73	-73	-73	-73	-73	-74

T$_8$: $(\tilde{C}_{\mathrm{A}*}+\Delta\tilde{C}_{\mathrm{A}*})<0\wedge[\tilde{C}_{\mathrm{A}}+(\tilde{C}_{\mathrm{A}*}+\Delta\tilde{C}_{\mathrm{A}*}-99)/100]\geq 0\rightarrow\tilde{C}_{\mathrm{A}}=\tilde{C}_{\mathrm{A}}+(\tilde{C}_{\mathrm{A}*}+\Delta\tilde{C}_{\mathrm{A}*}-99)/100$

O esforço computacional para gerar matriz $\Delta\tilde{C}_{\mathrm{A}*}$ foi perceptível e cresceu de forma não-linear quando comparada com o aumento do número de estados do modelo. Enquanto o tamanho das matrizes saltou de 10x10 para 30x30 (aumento de nove vezes no número de itens da matriz), o tempo computacional para calcular saiu de 2,74 s para 84,4 s. O ganho em termos de precisão neste exemplo talvez não compense o esforço computacional extra, tendo em vista os resultados apresentados na comparação das Figuras 3.23 e 3.24, onde a primeira ilustra a dinâmica quando o modelo possui dez macro estados e a última, trita macroestados.

Conforme as discretizações ficam cada vez menores, o programa gerado na linguagem

do NuSMV fica maior, conforme ilustrado na Figura 3.25, onde houve a necessidade de omissão de parte das matrizes de $\Delta\tilde{C}_{\mathrm{A}*}$ e $\Delta\tilde{h}_{*}$ em função de suas dimensões. A grande dimensão dos matrizes pode impactar na simulação e verificação do modelo, assim como na própria construção da matriz de salto. Stursberg *et al* (49) também utilizaram um estratégia de solução da EDO para geração de máquinas temporizadas e relataram um grande esforço computacional na etapa de construção da máquina quando comparada com outras etapas até a construção do modelo final.

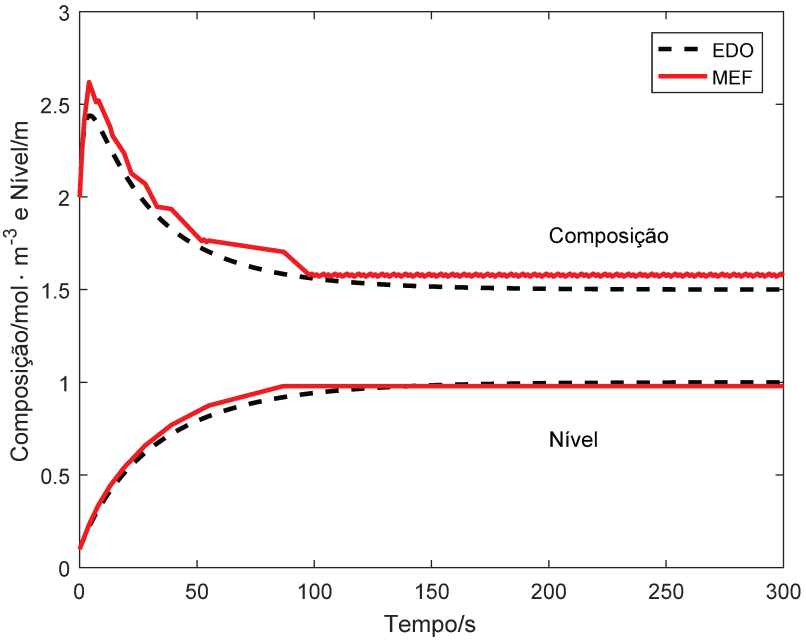


Figura 3.23: Comparação entre o resultado da máquina de estados e a equação diferencial do tanque considerando $n = 10$.

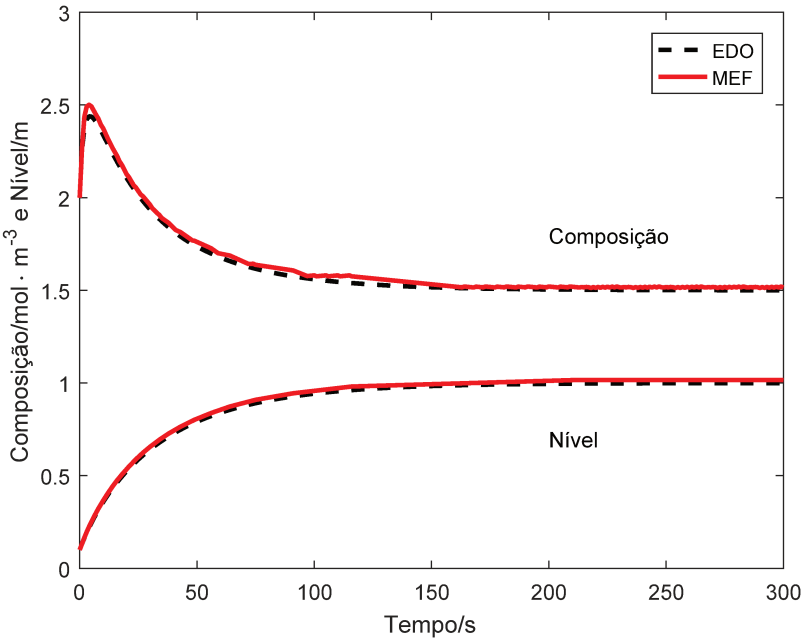


Figura 3.24: Comparação entre o resultado da máquina de estados e a equação diferencial do tanque considerando $n = 30$.

Figura 3.25: Programa gerado para o NuSMV com a implementação da máquina de estados do Exemplo 1.

```
MODULE main
DEFINE
cstr_trans1 := [[[[-2,-2,-2,-2,-2,-2,-2,-2,-2,-2], [omitido por falta de espaço],[17
  ↪ ,17,17,17,17,17,17,17,17,17]]]];
cstr_trans2 := [[[[-6,-7,-8,-9,-10,-11,-12,-13,-14,-15], [omitido por falta de
  ↪ espaço], [16,12,9,5,2,-1,-5,-8,-12,-15]]]];

DEFINE Q0 := 1; DEFINE Ca0 := 9;
VAR
tanque : cstr(Q0,Ca0,0,0,4,21,cstr_trans1,cstr_trans2);

MODULE cstr(in1,in2,y10,y1x0,y20,y2x0,cstr_trans1,cstr_trans2)
DEFINE
dy1 := cstr_trans1[in1][in2][y1][y2];
dy2 := cstr_trans2[in1][in2][y1][y2];
VAR
y1: 0..9; y1x : 0..99; y2: 0..9; y2x : 0..99;
ASSIGN
init(y1) := y10; init(y1x) := y1x0; init(y2) := y20; init(y2x) :=y2x0;

next(y1x) := case
(y1x + dy1 < 0) & ((y1 + (-99 + y1x + dy1)/100 < 0) | ((y1x + dy1) mod 100 = 0) ) :
  ↪ 0;
(y1x + dy1 > 99) & (y1 + (y1x + dy1)/100 > 9) : 99;
(y1x + dy1 < 0) & ((y1x + dy1) mod 100 != 0) : 100 + ((y1x + dy1) mod 100);
TRUE : (y1x + dy1) mod 100;
esac;
next(y1) := case
y1 + (y1x + dy1)/100 > 9 : 9;
(y1 + (-99 + y1x + dy1)/100 < 0) : 0;
(y1x + dy1 < 0) & (y1 + (-99 + y1x + dy1)/100 >= 0) : y1 + (-99 + y1x + dy1)/100;
TRUE : y1 + (y1x + dy1)/100;
esac;
next(y2x) := case
(y2x + dy2 < 0) & ((y2 + (-99 + y2x + dy2)/100 < 0) | ((y2x + dy2) mod 100 = 0) ) :
  ↪ 0;
(y2x + dy2 > 99) & (y2 + (y2x + dy2)/100 > 9) : 99;
(y2x + dy2 < 0) & ((y2x + dy2) mod 100 != 0) : 100 + ((y2x + dy2) mod 100);
TRUE : (y2x + dy2) mod 100;
esac;
next(y2) := case
y2 + (y2x + dy2)/100 > 9 : 9;
(y2 + (-99 + y2x + dy2)/100 < 0) : 0;
(y2x + dy2 < 0) & (y2 + (-99 + y2x + dy2)/100 >= 0) : y2 + (-99 + y2x + dy2)/100;
TRUE : y2 + (y2x + dy2)/100;
esac;

```

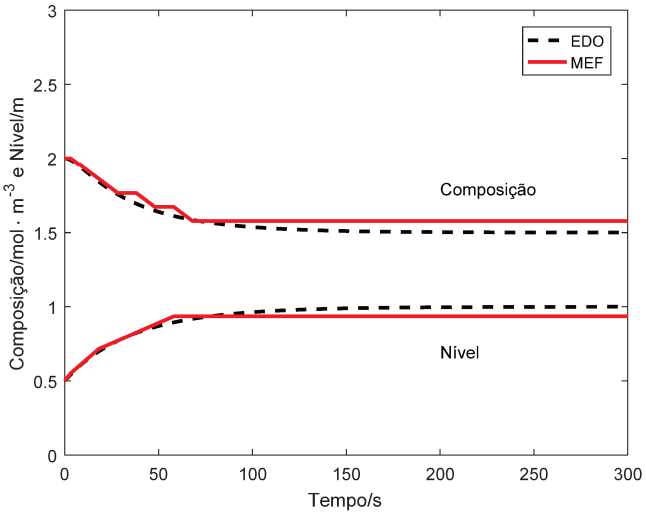
O comportamento do sistema em termos da rigidez do sistema de equações diferenciais foi testado modificando-se o coeficiente da válvula para $c_v = 5\,\mathrm{m}^{2,5}$ (também foi alterado $n = 20$ para melhorar a solução da máquina de estados). O grau de rigidez, calculado

como a razão entre o maior e o menor valor absoluto dos autovalores da matriz Jacobiana do sistema de equações (Equação 3.13) (52), mudou de 235,9 para $1{,}1794 \times 10^5$ quando o Jacobiano é avaliado para $h = 0{,}5\,\mathrm{m}$ e $C_\mathrm{A} = 2\,\mathrm{mol/m^3}$. Quanto maior o valor do grau de rigidez, mais difícil é a solução da equação diferencial por métodos numéricos explícitos.

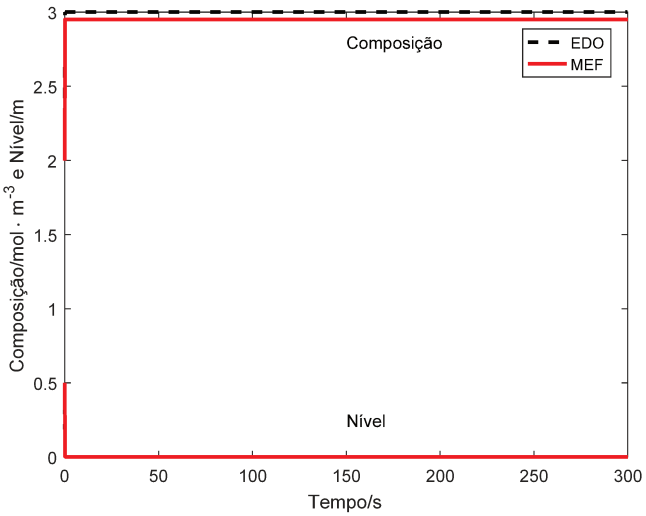
$$\frac{\left|\mathrm{Re}\bar{\lambda}\right|}{\left|\mathrm{Re}\underline{\lambda}\right|} \tag{3.13}$$

Ora, o método $\delta\pi$ produz máquinas de estados que avançam no tempo de forma explícita, semelhante ao método explícito de Euler (Equação 3.1) e o qual é sabido apresentar dificuldade numérica na solução de problemas rígidos. Todavia, como o sistema de equações neste trabalho é resolvido numericamente por métodos implícitos no Matlab (pelo comando ode15s) e a simulação da máquina é feita com passo fixo, não foi identificada nenhuma dificuldade numérica adicional ao executar a máquina no NuSMV.

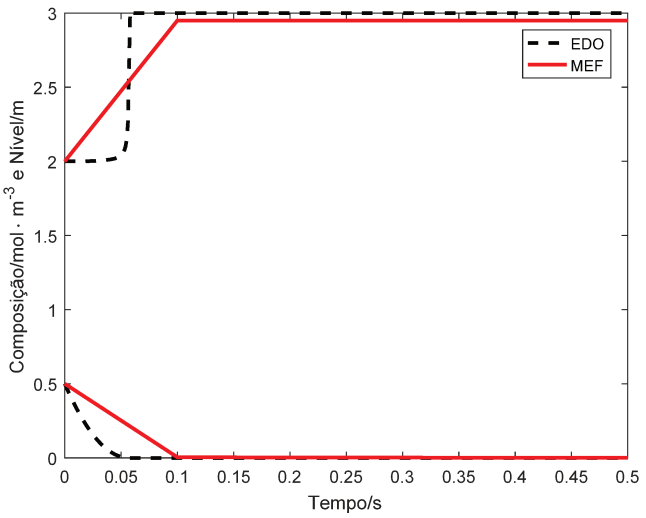
Usando passo de tempo $\Delta t = 0{,}1\,\mathrm{s}$ foi possível realizar um comparativo do sistema de EDOs do tanque com e sem rigidez. Foi observado um tempo médio de 1,48 s para simular no NuSMV o sistema com rigidez, enquanto o modelo sem rigidez levou, em média, 2,56 s, em que a diferença pode ser atribuída a alguma execução interna diferenciada da máquina no NuSMV, como ativação de transições diferentes. Ressalta-se que a única diferença entre as duas simulações foi o valor do c_v. A Figura 3.26 apresenta ambas as simulações e um destaque para a região inicial da simulação do modelo com rigidez.



(a)



(b)



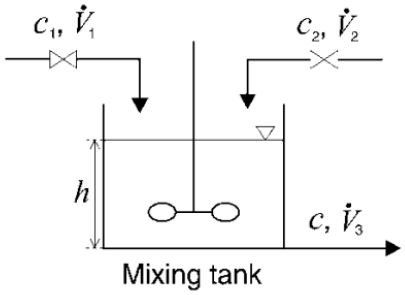
(c)

Figura 3.26: Simulação do tanque para a condição (a) não-rígida, (b) rígida e (c) um zoom nos primeiros 0,5 s da simulação com rigidez.

Finalmente, para fins de comparação com a literatura, o trabalho de Stursberg *et al*, um exemplo semelhante foi utilizado para construção de uma máquina temporizada para modelar o nível e composição de um tanque de mistura a partir de equações diferenciais (Equações 3.14 e 3.15). Este tanque possui duas alimentações com composições diferentes e opera de forma contínua. Uma ilustração esquemática do tanque e da discretização feita para o nível e a composição pode ser observada na Figura 3.27.

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \frac{1}{A}\left(Q_1 + Q_2 - c_v\sqrt{h}\right) \tag{3.14}$$

$$\frac{\mathrm{d}C}{\mathrm{d}t} = \frac{1}{Ah}\left(Q_1(C_1 - C) + Q_2(C_2 - C)\right) \tag{3.15}$$



^{D}h (m)	^{D}c (mole/l)
{[0.5, 0.7[, [0.7, 0.9[, [0.9, 1.1[, [1.1, 1.3[, [1.3, 1.5]}	{[1.20, 1.32[, [1.32, 1.44[, [1.44, 1.56[, [1.56, 1.68[, [1.68, 1.80]}

Figura 3.27: Sistema físico modelado no trabalho de Stursberg *et al* com as respectivas discretizações. Reproduzido com permissão de (49).

A máquina resultante, porém, possui quantidade de informação restrita quando comparada com as máquinas desenvolvidas pelo método $\delta\pi$, posto que a modelagem feita por Stursberg *et al* estima o tempo de uma transição como um intervalo de tempo máximo e mínimo que o sistema pode levar para sair de um estado para outro. Assim, não é possível recuperar quantitativamente, como feito no presente trabalho, a trajetória do sistema dinâmico, apenas extrair uma estimativa (dentro de um intervalo) do tempo total da trajetória. As transições entre os estados discretos estão apresentadas na Figura 3.28.

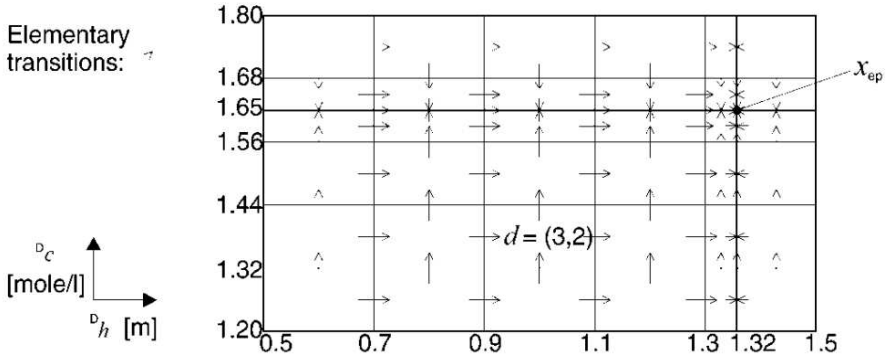


Figura 3.28: Transições possíveis da máquina de estados do tanque de mistura. Reproduzido com permissão de (49).

3.10 Ex.4: Equação Diferencial Ordinária de 2ª ordem

Na engenharia química alguns sistemas são adequadamente descritos por equações de 2ª ordem, como sistemas que envolvem difusão em estado estacionário. Neste exemplo, será introduzido um equipamento que constitui um sistema mecânico largamente utilizado nas indústrias químicas: a válvula de controle pneumática.

A válvula de controle pneumática (Figura 3.29) possui um diafragma que divide a parte superior da válvula (chamada de atuador) em duas câmaras. No centro do diafragma existe um suporte onde está presa a haste da válvula e uma mola. Na outra ponta da haste há o obturador (um plugue), que é o elemento que interrompe o escoamento do fluido pelo corpo da válvula. Na câmara formada acima do diafragma existe uma entrada de ar comprimido que pode pressurizar essa região e criar uma força sobre o diafragma, que se desloca e desce a haste, fechando a passagem do fluido com o obturador. Se a região acima do diafragma for despressurizada, a mola força o retorno do diafragma para a posição de repouso, levantando a haste e abrindo passagem para o fluido pelo corpo da válvula.

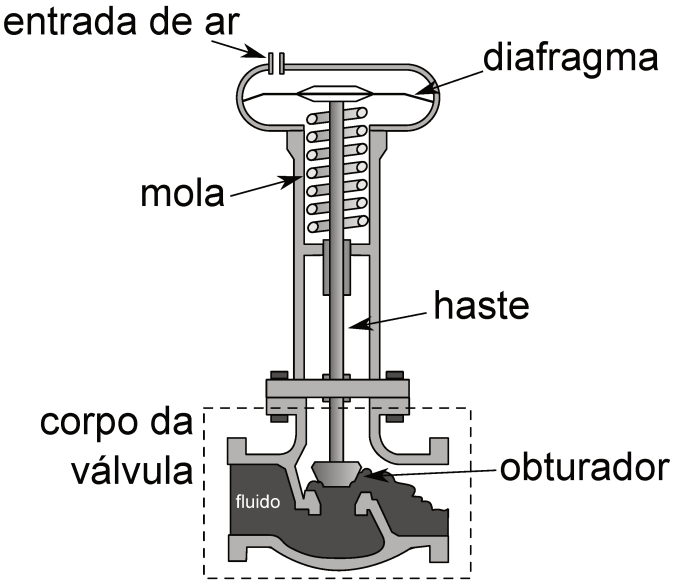


Figura 3.29: Esquema de uma válvula de controle pneumática.

Um modelo que descreve de forma simplificada o comportamento dinâmico dessa válvula está apresentado na Equação 3.16(53). Trata-se de uma equação diferencial de 2ª ordem para sistemas do tipo massa + mola amortecido.

$$M_{\mathrm{h}}\frac{\mathrm{d}^2a}{\mathrm{d}t^2} + c\frac{\mathrm{d}a}{\mathrm{d}t} + k_m a = \Delta P A_{\mathrm{d}} \tag{3.16}$$

Entretanto, na forma apresentada na Equação 3.16, a válvula se comportaria como uma válvula do tipo falha-fecha, ou seja, ar seria aplicado para abrir a válvula. Se a válvula fosse do tipo falha-abre, ou seja, ar é aplicado na válvula para fechá-la, seria necessário efetuar a transformação $a_o = 100 - a$ e aplicando-a na Equação 3.16, obteria-se a Equação 3.17.

$$-M_{\mathrm{h}}\frac{\mathrm{d}^2a_{\mathrm{o}}}{\mathrm{d}t^2} - c\frac{\mathrm{d}a_{\mathrm{o}}}{\mathrm{d}t} + k_m(100 - a_{\mathrm{o}}) = \Delta P A_{\mathrm{d}} \tag{3.17}$$

Considere, a a abertura da válvula (0 %-100 %) quando a válvula é do tipo falha-fecha, a_{o} a abertura da válvula quando ela é do tipo falha-abre, M_{h} a massa da haste, c a constante associada ao atrito da haste com os outros componentes da válvula, k_m a constante da mola, ΔP é a diferença de pressão entre os dois lados do diafragma e A_d é a área do diafragma.

A Equação 3.17 pode ser reescrita como um sistema de equações diferenciais com a introdução de uma nova variável y, conforme as Equações 3.18 e 3.19. Assim, uma vez demonstrado que a metodologia $\delta\pi$ permite a extração dos autômatos equivalentes a um

sistema de equações diferenciais ordinárias isso significa consequentemente a possibilidade também de representação de EDOs de segunda ordem, quando estas são reduzidas a um sistema de EDOs.

$$\frac{\mathrm{d}a_\mathrm{o}}{\mathrm{d}t} = y \tag{3.18}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -\frac{1}{M_\mathrm{h}}\left(cy - k_m(100 - a_\mathrm{o}) + \Delta P A_d\right) \tag{3.19}$$

A simulação desse sistema e comparação com o respectivo sistema de EDOs está apresentada na Figura 3.30, onde se observa uma boa concordância dos modelos. Para essa simulação foi utilizado $\Delta t = 0{,}01\,\mathrm{s}$, $n = 20$, $n_* = 100$ e o intervalo para a abertura foi de $[0; 100]$ e para a derivada de $[-40; 80]$. Ressalta-se que em todos os exemplos apresentados neste trabalho esses intervalos e a escolha do número de macroestados foram obtidos por tentativa e erro, comparando-se o resultado da máquina com o resultado da EDO até obter uma aproximação satisfatória.

Um último aspecto que merece destaque é a influência do passo de tempo usando para construção das matrizes de salto. Assim como na simulação de EDOs, a escolha de um valor adequado de Δt pode interferir na qualidade da solução e essa característica também foi observada nas máquinas de estados geradas pela presente metodologia. Uma comparação entre os passos de tempo de 0,01 s; 0,1 s e 1 s está ilustrada na Figura 3.31.

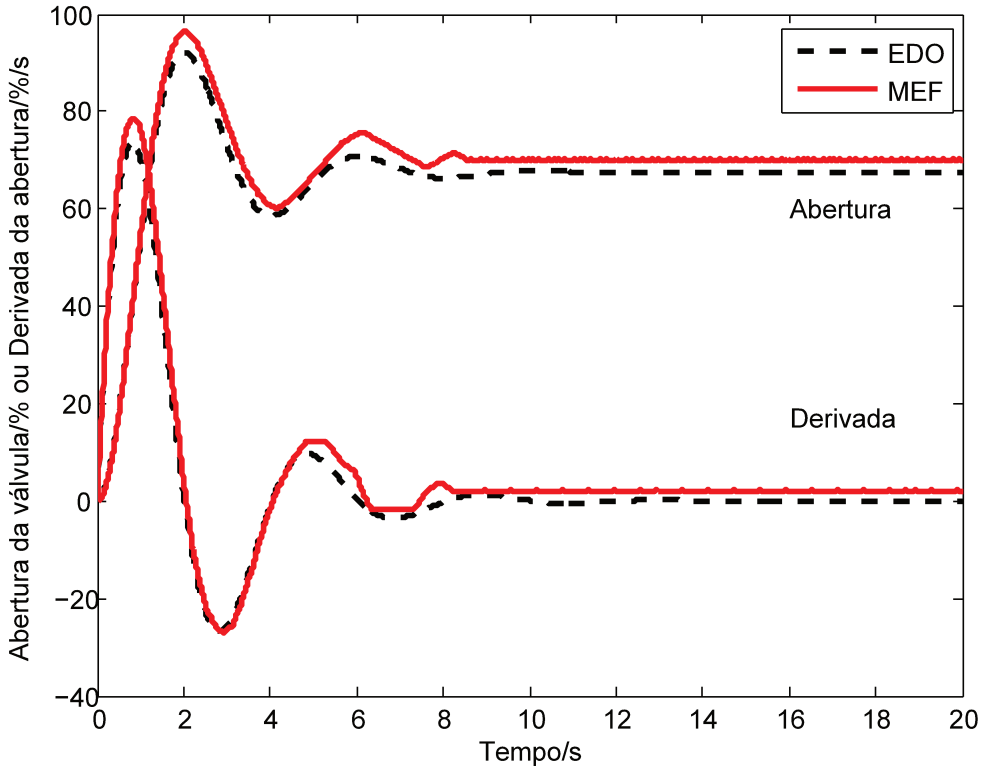


Figura 3.30: Comparação da máquina de estados que simula a abertura da válvula pneumática (e sua derivada) e o resultado da respectiva EDO.

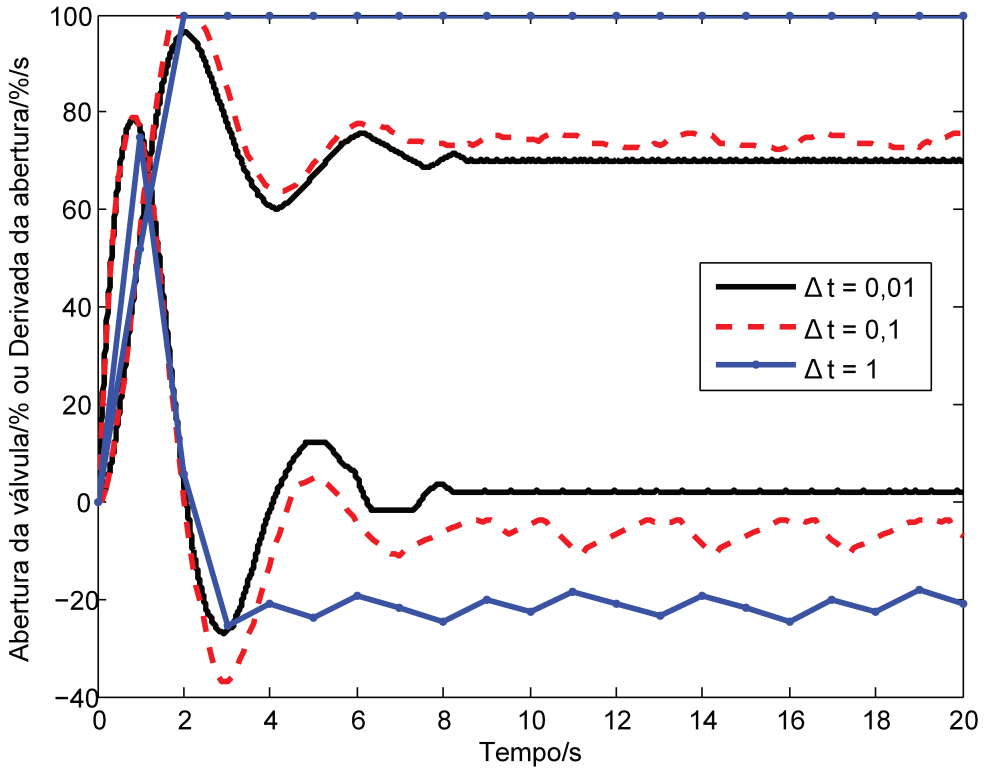


Figura 3.31: Efeito da escolha de Δt na construção da máquina de estados para o exemplo da válvula pneumática.

3.11 Ex.5: Equação Diferencial Parcial

A última classe de equações de grande importância para engenharia química é a de equações diferenciais parciais e este exemplo apresenta o potencial da metodologia $\delta\pi$ para modelagem de EDPs. Conforme demonstrado anteriormente, a redução de qualquer modelo a um sistema de equações ordinárias permite a extração do respectivo autômato e isso também é verdade para EDPs utilizando-se discretização do modelo por diferenças finitas.

Neste exemplo o problema a ser modelado é o perfil transiente de composição de uma partícula esférica porosa onde ocorre difusão de um soluto da película ao redor da partícula até o seu centro. Uma vez que o modelo da partícula seja discretizado, a partícula passa a ser representada por cascas esféricas e o modelo discretizado descreverá a composição em cada casca ao longo do tempo. A representação esquemática do sistema está apresentada na Figura 3.32. O modelo da partícula pode ser visto na Equação 3.20 e a discretização resultante na Equação 3.21.

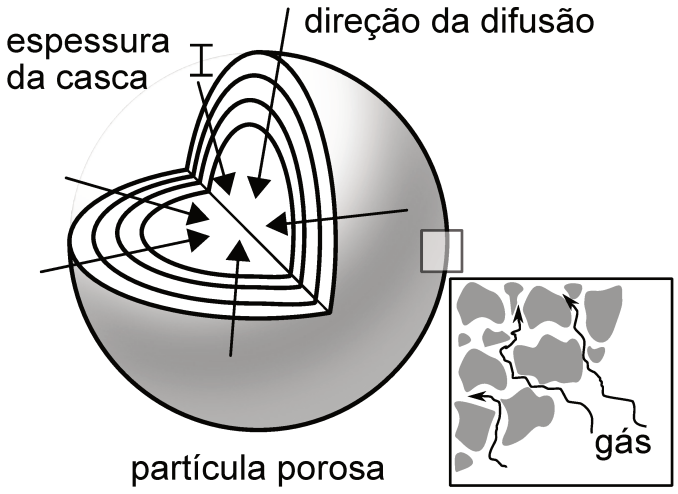


Figura 3.32: Representação esquemática de uma partícula porosa e as cascas que constituem o modelo matemático.

$$\frac{\partial C}{\partial t} = D_{AB}\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial C}{\partial r}\right) \tag{3.20}$$

$$\frac{\partial C_i}{\partial t} = D_{AB}\left[\frac{1}{r_i}\frac{C_{i+1}-C_{i-1}}{\Delta r} + \frac{C_{i+1}-2C_i+C_{i-1}}{\Delta r^2}\right] \tag{3.21}$$

Portanto, a Equação 3.21 quando agrupada para todas as cascas forma uma sistema de equações diferenciais ordinárias, passível de ser transformado em um máquina de estados.

No centro da partícula foi imposta uma condição de contorno de Neumann, ou seja, composta por uma derivada. Essa condição de contorno surge em virtude de centro da partícula não existir difusão, portanto, pode-se escrever

$$\left.\frac{\partial C}{\partial r}\right|_{r=0} = \frac{-3C_1 + 4C_2 - C_3}{2\Delta r} = 0$$

de onde se conclui que $C_1 = (4C_{i+1} - C_{i+2})/3$, ou seja, a concentração na casca esférica mais interna é explicitamente dependente das concentrações das duas cascas seguintes. A notação utilizada neste trabalho enumera a casca mais interna como 1. No outro extremo da partícula ($r = r_p$) para fins de simplificação foi considerada uma concentração constante e igual a $C_b = 10\,\mathrm{mol/m^3}$ (condição de contorno de Dirichlet).

Os perfis de composição ao longo do tempo para algumas cascas estão ilustrados na Figura 3.33. Nota-se que as composições finais não foram iguais na proximidade do estado estacionário, todavia, o comportamento transiente ainda é semelhante ao da equação diferencial. De qualquer maneira, o modelo ainda poderia ser utilizado em verificação formal para acompanhar a composição da partícula ao longo de um reator heterogêneo.

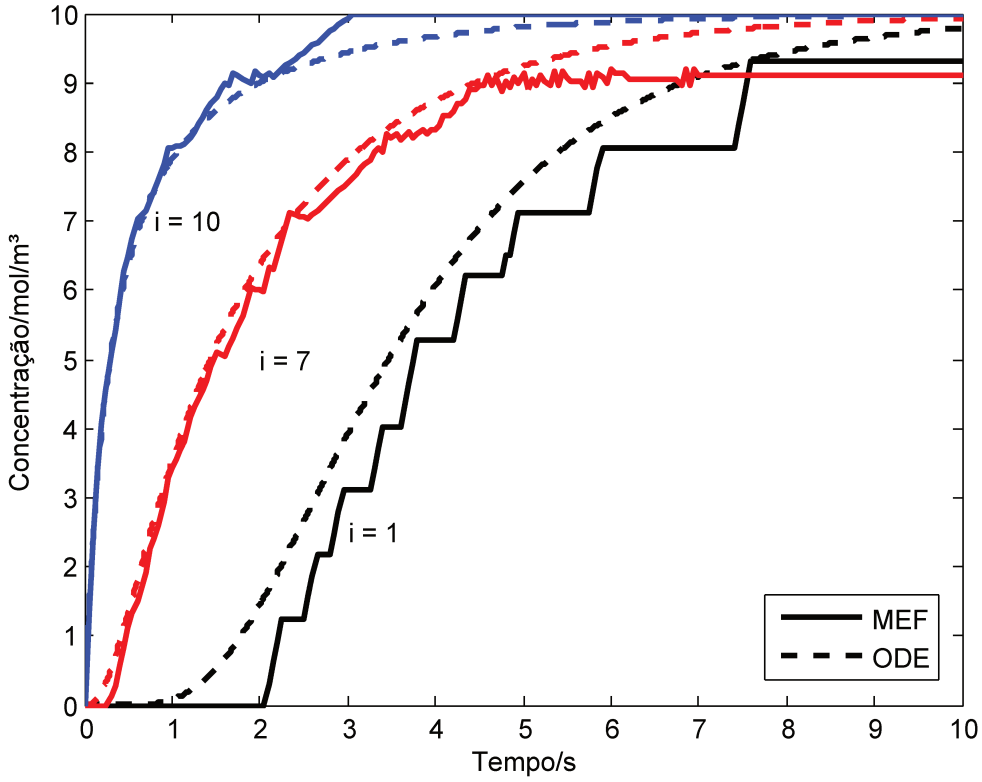


Figura 3.33: Perfil de concentração ao longo do tempo para a casca mais interna ($i = 1$), a 7ª casca e a casca mais externa do modelo da partícula.

Uma questão pertinente a este exemplo é a estabilidade da máquina de estados em

vista da discretização do domínio. Uma vez que o modelo na forma de máquina de estados reproduz o comportamento do modelo dinâmico original, espera-se que ele apresente comportamento instável quando a EDO for instável, e dentro de certos limites no passo de tempo (em virtude do distanciamento em relação ao modelo estável conforme apresentado na Figura 3.31) que também haja estabilidade no comportamento dinâmico da máquina.

Uma análise da estabilidade dos modelos produzidos pelo método $\delta\pi$ não é tarefa trivial, posto que ao contrário do critério de Von Neumman (54) que pode ser aplicado a equações diferenciais parciais, que são contínuas, as máquinas são discretas e as derivadas são fixadas em intervalos visando uma linearização por trechos, o que dificulta a previsão da estabilidade global da máquina.

3.12 Aplicabilidade

Diante dos exemplos apresentados, crê-se na eficáfia e aplicabilidade do método $\delta\pi$ na produção de máquinas de estados extendidos para modelagem de processos dinâmicos. Entretanto, conforme apresentado anteriormente, outras ferramentas, dentro de certas limitações, também poderiam ter sido utilizadas para tal finalidade.

O diferencial do método apresentado neste trabalho é principalmente a possibilidade de utilização de outras fontes da dinâmica do processo para construção de uma máquina quantitativa, ou seja, o comportamento dinâmico pode ser acompanhado e comparado com as equações/fontes originais. A classe de modelos mais próxima dessa formulação são as máquinas híbridas e durante a execução do presente trabalho foi possível identificar a existência de uma *software* chamado S-taliro cuja implementação e verificação de modelos híbridos aparenta ser compatível com diversas aplicações na engenharia química.

A lógica temporal empregada pelo S-taliro permite maior expressividade que a lógica ACTL, utilizada em outras ferramentas. O fato de usar modelos híbridos também reduz o esforço na construção do modelo a ser avaliado, uma vez que muitos modelos na engenharia química já assumem a forma de equações diferenciais. Há ainda um diferencial muito interessante que é a possibilidade de comunicação com outros programas via Simulink e isso permitiria a comunicação, por exemplo, com o *software* Aspen Dynamics, de modo que a verificação de modelos de simuladores de processos poderia ser feita, em tese, de maneira imediata, sem a necessidade de construção de modelos intermediários como ocorre

no método $\delta\pi$.

Infelizmente não foi possível encontrar a existência de padrões de propriedades para a lógica MTL (*Metric Temporal Logic*), usada pelo S-taliro e isso pode dificultar sensivelmente o uso da ferramenta na engenharia química.

Outras abordagens para construção das máquinas de estados a partir de abstrações das respectivas equações diferenciais também foram identificadas ((49) e (45), por exemplo), mas em geral, a quantidade de informação retida pela máquina é insuficiente para uma análise detalhada do transiente dos processos. Como é durante o transiente onde os acidentes podem acontecer, as abstrações podem esconder comportamentos importantes para análise e segurança dos processos.

Por outro lado, o método $\delta\pi$ sofre com a necessidade de experimentação em relação ao esquema de discretização, ou seja, a decisão sobre n, n_* e Δt que em diversas situações observadas neste trabalho parece apresentar um ótimo em termos de número de macroestados. Foi observado que o aumento irrestrito do número de macroestados nem sempre melhora a qualidade da solução. Além disso as equações diferenciais devem ser invariantes no tempo.

Finalmente, pode-se citar a relativa simplicidade do método $\delta\pi$ em sua implementação, que consiste, em sua essência, na execução de experimentos em diferentes condições para avaliação da matriz de salto e posterior implementação do modelo em máquina extendidas para cada variável (função incógnita) do modelo (lembrando que as transições sempre são as mesmas para cada variável). Uma tabela comparativa (Tabela 3.11) foi organizada para expor melhor as diferenças e semelhanças entre os métodos apresentados em (49) e (45), o *software* S-taliro e o método $\delta\pi$.

Tabela 3.11: Comparativo entre diferentes meios para produzir e simular máquinas a partir de equações diferenciais.

	Stursberg *et al* (49)	S-taliro (55)	Blouin (45)	Método $\delta\pi$
Modelo quantitativo	✗	✓	✗	✓
Padrões de propriedades da lógica temporal	✓	✗	✓	✓
Estratégia pode ser usada em outras linguagens de programação	✓	✗	✓	✓
Outras fontes de informação da dinâmica além de eq. diferenciais?	✗	✓	✗	✓
Aceita alta dimensionalidade dos modelos ($n > 0$)?	✓	✓	✗	✓

capítulo 4

Estudo de caso: verificação formal de um reator

Um estado estacionário é a condição em que um sistema se encontra em que não há mudança em suas variáveis. Muitos sistemas físicos possuem apenas um único estado estacionário para uma dada condição de entrada e condições iniciais.

A existência de múltiplos estados estacionários foi observada pela primeira vez em reatores químicos em 1918 na reação de oxidação da amônia (56) e até hoje se mostra um desafio em termos operacionais. Neste capítulo serão introduzidos alguns conceitos básicos de multiplicidade de estados e o porquê de sua ocorrência, especificamente, em reatores CSTR não-adiabáticos. O modelo de um reator (devidamente discretizado com o método $\delta\pi$) foi utilizado como estudo de caso para demonstrar o potencial da verificação de modelos ao explorar a dinâmica desse equipamento e direcionar a síntese do procedimento de partida até o estado estacionário desejado.

4.1 Multiplicidade de estados estacionários

É dito que um sistema apresenta multiplicidade de estados estacionários quando este sistema, em geral aberto, permite a existência de mais de um estado estacionário para o mesmo conjunto de parâmetros (56). Quando esta característica é identificada em um processo, pequenas mudanças em alguma variável podem levar a grandes variações nas condições operacionais do sistema, em vista da alteração do estado estacionário.

Diversos processos químicos e bioquímicos apresentam essa característica, como sistemas enzimáticos, reatores CSTR (*Continuous Stirred-Tank Reactor*), colunas de destilação, processos de absorção com reação química, polimerização de olefinas em leitos fluidizados, combustão e em diversos processos usados em manufatura e processamento de componentes eletrônicos (56).

Dentre as fontes mais importantes de multiplicidade de estados pode-se listar (56):

- Dependência não monotônica de uma ou mais taxas do processo em relação a uma ou mais variáveis de estado
- Realimentação de informação
- Operação em contra-corrente

Dada a abrangência do tema e a necessidade de um amplo arcabouço teórico para uma discussão aprofundada, o foco neste trabalho será para o caso clássico de reatores CSTR não-isotérmicos. No CSTR, tanto o balanço de massa quanto energia são funções da temperatura e suas curvas podem se interceptar em diversos pontos. Esses balanços podem ser vistos de duas maneiras (57): a partir do gráfico de conversão versus temperatura, onde as intersecções das equações dos balanços correspondem aos estados estacionários e a partir do gráfico das taxas de geração/remoção de calor, onde as curvas de geração e remoção de calor também se interceptam nos estados estacionários. Ambos os gráficos possuem mesmo formato e a forma geral está representada na Figura 4.1.

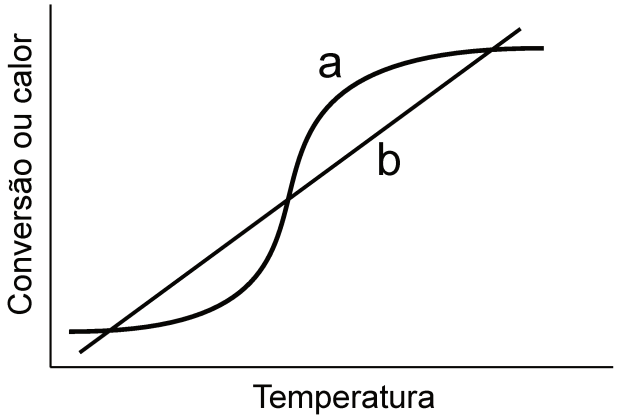


Figura 4.1: Multiplicidade de estados observada nas intersecções onde a curva (a) pode ser o balanço de massa ou calor gerado e (b) balanço de energia ou calor removido.

Carberry (58) cita ainda uma outra perspectiva para análise da existência de multiplicidade, agora sob o ponto de vista da conversão versus tempo de residência (razão

do volume do reator pela vazão volumétrica de alimentação). O gráfico dessas variáveis pode produzir até seis tipos diferentes de padrões, os quais são importantes na discussão da aderência do processo a um dado estado estacionário(mais informações podem ser consultadas na referência (58)). O uso do tempo de residência é conveniente na análise de multiplicidade de estados pela facilidade de manipulação da vazão de alimentação do reator.

Dependendo da vazão do fluido refrigerante no reator (e possivelmente outras variáveis do processo) a curva de remoção de calor pode se deslocar para a direita ou esquerda, provocando alterações nos estados estacionários. A Figura 4.2 ilustra diferentes situações de interceptação da curva de remoção com a curva de geração de calor, dando origem a estados estacionários de temperaturas baixa, intermediária e alta.

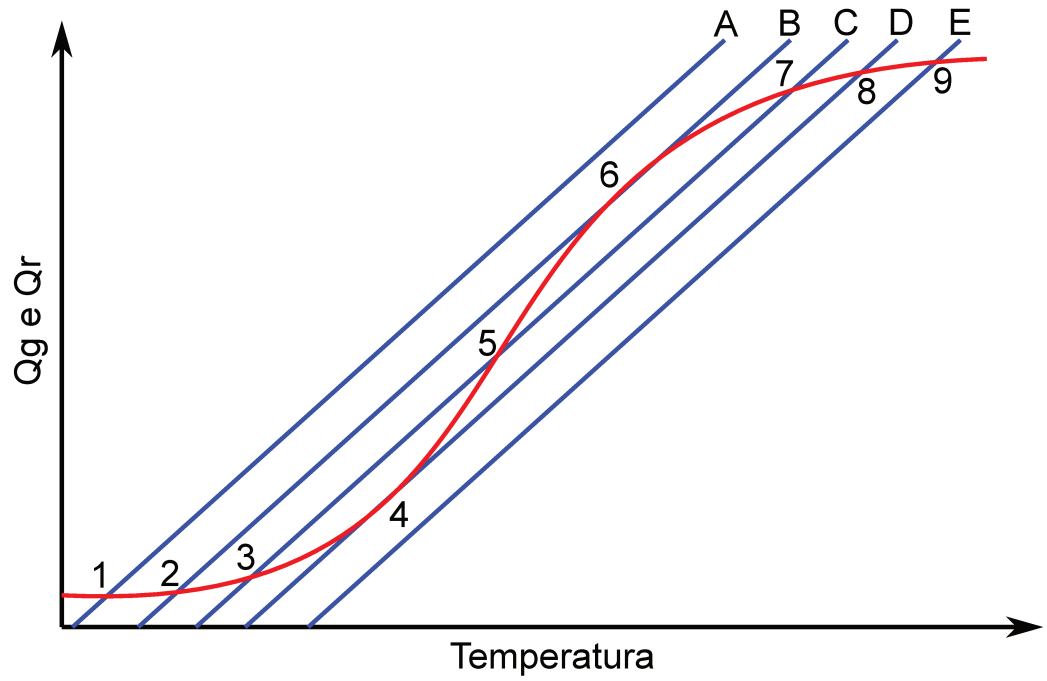


Figura 4.2: Sobreposição de várias curvas de remoção de calor e a curva de geração de calor. Adaptado de (59).

Considerando a curva C na Figura 4.2, se a temperatura do reator for inicialmente menor que a do ponto 3, a curva de geração está acima da curva de remoção e a tendência é o aumento da temperatura até atingir o ponto 3. Se, por outro lado, o reator estiver depois do ponto 7, a curva de remoção está acima da curva de geração, provocando resfriamento até o ponto 7. Utilizando o mesmo raciocínio, temperaturas entre os pontos 3-5 e 5-7 levam aos pontos 3 e 7 respectivamente, ou seja, as temperaturas se afastam do ponto 5. Portanto, pode-se afirmar que os pontos 7 e 3 são estados estacionários estáveis e o ponto

5, instável (59).

Em termos de análise estacionária, é possível projetar o reator para operar em um dos estados estáveis, preferencialmente aquele cuja conversão for superior. Entretanto, existe a dificuldade em termos operacionais de efetuar a partida do equipamento e atingir o estado desejado. Alguns exemplos da literatura, como encontrado em Schmidt *et al* (60), Mohl *et al* (61) e Scenna *et al* (62) ilustram o quão difícil é efetuar a partida de um equipamento com múltiplos estados estacionários.

Curiosamente, Schmidt *et al* (60) recomendam o uso de algoritmos numéricos em modelos de reatores de polimerização para buscar pela existência de estados estacionários de alta conversão em problemas de multiplicidade de estados estacionários. Essa busca pode ser delegada a algoritmos de zero de função, mas também pode de ser feita usando verificação formal, que além de identificar o estado, também permitiria traçar a trajetória que levaria o sistema até esse ponto de operação.

Na sessão a seguir, o método $\delta\pi$ foi empregado para modelar um CSTR com múltiplos estados estacionários como uma máquina de estados finitos. A verificação formal, então, foi utilizada para estudar a dinâmica desse equipamento.

4.2 Modelagem do CSTR com múltiplos estados estacionários

Para demonstrar o uso da verificação formal em um modelo gerado pelo método $\delta\pi$, foi considerado um reator CSTR onde ocorre uma reação simples do tipo A $\rightarrow$ B, com cinética conforme Equação 4.1. Esta taxa de reação foi obtida de um exemplo de multiplicidade de um CSTR de Green e Perry (57).

$$r = \exp\left(25 - \frac{10000}{T}\right) C \tag{4.1}$$

Para modelar o nível, composição e temperatura ao longo do tempo foram desenvolvidos os balanços de massa e energia, conforme as Equações 4.2 até 4.6. Equações 4.2 e 4.3 modelam a vazão volumétrica de descarga do reator e a taxa de transferência de calor da camisa do reator, que é preenchida com fluido refrigerante sob temperatura T_r.

$$Q = c_v\sqrt{h} \tag{4.2}$$

$$Q_h = UA(T_r - T) \tag{4.3}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \frac{Q_0 - Q}{A_T} \tag{4.4}$$

$$\frac{\mathrm{d}C}{\mathrm{d}t} = \frac{C_0 Q_0 - CQ}{V} - r - \frac{\mathrm{d}h}{\mathrm{d}t}\frac{C}{h} \tag{4.5}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{T_0 Q_0 - TQ}{V} - \frac{\Delta H_r r}{\rho c_p} + \frac{Q_h}{V\rho c_p} - \frac{\mathrm{d}h}{\mathrm{d}t}\frac{T}{h} \tag{4.6}$$

O reagente é introduzido diluído no reator em uma proporção de 1:5 com o solvente. As propriedades da solução foram aproximadas pelas propriedades da água por simplificação. As condições operacionais e as constantes do modelo estão resumidas na Tabela 4.1.

Se as condições operacionais forem mantidas conforme a Tabela 4.1, o sistema apresenta três estados estacionários, conforme evidenciado na Figura 4.3: um estado estável na temperatura de 435 K (conversão de 99,8 %), outro estado estável na temperatura de 288,8 K (conversão de 26,1 %) e um estado instável em 326,3 K. A intenção neste estudo de caso é manter o sistema operando na temperatura de 435 K em vista da alta conversão do reagente.

Tabela 4.1: Condições operacionais e constantes do modelo do reator.

Descrição	Símbolo	Valor
Vazão volumétrica de alimentação	Q_0	$0{,}01\,\mathrm{m^3/s}$
Concentração molar do reagente na alimentação	C_0	$10\,\mathrm{mol/m^3}$
Temperatura do fluido refrigerante	T_r	$280\,\mathrm{K}$
Temperatura de alimentação	T_0	$290\,\mathrm{K}$
Constante da válvula	c_v	$0{,}01\,\mathrm{m^{2,5}/s}$
Área transversal do reator	A_T	$1\,\mathrm{m^2}$
Densidade molar do fluido	ρ	$50\,\mathrm{mol/m^3}$
Entalpia de reação	ΔH_r	$-7 \times 10^4\,\mathrm{J/mol}$
Calor específico molar	c_p	$75\,\mathrm{J/molK}$
Coeficiente convectivo multiplicado pela área de troca térmica	UA	$10\,\mathrm{W/K}$

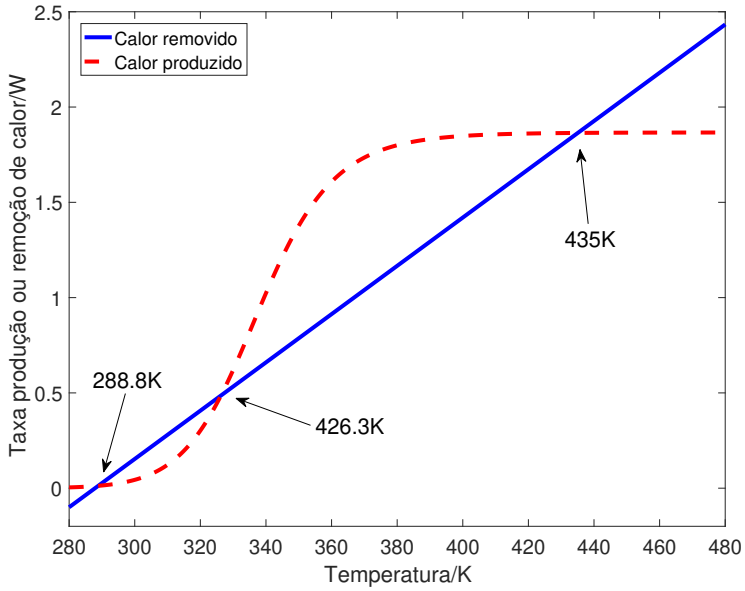


Figura 4.3: Curvas de taxas de produção e remoção de calor, cujas interseções são estados estacionários.

A vazão de alimentação do reagente também influencia na existência dos estados estacionários, conforme apresentado nas Figuras 4.4 e 4.5. A região entre aproximadamente 0-835 s de tempo de residência, por exemplo, possui três estados estacionários. Pelas condições apresentadas na Tabela 4.1 e para o nível de 1 m, o tempo de residência no reator é de 100 s, o que pelas Figuras 4.4 e 4.5 aponta que vazões 8 vezes menores ou 250 vezes maiores em relação a operação nominal ainda estarão dentro da região de multiplicidade.

A Figura 4.6 ilustra o resultado de diferentes condições iniciais da temperatura do reator (com nível mantido em 1 m e concentração inicial do reagente igual a zero, ou seja, o reator possui inicialmente apenas solvente e o reagente diluído é adicionado de forma repentina). O resultado da simulação da máquina de estados que modela esse sistema, sob mesmas condições iniciais, está apresentado na Figura 4.7. Nota-se que o comportamento da multiplicidade de estados estacionários foi herdada pela máquina de estados. A discretização das variáveis está ilustrada na Tabela 4.2.

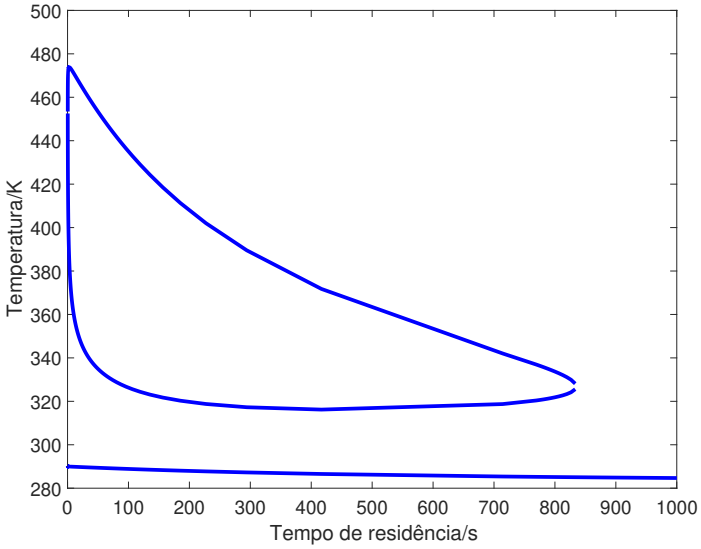


Figura 4.4: Temperaturas de operação no estado estacionário do reator.

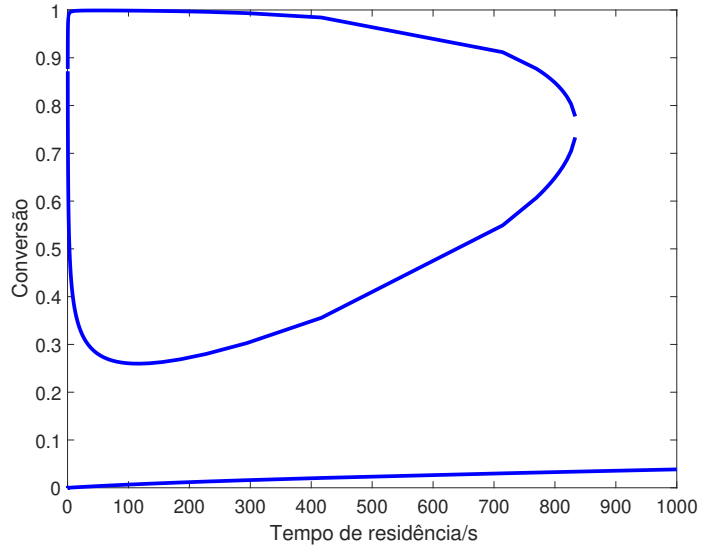


Figura 4.5: Conversão no estado estacionário do reator.

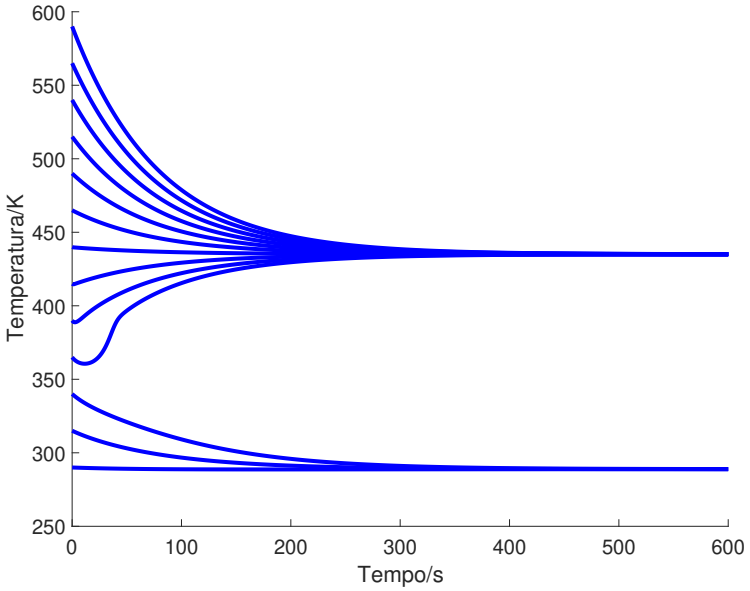


Figura 4.6: Simulação da equação diferencial do reator para diferentes temperaturas inciais, evidenciando os dois estado estacionários.

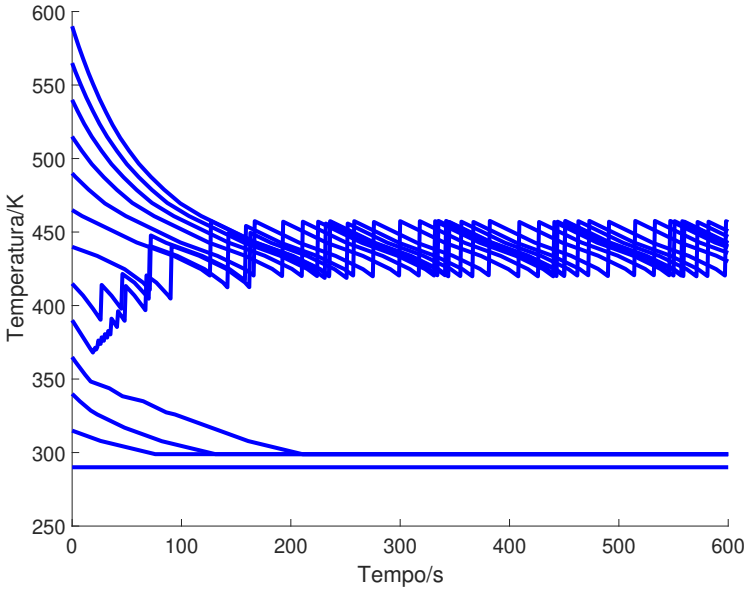


Figura 4.7: Simulação da máquina de estados do reator para diferentes temperaturas inciais, demonstrando que o modelo "herdou" os dois estado estacionários.

As discretizações apresentadas na Tabela 4.2 foram definidas ao comparar o comportamento das simulações da máquina de estados e com as equações diferenciais do reator. A vazão foi discretizada de maneira diferente porque ela não é uma variável de estado, ou seja, ela não possui derivada e por isso não precisa de microestados. O valor inferior do intervalo do nível foi escolhido diferente de zero para evitar divisão por zero no modelo.

Tabela 4.2: Parâmetros utilizados na discretização das variáveis do modelo.

Variável	Discretização	Intervalo
Nível	$n = 5/n_* = 100$	0,001-2 m
Concentração	$n = 5/n_* = 100$	0-10 mol/m^3
Temperatura	$n = 40/n_* = 100$	290-650 K
Vazão	$n = 2$	0-0,02 m^3/s

A temperatura da corrente de alimentação foi discretizada de acordo com cada problema de verificação para reduzir o número de estados do modelo.

As oscilações observadas na Figura 4.7 resultam do esquema de discretização, mas para fins de análise do presente reator elas são aceitáveis, visto que os estados estacionários são facilmente identificáveis.

Apesar da Figura 4.7 demonstrar a existência de múltiplos estados estacionários, a verificação formal também poderia ser usada para buscar por esses estados. Para tanto, é necessário especificar em lógica temporal esses estados estacionários.

4.3 Estados estacionários e lógica temporal

Uma vez que no estado estacionário as variáveis não mudam, as derivadas do modelo que descrevem o sistema são iguais a zero. As máquinas de estados construídas pelo método $\delta\pi$ também possuem estados estacionários, mas em certas circunstâncias quando uma variável não é discretizada suficientemente a máquina o descreve saltando entre dois ou mais estados, como observado na Figura 4.7. Portanto, se a MEF permanece limitada dentre dois estados $\mathbf{y}_1$ e $\mathbf{y}_2$, não necessariamente adjacentes, por um tempo indefinido após uma perturbação, pode-se dizer que ela atingiu o correspondente estado estacionário que seria atingido pelo modelo na forma de equações diferenciais. Il Moon (63) referiu-se a essa situação como um *loop* na máquina de estados e definiu a seguinte propriedade para referir-se a essa situação

$$\mathbf{EF\ EG\ y} \tag{4.7}$$

ou seja, essa propriedade expressa que "existe ao menos um caminho na execução da máquina (**EF**) em que haverá ao menos uma trajetória na qual o sistema sempre perma-

necerá naquele estado **y** (**EG y**)". Visualmente, pode-se fazer uma comparação da árvore de computação da máquina e a variável contínua como ilustrado na Figura 4.8.

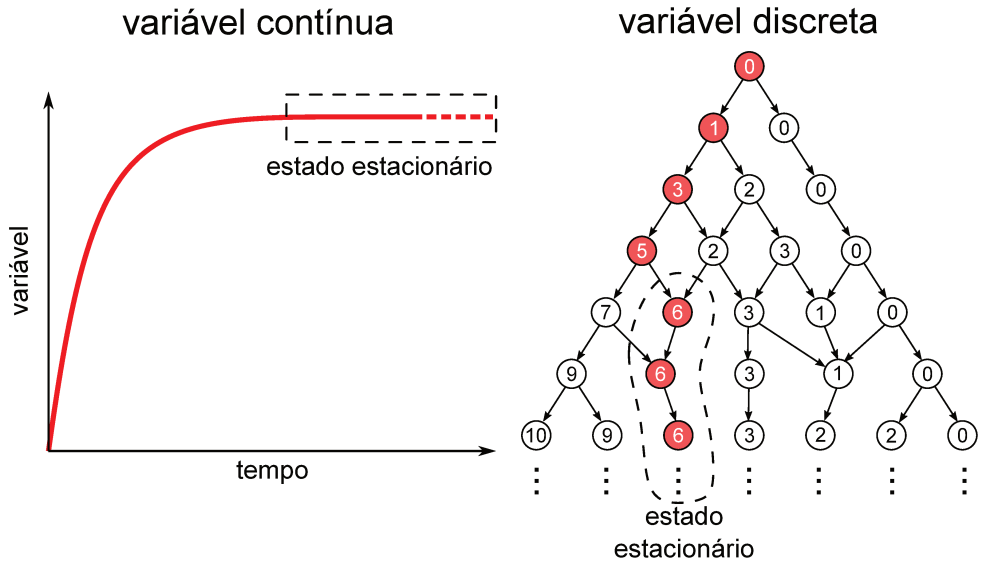


Figura 4.8: Comparativo entre o estado estacionário de uma variável contínua e a parcial execução de sua correspondente máquina de estados, onde os números indicam o macroestado (com destaque para o estado estacionário) e as setas, as possíveis transições.

Para o reator do presente trabalho, o estado estacionário de temperatura alta está limitado no intervalo $\tilde{T} = [14; 18] =]13; 19[$[1] (entre aproximadamente 409-450 K) e o de temperatura baixa em $\tilde{T} = [0; 2] = [0; 3[$ (entre aproximadamente 280-307 K). O intervalo da temperatura baixa foi definido pelo que se observa na Figura 4.7, onde temperaturas ligeiramente acima da temperatura de 290 K convergem na MEF para $\tilde{T} = 2$, mas se a temperatura inicial for $T = 290$, a MEF permanece em $\tilde{T} = 0$.

Suponha que apenas um dos estados estacionários é conhecido, por exemplo, o de temperatura baixa. Conhecendo-se ao menos um estado estacionário é possível localizar todos os outros múltiplos estados estáveis (se existirem) usando a verificação formal. Para isso, o modelo do reator foi modificado de modo que a temperatura inicial não foi fixada e o NuSMV testaria todas as temperaturas iniciais possíveis. A temperatura da corrente de alimentação foi definida em $T = 290$ K. Lançou-se mão da propriedade

$$\mathbf{AF\ EG}\ \tilde{T} < 3$$

cujo significado é "inevitavelmente todos os caminhos na execução da máquina levarão no futuro (**AF**) a pelo menos uma trajetória na qual o sistema sempre permanecerá no

[1] Essa igualdade é verdadeira porque $\tilde{T}$ é um número inteiro.

estado $\tilde{T} < 2$", uma vez existindo outros estados, essa propriedade será apontada como falsa e o contraexemplo indicará uma temperatura inicial do reator que o leve até o outro estado estacionário. Essa propriedade é ligeiramente diferente da apresentada na Equação 4.7 para expressar uma maior abrangência da existência do estado estacionário de temperatura baixa, de modo que ao contrário da propriedade da Equação 4.7 deseja-se neste momento verificar se **todas as execuções** da máquina de estados levarão em algum momento ao estado estacionário de temperatura baixa.

Após 868 s em um computador AMD A10 com 12 GB de memória RAM, o NuSMV avisou que a propriedade era de fato falsa e indicou que se a temperatura inicial for de $\tilde{T} = 31$ (569 K) o reator iria para um estado estacionário diferente daquele de temperatura baixa (Figura 4.9). Observa-se pela Figura 4.6 que temperaturas mais baixas levariam a esse mesmo estado estacionário, mas para a verificação o importante é encontrar um contraexemplo que prove que a propriedade é falsa.

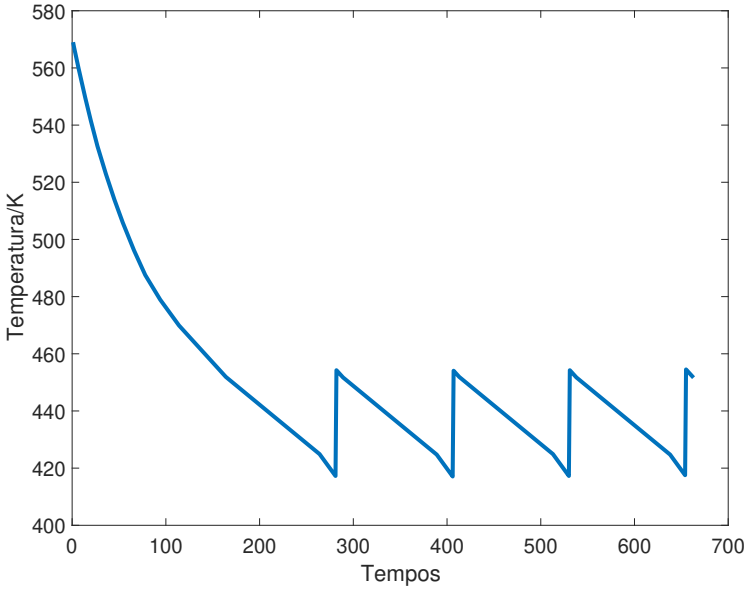


Figura 4.9: Prova da existência de um estado estacionário estável diferente do estado de temperatura baixa.

Nota-se pelo esforço computacional que para esse tipo de análise (existência de múltiplos estados) a verificação talvez não seja a melhor ferramenta, pois já foi demonstrado sem muita dificuldade na Figura 4.6 a existência dos dois estados. Porém, essa propriedade poderia ser usada em análises mais complexas, variando-se mais de uma condição inicial para definir "regiões de condições iniciais" que sempre levariam ao estado desejado.

Supondo que agora os dois estados são conhecidos, foi verificada a possibilidade da existência de um terceiro estado estável com a propriedade

$$\mathbf{AF\ EG}\ (\tilde{T} < 2) \vee (\tilde{T} > 13 \wedge \tilde{T} < 19),$$

que foi verificada como verdadeira pelo NuSMV, indicando que estes são os únicos estados estacionários estáveis para as condições operacionais estabelecidas na Tabela 4.1.

Uma vez que os estados eram conhecidos, pode-se determinar suas alcançabilidades. As verificações feitas deste ponto até o final do trabalho consideraram a variável manipulada como sendo a temperatura da corrente de alimentação porque é possível ter uma maior flexibilidade (e um modelo mais simples) para modificar temperatura do reator usando apenas uma variável. Se a variável manipulada fosse a vazão de fluido refrigerante na camisa do reator, seria necessário supor a existência de outra variável manipulada para efetuar o aquecimento. As transições da temperatura da alimentação foram modeladas como uma máquina não-determinista, ou seja, a cada passo do tempo, ela pode assumir qualquer valor dentro do seu domínio.

Em termos de alcançabilidade, a propriedade

$$\mathbf{AG\ EF}\ \tilde{T} > 13 \wedge \tilde{T} < 19$$

indica a possibilidade do sistema sempre chegar no futuro na situação em que a condição $\tilde{T} > 13 \wedge \tilde{T} < 19$ é verdadeira, independente do estado atual. Essa propriedade não indica que sempre é possível alcançar o estado estacionário, mas a possibilidade de pelo menos passar durante o transiente por essa faixa de temperaturas. O NuSMV indicou que essa propriedade é verdadeira, o que é um bom indicativo de que talvez, se mantido o nível do reator em 1 m, inicialmente sem reagente e frio, sempre é possível manipular a temperatura de alimentação para levar o reator próximo do estado estacionário de temperatura alta.

Por outro lado, a propriedade

$$\mathbf{AG\ EF}\ \tilde{T} < 2$$

foi apontada como falsa pelo NuSMV, retornando um contraexemplo em que a temperatura de alimentação foi mantida em aproximadamente 363 K e o reator avançou para o estado de temperatura alta. Como não houve nenhuma outra mudança na temperatura de alimentação, isso indica que não é possível retornar para estado de temperatura baixa

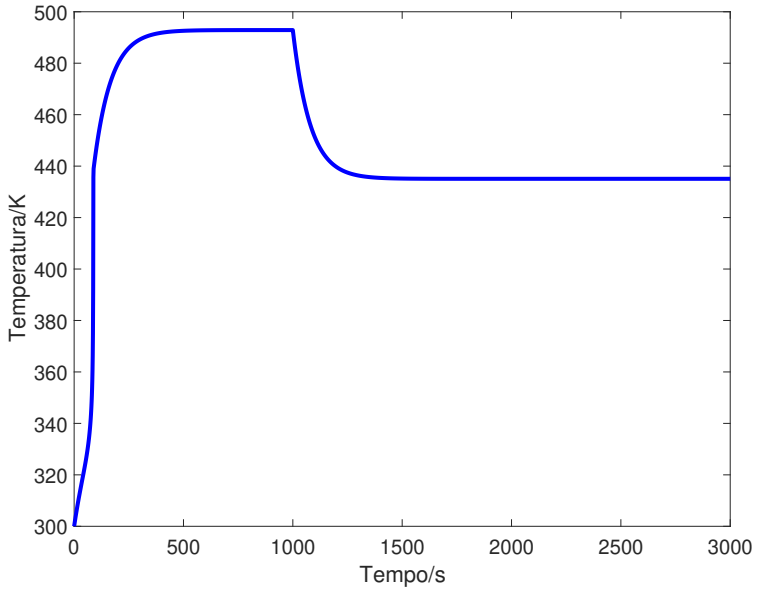


Figura 4.10: Perfil de temperatura do reator quando a alimentação é inicialmente 363 K e depois alterada para 290 K em $t = 1000$ s.

alterando apenas a temperatura de alimentação. Esse fato é observado fazendo-se a simulação do sistema (com as EDOs) com temperatura de alimentação inicial de 363 K e aos 1000 s alterando-se a temperatura para 290 K. Conforme Figura 4.10, a temperatura do reator diminui, mas se mantém no estado estacionário mais alto.

Portanto, se a temperatura de alimentação mais baixa possível for de 290 K, o reator não seria capaz de sair do estado estacionário de temperatura alta, devendo-se recorrer a outras variáveis como a vazão de fluido refrigerante da camisa do reator, para resfriar suficientemente o meio reacional.

4.4 Partida do equipamento

Uma outra interessante aplicação da verificação formal é que a negação da propriedade da Equação 4.7 tem o potencial de ser usada para traçar a trajetória de partida do equipamento até o estado especificado, posto que ela expressaria que "não existe nenhum caminho na execução da máquina em que haverá ao menos uma trajetória na qual o sistema sempre permanecerá naquele estado $\mathbf{y}$". Uma vez que a variável manipulada nesta análise é a temperatura da corrente de alimentação, o contraexemplo conteria o perfil da temperatura da alimentação para chegar no estado desejado.

Para a análise da partida do reator, a temperatura de alimentação foi discretizada com

$n = 2$. O valor superior do intervalo da temperatura de alimentação foi 436 K e o inferior 290 K, para que durante a execução do programa fossem avaliadas as temperaturas de 290 K e 363 K (90 °C), esta última é um valor arbitrário, mas factível do ponto de vista prático, para evitar variações instantâneas absurdas.

Suponha que inicialmente o reator esteja com nível de 1 m, sem reagente e frio. Pode-se escrever as propriedades para cada estado estacionário como:

$$\neg\mathbf{EF}\ \mathbf{EG}\ \tilde{T} < 2$$

$$\neg\mathbf{EF}\ \mathbf{EG}\ (\tilde{T} > 13 \wedge \tilde{T} < 19).$$

Como explicado anteriormente, o estado estacionário de interesse é o de temperatura alta, visto que a conversão é mais alta, portanto maior atenção foi dada a essa condição. Para garantir que a trajetória encontrada seria atingida com a temperatura de alimentação mínima, a propriedade foi modificada para

$$\neg\mathbf{EF}\ \mathbf{EG}\ (\tilde{T} > 13 \wedge \tilde{T} < 19 \wedge \tilde{T_0} = 0)$$

reforçando as condições de discretização usadas, $T_0 = 290$ K. A verificação feita pelo NuSMV indicou o perfil de temperatura de alimentação mostrado na Figura 4.11, na forma de dois degraus: o primeiro para a temperatura mais alta de 363 K e o seguinte de volta para 290 K, 85 s depois. Durante a transição de volta para 290 K o contraexemplo fornecido pelo NuSMV apresentou algumas oscilações entre os dois patamares de temperatura, mas que foram descartados por não terem se mostrado importantes na partida do reator.

A verificação dessa propriedade durou aproximadamente 267 s. Em termos de consumo de memória RAM foi observado um pico de 1,8 GB consultando-se periodicamente o Monitor de Recursos do Windows. Portanto, se o modelo fosse mais complexo, memória poderia se tornar um problema em computadores com recursos mais limitados.

A temperatura do reator de acordo com a MEF do reator está ilustrada na Figura 4.12 e juntamente com a Figura 4.11 vê-se que a temperatura da alimentação só permaneceu no valor mais alto tempo suficiente para que o reator pudesse transitar sozinho para o estado mais quente. A temperatura a partir da qual o reator transita naturalmente para o estado estacionário mais quente é chamada de temperatura de ignição (59).

A simulação das equações diferenciais corrobora que se a alimentação for mantida sob 363 K por 85 s, a temperatura do reator é elevada suficientemente até a ignição. O perfil

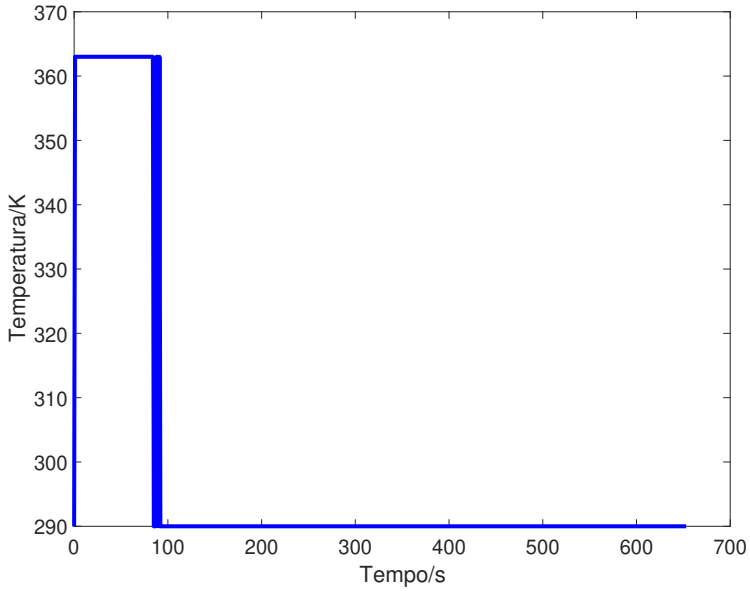


Figura 4.11: Perfil de temperatura da alimentação encontrado pela verificação do modelo.

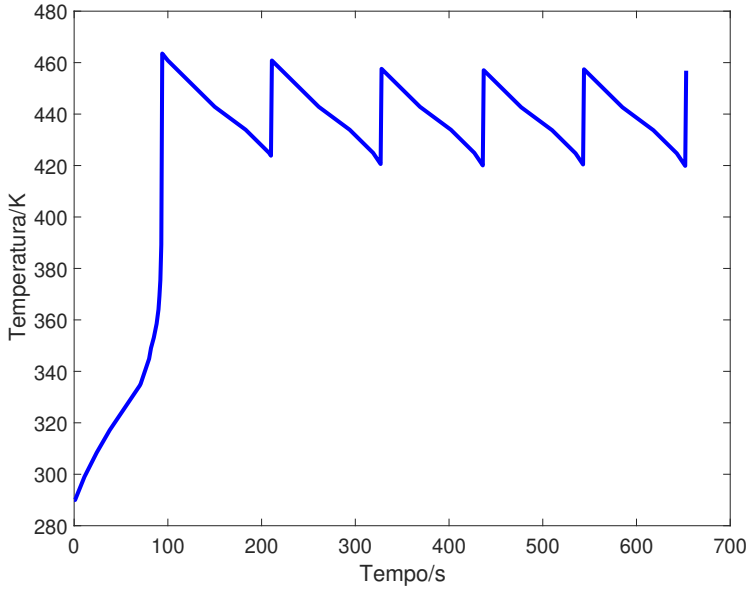


Figura 4.12: Perfil de temperatura do reator durante a partida até o estado estacionário de temperatura alta.

de temperatura utilizado na Figura 4.13 desconsidera as oscilações observadas no final do segundo degrau de temperatura da alimentação.

Assim, o procedimento de partida nestas condições seria:

1. Circular o fluido refrigerante com a vazão prevista para o estado estacionário

2. Encher o tanque com solvente até o nível de operação (1 m)

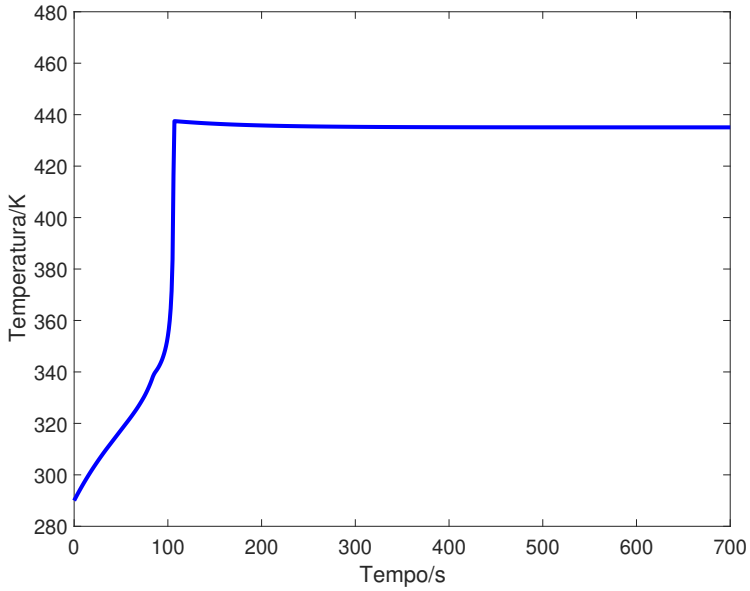


Figura 4.13: Perfil de temperatura do reator simulando as EDOs e com o perfil da temperatura de alimentação da Figura 4.11.

3. Alimentar o reagente em temperatura ambiente

4. Aquecer a alimentação até 363 K por 85 s

5. Desligar o aquecimento e aguardar o sistema atingir o estado estacionário

Em termos operacionais, um aquecimento rápido em curto período de tempo pode ser feito com resistores elétricos adequadamente dimensionados para o processo.

A última situação analisada com a verificação formal neste reator foi sua partida inicialmente vazio (ao invés de preenchido com solvente). As mudanças feitas no programa do NuSMV para essa verificação foram a alteração da condição inicial do nível para 0,001 m e a vazão de alimentação foi fixada em 0,01 m^3/s. A propriedade foi mantida a mesma.

A primeira tentativa de verificação desse novo modelo foi de aproximadamente 32 h, porém, notou-se posteriormente que este tempo (contabilizado pelo próprio NuSMV) pode estar errado, visto que o programa passou várias horas aguardando o comando do usuário para exibir o contraexemplo. Uma segunda tentativa mostrou que em menos de 5 h o NuSMV indica que a propriedade é falsa, mas levaria-se ainda algum tempo (não contabilizado) para produzir o devido contraexemplo. Não se determinou o porquê da produção do contraexemplo demorar tanto tempo nesta verificação. O consumo de memória RAM foi de aproximadamente 2 GB (observado posteriormente a produção do contraexemplo).

O perfil de temperatura da alimentação obtido pela verificação quando o tanque está enchendo está ilustrado na Figura 4.14. Trata-se de um perfil sensivelmente diferente do que foi encontrado quando o nível estava fixado. Na realidade, este perfil é impraticável porque os picos de temperatura são muito curtos para serem implementados em um processo real (a temperatura não conseguiria variar tão rápido). Apesar da viabilidade ser pequena, o perfil aparenta ser eficaz ao direcionar tanto o modelo na forma de MEF (Figura 4.15) quanto as EDOs (Figura 4.16) para o estado estacionário de temperatura alta.

Os picos poderiam ser evitados alterando-se o modelo diretamente na linguagem do NuSVM para evitar que ocorram transições da temperatura em períodos de tempo tão curtos.

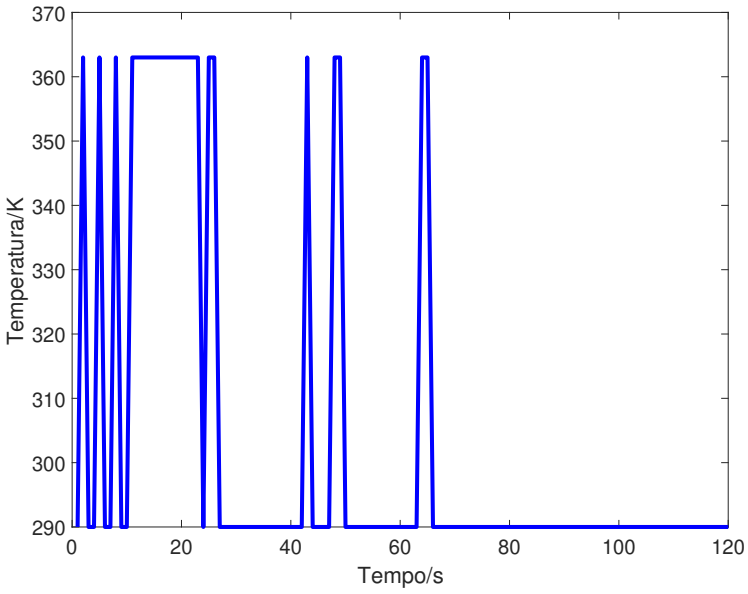


Figura 4.14: Perfil de temperatura da alimentação que leva ao estado estacionário de temperatura alta quando o nível não está fixado.

Para contornar a dificuldade imposta pela execução do perfil de temperatura da Figura 4.14, optou-se por uma simplificação ao transformá-la em dois degraus semelhante ao que foi visto na verificação com nível fixo. O primeiro degrau iria até 363 K no instante inicial e outro de volta para 290 K, 27 s depois, pois nessa região a temperatura de alimentação transita com maior frequência até o patamar mais alto (Figura 4.17). Os três últimos picos foram descartados.

O perfil simplificado aparenta produzir resultados semelhantes ao perfil sem simplificação, de modo que o sistema continua se dirigindo ao estado estacionário de temperatura

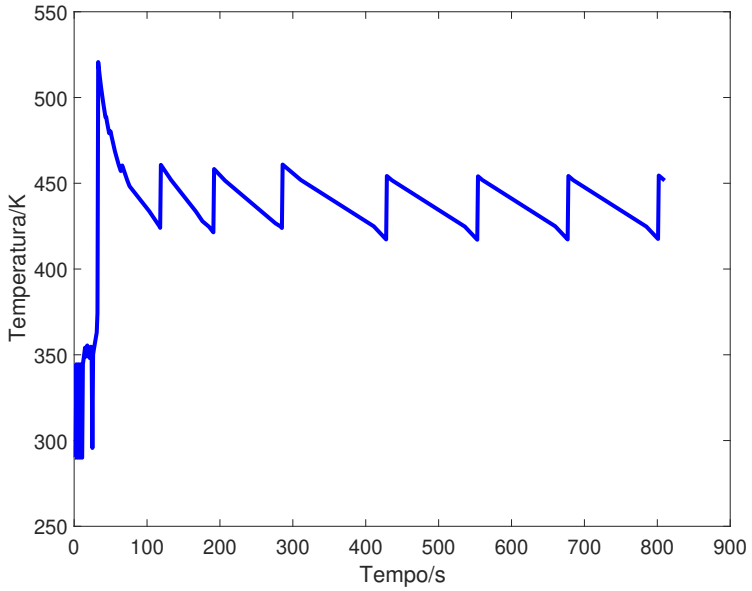


Figura 4.15: Temperatura do reator quando a temperatura da alimentação segue a trajetória da Figura 4.14 (modelo em MEF).

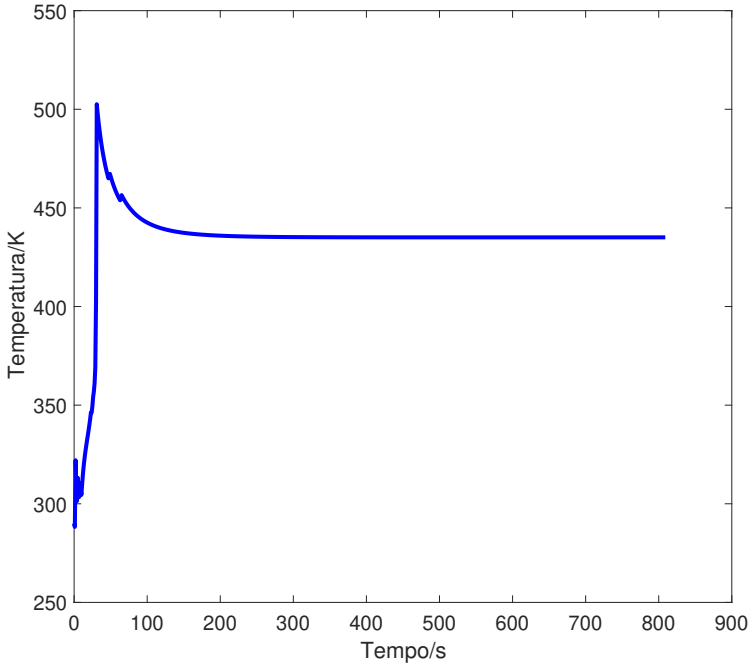


Figura 4.16: Temperatura do reator quando a temperatura da alimentação segue a trajetória da Figura 4.14 (sistema de EDOs).

alta, conforme observa-se na Figura 4.18.

Portanto, o procedimento de partida quando o reator está frio e vazio seria:

1. Circular o fluido refrigerante com a vazão prevista para o estado estacionário

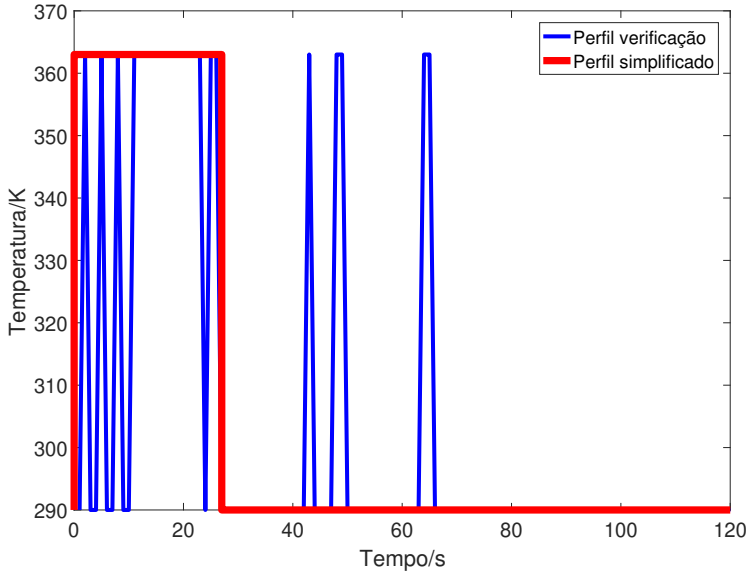


Figura 4.17: Simplificação do perfil de temperatura da alimentação.

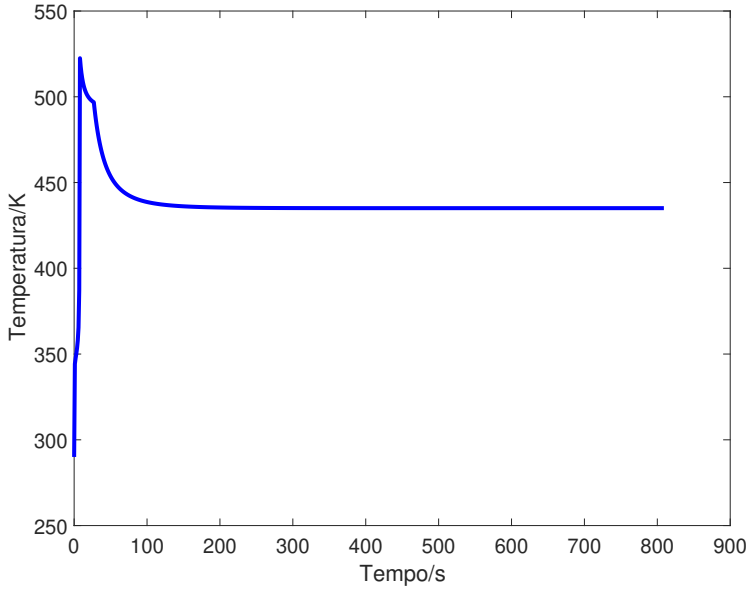


Figura 4.18: Simulação com EDOs da temperatura do reator usando o perfil simplificado.

2. Iniciar o enchimento do tanque com o reagente até o nível de operação (1 m)

3. Simultaneamente com o enchimento, aquecer a alimentação até 363 K por 27 s

4. Desligar o aquecimento e aguardar o sistema atingir o estado estacionário

De maneira semelhante, outras variáveis também poderiam ser acompanhadas a fim de analisar o perfil para partida de um determinado processo, como a vazão de fluido refrigerante.

Finalmente, deve-se ressaltar que a verificação de modelos não garante a partida mais eficiente (em termos de alguma função objetivo) porque não se trata de uma otimização, mas os resultados encontrados são interessantes do ponto de vista operacional porque dão um indicativo de como poderia ser o procedimento geral para partida do processo.

capítulo 5

Conclusões

O desenvolvimento de modelos de sistemas contínuos na forma de autômatos finitos tem sido um desafio nas últimas décadas para cientistas e engenheiros da computação. Diversos trabalhos foram publicados para diferentes classes de autômatos e uma sistematização simples o suficiente para construção dos modelos para engenheiros que não são da área da computação facilitaria a exploração da verificação formal em áreas como a engenharia química.

Neste sentido, o presente trabalho é mais um passo na tentativa de sistematizar o desenvolvimento de modelos que sejam adequados para verificação formal, especificamente de processos químicos. A metodologia aqui apresentada e chamada de $\delta\pi$ tem se mostrado promissora no que diz respeito à simplicidade de implementação, tendo sido demonstrada neste trabalho sua aplicação em sistemas de EDOs, EDOs de 2^{a} ordem e até EDPs, com equações invariantes no tempo. Também foi exemplificada construção da máquina diretamente a partir de dados experimentais. Ao contrário de vários exemplos encontrados na literatura que se limitavam, em geral, a simples tanques de mistura, demonstrou-se a aplicação na modelagem do nível de um tanque (que é uma equação não-linear), dinâmica de válvulas de controle, reator CSTR e até a composição dentro de uma partícula porosa.

Apesar da complexidade dos modelos, na prática, a técnica necessita apenas de informações sobre a dinâmica do processo codificada na forma de uma matriz de salto. Essa matriz pode ser construída a partir de equações diferenciais ou de outras fontes de conhecimento da dinâmica do processo. Além da matriz de salto a construção da máquina de estados com o método $\delta\pi$ ainda necessita do *range* de cada variável, o número de macro

e microestados e o passo de tempo. Essas informações são específicas de cada modelo e não exigem conhecimento de verificação formal, o que é uma vantagem para quem não é especialista em verificação.

O método $\delta\pi$, neste sentido, abre a possibilidade de desenvolvimento das máquinas de estados a partir de modelos já implementados em simuladores de processos, o que impactaria positivamente no tempo de desenvolvimento de programas de verificação especializados em processos industriais.

Foi apresentado ainda neste trabalho um estudo de caso onde foi desenvolvido o modelo de máquinas de estados de um reator CSTR com múltiplos estados estacionários. Demonstrou-se que a máquina resultante herdou a característica de multiplicidade e foi possível verificar o modelo a fim de comprovar alcançabilidade dos estados e até a geração de procedimentos de partida do equipamento. Em algumas situações a verificação apresentou um consumo considerável de memória RAM que foi contornado com diminuição da discretização de algumas variáveis.

Apesar de não ser a forma mais eficiente de estudar o comportamento de sistemas com múltiplos estados estacionários, a metodologia $\delta\pi$ abre a possibilidade de empregar a verificação formal em aplicações ainda não relatadas na literatura.

Talvez a principal limitação da abordagem $\delta\pi$ identificada até o presente momento é a necessidade de um certa quantidade de testes para ajuste das discretizações, uma vez que não foi possível determinar regras práticas para a decisão a respeito dessas variáveis ainda. O número de estados também cresce rapidamente se a quantidade de faixas discretizadas for aumentada exageradamente, o que poderia comprometer a verificação. Assim, existe uma decisão a ser tomada entre qualidade do modelo e viabilidade em termos de tempo de verificação.

Há limitações da técnica também em termos de descontinuidades, por exemplo, situações onde ocorra divisão por zero. Neste caso, a discretização pode ser planejada para que o modelo não seja avaliado nestes pontos durante a construção da máquina de estados.

Em termos de implementação da técnica, a etapa de pré-processamento do modelo, que contempla a solução das equações diferenciais, foi implementada em Matlab e a etapa de verificação/simulação da máquina de estados foi feita no NuSMV. O NuSMV foi escolhido como *software* de verificação pela facilidade de implementação dos modelos em sua linguagem e pela possibilidade de executar programas a partir de comandos externos, ou

seja, a partir do Matlab era possível chamar o executável do NuSMV e fazê-lo executar comandos específicos para carregar o modelo, a verificação e exportar os resultados em arquivos de texto. Porém, a metodologia proposta neste trabalho pode ser implementada em praticamente qualquer linguagem de programação e *software* de verificação, basta que este último dê suporte à máquinas estendidas.

Acredita-se que o método $\delta\pi$ possa ser melhorado se outras estratégias para discretização das variáveis forem empregadas ao invés de discretizá-la em intervalos iguais, melhor equacionamento para as transições reduzindo o total de quatro por máquina e até a eventual eliminação da máquina $\mathcal{M}_\delta$ e substituição por alguma outra abordagem mais simples, o que contribuiria na redução do número de estados do modelo.

A análise mais aprofundada no ordenamento de variáveis dos diagramas de decisão binárias do NuSMV para modelos construídos com o método $\delta\pi$ também pode contribuir para construção de modelos mais eficientes em termos de verificação (conforme Johnston *et al* (64), o ordenamento de variáveis pode impactar fortemente no consumo de memória e eficiência da verificação).

Olhando para o futuro, certamente ainda há muita margem para pesquisa em verificação formal na engenharia química, especialmente no uso de máquinas híbridas, que tem se mostrado foco de diversas pesquisas na ciência da computação. Durante o desenvolvimento deste trabalho, o direcionamento sempre foi a construção de máquinas simples, visto que a adição de maior complexidade ao modelo (tempo, variáveis contínuas, probabilidades, etc.) impactaria na sua implementação e verificação. Entretanto, novas ferramentas têm ganhado destaque nos últimos anos e elas podem auxiliar em pesquisas futuras na engenharia química.

Uma dessas ferramentas é o S-taliro, baseada em modelos híbridos. Esse *software* verifica modelos implementados no Simulink, que já é utilizado há muitos anos na engenharia química. Se por um lado isso auxiliaria na implementação do modelo, por outro, não foi identificada a existência de padrões de propriedades para a lógica MTL do S-taliro, o que parece ser um grande entrave, uma vez que a escrita da propriedade é um gargalo na verificação de modelos.

Por fim, o presente trabalho é uma contribuição não apenas para a engenharia química, mas também para a engenharia e ciência da computação, dada a dificuldade de modelagem de sistemas físicos e integração com os modelos de *hardware* e *software*. A importância

dessa integração é tão grande que cunhou-se o termo sistemas ciberfísicos (*Cyber-physical systems*) e cujas pesquisas nessa área são destinadas a estudar a interação entre os elementos que constituem o sistema físico com os elementos de *software* dos dispositivos desse sistema. Mesmo que a metodologia $\delta\pi$ não seja de alta eficiência para verificação, ainda se mantém demonstrada uma rota alternativa para construção de máquinas de estados de forma automática.

Referências Bibliográficas

1 BODESTEDT, O. *Dictionary Industrial Automation and Control*. [S.l.]: Studentlitteratur, 1993.

2 MOON, I.; POWERS, G. J.; BURCH, J. R.; CLARKE, E. M. Automatic verification of sequential control systems using temporal logic. *AIChE Journal*, Wiley Online Library, v. 38, n. 1, p. 67–75, 1992.

3 BAIER, C.; KATOEN, J. *Principles of Model Checking*. [S.l.]: MIT Press, 2008.

4 SIPSER, M. *Introdução à teoria da computação*. 2. ed. [S.l.]: Cengage Learning, 2007.

5 BéRARD, B.; BIDOIT, M.; FINKEL, A.; LAROUSSINIE, F.; PETIT, A.; PETRUCCI, L.; MCKENZIE, P. S. P. *Systems and software verification*: Model-checking techniques and tools. [S.l.]: Springer, 2001.

6 CASSANDRAS, C. G.; LAFORTUNE, S. *Discrete event systems*. [S.l.]: Springer, 2008.

7 MACHADO, J.; DENIS, B.; LESAGE, J.-J. Formal verification of industrial controllers: with or without a plant model? In: LISBOA. *Proc. of 7th Portuguese Conference on Automatic Control, CONTROLO*. [S.l.], 2006. v. 6.

8 ALUR, R. *Principles of cyber-physical systems*. [S.l.]: The MIT Press, 1966.

9 CHENG, K. T.; KRISHNAKUMAR, A. S. Automatic functional test generation using the extended finite state machine model. In: ACM. *Proceedings of the 30th international Design Automation Conference*. [S.l.], 1993. p. 86–91.

10 PARK, T.; BARTON, P. I. Formal verification of sequence controllers. *Computers & Chemical Engineering*, Elsevier, v. 23, n. 11, p. 1783–1793, 2000.

11 NORMAN, G.; PARKER, D. *Quantitative Verification: Formal Guarantees for Timeliness, Reliability and Performance*. [S.l.], 2014.

12 BEHRMANN, G.; DAVID, A.; LARSEN, K. G. *A Tutorial on Uppaal 4.0*. 2006. Http://www.uppaal.com/admin/anvandarfiler/filer/uppaal-tutorial.pdf. Acesso em: 02 de junho de 2015. Disponível em: ⟨http://www.uppaal.com/admin/anvandarfiler/filer/uppaal-tutorial.pdf⟩.

13 KWIATKOWSKA, M.; NORMAN, G.; PARKER, D. Prism: Probabilistic symbolic model checker. In: *Computer performance evaluation: modelling techniques and tools*. [S.l.]: Springer, 2002. p. 200–204.

14 RABINER, L. R.; JUANG, B.-H. An introduction to hidden markov models. *ASSP Magazine, IEEE*, IEEE, v. 3, n. 1, p. 4–16, 1986.

15 ALUR, R. Formal verification of hybrid systems. In: IEEE. *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*. [S.l.], 2011. p. 273–278.

16 HENZINGER, T. A. *The theory of hybrid automata*. [S.l.]: Springer, 2000.

17 RILEY, D.; KOUTSOUKOS, X. Probabilistic verification of a biodiesel production system using statistical model checking. *Mathematical and Computer Modelling of Dynamical Systems*, Taylor & Francis, v. 20, n. 5, p. 452–469, 2014.

18 FISHER, M. *An introduction to practical formal methods using temporal logic*. [S.l.]: Wiley, 2011.

19 BISPO, C. A. F. *Introdução a Lógica Matemática*. [S.l.]: Cengage CTP, 2011.

20 PARKER, D. *Probabilistic temporal logics*. 2011. Http://www.prismmodelchecker.org/lectures/pmc/04-prob_logics.pdf. Acesso em: 10 de junho de 2015. Disponível em: ⟨http://www.prismmodelchecker.org/lectures/pmc/04-prob_logics.pdf⟩.

21 GALLEGOS, I.; OCHOA, O.; GATES, A. Q.; ROACH, S.; SALAMAH, S.; VELA, C. A property specification tool for generating formal specifications: Prospec 2.0. In: *SEKE*. [S.l.: s.n.], 2008. p. 273–278.

22 HANAZUMI, S.; MELO, A. C. V. de; PĂSĂREANU, C. S. From test purposes to formal jpf properties. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 40, n. 1, p. 1–5, 2015.

23 DWYER, M. B.; AVRUNIN, G. S.; CORBETT, J. C. Patterns in property specifications for finite-state verification. In: IEEE. *Software Engineering, 1999. Proceedings of the 1999 International Conference on*. [S.l.], 1999. p. 411–420.

24 ALAVI, H.; AVRUNIN, G.; CORBETT, J.; DILLON, L.; DWYER, M.; PASAREANU, C. *Property Pattern Mappings for CTL*. 2017.

25 BAIER, C.; KATOEN, J.-P. *Principles of model checking*. [S.l.]: MIT Press, 2008.

26 LIGGESMEYER, P.; ROTHFELDER, M.; RETTELBACH, M.; ACKERMANN, T. Qualitätssicherung software-basierter technischer systeme–problembereiche und lösungsansätze. *Informatik-Spektrum*, v. 21, n. 5, p. 249–258, 1998.

27 WOODCOCK, J.; LARSEN, P. G.; BICARREGUI, J.; FITZGERALD, J. Formal methods: Practice and experience. *ACM Computing Surveys (CSUR)*, ACM, v. 41, n. 4, p. 19, 2009.

28 RAY, S. *Scalable techniques for formal verification*. [S.l.]: Springer, 2010.

29 SILVA, A. de M. *Aplicação de verificação de modelos a programas de CLP: explorando a temporização*. 121 p. Tese (Mestrado) — Instituto Militar de Engenharia, Rio de Janeiro, 2008.

30 GOVIL, N.; AGRAWAL, A.; TIPPENHAUER, N. O. On ladder logic bombs in industrial control systems. *arXiv preprint arXiv:1702.05241*, 2017.

31 KOTTLER, S.; KHAYAMY, M.; HASAN, S. R.; ELKEELANY, O. Formal verification of ladder logic programs using nusmv. In: IEEE. *SoutheastCon, 2017*. [S.l.], 2017. p. 1–5.

32 RAWLINGS, B. C.; KIM, J.; MOON, I.; YDSTIE, B. E. Symbolic verification of control systems and operating procedures. *Industrial & Engineering Chemistry Research*, ACS Publications, v. 53, n. 13, p. 5299–5310, 2014.

33 SAMPAIO, L. R. *Validação visual de programas Ladder baseada em modelos*. 78 p. Tese (Mestrado) — Universidade Federal de Campina Grande, Campina Grande, 2011.

34 TURK, A. L. *Event modeling and verification of chemical processes using symbolic model checking*. 181 p. Tese (PhD) — Carnegie Mellon University, Pittsburgh, 1999.

35 CLARKE, E.; GRUMBERG, O.; JHA, S.; LU, Y.; VEITH, H. Progress on the state explosion problem in model checking. In: SPRINGER. *Informatics*. [S.l.], 2001. p. 176–194.

36 NUSMV. *About UPPAAL*. 2018. Disponível em: ⟨http://nusmv.fbk.eu/⟩.

37 UPPAAL. *About UPPAAL*. 2015. Http://www.uppaal.org/. Acesso em: 15 de junho de 2015. Disponível em: ⟨http://www.uppaal.org/⟩.

38 KWIATKOWSKA, M.; NORMAN, G.; PARKER, D. PRISM 4.0: Verification of probabilistic real-time systems. In: GOPALAKRISHNAN, G.; QADEER, S. (Ed.). *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*. [S.l.]: Springer, 2011. (LNCS, v. 6806), p. 585–591.

39 MATHWORKS. *CheckMate verification tool*. 2015. Http://www.mathworks.com/matlabcentral/fx_files/15441/3/content/ doc/-verify/verify.htm. Acesso em: 15 de junho de 2015. Disponível em: ⟨http://www.mathworks.com/matlabcentral/fxfiles/15441/3/content/doc/verify/verify.htm⟩.

40 BARNER, S.; BEN-DAVID, S.; GRINGAUZE, A.; STERIN, B.; WOLFSTHAL, Y. An algorithmic approach to design exploration. In: *FME 2002: Formal Methods?Getting IT Right*. [S.l.]: Springer, 2002. p. 146–162.

41 MARGOLIS, D. P. *A methodology for synthesizeing logic models of operating procedures for process verification*. 196 p. Tese (PhD) — Carnegie Mellon University, Pittsburgh, 2003.

42 ANTONIOTTI, M.; POLICRITI, A.; UGEL, N.; MISHRA, B. Model building and model checking for biochemical processes. *Cell biochemistry and biophysics*, Springer, v. 38, n. 3, p. 271–286, 2003.

43 CARRILLO, M.; GÓNGORA, P. A.; ROSENBLUETH, D. A. An overview of existing modeling tools making use of model checking in the analysis of biochemical networks. *Frontiers in plant science*, Frontiers Media SA, v. 3, 2012.

44 MILAM, D. E. *Synthesizing modular logic models of chemical engineering process equipment and control systems for verification*. 169 p. Tese (PhD) — Carnegie Mellon University, Pittsburgh, 2003.

45 BLOUIN, S. *Finite-state machine abstractions of continuous systems*. 179 p. Tese (PhD) — Queenś University, Ontario, 2003.

46 THOMBRE, M. *Application of Discrete-Event Dynamic Systems in Plant Analysis and Control*. 111 p. Tese (PhD) — Norwegian University of Science and Technology, Trondheim, 2017.

47 RADULESCU, O.; SAMAL, S. S.; NALDI, A.; GRIGORIEV, D.; WEBER, A. Symbolic dynamics of biochemical pathways as finite states machines. In: SPRINGER. *International Conference on Computational Methods in Systems Biology*. [S.l.], 2015. p. 104–120.

48 MEENAKSHI, B.; BHATNAGAR, A.; ROY, S. Tool for translating simulink models into input language of a model checker. *Formal Methods and Software Engineering*, Springer, p. 606–620, 2006.

49 STURSBERG, O.; KOWALEWSKI, S.; ENGELL, S. On the generation of timed discrete approximations for continuous systems. *Mathematical and Computer Modelling of Dynamical Systems*, Taylor & Francis, v. 6, n. 1, p. 51–70, 2000.

50 D'AURIOL, B. J. A finite state machine model to support the visualization of complex dynamic systems. In: *MSV*. [S.l.: s.n.], 2006. p. 304–310.

51 PREU, J.; KOWALEWSKI, S.; WONG-TOI, H.; HENZINGER, T. A. *et al.* An algorithm for the approximative analysis of rectangular automata. In: SPRINGER. *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*. [S.l.], 1998. p. 228–240.

52 LAMBERT, J. D. *Numerical Methods for Ordinary Differential Systems*: The initial value problem. 1. ed. [S.l.]: Wiley, 1991.

53 TEIXEIRA, H. C. G.; CAMPOS, M. C. M. M. *Controles típicos de equipamentos e processos industriais*. [S.l.]: Blucher, 2006.

54 SMITH, G. D. *Numerical solution of partial differential equations: finite difference methods*. [S.l.]: Oxford university press, 1985.

55 HOXHA, B.; BACH, H.; ABBAS, H.; DOKHANCHI, A.; KOBAYASHI, Y.; FAINEKOS, G. Towards formal specification visualization for testing and monitoring of cyber-physical systems. In: *Int. Workshop on Design and Implementation of Formal Tools and Systems*. [S.l.: s.n.], 2014.

56 ELNASHAIE, S.; UHLIG, F. *Numerical Techniques for Chemical and Biological Engineers Using MATLAB*. [S.l.]: Springer, 2007.

57 GREEN, D. W.; PERRY, R. H. *Perry's chemical engineers' handbook*. [S.l.]: McGraw-Hill, 1999.

58 CARBERRY, J. J.; VARMA, A. *Chemical reaction and reactor engineering*. [S.l.]: Marcel Dekker, 1986.

59 BEQUETTE, B. W. *Process dynamics*: Modeling, analysis, and simulation. [S.l.]: Prentice Hall, 1998.

60 SCHMIDT, A.; CLINCH, A.; RAY, W. The dynamic behaviour of continuous polymerization reactors-iii: An experimental study of multiple steady states in solution polymerization. *Chemical Engineering Science*, Elsevier, v. 39, n. 3, p. 419–432, 1984.

61 MOHL, K.-D.; KIENLE, A.; GILLES, E.-D.; RAPMUND, P.; SUNDMACHER, K.; HOFFMANN, U. Steady-state multiplicities in reactive distillation columns for the production of fuel ethers mtbe and tame: theoretical analysis and experimental verification. *Chemical Engineering Science*, Elsevier, v. 54, n. 8, p. 1029–1043, 1999.

62 SCENNA, N. J.; BENZ, S. J.; FRANCESCONI, J. A.; RODRÍGUEZ, N. H. Start-up of homogeneous azeotropic distillation columns with multiple steady states. *Industrial & engineering chemistry research*, ACS Publications, v. 43, n. 2, p. 553–565, 2004.

63 MOON, I. *Automatic verification of discrete chemical process control systems*. 191 p. Tese (PhD) — Carnegie Mellon University, Pittsburgh, 1992.

64 JOHNSTON, W.; WINTER, K.; BERG, L. van den; STROOPER, P.; ROBINSON, P. Model-based variable and transition orderings for efficient symbolic model checking. In: SPRINGER. *International Symposium on Formal Methods*. [S.l.], 2006. p. 524–540.