CLEBER ALVES SARMENTO

MODELAGEM DE PROGRAMAS E SUA VERIFICAÇÃO PARA CONTROLADORES PROGRAMÁVEIS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia.

SÃO PAULO
2008

CLEBER ALVES SARMENTO

MODELAGEM DE PROGRAMAS E SUA VERIFICAÇÃO PARA CONTROLADORES PROGRAMÁVEIS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia.

Programa: Engenharia Mecânica

Área de Concentração:
Engenharia de Controle e Automação Mecânica – Engenharia Mecatrônica

Orientador:
Prof. Dr. Diolino José dos Santos Filho

SÃO PAULO
2008

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 16 de Janeiro de 2008

Assinatura do autor:_____

Assinatura do orientador:_____

FICHA CATALOGRÁFICA

Sarmento, Cleber Alves
Modelagem de programas e sua verificação para controladores programáveis / C.A. Sarmento. -- ed.rev. -- São Paulo, 2008.
151 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Controladores programáveis 2.Linguagem de programação LD 3.Modelagem e verificação de máquinas de estados finitos estendidas (MEFEs) I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II. t.

DEDICATÓRIA

Aos meus pais, Isabel e Francisco, ao meu irmão Alisson e, de uma forma especial, à minha amada esposa Renata.

AGRADECIMENTOS

De uma forma inicial, quero manifestar o meu mais sincero agradecimento a Deus pela vida de todos aqueles que mencionarei a seguir, os quais contribuíram de alguma maneira para o desenvolvimento deste trabalho. Peço a Deus que, de uma forma misericordiosa, continue, todos os dias, abençoando a vida e a família de todas estas pessoas.

Sou profundamente grato ao Prof. Diolino José dos Santos Filho, pessoa que me confiou a oportunidade de encarar o desafio de realizar este trabalho, e que tem me ensinado, ao longo de toda a nossa convivência, que a humildade é uma das virtudes mais nobres que podemos ter em nossas vidas.

Agradeço as valiosas contribuições e discussões que foram promovidas pelos professores Paulo E. Miyagi e José Reinaldo Silva (PMR-USP), Luis Alberto M. Riascos (UFABC), e ao apoio que recebi do Prof. Wanderson de Oliveira Assis (IMT).

Aos amigos Marcosiris Pessoa e Osvaldo Asato (do grupo de pesquisa do Prof. Diolino), pelas oportunidades nas quais me auxiliaram com as pesquisas bibliográficas e com as revisões do texto. Agradeço também a secretária Andréa Parizotto (FSA), por sua disposição contínua, ajudando-me constantemente na marcação das reuniões realizadas com o Prof. Diolino, além da Adriana, Daniela, Graziele, Lucas, Maria e Mônica (FSA).

Ao apoio de Valdomiro Alves de Cerqueira, Adriano Ribolla e Marco Antônio Vilas Boas, os quais fazem parte da empresa em que trabalho, e aos diversos colegas da Poli: Roni Sérgio Viola, Percy I. J. Kaneshiro, José I. Garcia Melos, Roy Andres G. Morales, Pedro M. G. Del Foyo, Cristina T. M. Matsusaki, Fabrício Junqueira, Gladys D. B. Gustin, Regina C. Cantele, Gilberto (Beto do grupo GAS-PCS), Marcelo Colpas da Silva, Marco Aurélio P. Franco, Demétrio dos Santos, Rodrigo de Moraes, Marcelo Acácio Rodrigues e Marcelo A. da Costa (PEA).

Finalmente, quero demonstrar o meu mais profundo e sincero agradecimento aos meus queridos pais, Francisco e Isabel, ao meu irmão Alisson e à minha amada esposa Renata, os quais aprenderam a compreender a importância deste trabalho para a minha vida.

RESUMO

Os sistemas produtivos (SPs) podem utilizar controladores programáveis (CPs) como dispositivos de realização do controle. Neste contexto, programas de controle executados por estes CPs podem ser desenvolvidos de forma que não estejam em conformidade com as especificações de projeto, o que poderá provocar o surgimento de erros funcionais associados à execução de tais programas de controle, erros estes que podem levar os SPs sob controle a um estado que poderá implicar em acidentes envolvendo equipamentos, pessoas e o meio-ambiente. Esta questão tem motivado o surgimento de diversas abordagens para identificar a existência de erros em programas de controle de CPs, de forma a permitir a correção destes erros e garantir, conseqüentemente, maior confiabilidade operacional.

O presente trabalho tem por objetivo identificar a existência de erros em programas de controle baseados em LD (Ladder Diagram). Para isto, propõe-se um procedimento de desenvolvimento de modelos baseados em máquinas de estados finitos estendidas (MEFEs), que são gerados a partir do mapeamento de cada um dos *rungs* contidos no programa de controle que se deseja identificar erros. Uma vez desenvolvidos os modelos em MEFEs, torna-se possível a utilização de uma ferramenta computacional de verificação, própria para estabelecer se os modelos verificados satisfazem determinadas proposições estabelecidas em lógica temporal.

Uma proposição em lógica temporal está relacionada a um estado específico do programa de controle modelado, sendo que o objetivo da verificação é o de estabelecer se a proposição estipulada é atendida ou não. Se um determinado estado específico for, por exemplo, um estado indesejado do programa de controle modelado, e este estado for possível de ser alcançado como resultado do processo de verificação realizado, isto impactará na não conformidade do programa de controle com as especificações comportamentais estipuladas na forma de proposição em lógica temporal, indicando haver, portanto, um erro neste programa de controle modelado.

Palavras-chave: Controladores programáveis. Linguagem de programação LD. Modelagem e verificação de máquinas de estados finitos estendidas (MEFEs).

ABSTRACT

Productive systems (PS) can use programmable logic controllers (PLCs) as the devices of accomplishment of the control. In this context, control programs executed by these PLCs can be developed in a way so that they can be in non-conformity with the project specifications, and this fact may result in functional errors related to the control programs execution. These errors can take the PS under control to a state which can lead into accidents involving equipment, people and the environment. This fact has motivated the appearance of different approaches so as to identify the existence of these errors in PLC control programs so that they can be corrected and assure a greater operational reliability.

This work aims at identifying the existence of errors in control programs based on Ladder Diagram (LD). In order to accomplish that, a modeling procedure that generates extended finite state machines (EFSMs) is proposed from the mapping of each one of the rungs in the control program whose errors are to be identified. Once the models based on EFSMs are developed it becomes possible to use a computational verification tool, specifically designed to determine if the verified models fulfill determined propositions established in temporal logic.

A proposition in temporal logic is related to a specific state of the modeled control program and the objective of the verification is to establish whether the proposition is fulfilled or not. If a determined specific state, for example, is an unwanted state of the modeled control program and if this state is reachable as a result of the verification process, this will impact in the non-conformity of the control program with the behavior specifications established in a temporal logic proposition, indicating an error in this modeled control program.

Keywords: Programmable logic controllers. LD programming language. Modeling and verification of extended finite state machines (EFSM).

LISTA DE ILUSTRAÇÕES

Figura 2.1 – Representação estrutural de um sistema de controle de SEDs (NAKAMOTO, 2002) 25

Figura 2.2 – Ciclo operacional de um CP 29

Figura 2.3 – Representação de um determinado programa de controle escrito em LD 33

Figura 2.4 – Exemplo de rung com um elemento contador do tipo CTU de endereço C5:0 38

Figura 2.5 – Representação gráfica que mostra a relação entre a condição lógica (no mesmo rung) que precede o elemento contador do tipo CTU, os seus endereços e os seus estados 39

Figura 2.6 – Exemplo de rung com um elemento temporizador do tipo TON de endereço T4:0 40

Figura 2.7 – Representação gráfica que mostra a relação entre a condição lógica (no mesmo rung) que precede o elemento temporizador do tipo TON, os seus endereços e os seus estados 42

Figura 3.1 – Exemplo de representação gráfica de um AF 45

Figura 3.2 – AFD que modela o comportamento do funcionamento de um interruptor 46

Figura 3.3 – Exemplo de uma máquina de Mealy 48

Figura 3.4 – Exemplo de modelo de uma máquina de preparação de café 51

Figura 3.5 – Exemplo de aplicação de canais de comunicação externos 53

Figura 3.6 – Exemplo de aplicação de canais de comunicação internos 54

Figura 3.7 – Exemplo da aplicação de um estado compromissado 56

Figura 3.8 – Exemplo para a aplicação de um estado compromissado com canais de

comunicação externos 57

Figura 3.9 – Exemplo da aplicação de um estado compromissado 58

Figura 3.10 – Representação gráfica de um modelo com MEFEs desenvolvidas por Robinson-Mallett et al. (2005) utilizando a ferramenta UPPAAL 59

Figura 4.1 – Modelo que representa os elementos constituintes de um processo de verificação de modelos de sistemas concorrentes de estados finitos 65

Figura 5.1 – Itens que compõem o PFS 80

Figura 5.2 – Modelo PFS que representa o procedimento de modelagem proposto 82

Figura 5.3 – Exemplo de um rung de um determinado programa de controle em LD 83

Figura 5.4 – Exemplo de um modelo principal 84

Figura 5.5 – Exemplos de modelos auxiliares das entradas 89

Figura 5.6 – Modelo auxiliar do elemento contador do tipo CTU de endereço C5:0 90

Figura 5.7 – Modelo auxiliar do elemento temporizador do tipo TON de endereço T4:0 . 93

Figura 5.8 – Exemplo de um determinado programa de controle em LD com dois rungs . 96

Figura 5.9 – Exemplo de um modelo principal contendo os dois rungs apresentados na Figura 5.8 97

Figura 5.10 – Exemplo de um determinado programa de controle em LD com dois rungs 98

Figura 5.11 – Exemplo de um modelo principal que apresenta o mapeamento dos dois rungs apresentados na Figura 5.10 98

Figura 5.12 – Exemplo de um modelo principal que modela o rung da Figura 2.4 100

Figura 5.13 – Exemplo de um modelo principal que modela o rung da Figura 2.6 101

Figura 6.1 – Ilustração simplificada do sistema de alarme proposto por Moon (1994) ... 104

Figura 6.2 – Programa de controle do sistema de alarme (MOON, 1994) 105

Figura 6.3 – Modelo principal criado a partir do programa de controle apresentado na Figura 6.2 106

Figura 6.4 – Modelos auxiliares dos endereços de entrada 107

Figura 6.5 – Rung a ser corrigido do programa de controle do sistema de alarme (MOON, 1994) 109

Figura 6.6 – Modelo principal corrigido do programa de controle do sistema de alarme 109

Figura 6.7 – Rungs 0010 e 0011 do programa de controle do equipamento queimador de gás (ZOUBEK, 2004) 111

Figura 6.8 – Modelo dos dois rungs da Figura 6.7 (ZOUBEK, 2004) 112

Figura 6.9 – Esquema simplificado do equipamento queimador de gás (ZOUBEK, 2004) 113

Figura 6.10 – Programa de controle do queimador de gás – Parte 1 (ZOUBEK, 2004) .. 114

Figura 6.11 – Programa de controle do queimador de gás – Parte 2 (ZOUBEK, 2004) .. 115

Figura 6.12 – Modelo principal proposto pelo presente trabalho do programa de controle do equipamento queimador de gás 118

Figura 6.13 – Modelos auxiliares dos endereços de entrada 119

Figura 6.14 – Modelos auxiliares dos elementos temporizadores do tipo TON 120

Figura 6.15 – Endereços corrigidos depois da verificação de número “1” 123

Figura 6.16 – Rung 0013 corrigido depois da realização da verificação de número “1” . 124

Figura 6.17 – Modelo principal resultante da aplicação de quatro correções 125

Figura A.1 – Regras de construção de programa de controle em linguagem LD.....133

Figura B.1 – Modelo da Figura 5.4.....135

Figura B.2 – Modelos da Figura 5.5.....135

Figura B.3 – Modelo da Figura 5.9.....136

Figura B.4 – Modelo da Figura 5.11.....136

Figura B.5 – Modelo da Figura 5.12.....137

Figura B.6 – Modelo da Figura 5.13.....137

Figura C.1 – Exemplo de um autômato temporizado com uma variável do tipo relógio. 138

LISTA DE TABELAS

Tabela 2.1 – Classificação de possíveis linguagens para a programação de CPs..... 31

Tabela 2.2 – Elementos contatos que podem ser utilizados em programas de controle desenvolvidos em LD..... 34

Tabela 2.3 – Associação entre endereços e elementos contatos..... 35

Tabela 2.4 – Associação entre endereços e elementos bobinas..... 36

Tabela 4.1 – Operadores lógicos..... 69

Tabela 4.2 – Operadores temporais: quantificadores de caminho..... 69

Tabela 4.3 – Operadores temporais: quantificadores de estado..... 70

Tabela 4.4 – Proposições na versão simplificada da lógica temporal CTL: exemplos da ocorrência de p e q em uma estrutura combinatória de estados simbólicos .. 70

Tabela 4.5 – Proposições da versão simplificada da lógica temporal CTL e suas diferentes formas de representação..... 72

Tabela 4.6 – Exemplos de proposições..... 73

Tabela 6.1 – Descrição das situações a serem verificadas no modelo do programa do sistema de alarme de Moon (1994)..... 108

Tabela 6.2 – Endereços do programa de controle do equipamento queimador de gás..... 116

Tabela 6.3 – Descrição das situações a serem verificadas no modelo do programa do queimador de gás de Zoubek (2004)..... 121

SUMÁRIO

Capítulo 1 ..... **16**

1. Introdução **16**

1.1 Motivação 20

1.2 Objetivo 21

1.3 Justificativa 21

1.4 Organização do trabalho 22

Capítulo 2 ..... **24**

2. Controladores Programáveis e a Linguagem LD **24**

2.1 Considerações iniciais 24

2.2 Controladores programáveis 26

2.3 Ciclo operacional de um CP 28

2.4 Introdução à linguagem LD 30

2.4.1 Elementos do programa de controle 33

Capítulo 3 ..... **43**

3. Máquinas de Estados Finitos Estendidas **43**

3.1 Autômatos finitos 43

3.2 Máquinas de Mealy 47

3.3 Máquinas de estados finitos estendidas 49

3.3.1 Exemplos de aplicações de máquinas de estados finitos estendidas 50

3.3.2 Canais de comunicação entre MEFEs 52

3.3.3 Estados compromissados 55

3.3.4 UPPAAL e a modelagem de MEFEs 58

3.3.5 Definição matemática adotada para o estudo de MEFEs 60

Capítulo 4 ..... **63**

4. Verificação de Modelos **63**

4.1 Considerações iniciais 63

4.2 A verificação de modelos 64

4.2.1 Proposicões da lógica temporal CTL 66

4.2.2 Sintaxe das expressões proposicionais 67

4.2.3 Operadores de lógica temporal 68

4.2.4 Exemplos de proposições ... 72
4.3 A ferramenta de modelagem e verificação UPPAAL ... 73
4.3.1 Justificativas para a utilização da ferramenta UPPAAL ... 74
4.4 Considerações sobre as propriedades de alcançabilidade e de segurança ... 76

Capítulo 5 ... 79

5. Procedimento para a Modelagem de MEFEs a partir de Programas de Controle Desenvolvidos em LD ... 79

5.1 Considerações iniciais ... 79
5.2 Passos do procedimento para a modelagem de MEFEs a partir de programas de controle desenvolvidos em LD ... 83
5.2.1 Parametrização inicial do modelo principal ... 83
5.2.2 Parametrização dos modelos auxiliares das entradas ... 87
5.2.3 Parametrização dos modelos auxiliares dos elementos contadores do tipo CTU ... 89
5.2.4 Parametrização dos modelos auxiliares dos elementos temporizadores do tipo TON ... 91
5.2.5 Declaração das variáveis associadas aos endereços do programa de controle ... 94
5.2.6 Mapeamento dos rungs ... 96
5.2.7 Inserção de estados devido aos modelos auxiliares dos elementos contadores do tipo CTU ... 99
5.2.8 Inserção de estados devido aos modelos auxiliares dos elementos temporizadores do tipo TON ... 100
5.2.9 Inclusão do tempo de varredura considerado ... 102

Capítulo 6 ... 103

6. Estudo de casos ... 103

6.1 O Trabalho de Moon ... 103
6.1.1 Programa de controle do sistema de alarme e proposta de modelagem ... 105
6.1.2 Verificação do modelo proposto ... 107
6.2 O Trabalho de Zoubek ... 110
6.2.1 Programa de controle do equipamento queimador de gás ... 112
6.2.2 Proposta de modelagem ... 116
6.2.3 Verificação do modelo proposto ... 121

Capítulo 7 ... 128

7. Conclusões ... 128

7.1 Observações complementares ... 129

7.2 Trabalhos futuros 130

Anexo A - Dados complementares sobre a linguagem LD 132

Anexo B – Estados compromissados 135

Anexo C – Autômatos temporizados 138

Referências Bibliográficas 141

CAPÍTULO 1

1. INTRODUÇÃO

Nas últimas décadas o crescente desenvolvimento da tecnologia tem contribuído para que o controle dos sistemas chamados sistemas produtivos (SPs) possa ser realizado utilizando-se controladores programáveis (CPs). Em conseqüência disso, o comportamento dinâmico dos SPs onde os CPs são aplicados depende, essencialmente, dos programas de controle executados por tais CPs. Em geral, estes programas de controle são desenvolvidos de acordo com a experiência adquirida por programadores na linguagem de programação adotada, fato este que pode contribuir para o surgimento de erros funcionais associados à execução de tais programas de controle. Neste sentido, estes erros podem levar os SPs sob controle a um estado que pode implicar em acidentes envolvendo equipamentos, pessoas e o meio-ambiente. Devido a isso, tem-se observado nos últimos anos o surgimento de diversas abordagens para se identificar a existência de erros em programas de controle, de forma que, a partir da correção dos mesmos, tem sido possível atribuir aos SPs uma maior confiabilidade operacional.

Os SPs são sistemas que podem perceber e causar ocorrências quando interagem com seus elementos e/ou com o ambiente ao seu redor através da percepção e geração de estímulos que podem ser chamados de eventos. Sistemas com estas características são chamados de sistemas a eventos discretos (SEDs) e, conforme pode ser visto em Hellgren; Fabian e Lennartson (2001); Nakamoto (2002); Zanma; Miyabayashi e Ishida (2004), os SPs são sistemas que pertecem à classe dos SEDs.

Por sua vez, os SEDs podem ser definidos como sistemas dinâmicos que evoluem de acordo com a ocorrência de eventos de duração finita em instantes de tempo que, em geral, são não determinísticos. Em SEDs, os valores das variáveis que representam o estado desses sistemas são modificados de forma discreta, a partir da ocorrência de eventos considerados instantâneos, sendo que tais valores permanecem inalterados entre a ocorrência de eventos consecutivos. Essas variáveis apresentam características qualitativas

(SANTOS FILHO, 1998) e assumem valores discretos como, por exemplo, números inteiros e variáveis lógicas (VILLANI et al., 2005). Por este motivo, considerar que os SPs evoluem dinamicamente através da ocorrência de eventos permite classificá-los como SEDs.

De uma forma concisa pode-se dizer que os SPs são sistemas cuja finalidade é a produção de itens (MATSUSAKI, 2004), sendo tais sistemas constituídos a partir da integração física e/ou virtual de diversas estações de trabalho, linhas flexíveis e células de manufatura, capazes de realizar uma variedade de processos de forma independente baseado no controle adequado do fluxo de itens concatenado ao sistema (SANTOS FILHO, 1998).

Em relação ao controle dos SPs, de acordo com Lee; Hsu (2004), diversos SPs utilizam controladores programáveis (CPs) como dispositivos de realização do controle, atendendo aos requisitos de confiabilidade, robustez e possibilidade de programação e de reprogramação (KURFESS, 2005). Historicamente, tais requisitos não eram possíveis de serem oferecidos pelos sistemas de controle de SPs baseados em circuitos elétricos compostos por relés eletromagnéticos, fato este que, aliado ao crescente desenvolvimento da tecnologia dos circuitos integrados, contribuiu para o surgimento dos CPs no final da década de 60, sendo que até os dias atuais, os CPs ainda continuam passando por aprimoramentos. Em Santos Filho (1998), já diagnosticava-se o fato dos CPs serem continuamente aprimorados de forma a permitir que os SPs controlados pudessem atingir um elevado nível de automação, flexibilidade e integração, princípios estes que ainda são fundamentais para empresas que desejam se manter competitivas frente um comportamento baseado na maximização de critérios de controle de qualidade e flexibilidade operacionais condizentes com as necessidades impostas pelos consumidores.

Os CPs realizam o controle de SPs através da execução cíclica de programas de controle que são armazenados em sua memória. No sentido de viabilizar o desenvolvimento de programas de controle para CPs de acordo com as necesidades atuais, diversas linguagens de programação têm sido criadas, sendo que cinco delas foram padronizadas pela norma IEC 61131 (PENG; ZHOU, 2004): IL (*instruction list* – lista de instruções), ST (*structured text* – texto estruturado), FBD (*function block diagram* – diagrama de bloco de função), SFC (*sequential function chart* – diagrama de função seqüencial) e LD (*ladder diagram* – diagrama escada). Destas cinco linguagens, a LD tem sido, ainda hoje, fortemente utilizada

para o desenvolvimento de programas de controle de SPs (LEE; HSU, 2004; LEE; ZANDONG; LEE, 2004; KURFESS, 2005; LUCAS; TILBURY, 2005; WAGNER et al., 2006). Este fato é reforçado por uma pesquisa realizada nos Estados Unidos pela Control Engineering (2002) apud Lucas; Tilbury (2003) que mostra que 96% dos desenvolvedores de programas de controle para CPs utilizam a linguagem LD, seja ela de forma isolada ou de forma combinada com outras linguagens.

De acordo com Uzam; Jones (1998); Lucas; Tilbury (2003); Lee; Hsu (2004); Peng; Zhou (2004), programadores de CPs ao desenvolverem um programa de controle em LD utilizam-se, geralmente, de métodos heurísticos de programação, os quais são susceptíveis à ocorrência de erros no programa de controle desenvolvido. No presente trabalho, considera-se que um erro identificado em um programa de controle desenvolvido em LD pode ser proveniente do equívoco cometido por um programador durante a associação de endereços do CP a elementos do programa de controle utilizados ou, ainda, pode ser proveniente do fato de um programador utilizar-se de métodos heurísticos de programação, o que pode resultar na ocorrência de problemas relacionados aos intertravamentos considerados no programa de controle e que determinam o comportamento dinâmico do SP controlado. Outro fator ainda a ser considerado e que é abordado está associado aos possíveis problemas de intertravamento que surgem em programas de controle desenvolvidos em LD decorrentes de valores indesejados associados aos endereços de entrada do CP, valores estes que podem ser diferentes dos valores esperados em um determinado instante, em virtude de acionamentos indevidos ou de anomalias nos dispositivos de detecção e de comando do SP controlado, anomalias estas que podem ser, por exemplo, a interrupção de um fio elétrico, um curto-circuito nos dispositivos mencionados ou um defeito no próprio dispositivo de entrada do CP (PARR, 2003).

Em geral, estes erros considerados são difíceis de serem identificados, principalmente, devido à dificuldade que existe em se analisar a representação e o processamento de paralelismos, concorrências e seqüenciamentos de processos em um programa de controle desenvolvido em LD (SUESUT et al., 2004); (ZOUBEK, 2004). Em virtude disso, várias abordagens têm sido propostas com o objetivo de identificar e/ou inibir a ocorrência de tais erros.

Dentre estas abordagens, é possível encontrar as que seguem uma sistemática na qual é

criado um modelo do sistema de controle de interesse em rede de Petri (RdP) e, em seguida, o modelo é convertido em um programa de controle escrito em LD (UZAM; JONES; AJLOUNI, 1996; ZHOU; VENKATESH, 1999; CHIRN; MCFARLANE, 2000; LEE; ZANDONG; LEE, 2004; PENG; ZHOU, 2004; SUESUT et. al, 2004). Outra possível abordagem realiza o processo inverso, conforme é visto em Peng; Zhou (2004), resultando em redes de Petri de difícil interpretação em virtude da complexidade do procedimento.

Outras abordagens propostas baseiam-se na execução de testes manuais, onde o programa de controle sob análise é executado por um CP e, então, são analisados os resultados da execução deste programa a partir da ocorrência de eventos simulados por um programador (ŠUSTA, 2003; SMITH; SIMPSON, 2005). Porém, estes testes manuais podem não apresentar resultados confiáveis quanto à identificação de erros no programa de controle analisado, uma vez que podem não considerar todas as possíveis combinações dos valores associados aos endereços (de entrada do CP) utilizados no programa de controle em questão.

A abordagem que será adotada pelo presente trabalho é derivada das propostas de Moon (1994); Rossi, Schnoebelen (2000); De Smet, Rossi (2002); Zoubek, Roussel e Kwiatkowska, (2003); Zoubek (2004), as quais apresentam procedimentos semelhantes entre si, seguindo um conjunto de métodos baseado em duas etapas:

- a realização de um procedimento de modelagem utilizando-se alguma técnica de modelagem de SEDs, mapeando um programa de controle existente desenvolvido em linguagem LD e;
- a verificação do modelo obtido a partir da realização do procedimento de modelagem em questão.

Nestes trabalhos de Moon (1994); Rossi, Schnoebelen (2000); De Smet, Rossi (2002); Zoubek, Roussel e Kwiatkowska, (2003); Zoubek (2004), o termo “verificação” apresenta um significado bastante semelhante, o qual, por sua vez, assemelha-se com o que será adotado pelo presente trabalho (definição a seguir), baseando-se nas definições encontradas em Hoppe; Meseguer (1993); IEC 61508 (1998); Andersson; Runeson (2002); Zoubek (2004); Wang; Tan (2006).

Definição 1: verificação é um processo onde são realizados testes exaustivos em modelos

de estados finitos desenvolvidos a partir de programas de controle existentes, executados por um CP, com o objetivo de identificar se um determinado estado específico do programa modelado pode ser alcançado ou se tal estado específico não pode ser alcançado.

O estado específico citado na Definição 1 é formado pela combinação de um conjunto de endereços pertencentes ao programa de controle de interesse e por seus respectivos valores, sendo tais endereços especificados por entradas, saídas, memórias, contadores e temporizadores. Caso um determinado estado específico indesejado seja alcançado depois de realizada a verificação do modelo, considera-se que isso indicará a presença de um erro no programa de controle modelado, ou ainda, caso um determinado estado específico desejado não seja alcançado depois de realizada a verificação do modelo, considera-se que isso também indicará a presença de um erro no programa de controle modelado.

Conforme relata Zoubek (2004), o termo verificação tem sido comumente confundido com o termo "validação" em questões referentes aos sistemas de controle de SPs e, por este motivo, o presente trabalho apresenta uma proposta de definição para o termo "validação", baseando-se nas definições encontradas em Hoppe; Meseguer (1993); IEC 61508 (1998); Santos Filho (1998); Andersson; Runeson (2002); Zoubek (2004); Wang; Tan (2006), de forma a estabelecer uma diferença entre ambos os termos nas questões envolvendo os SPs.

Definição 2: validação refere-se ao processo que é executado na fase conclusiva do projeto de um SP, onde são realizados testes para se determinar se o sistema de controle do SP desenvolvido satisfaz às expectativas dos clientes e/ou usuários que farão uso do sistema.

Desta forma, é possível dizer que não é necessária a realização do processo de verificação para que seja realizado o processo de validação de um SP. Entretanto, o processo de verificação deve ser fortemente desejado pelos clientes e/ou usuários deste SP, de forma que se constitua num pré-requisito do processo de validação.

1.1 MOTIVAÇÃO

Conforme foi apresentado na introdução do presente trabalho, os erros aqui considerados em programas de controle desenvolvidos em LD são possíveis de ocorrer, difíceis de serem identificados e podem ser responsáveis por acidentes envolvendo os equipamentos

utilizados no SP sob controle, os operadores e/ou usuários que interagem com este SP e o meio-ambiente no qual este SP está inserido (ZOUBEK, 2004).

Pode-se considerar que a identificação desses erros é uma tarefa que depende da realização de procedimentos que podem ser adotados e da experiência adquirida por quem realizará essa análise.

1.2 OBJETIVO

O objetivo do presente trabalho consiste em apresentar uma proposta para a modelagem e posterior verificação de modelos de máquinas de estados finitos estendidas (MEFEs) gerados a partir de programas de controle existentes desenvolvidos em LD, tendo-se como premissa o ciclo operacional de processamento do CP responsável pela execução de tais programas.

Neste sentido, os modelos gerados a partir desta proposta de modelagem permitem a utilização de ferramentas computacionais capazes de realizar a sua verificação, tornando possível a identificação de erros nestes modelos e, conseqüentemente, no programa de controle modelado.

1.3 JUSTIFICATIVA

Conforme foi apresentado, diversos problemas estão relacionados à utilização da linguagem LD e, apesar disso, diversos desenvolvedores de programas de controle de CPs ainda utilizam-se dessa linguagem. Aliado a isso, existe o fato de que há determinadas famílias de CPs que somente podem ser programados através da linguagem LD (ZOUBEK, 2004; ALLEN-BRADLEY, 2006). Estes fatores justificam a necessidade de desenvolvimento de procedimentos que possibilitem a identificação de erros em programas de controle desenvolvidos em LD.

Com a aplicação do procedimento de modelagem proposto pelo presente trabalho, os programas de controle considerados e que foram desenvolvidos utilizando-se a linguagem LD, podem ser modelados através de MEFEs, sendo que estes modelos, uma vez gerados, podem ser submetidos a testes exaustivos a fim de se determinar se um estado específico

do programa de controle modelado pode ser alcançado ou se tal estado específico não pode ser alcançado.

A partir do processo de verificação proposto é possível dizer se o programa de controle modelado apresenta os erros que são considerados pelo presente trabalho, evitando-se, assim, que a ocorrência de tais erros possa resultar em quaisquer prejuízos relacionados ao SP sob controle.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho foi organizado em sete capítulos. No Capítulo 1, estão sendo apresentados a introdução, a motivação, o objetivo e a justificativa deste trabalho. O Capítulo 2 abordará, com um maior nível de aprofundamento, o assunto controladores programáveis (CPs). Serão apresentadas questões referentes ao ciclo operacional de um CP, sendo tais questões fundamentais para se determinar os requisitos que os modelos em MEFEs propostos pelo presente trabalho devem atender. Encerra-se o Capítulo 2 com uma introdução à linguagem de programação LD, abordando-se, por fim, as diversas características dos elementos de programa de controle considerados por este trabalho, os quais são fundamentais para a realização do procedimento (apresentado no Capítulo 5) de modelagem de MEFEs a partir de programas de controle em LD.

No Capítulo 3 serão apresentadas as máquinas de estados finitos estendidas (MEFEs), técnica de modelagem utilizada no presente trabalho no procedimento que será apresentado no Capítulo 5. O capítulo será iniciado com uma introdução sobre autômatos finitos, passando-se, em seguida, por uma discussão sobre máquinas de estados finitos (MEFs) e sendo encerrado, finalmente, com uma abordagem sobre MEFEs.

O Capítulo 4 será iniciado com uma discussão sobre a verificação de modelos e sua relação com a checagem de modelos. Serão apresentadas as proposições da versão simplificada da lógica temporal CTL, as expressões proposicionais, os operadores de lógica temporal e exemplos de proposições. Em seguida, será realizada uma abordagem sobre a ferramenta de modelagem e verificação de modelos adotada pelo presente trabalho, discutindo-se também as justificativas para a sua escolha. Encerra-se o Capítulo 4 com algumas considerações sobre as propriedades de alcançabilidade e de segurança de modelos de

estados finitos que podem ser submetidos ao processo de verificação.

No Capítulo 5 será apresentado o procedimento proposto pelo presente trabalho para a realização da modelagem de MEFEs a partir de programas de controle desenvolvidos em LD e executados por um CP. Aspectos discutidos no Capítulo 2 (ciclo operacional de um CP, tempo de varredura, elementos do programa de controle etc.) e no Capítulo 3 (MEFEs, estados compromissados, canais de comunicação etc.) serão novamente abordados neste capítulo em virtude dos modelos em MEFEs que serão apresentados.

O Capítulo 6 contemplará dois trabalhos onde há a modelagem e a verificação de modelos que foram obtidos a partir de programas de controle existentes desenvolvidos em LD. São estes trabalhos: o de Moon (1994) e o de Zoubek (2004). Os programas de controle apresentados em tais trabalhos serão modelados de acordo com o procedimento de modelagem apresentado no Capítulo 5, e os resultados obtidos pelo presente trabalho serão comparados com os resultados obtidos por Moon (1994) e Zoubek (2004).

No Capítulo 7 serão apresentadas as conclusões e as propostas para trabalhos futuros. A bibliografia encerrará o presente trabalho.

CAPÍTULO 2

2. CONTROLADORES PROGRAMÁVEIS E A LINGUAGEM LD

Este capítulo abordará com um maior nível de aprofundamento o assunto controladores programáveis (CPs), ressaltando-se o fato de que, conforme foi apresentado no Capítulo 1, o CP é um dispositivo de realização de controle que tem sido fortemente aplicado ao controle de SPs. Serão apresentadas diversas questões referentes ao ciclo operacional de um CP, sendo tais questões fundamentais para se determinar os requisitos que os modelos em MEFEs propostos pelo presente trabalho devem atender. O capítulo será finalizado com uma introdução à linguagem de programação LD[1], sendo apresentadas as características dos elementos de programa de controle considerados por este trabalho, o que se faz necessário em virtude da execução do procedimento de modelagem de MEFEs apresentado no Capítulo 5.

2.1 CONSIDERAÇÕES INICIAIS

Conforme a discussão iniciada no Capítulo 1, os SPs podem ser classificados como SEDs,

[1] No Anexo A encontram-se informações complementares sobre esta linguagem.

sistemas estes que são caracterizados por serem sistemas dinâmicos e que evoluem de acordo com a ocorrência de eventos considerados instantâneos e de duração finita, em instantes de tempo que, em geral, são não determinísticos.

Em Nakamoto (2002), é possível encontrar uma proposta, a qual é derivada de Miyagi (1996), para a representação em diagrama de blocos de um sistema de controle de SEDs, representação esta que está sendo ilustrada pela Figura 2.1.

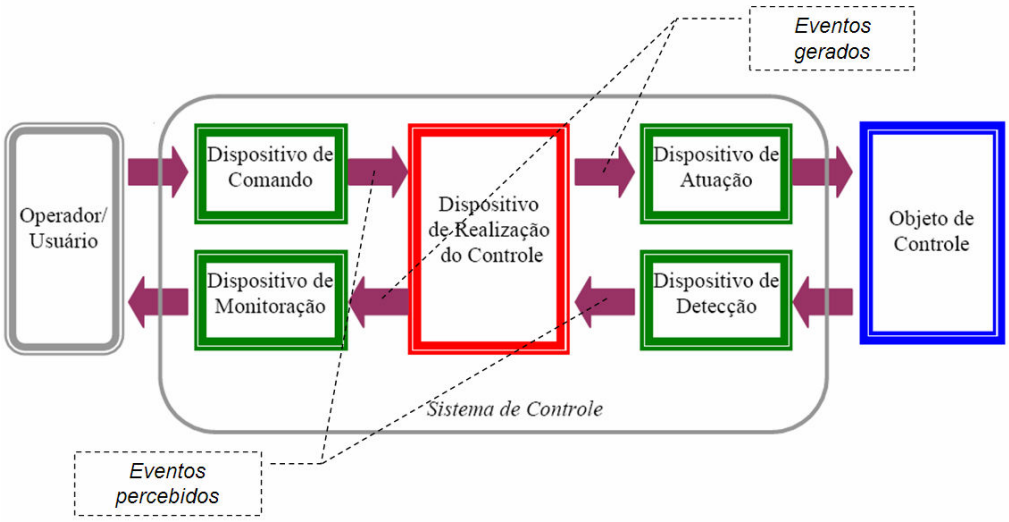


Figura 2.1 – Representação estrutural de um sistema de controle de SEDs (NAKAMOTO, 2002)

No lado direito do dispositivo de realização do controle (Figura 2.1) observa-se a presença de dispositivos de atuação e de detecção, os quais interagem com o objeto de controle. O dispositivo de atuação caracteriza-se por enviar sinais ao objeto de controle, o qual poderá ser, por exemplo, um dispositivo de fixação de uma determinada peça. O dispositivo de atuação pode ser formado por válvulas solenóides, relés eletromagnéticos, servo-motores etc. Já o dispositivo de detecção caracteriza-se por receber sinais do objeto de controle, sinais estes que podem ser referentes, por exemplo, à confirmação da ação executada pelos dispositivos de atuação. O dispositivo de detecção pode ser constituído por chaves-limites de fim-de-curso, sensores eletromagnéticos, potenciômetros, chaves-fotoelétricas, termostatos, tacômetros, *resolvers*, codificadores etc.

Ainda na Figura 2.1, observa-se do lado esquerdo do dispositivo de realização do controle a presença de dispositivos de comando e de monitoração. O dispositivo de comando caracteriza-se por receber sinais que são gerados a partir de eventos realizados por

operadores/usuários, sinais estes que podem estar relacionados a uma determinada informação para execução da seqüência de produção de certo produto. O dispositivo de comando pode ser composto por chaves de seleção e/ou rotativas, botoeiras, alavancas etc. Já o dispositivo de monitoração carcateriza-se por disponibilizar informações sobre o sistema sob controle para os operadores/usuários. Tal dispositivo pode ser constituído por lâmpadas sinalizadoras, buzinas, alarmes, monitores de vídeo, Interfaces Homem-Máquina (IHMs) etc.

Desta forma, explicita-se o fluxo de eventos gerados e percebidos pelo dispositivo de realização de controle constituindo um sistema de malha fechada que interage com o ambiente.

2.2 CONTROLADORES PROGRAMÁVEIS

Conforme Groover (2000), a primeira idéia de concepção de CP surgiu em 1968, quando o inglês Richard Morley propôs as primeiras especificações para este tipo de controlador, o qual recebeu o nome de Modicon[2] Model 084.

No mesmo ano em que Morley criou o CP, a Hydromatic Division da General Motors Corporation concebeu um conjunto de especificações para um controlador com características semelhantes às apresentadas pelo controlador de Morley, especificações estas que foram motivadas pelo alto custo e pela baixa flexibilidade dos sistemas de controle de SPs construídos a partir de circuitos elétricos compostos por relés eletromagnéticos, sistemas de controle estes que eram largamente utilizados pela General Motors Corporation em linhas de transferência e outros sistemas automatizados (ERICKSON, 1996; PENG; ZHOU, 2004). Dentre essas especificações propostas pela General Motors Corporation, destacavam-se:

[2] MODICON é uma composição originária de MOdular DIgital CONtroller.

- O dispositivo de realização do controle deveria ser programável e re-programável;
- Deveria ser projetado para operar em ambiente industrial;
- Deveria ser capaz de receber um sinal de 120VAC a partir de botões e chaves limites de fim-de-curso;
- Suas saídas deveriam suportar um número elevado de chaveamentos de cargas como motores e relés;
- Seu preço de aquisição e instalação deveria ser competitivo em relação aos relés eletromagnéticos e dispositivos lógicos de estado sólido em uso.

Desde então, os CPs, de um modo geral, têm passado por aprimoramentos que foram sendo realizados com o passar dos anos para que tais dispositivos de realização de controle pudessem se adequar às crescentes exigências das empresas por flexibilidade e integração. Por outro lado, em linhas gerais, a concepção dos CPs tem se mantido a mesma com o passar dos anos e tal concepção é apresentada, neste trabalho, como sendo composta por três itens considerados como relevantes: memória, unidade de processamento e dispositivos de entrada e saída (ALLEN-BRADLEY, 2006; SIEMENS, 2006).

A memória tem a função de armazenar o programa de controle do CP e os valores associados aos endereços do CP. Por sua vez, os endereços podem ser considerados como variáveis utilizadas pelo programa de controle do CP, podendo tais endereços ser classificados em dois tipos:

- Os endereços externos: estão associados ao endereçamento dos canais físicos dos dispositivos de entrada e saída do CP. Os canais físicos dos dispositivos de entrada recebem os sinais enviados pelo SP sob controle e os canais físicos dos dispositivos de saída enviam sinais para este mesmo SP sob controle;
- Os endereços internos: estão associados aos demais endereços que não estão associados ao endereçamento dos canais físicos dos dispositivos de entrada e saída do CP, estando associados a elementos do programa de controle que são utilizados para o desenvolvimento e a execução do programa de controle como, por exemplo, endereços auxiliares, contadores e temporizadores.

A unidade de processamento tem a função de executar de forma seqüencial as três etapas do ciclo operacional de um CP que é ilustrado na Figura 2.2.

Os dispositivos de entrada e saída são circuitos eletrônicos que realizam, respectivamente, a captação de sinais através de canais físicos de entrada (provenientes dos dispositivos de comando e/ou de detecção), e o envio de sinais que gerarão ações para os dispositivos de monitoração e/ou atuação, sinais estes que são disponibilizados através de canais físicos de saída.

2.3 CICLO OPERACIONAL DE UM CP

Em geral, o ciclo operacional de um CP, conforme está sendo apresentado na Figura 2.2, está baseado na execução seqüencial e cíclica das seguintes etapas (MIYAGI, 1996; DE SMET et al., 2000; DE SMET; ROSSI, 2002; HACKWORTH; HACKWORTH, 2003; PARR, 2003; ZOUBEK, ROUSSEL, KWIATKOWSKA, 2003):

a) atualização do valor dos endereços de entrada (amostragem do valor dos endereços de entrada);
b) processamento do programa de controle (armazenado na memória do CP);
c) atualização do valor dos endereços de saída.

O tempo de execução deste ciclo operacional de um CP recebe o nome de tempo do ciclo de varredura (*scan cycle time*) ou, simplesmente, tempo de varredura[3] (*scan time*) (SANTOS FILHO, 2000; PARR, 2003), termo este que será o adotado ao longo do presente trabalho. Neste sentido, ressalta-se que este ciclo operacional apresentado na Figura 2.2 é o que será considerado no processo de desenvolvimento de MEFEs que será apresentado no Capítulo 5.

[3] É importante ressaltar que CPs tradicionais utilizam-se desta tecnologia para executar o controle (ALLEN-BRADLEY, 2006; SIEMENS, 2006).

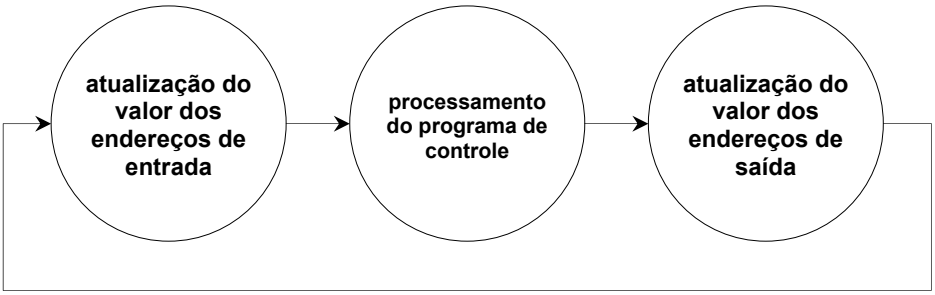


Figura 2.2 – Ciclo operacional de um CP

Na etapa (a) deste ciclo operacional de um CP, os estímulos físicos que são enviados ao CP (em geral, sinais em forma de tensão e/ou corrente elétrica), a partir dos dispositivos de detecção e/ou de comando, são captados pelos dispositivos de entrada do CP, convertidos em valores discretos do tipo booleano, associados aos endereços onde foram amostrados e armazenados na memória do CP. Em geral, um estímulo elétrico de 24VDC recebido por um CP é associado a um nível lógico verdadeiro (igual a "1") e um estímulo elétrico de 0VDC é associado a um nível lógico falso (igual a "0") (KURFESS, 2005). Esses valores lógicos são fundamentais para se determinar os resultados das operações efetuadas na etapa (b). Na etapa (c), os resultados obtidos na etapa anterior são repassados aos dispositivos de saída do CP, os quais enviam sinais em forma de tensão e/ou corrente elétrica aos dispositivos de atuação e/ou de monitoração[4].

Considerando-se as questões relativas ao tempo de varredura, segundo Miyagi (1996), quando a evolução dos estados internos do CP é definida temporalmente, considera-se a existência de um tipo de "relógio" interno do CP. Como a operação do CP possui um intervalo de tempo (considerado) básico definido pelo tempo de varredura, o "relógio" interno do CP atua segundo um número inteiro de períodos deste tempo de varredura.

[4] Assume-se no presente trabalho, conforme Miyagi (1996); Hackworth, Hackworth (2003), que os valores dos estados internos (endereços internos) e das saídas (endereços externos de saída) permanecem fixos desde o início da etapa em que os valores dos endereços de saída são atualizados até o fim do período de amostragem dos sinais dos endereços de entrada.

Desta forma, uma vez que o tempo de varredura é considerado constante, o "relógio" interno pode ser representado por pulsos gerados a cada ciclo, combinados com operadores aritméticos de forma adequada. Concluindo esta colocação, Miyagi (1996) descreve que elementos temporizadores podem utilizar-se do tempo de varredura do CP como base de tempo, fato este que também é apresentado em Parr (2003), de forma que tal hipótese é fundamental para suportar o procedimento de modelagem proposto pelo presente trabalho.

Porém, considerar que o tempo de varredura tenha sempre um valor constante pode implicar em uma aproximação, pois conforme é apresentado em Erickson (1996); Parr (2003); Allen-Bradley (2006), o tempo de varredura de um CP é determinado pelo valor do tempo de execução de seu ciclo operacional, e este tempo poderá ter um valor não constante devido à etapa de processamento do programa de controle. Durante a execução da etapa em questão, devido aos possíveis desvios condicionais e incondicionais que possam existir para a execução de *rungs* (linhas de programação em LD) e/ou sub-rotinas, e da possível existência de elementos do programa de controle com elevado tempo de execução e cuja execução seja realizada em diferentes intervalos de tempo (BOLTON, 2006), poderá ocorrer uma variação do valor do tempo de varredura em relação a um valor constante (PARR, 2003).

Em virtude deste fato, o presente trabalho considera o valor máximo do tempo de varredura (tempo de varredura crítico), assumindo tal valor como sendo constante e utilizando-o para o desenvolvimento do modelo principal (Capítulo 5) do programa de controle que se deseja modelar.

2.4 INTRODUÇÃO À LINGUAGEM LD

No capítulo introdutório deste trabalho foram apresentadas cinco linguagens de programação de CPs que são padronizadas pela norma IEC 61131: Sequential Function Chart (SFC), Ladder Diagram (LD), Function Block Diagram (FBD), Instruction List (IL) e Statement List (SL). Estas cinco linguagens fazem parte de um conjunto de linguagens para a programação de CPs que podem ser vistas na Tabela 2.1, a qual foi obtida em Miyagi (1996).

Tabela 2.1 – Classificação de possíveis linguagens para a programação de CPs

Tipo	**Linguagem**	**Carater**		
		lógica	ordenação	funções complexas
textuais	álgebra de Boole	X		
	IL (Instruction List)	X		
	ST (Structured Text)	X		
gráficas	LD (Ladder Diagram)	X		X
	FBD (Function Block Diagram)	X		X
	fluxograma		X	X
	elementos SFC (Sequential Function Chart)		X	
tabulares	tabela de decisão		X	

Essa diversidade de linguagens de programação de CPs pode ser atribuída ao fato de que desde o surgimento de tais equipamentos, os seus fabricantes têm buscado diferentes formas de demonstrar aos seus clientes diferenciais competitivos em relação aos CPs concorrentes (WAGNER et al., 2006). Tal diversidade impôs aos profissionais que trabalhavam com CPs a necessidade de conhecerem essas diferentes linguagens. Diante destes e de outros fatos, o comitê técnico TC65 do IEC (International Electrotechnical Commission) iniciou a elaboração da norma IEC 61131, a qual, em uma das suas oito partes em que é organizada, trata da padronização de linguagens de programação para CPs, tendo estabelecido cinco linguagens padronizadas, entre as quais está a linguagem LD.

A representação de um programa de controle desenvolvido em LD apresenta uma estrutura semelhante a uma escada (daí o nome em inglês *ladder*), onde os degraus desta escada ou *rungs* (termo comumente utilizado para as linhas de programação em LD) são os locais para a colocação dos elementos do programa de controle, cuja proposta de definição aderente ao presente trabalho é apresentada a seguir:

Definição 3: elementos do programa de controle referem-se a todos os elementos gráficos pertencentes à linguagem LD que devem ser dispostos de uma forma organizada sobre os *rungs* do programa de controle e que, de acordo com a varredura do CP, determinam os resultados do processamento do programa de controle executado, podendo tais resultados serem associados aos endereços de saída do CP.

Assim, os elementos do programa de controle que são vistos no presente trabalho e considerados no procedimento de modelagem (Capítulo 5) estão os contatos do tipo NA

(normalmente aberto) e NF (normalmente fechado), as bobinas simples e as com retenção (*latch* e *unlatch*), o contador do tipo CTU (*count up*) e o temporizador do tipo TON (*timer on-delay*), de forma que as características destes elementos serão apresentadas adiante.

Como exemplos da associação entre endereços e elementos do programa de controle, conforme a definição proposta, tem-se: o endereço auxiliar de 1 bit dado por B3:0/0 (endereço do tipo booleano) pode ser associado a um elemento contato do tipo NA, NF ou a um elemento bobina, conforme pode ser visto na Figura 2.3 (*rungs* 0000 e 0002); o endereço T4:0.ACC (endereço do tipo inteiro) é associado a um elemento do programa de controle do tipo temporizador, indicando o valor do tempo acumulado pelo mesmo, e pode ser utilizado por um elemento do programa de controle que realiza uma operação de adição de valores inteiros (*rung* 0004 da Figura 2.3).

É importante destacar que a representação de um programa de controle escrito em LD que pode ser vista na Figura 2.3 utiliza a simbologia/codificação empregada em CPs da família SLC 500 da empresa Allen-Bradley. O presente trabalho optou por esta família de CPs em virtude de sua larga utilização e porque foi utilizada em Zoubek (2004), que trata-se de um dos dois estudos de casos que serão abordados no Capítulo 6.

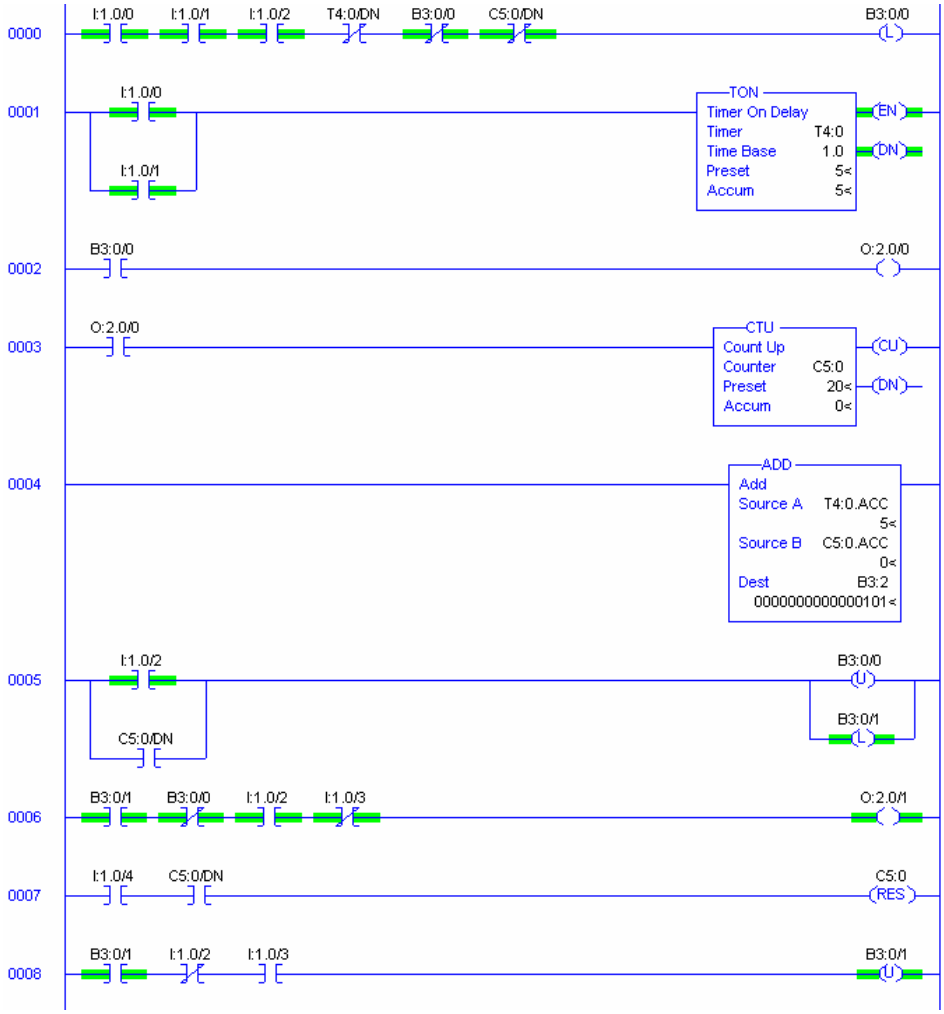


Figura 2.3 – Representação de um determinado programa de controle escrito em LD

A possibilidade de utilização de endereços do tipo booleano, como foi visto no caso do endereço auxiliar B3:0/0, confere à linguagem LD a possibilidade de poder ter seu programa de controle sendo representado através de expressões booleanas (MOON, 1994; MIYAGI, 1996; DE SMET et al., 2000; CHMIEL; HRYNKIEWICZ; MUSZYŃSKI, 2002; ZOUBEK, 2004). Tal fato contribuiu para que se pudesse utilizar MEFEs (Capítulo 3) no procedimento de modelagem (de MEFEs) a partir de programas de controle desenvolvidos em LD (Capítulo 5).

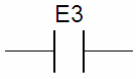
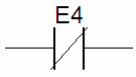
2.4.1 Elementos do programa de controle

A seguir serão abordados os elementos do programa de controle que serão contemplados no procedimento de modelagem a ser apresentado por este trabalho.

2.4.1.1 Contatos

Nos programas desenvolvidos em LD, elementos do programa de controle do tipo contato, a partir daqui, simplesmente, elementos contatos, podem ser associados a endereços externos e internos, conforme a necessidade exigida no desenvolvimento de tais programas. A representação, descrição e função dos dois tipos de elementos contatos, em geral, mais utilizados em programas de controle escritos em LD (CHMIEL; HRYNKIEWICZ; MUSZYŃSKI, 2002) podem ser vistas na Tabela 2.2[5]. É importante observar que foi adotada a grandeza física tensão elétrica na descrição da função dos elementos contatos abordados nesta tabela.

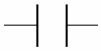
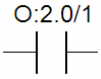
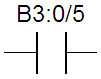
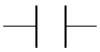
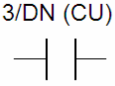
Tabela 2.2 – Elementos contatos que podem ser utilizados em programas de controle desenvolvidos em LD

Representação	**Descrição**	**Função**
	Elemento contato do tipo normalmente aberto (NA) associado a uma entrada discreta do CP, cujo endereço é aqui representado por E3.	Verifica se na entrada E3 do CP há uma tensão de 24VDC. Em caso afirmativo, diz-se que o elemento contato do tipo NA está ativado (nível lógico verdadeiro) e que o mesmo permite o fluxo de energia à sua direita, desde que haja fluxo à sua esquerda. Se o valor de tensão na entrada E3 for de 0VDC, diz-se que tal elemento contato está desativado (nível lógico falso) e, portanto, não é permitido o fluxo de energia à sua direita.
	Elemento contato do tipo normalmente fechado (NF) associado a uma entrada discreta do CP, cujo endereço é aqui representado por E4.	Verifica se na entrada E4 do CP há uma tensão de 0VDC. Em caso afirmativo, diz-se que o elemento contato do tipo NF está ativado (nível lógico verdadeiro) e que o mesmo permite o fluxo de energia à sua direita, desde que haja fluxo à sua esquerda. Se o valor de tensão na entrada E4 for de 24VDC, diz-se que tal elemento contato está desativado (nível lógico falso) e, portanto, não é permitido o fluxo de energia à sua direita.

[5] Os elementos contatos apresentados não possuem capacidade de retenção de sinal, o que é considerado no procedimento de mapeamento de programas de controle desenvolvidos em LD para MEFEs.

Além dos endereços de entrada (endereços externos), também podem ser associados aos elementos contatos, endereços que podem se referir às saídas (endereços externos) ou aos endereços internos do CP. Na Tabela 2.3 podem ser vistos alguns exemplos da associação entre elementos contatos e endereços (o que constitui uma instrução), exemplos estes que estão utilizando a simbologia/codificação empregada em CPs da família SLC 500 da empresa Allen-Bradley.

Tabela 2.3 – Associação entre endereços e elementos contatos

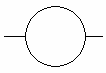
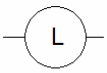
Instrução	**Descrição**
I:1.0/0 	Verifica se o endereço de entrada dado por I:1.0/0 possui uma tensão de 24VDC, o que caracterizará essa instrução como estando ativada (nível lógico verdadeiro).
O:2.0/1 	Verifica se o endereço de saída dado por O:2.0/1 possui um nível lógico verdadeiro, o que caracterizará essa instrução como estando ativada.
B3:0/5 	Verifica se o endereço auxiliar B3:0/5 possui um nível lógico verdadeiro, o que caracterizará essa instrução como estando ativada.
T4:1/DN (EN) (TT) 	Verifica se um dos três bits DN, EN ou TT associados ao elemento temporizador T4:1 possui um nível lógico verdadeiro, o que caracterizará essa instrução como estando ativada.
C5:3/DN (CU) (OV) 	Verifica se um dos três bits DN, CU ou OV associados ao elemento contador C5:3 possui um nível lógico verdadeiro, o que caracterizará essa instrução como estando ativada.

2.4.1.2 Bobinas

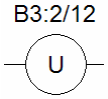
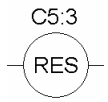
Segundo Miyagi (1996), os elementos do programa de controle do tipo bobina, a partir daqui, simplesmente, elementos bobinas, podem ser classificados em: bobina comum, bobina inversa, bobina de *set*/*reset*, bobina com memória e bobina detectora de variação. No presente trabalho serão abordados, apenas, os elementos bobinas do tipo comum e os do tipo *set*/*reset*, não havendo, porém, qualquer restrição para se considerar os outros tipos nas questões de modelagem apresentadas pelo presente trabalho[6]. Há, ainda, outro tipo de elemento bobina que é contemplado pelo presente trabalho, o qual está associado ao *reset* do valor de contagem do elemento do programa de controle contador do tipo CTU, sendo que as suas características serão apresentadas adiante.

Conclui-se esta subseção com a Tabela 2.4, onde são apresentados exemplos de associação entre elementos bobinas e endereços (formando uma instrução), exemplos estes que também utilizam a simbologia/codificação empregada em CPs da família SLC 500.

Tabela 2.4 – Associação entre endereços e elementos bobinas

Instrução	**Descrição**
O:2.1/3 	Se a condição lógica que está à esquerda do elemento bobina tiver resultado lógico verdadeiro, então o elemento bobina assumirá um nível lógico verdadeiro e no endereço de saída O:2.1/3 haverá uma tensão de 24VDC, enquanto o nível lógico à esquerda do elemento bobina se mantiver verdadeiro.
B3:2/12 	Se a condição lógica que está à esquerda do elemento bobina tiver um resultado lógico verdadeiro, então o endereço auxiliar B3:2/12 será ativado e possuirá nível lógico verdadeiro. Uma vez ativado esse elemento bobina, mesmo se o nível lógico do lado esquerdo dele for falso, o mesmo permanecerá com nível lógico verdadeiro. Essa instrução somente será desativada se for acionado o elemento bobina do tipo *unlatch* com o

[6] Pelo fato do presente trabalho adotar a simbologia/codificação utilizada por CPs da família SLC 500, para a representação dos elementos bobinas do tipo set/reset serão utilizados os termos "*latch*" (para *set*) e "*unlatch*" (para *reset*). Outras simbologias/codificações adotadas por outros CPs para a representação dos elementos bobinas do tipo *latch* e *unlatch* não são uma restrição para uma futura adaptação ao presente trabalho.

Instrução	Descrição
	mesmo endereço desse elemento bobina, ou seja, B3:2/12. Além disso, também é necessário que o nível lógico do lado esquerdo desse elemento bobina do tipo *latch* seja falso e que na seqüência de execução do programa de controle preceda o elemento bobina do tipo *unlatch*.
	Se a condição lógica que está à esquerda desse elemento bobina tiver um nível lógico verdadeiro, então o endereço auxiliar B3:2/12 será desativado e possuirá nível lógico falso, desde que também o nível lógico do lado esquerdo do elemento bobina do tipo *latch* de mesmo endereço também seja falso. Uma vez desativado, ainda que o nível lógico presente do lado esquerdo desse elemento bobina seja falso, o nível lógico da mesma será mantido como falso. Esse valor somente deixará de ser falso se for acionada o elemento bobina do tipo *latch* de mesmo endereço, ou seja, B3:2/12.
	Se a condição lógica que está à esquerda do elemento bobina tiver nível lógico verdadeiro, então o valor acumulado no elemento contador de endereço C5:3 sofrerá uma operação de *reset*, isto é, terá um valor acumulado igual a "0". Enquanto o nível lógico à esquerda desse elemento bobina for verdadeiro, não será possível o incremento deste elemento contador.

2.4.1.3 Contadores

Elementos do programa de controle do tipo contador, a partir daqui, simplesmente, elementos contadores, são, em geral, utilizados (em SPs) para se determinar, por exemplo, o valor do volume de produção de um determinado produto durante um dado intervalo de tempo. Esta contagem pode ser realizada, por exemplo, através de estímulos enviados a um endereço de entrada do CP a partir de um sensor de detecção de peça (dispositivo de detecção) ou a partir de um botão de indicação de peça produzida (dispositivo de comando).

A característica de funcionamento do elemento contador do tipo CTU (*count up*) utilizado em CPs da família SLC 500 é semelhante à característica de funcionamento do elemento contador do tipo CTU padronizado pela norma IEC 61131. Este fato contribui para que a MEFE do elemento contador do tipo CTU (utilizado em CPs da família SLC 500) tenha que sofrer poucas modificações para adequar-se à uma eventual modelagem deste mesmo elemento contador de acordo com a padronização estabelecida pela norma IEC 61131.

Para ilustrar a disposição física em um *rung* de um elemento contador do tipo CTU e as suas características funcionais, recorre-se ao exemplo da Figura 2.4. Neste exemplo,

sempre que ocorrer uma mudança de nível lógico falso para nível lógico verdadeiro do endereço de entrada I:1.0/0 associado ao elemento contato do tipo NA, o valor de contagem no elemento contador do tipo CTU (endereço C5:0) será incrementado em uma unidade, ficando este valor armazenado em um endereço interno dado por C5:0.ACC.

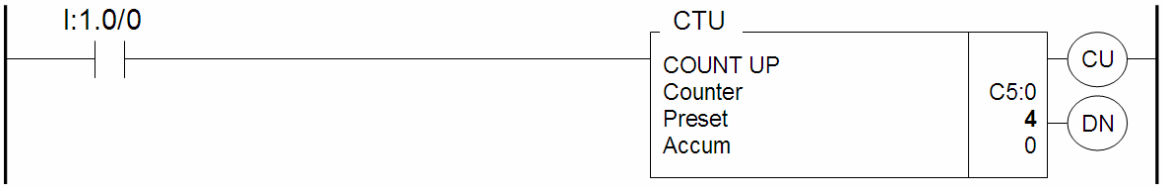


Figura 2.4 – Exemplo de rung com um elemento contador do tipo CTU de endereço C5:0

Tendo-se como referência o exemplo da Figura 2.4, a seguir são apresentados os componentes que fazem parte de um elemento contador do tipo CTU:

- *Counter*: estabelece o endereço (interno) do elemento contador. No exemplo da Figura 2.4, este endereço é dado por C5:0;
- *Preset*: é um valor limiar fixo que delimita o valor de incremento de contagem do elemento contador do tipo CTU. No exemplo da Figura 2.4, este valor é igual a "4";
- *Accum*: valor incrementado no elemento contador do tipo CTU, o qual é armazenado no endereço interno C5:0.ACC. Quando este valor for igual ao valor estabelecido em *Preset*, um endereço de formato booleano associado a este elemento contador terá modificado seu nível lógico de falso para verdadeiro. Este endereço é o *Count Done Bit* e seu formato é constituído pela junção entre o endereço do elemento contador do tipo CTU a que pertence e a extensão .DN (*done*). Assim, baseando-se no exemplo apresentado na Figura 2.4, este endereço é dado por C5:0.DN, o qual pode ser associado a elementos contatos do tipo NA e/ou NF.

O funcionamento do elemento contador do tipo CTU, baseando-se no exemplo da Figura 2.4, pode ser descrito da seguinte forma: quando este elemento contador é precedido no *rung* em que se encontra por uma mudança de nível lógico falso para verdadeiro, incrementa-se em uma unidade o valor do componente *Accum* (C5:0.ACC). Para que um novo incremento seja possível, esse nível lógico deve ser modificado para falso e, depois, para verdadeiro. Quando, finalmente, o valor do componente *Accum* for igual ao valor

estabelecido em *Preset*, o endereço associado a esse elemento contador que indica o final de incremento (C5:0.DN) terá seu nível lógico modificado de falso para verdadeiro. Outro endereço que está associado ao elemento contador e que indica que este elemento está sendo ativado é conhecido por *Count Up Bit*. O formato deste endereço é constituído pela junção entre o endereço do elemento contador do tipo CTU a que pertence e pela extensão .CU (*count up*). Assim, para um elemento contador do tipo CTU cujo endereço é C5:0, o endereço do *Count Up Bit* será dado por C5:0.CU. Ainda em relação à Figura 2.4, observam-se duas bobinas (CU e DN) do lado direito do elemento contador, as quais fazem parte deste elemento. Estas bobinas servem para representar visualmente, respectivamente, o acionamento do *Count Up Bit* e do *Count Done Bit*.

Na Figura 2.5, pode-se observar a relação entre a condição lógica (no mesmo *rung*) que precede o elemento contador do tipo CTU, os seus endereços e os seus estados, relação esta que baseia-se no exemplo da Figura 2.4.

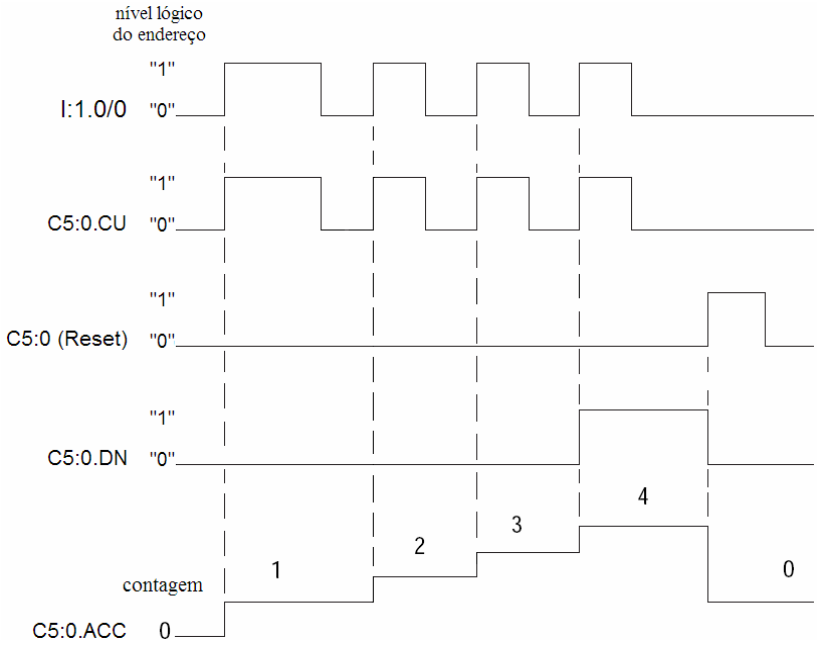


Figura 2.5 – Representação gráfica que mostra a relação entre a condição lógica (no mesmo rung) que precede o elemento contador do tipo CTU, os seus endereços e os seus estados

2.4.1.4 Temporizadores

Conforme Miyagi (1996), nos sistemas de controle de SEDs nota-se a necessidade não

somente de funções lógicas como AND/OR e funções de auto-retenção ou memória, mas também de elementos temporizadores que provoquem atrasos aos sinais. Assim, conforme é apresentado por Miyagi (1996), três tipos de elementos do programa de controle do tipo temporizador, a partir daqui, simplesmente, elementos temporizadores, podem ser citados: TP (*timer pulse*), TON (*timer on-delay*) e TOF (*timer off-delay*). O presente trabalho abordará o elemento temporizador do tipo TON, o qual pode ser considerado como o mais comumente utilizado para o desenvolvimento de programas de controle em LD (PARR, 2003). Para tal elemento temporizador, o presente trabalho apresenta, no Capítulo 5, uma proposta para a sua modelagem através de MEFEs e, no Capítulo 6, apresenta um estudo de caso com a aplicação do modelo deste elemento temporizador.

A representação do elemento temporizador do tipo TON empregado em CPs da família SLC 500 diferencia-se da representação do elemento temporizador do tipo TON proposto pela IEC 61131, porém ambos possuem as mesmas características funcionais. A Figura 2.6 apresenta o exemplo de um *rung* com um elemento temporizador do tipo TON de endereço T4:0.

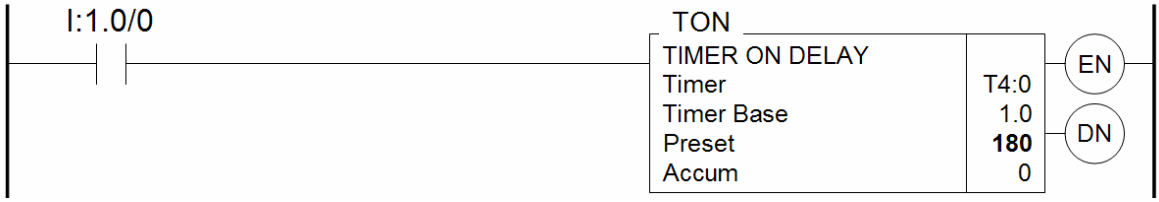


Figura 2.6 – Exemplo de rung com um elemento temporizador do tipo TON de endereço T4:0

No exemplo apresentado na Figura 2.6 sempre que ocorrer uma mudança de nível lógico falso para nível lógico verdadeiro do endereço de entrada I:1.0/0 associado a um elemento contato do tipo NA, o valor do tempo acumulado no elemento temporizador do tipo TON (endereço T4:0) será incrementado, ficando este valor armazenado em um endereço interno dado por T4:0.ACC.

Tendo-se como referência o exemplo apresentado na Figura 2.6, a seguir são apresentados os componentes que fazem parte de um elemento temporizador do tipo TON:

- *Timer*: estabelece o endereço (interno) do elemento temporizador do tipo TON. No exemplo da Figura 2.6, este endereço é dado por T4:0;

- *Timer Base*: é o fator multiplicativo para se determinar a precisão do incremento do tempo do elemento temporizador do tipo TON. Este valor pode ser 0,01s, 0,1s ou 1,0s;
- *Preset*: é um valor limiar fixo que delimita o valor de incremento do tempo acumulado no elemento temporizador do tipo TON. No exemplo apresentado na Figura 2.6, este valor é de 180s. Assim, se durante um período de 180s o elemento temporizador do tipo TON permanecer ativado de forma ininterrupta, ao término deste período um endereço de formato booleano associado a este elemento temporizador terá modificado seu nível lógico de falso para verdadeiro. Este endereço é conhecido por *Timer Done Bit* e seu formato é constituído pela junção entre o endereço do elemento temporizador do tipo TON a que pertence e a extensão .DN (*done*). Assim, no exemplo apresentado na Figura 2.6, este endereço é dado por T4:0.DN, o qual pode ser associado a elementos contatos do tipo NA e/ou NF;
- *Accum*: valor acumulado pelo elemento temporizador do tipo TON. Quando o valor de *Accum* for igual ao valor de *Preset*, o endereço T4:0.DN passará de nível lógico falso para nível lógico verdadeiro. No exemplo da Figura 2.6, o valor do tempo que é acumulado é armazenado no endereço interno T4:0.ACC.

Outro endereço que também está associado ao elemento temporizador do tipo TON e que indica a ativação do mesmo é conhecido por *Timer Enable Bit*. O formato de tal endereço é constituído pela junção entre o endereço do elemento temporizador do tipo TON a que pertence e a extensão .EN (*enable*). Assim, para um elemento temporizador do tipo TON cujo endereço é T4:0, o endereço do *Timer Enable Bit* será dado por T4:0.EN, endereço este que também pode ser associado a elementos contatos do tipo NA e/ou NF. Ainda em relação à Figura 2.6, observam-se duas bobinas (EN e DN) do lado direito do elemento temporizador, as quais fazem parte deste elemento. Estas bobinas servem para representar visualmente, respectivamente, o acionamento do *Timer Enable Bit* e do *Timer Done Bit*.

Finalmente, o último endereço que está associado ao elemento temporizador do tipo TON e que indica a sua atividade (permancendo ativo enquanto o elemento temporizador estiver acumulando tempo) é conhecido por *Timer Timing Bit*. O formato deste endereço é constituído pela junção entre o endereço do elemento temporizador do tipo TON a que

pertence e a extensão .TT (*timer timing*). Assim, para um elemento temporizador cujo endereço é T4:0, o endereço do *Timer Timing Bit* será dado por T4:0.TT, endereço este que também pode ser associado a elementos contatos do tipo NA e/ou NF (o *Timer Timing Bit* não será contemplado no modelo do elemento temporizador do tipo TON apresentado no presente trabalho). A diferença entre os endereços T4:0.EN e T4:0.TT é que T4:0.EN permance ativo mesmo depois do término do acúmulo de tempo realizado pelo elemento temporizador do tipo TON (endereço T4:0), desde que o mesmo ainda se mantenha ativado.

Na Figura 2.7 pode-se observar a relação entre a condição lógica (no mesmo *rung*) que precede o elemento temporizador do tipo TON, os seus endereços e os seus estados, relação esta que baseia-se no exemplo da Figura 2.6.

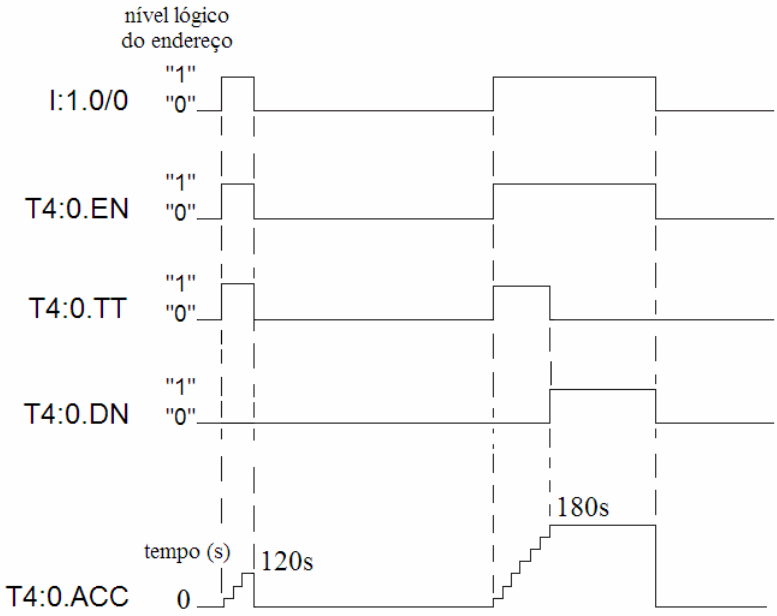


Figura 2.7 – Representação gráfica que mostra a relação entre a condição lógica (no mesmo rung) que precede o elemento temporizador do tipo TON, os seus endereços e os seus estados

Desta forma, estes elementos do programa de controle que foram descritos formam a base estrutural para o desenvolvimento das MEFEs que modelam tais elementos, conforme será apresentado no Capítulo 5.

CAPÍTULO 3

3. MÁQUINAS DE ESTADOS FINITOS ESTENDIDAS

Neste capítulo serão apresentadas as máquinas de estados finitos estendidas (MEFEs), técnica de modelagem utilizada no presente trabalho no procedimento que será apresentado no Capítulo 5. As MEFEs são consideradas como uma extensão das máquinas de estados finitos (MEFs) e estas, por sua vez, são derivadas dos autômatos finitos (AFs). Nesta direção, o capítulo será iniciado com uma introdução sobre autômatos finitos, passará, em seguida, por uma discussão sobre MEFs e será encerrado, finalmente, com uma abordagem sobre MEFEs.

3.1 AUTÔMATOS FINITOS

Em geral, as definições que são propostas para o termo "autômato" consideram a sua associação ao termo "finito", fato este que, conforme Cohen (1990), diz respeito ao número de possíveis estados deste autômato e ao número de símbolos do alfabeto de entrada (que será apresentado adiante) serem ambos finitos.

Assim, de forma introdutória, de acordo com Cohen (1990), um autômato finito (AF) é um modelo composto por:

1. um conjunto finito de estados, sendo que um destes estados é designado como sendo o estado inicial, e alguns destes estados (talvez nenhum) são designados como sendo os estados finais;

2. um alfabeto (de símbolos) de entrada, a partir do qual são formadas strings[7], de forma que deve ser lido um símbolo por vez;

3. um conjunto finito de transições que indica para cada estado e para cada símbolo do alfabeto de entrada qual será o próximo estado a ser ativado.

Para ilustrar o que foi descrito nesta definição, apresenta-se, a seguir, um determinado exemplo: se o estado ativo de um AF for *s* e o símbolo *e* (pertencente ao alfabeto de entrada associado a este AF) for reconhecido, então o estado ativo do AF em questão passará a ser *s*', executando-se uma função de transição dada por $s' = f(s, e)$.

Onde:

s' é o próximo estado a partir de *s*.

f é uma função de transição do tipo $S \times E \rightarrow S$ (onde *S* refere-se ao conjunto finito de estados do AF e *E* refere-se ao alfabeto de entrada associado a este AF).

Um AF pode ser representado (graficamente) por um grafo de transição entre estados (DEVADAS; KEUTZER; KRISHNAKUMAR, 1991), grafo este constituído por nós e arcos orientados, onde os nós representam os estados do AF e os arcos orientados representam as transições entre os estados deste AF (ZHOU, VENKATESH, 1999). Um exemplo desta representação gráfica que descreve a transição realizada pela função

[7] Conforme Hopcroft; Ullman e Motwani (2001), uma string (palavra ou, ainda, cadeia) é uma seqüência finita de símbolos escolhidos de algum alfabeto. Por exemplo, 01101 é uma string do alfabeto binário $\sum = \{0, 1\}$. A string 111 é outra escolhida deste alfabeto.

apresentada anteriormente é ilustrado na Figura 3.1.

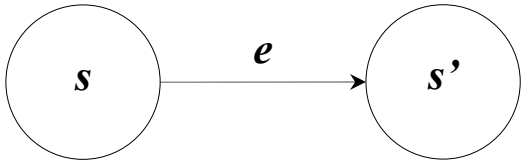


Figura 3.1 – Exemplo de representação gráfica de um AF

Os AFs podem ser utilizados para se realizar a modelagem de uma considerável variedade de elementos de hardware e software de sistemas computacionais e, conforme Hopcroft; Ullman e Motwani (2001), podem ser descritos alguns exemplos de sua utilização:

a) softwares para desenvolver e simular circuitos lógicos digitais;
b) analisadores léxicos de softwares compiladores;
c) softwares para examinar corpos de texto a fim de se encontrar ocorrências de palavras, frases ou outros padrões.

Conforme é apresentado em Menezes (2005), os AFs podem ser classificados como sendo determinísticos ou não deteminísticos. O termo "determinístico" associado a um AF, de acordo com Hopcroft; Ullman e Motwani (2001), refere-se ao fato de que para cada símbolo reconhecido pelo AF existe um e somente um estado bem definido para o qual o AF pode transitar a partir de seu estado atual; por outro lado, os AFs não-determinísticos podem ativar um conjunto de estados alternativos. A característica determinística presente em AFs é herdada pelas MEFEs propostas pelo presente trabalho[8], devido à necessidade existente em se contemplar nestas MEFEs a execução seqüencial e cíclica do ciclo operacional de um CP, e do processamento do programa de controle desenvolvido em LD.

Na literatura acadêmica encontram-se diversos trabalhos que tratam da modelagem de SEDs sendo realizada utilizando-se autômatos finitos determinísticos (AFDs) como, por

[8] MEFEs do modelo principal e dos modelos auxiliares dos elementos contadores e temporizadores (tais modelos serão apresentados no Capítulo 5).

exemplo, os trabalhos de Denning; Dennis e Qualitz (1978); Cassandras (1993); Hopcroft; Ullman e Motwani (2001); Wagner et al. (2006). Nestes trabalhos observa-se que os símbolos pertencentes ao alfabeto de entrada do AFD em questão são considerados como sendo os eventos discretos do SED modelado.

Conforme Hopcroft; Ullman e Motwani (2001), um exemplo desenvolvido em AFD para a modelagem de SEDs é o caso do modelo de um interruptor do tipo liga/desliga que é ilustrado na Figura 3.2. O modelo tem a capacidade de "memorizar" o seu estado atual, ou seja, se está no estado ligado ou no estado desligado, sendo que o símbolo (evento discreto) representado por pressionar, neste modelo, refere-se a um evento realizado por um usuário para modificar os estados descritos: se o AFD em questão estiver no estado desligado, a ocorrência do símbolo pressionar ativará o estado ligado e, quando o AFD se encontrar no estado ligado, a ocorrência do mesmo símbolo ativará o estado desligado.

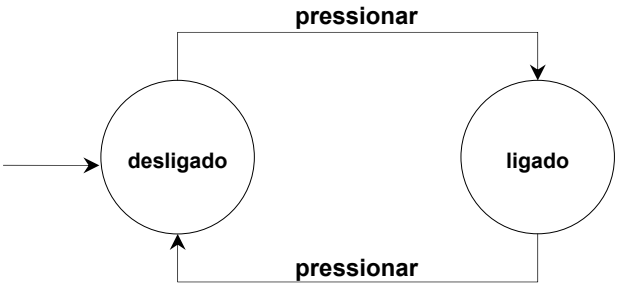


Figura 3.2 – AFD que modela o comportamento do funcionamento de um interruptor

Finalizando-se esta abordagem sobre os AFDs, ressalta-se que os modelos propostos e desenvolvidos em MEFEs que serão apresentados no decorrer deste trabalho herdam a caracterítica determinística presente em um AFD (HESSEL, 2007). Entretanto, numa MEFE, em vez de ser necessário o reconhecimento de símbolos pertencentes ao alfabeto de entrada para a execução da transição entre estados, é necessária a satisfação de condições de restrição impostas por expressões matemáticas às transições entre estados. Uma vez que tais condições de restrição sejam satisfeitas, a MEFE poderá transitar, a partir de um estado atual para um e somente um estado bem definido, não havendo a possibilidade de que sejam assumidos quaisquer estados alternativos.

Tendo-se finalizado estas considerações sobre os AFs, a seguir, será iniciada uma abordagem que discutirá sobre as máquinas de estados finitos (MEFs), dando-se ênfase às máquinas de Mealy.

Segundo Cohen (1990), os AFs com capacidade de execução de símbolos (de um alfabeto de saída) foram introduzidos, de forma independente, por George H. Mealy, em 1955, e por Edward F. Moore, em 1956. Tais variações de AFs passaram a ser chamadas, respectivamente, máquinas de Mealy e máquinas de Moore, em homenagem aos seus criadores, de forma que o objetivo de George H. Mealy e de Edward F. Moore foi o de projetar um modelo matemático que pudesse ser aplicado ao estudo de circuitos seqüenciais e, conforme consta em Booth (1967), estas máquinas são consideradas exemplos de MEFs.

3.2 MÁQUINAS DE MEALY

Uma diferença que existe entre as máquinas de Mealy e as máquinas de Moore está na forma pela qual os símbolos (pertencentes a um determinado alfabeto de saída) são executados: em uma máquina de Mealy, tais símbolos são executados durante a execução das transições entre os estados desta máquina; numa máquina de Moore, estes símbolos são executados com a ativação dos estados da máquina em questão (COHEN, 1990). No presente trabalho, as MEFEs que são utilizadas nos modelos propostos herdam a característica presente numa máquina de Mealy, onde o reconhecimento (restrição) e a execução (ação) de símbolos ocorrem na transição entre estados.

A máquina de Mealy trata-se de um modelo matemático que também herda a característica determinística presente nos AFDs, pois, conforme consta em Cohen (1990), cada estado desta máquina deve ter cada uma de suas transições (que partem do mesmo) associadas a um único símbolo pertencente ao alfabeto de entrada de tal máquina.

A partir destas considerações iniciais, apresenta-se uma definição matemática de uma máquina de Mealy, a qual, conforme Wagner et al. (2006), trata-se de um modelo formado por uma tupla do tipo (I, Q, S, s_0, f, w).

Onde:

> I representa um alfabeto de entrada (conjunto finito e não-vazio de símbolos).
>
> Q representa um alfabeto de saída (conjunto finito e não-vazio de

símbolos).

S representa um conjunto finito de estados.

$s_0 \in S$ representa o estado inicial.

f representa uma função de transição entre estados do tipo f: $S \times I \rightarrow S$.

w representa uma função de execução de símbolos do alfabeto de saída, sendo tal função do tipo w: $S \times I \rightarrow Q$.

A Figura 3.3 ilustra um exemplo de máquina de Mealy. Neste exemplo, observa-se que estando tal máquina, por exemplo, em um estado atual s_3, para que a mesma evolua para um estado futuro s_0 é necessário que a mesma reconheça o símbolo "0" pertencente ao alfabeto de entrada, alfabeto este que, neste caso, é composto pelos símbolos "0" e "1". O reconhecimento do símbolo "0" implica na execução do símbolo "1" pertencente ao alfabeto de saída, o qual também é composto pelos símbolos "0" e "1".

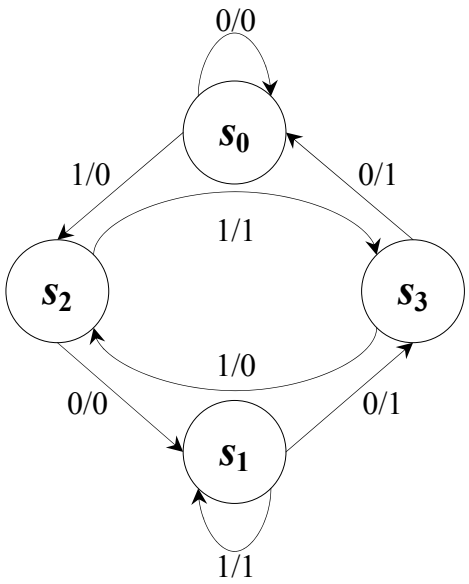


Figura 3.3 – Exemplo de uma máquina de Mealy

Para modelar os sistemas que fazem parte do escopo deste trabalho, faz-se necessário utilizar técnicas de modelagem que herdem a característica de uma máquina de Mealy, onde a restrição (reconhecimento de símbolos) e a ação (execução de símbolos) ocorrem na transição entre estados, sendo necessário também contemplar os seguintes requisitos

adicionais:

1. a primeira necessidade está no fato de que a restrição que é imposta (através do reconhecimento de símbolos pertencentes ao alfabeto de entrada) para se executar uma transição entre estados (máquina de Mealy), deve ser consituída por uma expressão relacional composta por variáveis de dados (do tipo inteiro ou booleano) e por valores numéricos inteiros ou booleanos;

2. outra necessidade consiste no fato de que a execução (de símbolos pertencentes ao alfabeto de saída) que ocorre depois que é satisfeita a restrição imposta às transições entre estados (máquina de Mealy), deve contemplar expressões de atribuição de valor compostas por variáveis de dados (do tipo inteiro ou booleano) e por valores numéricos inteiros ou booleanos.

Conforme será apresentado na seção a seguir, as MEFEs possuem a capacidade de atender a estes requisitos adicionais.

3.3 MÁQUINAS DE ESTADOS FINITOS ESTENDIDAS

Esta seção abordará o assunto máquinas de estados finitos estendidas (MEFEs), técnica utilizada no presente trabalho para a modelagem de programas de controle desenvolvidos em LD e executados por CPs. A próxima subseção (3.3.1) abordará alguns exemplos onde há diferentes aplicações de MEFEs. Nas subseções posteriores serão discutidos diversos aspectos referentes ao processo de modelagem de MEFEs utilizando-se a ferramenta de modelagem e verificação adotada pelo presente trabalho, chamada UPPAAL[9]. Tais aspectos referem-se, por exemplo, aos canais de comunicação entre MEFEs e aos estados

[9] O nome UPPAAL surgiu da junção dos nomes das universidades Uppsala e Aalborg, às quais pertencem os pesquisadores que têm desenvolvido a ferramenta computacional em questão. Diversos aspectos associados a esta ferramenta serão abordados no presente trabalho (http://www.uppaal.com).

compromissados, sendo que ambos são itens que serão vistos aplicados aos modelos em MEFEs propostos neste trabalho. A subseção 3.3.5 encerra este capítulo, apresentando a definição matemática para MEFE que será adotada neste trabalho.

3.3.1 Exemplos de aplicações de máquinas de estados finitos estendidas

No trabalho de Cheng; Krishnakumar (1993) é proposto um método de transformação da descrição de um circuito seqüencial dada em VHDL (VHSIC[10] Hardware Description Language) em uma MEFE. Desta forma, foi possível a execução de testes funcionais nos modelos criados em MEFE, de maneira a se evitar que erros pudessem ser observados somente depois da implantação da descrição em VHDL nos circuitos seqüenciais. Em um dos modelos de MEFEs apresentados no trabalho de Cheng; Krishnakumar (1993), observa-se a existência de variáveis de dados do tipo inteiro e do tipo booleano, as quais foram utilizadas tanto para a criação de restrições associadas às transições quanto para a criação de expressões de atribuição de valor. Uma restrição importante observada neste trabalho de Cheng; Krishnakumar (1993) é a não existência de canais de comunicação entre MEFEs.

No trabalho de Hong et al. (2002), encontra-se um exemplo onde uma MEFE realiza a modelagem do funcionamento de uma máquina de preparação de café, conforme é apresentado na Figura 3.4.

[10] VHSIC (ou Very High Speed Integrated Circuit) é um tipo de circuito lógico digital. O termo foi cunhado pelo departamento de defesa estadunidense durante um projeto que levou ao desenvolvimento da linguagem VHDL.

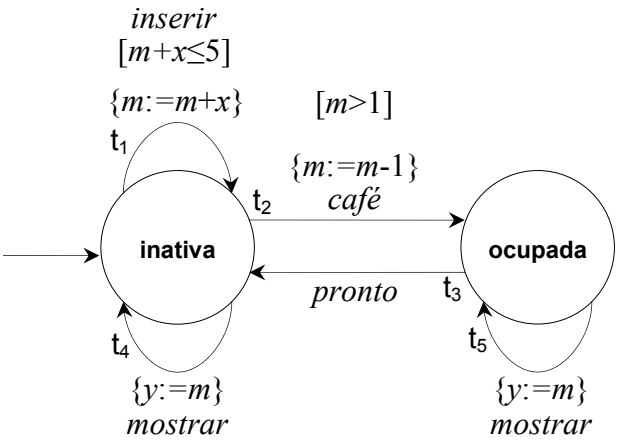


Figura 3.4 – Exemplo de modelo de uma máquina de preparação de café

A seguir, apresentam-se os elementos que constituem esta MEFE da Figura 3.4, tendo-se como referência a definição matemática para MEFEs que é proposta por Hong et al. (2002), e que será apresentada na última subseção deste capítulo. São os elementos:

- Estados: inativa e ocupada;
- Estado inicial: inativa;
- Eventos: *inserir*, *café*, *pronto* e *mostrar*;
- Transições: t_1, t_2, t_3, t_4 e t_5;
- Variáveis de dados: *x*, *m* e *y*;
- Restrições à execução de transições: $m+x \leq 5$ (t_1) e $m>1$ (t_2);
- Atribuição de valores às variáveis a partir da execução de transições: $m:=m+x$ (t_1), $m:=m-1$ (t_2) e $y:=m$ (t_4 e t_5).

O funcionamento desta máquina pode ser interpretado da seguinte forma: inicialmente, assumindo-se que a MEFE em questão esteja no estado inativa e que o valor de *m* (créditos acumulados na máquina) e o valor de *x* (créditos inseridos pelo usuário da máquina) sejam iguais a "0", é possível, somente, a execução da transição t_4, exeutando-se o evento *mostrar* e atualizando-se o valor de *y* (que indica o valor de créditos da máquina) através da expressão $y:=m$.

Para executar-se t_1 é necessário que a MEFE receba o evento *inserir*, pois a condição de restrição imposta por $m+x \leq 5$ está sendo satisfeita. Então, ao receber o evento *inserir*, executa-se a expressão $m:=m+x$ na transição t_1. Como *m* e *x* têm valor "0", então o valor de *m* permanecerá em "0". Se forem inseridos, por exemplo, 2 créditos (*x* igual a "2"), quando

a MEFE receber o evento *inserir*, o valor de *m* será atualizado para "2", permitindo-se, assim, que seja possível a execução de t_2, pois satisfaz-se a condição de restrição imposta por $m>1$. Ao executar-se t_2, executa-se o evento *café*, decrementa-se o valor de *m* ($m:=m-1$) e ativa-se o estado ocupada. Neste estado, a MEFE executa a transição t_5, executando-se o evento *mostrar* e atualizando-se o valor de *y* ($y:=m$), de forma que é mostrada ao usuário a quantidade de créditos presentes na máquina. Somente quando a MEFE receber o evento *pronto* é que a mesma irá para o estado inativa, executando-se a transição t_3.

Outro trabalho onde podem ser encontradas MEFEs é o trabalho de Huang; Liu (2004), o qual as utiliza para identificar a quantidade de subsistemas que poderiam ser desabilitados em um determinado circuito eletrônico de forma a ocorrer o menor consumo possível de energia elétrica. Huang; Liu (2004) comentam que blocos de controle que comumente são utilizados em circuitos digitais seqüenciais poderiam ser modelados através de MEFs. Porém, os circuitos digitais que são tratados por Huang; Liu (2004) precisam considerar elementos como contadores, registradores de deslocamento, pilhas e filas, e então, torna-se necessária a utilização de MEFEs, pois estas permitem que sejam levadas em consideração variáveis booleanas e inteiras necessárias à modelagem. Conforme observado em Cheng; Krishnakumar (1993), há também nesta proposta a restrição de não se poder representar canais de comunicação entre MEFEs.

O próximo trabalho a ser apresentado é o trabalho de Robinson-Mallett et al. (2005). Para que a discussão deste trabalho seja realizada, é necessário conhecer a função dos canais de comunicação e dos estados compromissados encontrados em tal trabalho, que são itens que podem ser aplicados a modelos desenvolvidos com a ferramenta UPPAAL. Da mesma forma que Robinson-Mallett et al. (2005), o presente trabalho também utiliza estes itens nos modelos de MEFEs propostos e, por este motivo, os canais de comunicação e os estados compromissados serão abordados nas duas próximas subseções (3.3.2 e 3.3.3), dada a importância destes conteúdos no contexto do presente trabalho.

3.3.2 Canais de comunicação entre MEFEs

Os canais de comunicação são recursos que podem ser aplicados ao desenvolvimento de MEFEs, sendo utilizados em situações onde se deseja que ao se realizar uma ação em uma determinada MEFE haja uma reação em outra MEFE (ou outras MEFEs). Tais canais

podem ser aplicados ao desenvolvimento de modelos utilizando-se a ferramenta UPPAAL (YI; PETTERSSON; DANIELS, 1994; LARSEN; PETTERSSON; YI, 1997) e podem ser classificados como sendo de dois tipos:

- canais de comunicação externos e;
- canais de comunicação internos.

Os canais de comunicação externos, também conhecidos como canais de sincronismo, utilizam símbolos que são constituídos por letras e pelos sufixos "?" e "!" designando, respectivamente, eventos de recepção e de emissão entre MEFEs (HESSEL, 2007). Na Figura 3.5 é possível observar um exemplo de aplicação de canais de comunicação externos, onde um canal de comunicação é estabelecido entre duas MEFEs através de a! e de a?.

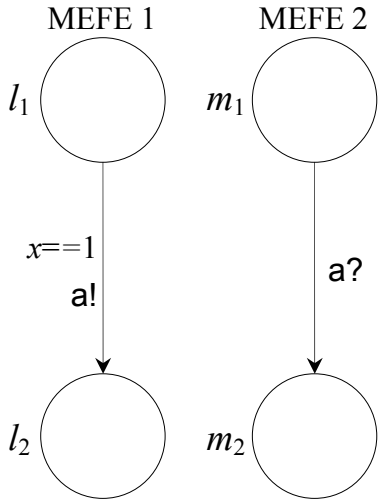


Figura 3.5 – Exemplo de aplicação de canais de comunicação externos

Conforme o exemplo da Figura 3.5, partindo-se de uma condição inicial onde os estados ativos das duas MEFEs são "l_1" e "m_1", para que a MEFE 2 possa evoluir de "m_1" para "m_2", é necessário que ela receba um evento gerado a partir do canal de comunicação externo de emissão a!, recebendo o evento através do canal de comunicação externo de recepção a?. Os canais de comunicação externos de recepção, enquanto não recebem o evento gerado pelo canal de comunicação externo de emissão ao qual está associado, apresentam uma condição de restrição à execução das transições onde estão dispostos. Portanto, na MEFE 2, a? representa uma condição de restrição. Logo, para que possa ocorrer a transição entre os estados das MEFEs em questão, a MEFE 1, que possui uma

condição de restrição dada pela expressão x==1, precisa satisfazer tal restrição, para que o canal de comunicação externo de emissão a! seja executado e realize, desta forma, a habilitação da restrição da MEFE 2.

Canais de comunicação internos são compostos por variáveis de dados do tipo inteiro ou booleano, cujos valores são compartilhados entre MEFEs (na subseção 3.3.5 serão mostrados os subconjuntos V_I e V_O, os quais contemplam estes canais de comunicação internos). Na Figura 3.6 é possível observar um exemplo de aplicação de canais de comunicação internos, onde um canal de comunicação é estabelecido entre duas MEFEs (MEFE 1 e MEFE 2) utilizando-se a variável x.

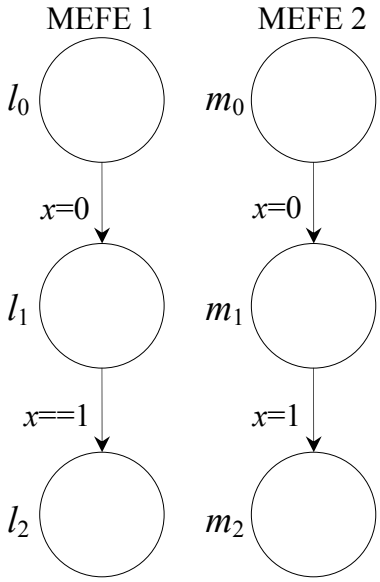


Figura 3.6 – Exemplo de aplicação de canais de comunicação internos

Conforme o exemplo da Figura 3.6, partindo-se de uma condição onde os estados ativos das duas MEFEs são "l_1" e "m_1" e a variável x possui valor zero, para que a MEFE 1 possa evoluir de "l_1" para "l_2" é necessário que a condição de restrição imposta pela expressão x==1 seja satisfeita. Neste cenário, como x possui valor igual a zero, essa evolução não será possível neste instante. Por sua vez, na MEFE 2, não há quaisquer condições de restrição entre os estados "m_1" e "m_2"; há somente uma operação de atribuição de valor à variável x, onde atribui-se o valor "1" à essa variável. Desta forma, quando houver a evolução do estado "m_1" para o estado "m_2" na MEFE 2, o valor de x fará com que a condição de restrição existente na transição entre l_1 e l_2 possa ser satisfeita, permitindo-se, então, a evolução da MEFE 1.

No presente trabalho faz-se necessária a utilização destes dois tipos de canais de comunicação para atender à necessidade de executar a comunicação entre a MEFE principal e as auxiliares, sendo necessário tratar a questão do sincronismo com prioridade, conforme será visto com mais detalhes na subseção a seguir (3.3.3), com a apresentação dos estados compromissados.

3.3.3 Estados compromissados

Os estados compromissados são estados que, ao serem ativados, devem ser imediatamente desativados, de forma a ser executada, prioritariamente, a transição que parte de tal estado, desde que a condição de restrição de tal transição seja satisfeita. Segundo Pettersson (1999), os estados compromissados (ou locais compromissados) foram criados com o objetivo de se obter flexibilidade para o desenvolvimento de modelos e melhorar o desempenho computacional nas verificações de tais modelos, de maneira que a utilização destes estados compromissados, na ferramenta UPPAAL, proporciona duas reduções potenciais na quantidade armazenada de estados explorados por esta ferramenta:

- reduz-se o valor de interações entre os modelos e, conseqüentemente, o número de estados explorados armazenados na memória da ferramenta;
- uma seqüência de estados compromissados é semanticamente executada de forma instantânea, de forma que armazena-se na memória da ferramenta somente o estado anterior a esta seqüência, e os estados compromissados desta seqüência não são armazenados.

Como resultado da utilização de estados compromissados, explora-se e armazena-se um número significativamente menor de estados simbólicos (PETTERSSON, 1999), os quais serão apresentados na definição matemática de MEFEs que será vista na subseção 3.3.5. Os estados compromissados podem ser aplicados a modelos de autômatos temporizados estendidos com variáveis de dados e a modelos de MEFEs, sendo que em Larsson (2000) encontram-se regras que estão relacionadas à evolução dos modelos que contêm tais estados. Neste sentido, as regras que são aplicadas às MEFEs serão apresentadas através de dois exemplos.

No exemplo ilustrado na Figura 3.7 apresentam-se duas MEFEs que fazem parte de um modelo composto por outras MEFEs, as quais não são ilustradas. A utilização de um

estado compromissado na MEFE 1 deve-se ao fato de que o objetivo do modelo é o de fazer com que o valor do tipo inteiro dado pelo número "5" seja enviado à variável j, de tal forma que, mesmo se houver outras MEFEs no modelo em condições para realizar uma atribuição de valor à variável j, a transmissão do valor "5" para a variável j será priorizada durante a simulação e/ou verificação do modelo, pois tão logo o estado l_2 seja ativado, o mesmo deverá ser imediatamente desativado (por ser do tipo compromissado).

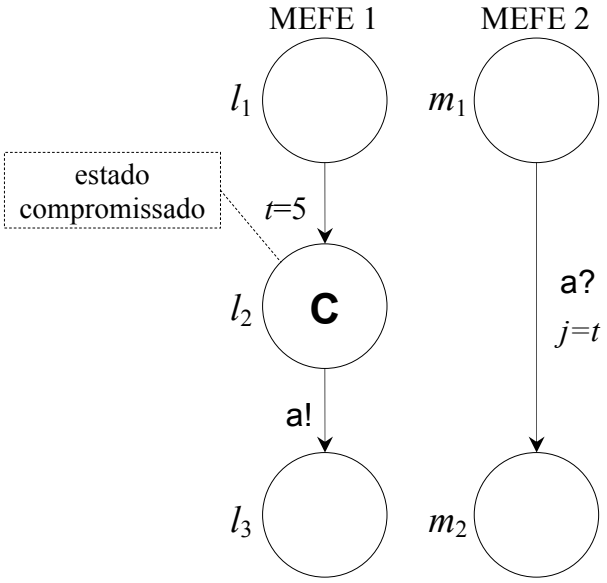


Figura 3.7 – Exemplo da aplicação de um estado compromissado

O próximo exemplo apresentado pelo presente trabalho é adaptado de Larsen; Pettersson e Yi (1997), sendo ilustrado na Figura 3.8. Neste exemplo, apresentam-se as MEFEs 1, 2 e 3, as quais fazem parte de um modelo composto por diversas MEFEs que interagem entre si. O objetivo das MEFEs ilustradas na Figura 3.8 é o de realizar o seguinte procedimento: quando a MEFE 1 executar o sincronismo com a MEFE 2, imediatamente em seguida, a MEFE 1 deve também realizar o sincronismo com a MEFE 3, sendo que ambos os sincronismos são obtidos através da utilização de canais de comunicação externos.

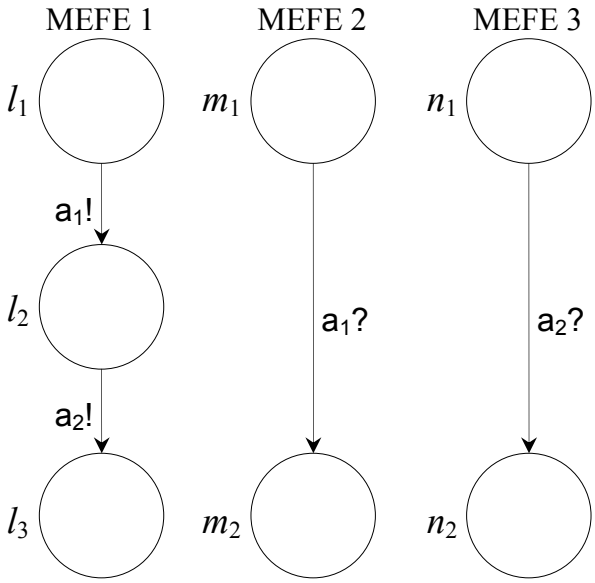


Figura 3.8 – Exemplo para a aplicação de um estado compromissado com canais de comunicação externos

Entretanto, da forma como é ilustrada na Figura 3.8, tal operação poderá não ser executada conforme a premissa estabelecida. O problema poderá ocorrer da seguinte forma: quando a MEFE 1 executa a transição entre os estados "l_1" e "l_2", executa-se o sincronismo entre as MEFEs 1 e 2 através dos canais de comunicação externos $a_1!$ e $a_1?$. Ao ativar o estado "l_2", a MEFE 1 não é forçada a desativar este estado e ativar o estado "l_3" e, como conseqüência disto, as outras MEFEs pertencentes ao modelo poderão realizar a execução de suas transições caso estas estejam habilitadas, fato este que não respeitará a premissa que foi estabelecida para a execução das MEFEs 1, 2 e 3.

Para fazer com que a premissa para a execução das MEFEs 1, 2 e 3 seja respeitada, deve ser designado o estado "l_2" da MEFE 1 como sendo um estado compromissado. Assim, tão logo o estado compromissado "l_2" seja ativado, o mesmo deve ser imediatamente desativado durante a simulação e/ou verificação do arranjo, permitindo-se que o sincronismo entre a MEFE 1 e a MEFE 3 seja realizado imediatamente depois do sincronismo entre a MEFE 1 e a MEFE 2. Na Figura 3.9 apresenta-se o modelo resultante da aplicação do estado compromissado ao local "l_2" da MEFE 1.

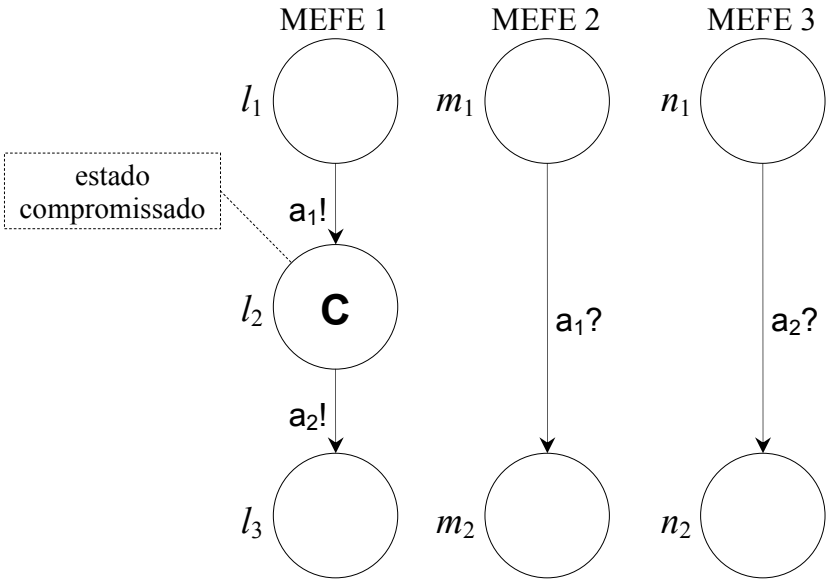


Figura 3.9 – Exemplo da aplicação de um estado compromissado

3.3.4 UPPAAL e a modelagem de MEFEs

O trabalho de Robinson-Mallett et al. (2005) apresenta que uma MEFE é uma extensão das máquinas de Mealy, às quais são adicionadas restrições à execução das transições e operações de atualização de variáveis depois das restrições serem satisfeitas, sendo que o trabalho trata da geração de mínimas Seqüências de Distinção[11] para modelos de MEFs e MEFEs. Uma Seqüência de Distinção deve ser capaz de permitir a identificação da execução de cada estado de uma máquina de estados, a partir da observação do comportamento das saídas desta máquina. Robinson-Mallett et al. (2005) utilizaram a ferramenta de modelagem e verificação UPPAAL para a construção dos modelos de MEFEs utilizados em seu trabalho e um exemplo é ilustrado na Figura 3.10.

[11] Entende-se por mínimas Seqüências de Distinção o menor caminho percorrido em termos de número de transições entre estados envolvendo a comunicação entre MEFEs com o objetivo de se avaliar se determinados estados são alcançáveis.

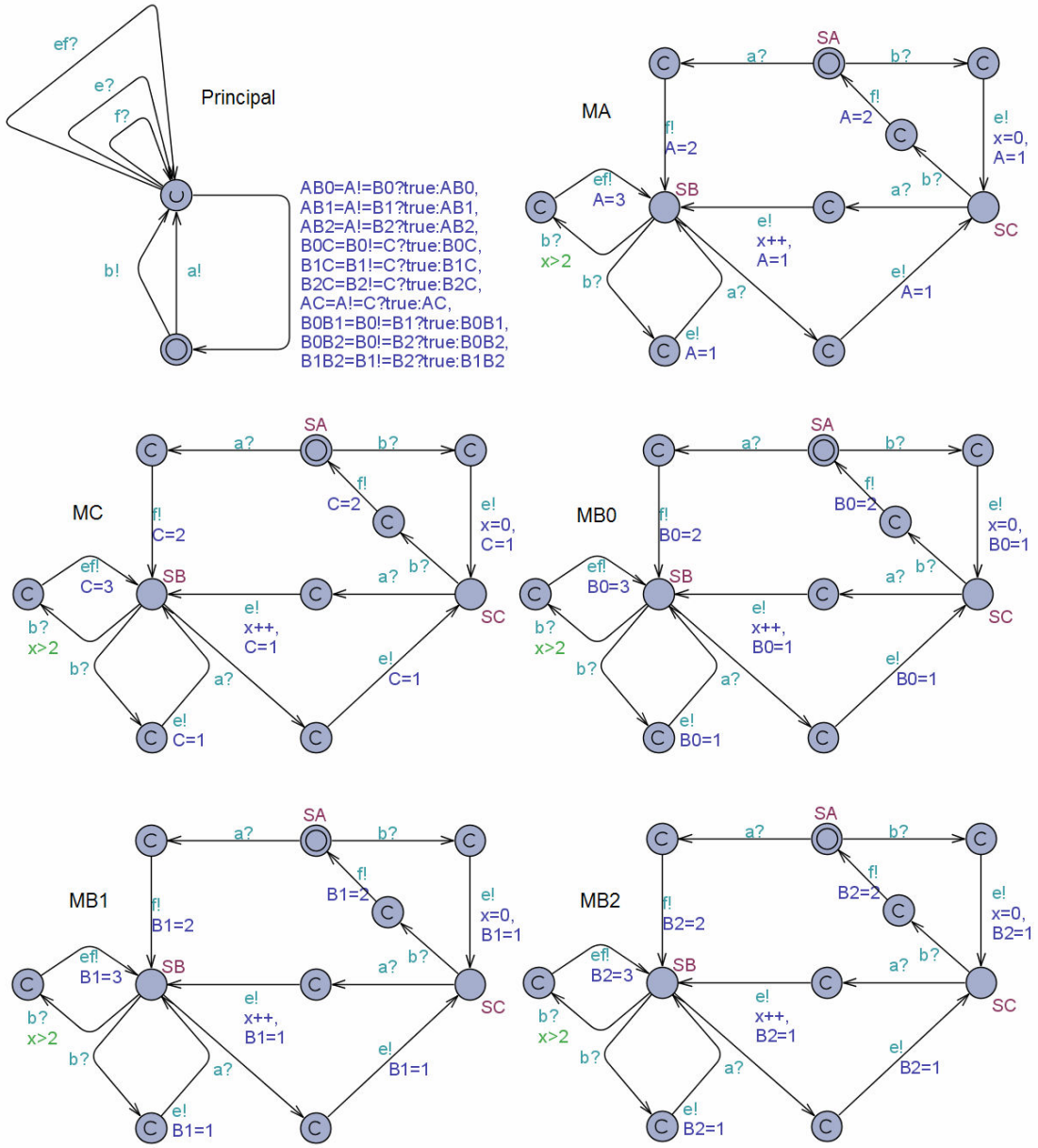


Figura 3.10 – Representação gráfica de um modelo com MEFEs desenvolvidas por Robinson-Mallett et al. (2005) utilizando a ferramenta UPPAAL

O objetivo da Figura 3.10 é ilustrar a modelagem de MEFEs com a ferramenta UPPAAL, de forma que nesta figura observam-se os seguintes elementos:

- canais de comunicação externos entre as MEFEs (a? e a!, b? e b!, d? e d!, e? e e! e ef? e ef!);
- estados compromissados;
- condições de restrição compostas pela variável de dados do tipo inteiro x à

execução de determinadas transições (x>2, x<=2 e x==0);

- operação para o incremento do valor da variável x (x++);
- operações para a atualização do valor de variáveis booleanas[12] (AB0, AB1, AB2, B0C, B1C, B2C, AC, B0B1, B0B2 e B1B2) na MEFE Principal;
- variáveis de dados do tipo inteiro (A, B0, B1, B2 e C).

Um exemplo de uma possível execução deste modelo da Figura 3.10 é o seguinte: se a MEFE MA estiver com o estado SA ativado e receber (via canal de comunicação externo de recepção a?) a habilitação da MEFE Principal para a execução da transição entre o estado SA e o estado compromissado imediatamente depois de SA, ao ativar o estado compromissado imediatamente depois do estado SA, a MEFE Principal não poderá executar quaisquer de suas transições, pois na MEFE MA está ativo o estado compromissado. Este estado compromissado deve ser imediatamente desativado e, então, é executado o canal de comunicação externo f!, fazendo com que a MEFE MA ative o estado SB e que a transição na MEFE Principal onde há o canal de comunicação externo f? seja executado, realizando-se um sincronismo entre estas MEFEs.

3.3.5 Definição matemática adotada para o estudo de MEFEs

A definição matemática que é adotada, neste trabalho, para o estudo de MEFEs é a que é proposta por Hong et al. (2002), cuja definição proposta é válida para MEFEs classificadas como determinísticas. Desta forma, conforme Hong et al. (2002), uma MEFE é um modelo matemático formado por uma tupla do tipo (S, S_0, E, V, T).

Onde:

---

[12] Este tipo de expressões apresentadas na MEFE Principal onde são atribuídos valores às variáveis booleanas constituem-se em um recurso adicional da ferramenta UPPAAL para a modelagem de MEFEs. Por exemplo, a expressão AB0=(A!=B0 ? true : AB0) significa: se (?) o valor de A for diferente (!=) do valor de B0, atribui-se nível lógico verdadeiro (true) à variável AB0. Caso contrário (:), atribui-se o valor atual de AB0.

S representa um conjunto finito de estados.

$S_0 \subseteq S$ representa um conjunto de estados iniciais.

E representa um conjunto finito de eventos (de recepção e de emissão).

V representa um conjunto finito de variáveis de dados do tipo inteiro e/ou booleano que são subdivididas em três conjuntos distintos formados por V_I, V_L, V_O, onde V_I refere-se às variáveis globais de entrada, V_L refere-se às variáveis locais e V_O refere-se às variáveis globais de saída.

T representa um conjunto finito de transições.

Segundo Hong et al. (2002), uma transição de uma MEFE é representada por uma tupla do tipo (s, e, g, A, s'), onde s e $s' \in S$, $e \in E$, g representa condições de restrição compostas por expressões formadas por números inteiros e pelas variáveis pertencentes a $V_I \cup V_L$, e A representa um conjunto formado por expressões relacionais para atribuição de valor às variáveis pertencentes a $V_L \cup V_O$.

Em Hong et al. (2002), o conjunto finito de eventos E está relacionado a eventos externos, e Hessel (2007), utilizando-se do mesmo exemplo de Hong et al. (2002), propõe que aos eventos de recepção e de emissão em uma MEFE associe-se uma tupla de parâmetros formais. No presente trabalho assume-se que o conjunto de eventos de recepção e de emissão E é formado pelos canais de comunicação externos (de recepção e de emissão) existentes num modelo constituído por MEFEs como, por exemplo, os canais que foram apresentados no exemplo da Figura 3.10.

Para a utilização da ferramenta UPPAAL é necessário representar-se estados simbólicos, os quais servirão para a estipulação de proposições em lógica temporal para a verificação de modelos. Em Hessel (2007), que é um trabalho que utiliza-se das contribuições do trabalho de Hong et al. (2002) para tratar do assunto MEFEs, encontra-se uma abordagem matemática para os estados simbólicos de MEFEs, a qual será a adotada pelo presente trabalho. Desta forma, de acordo com Hessel (2007), um estado simbólico é formado por uma tupla do tipo (l, σ), onde l representa o estado ativo da MEFE e σ é um mapeamento de valores às variáveis de V, sendo tais valores pertencentes ao domínio das variáveis de V. Por uma questão de convenção, a letra l (que representa o estado ativo de uma MEFE)

utilizada por Hessel (2007) será substituída pela letra *s*, a qual vem sendo utilizada para representar estados no presente trabalho. Portanto, um estado simbólico será designado com sendo formado por uma tupla do tipo (s, σ). Neste contexto, resta ainda ressaltar que no trabalho de Hessel (2007), a definição matemática para os estados simbólicos é realizada levando-se em consideração somente uma MEFE, fato este que implica numa limitação da abrangência de *s*, o qual é restrito a apenas um estado ativo de uma determinada MEFE. Em virtude do fato de que no presente trabalho diversos modelos desenvolvidos são compostos por mais de uma MEFE, o presente trabalho considerará que *s* representa o conjunto dos estados ativos de todas as MEFEs que constituem um determinado modelo de interesse.

O estado simbólico é importante para o processo de verificação de modelos, pois, com a utilização de ferramentas computacionais de verificação, pode-se avaliar se um determinado conjunto de variáveis e estados pertencentes ao modelo sob verificação poderá ser alcançado. Quando este modelo sob verificação trata-se do modelo de um determinado programa de controle desenvolvido em LD para um CP, é possível investigar se um estado específico deste programa de controle modelado poderá ser alcançado, evidenciando-se, assim, a presença ou a ausência de erros no programa de controle modelado. Neste sentido, no próximo capítulo serão apresentadas abordagens em relação à verificação de modelos.

CAPÍTULO 4

4. VERIFICAÇÃO DE MODELOS

Este capítulo será iniciado com uma abordagem sobre a verificação de modelos e sua relação com a checagem de modelos. Serão apresentadas as proposições da versão simplificada da lógica temporal CTL, as expressões proposicionais, os operadores de lógica temporal e exemplos de proposições. Em seguida, será realizada uma abordagem sobre a ferramenta de modelagem e verificação de modelos adotada pelo presente trabalho, discutindo-se também as justificativas para a sua escolha. O capítulo será encerrado com considerações sobre as propriedades de alcançabilidade e de segurança de modelos de estados finitos que podem ser submetidos à verificação.

4.1 CONSIDERAÇÕES INICIAIS

Conforme consta no Capítulo 1, "*verificação é um processo onde são realizados testes exaustivos em modelos de estados finitos desenvolvidos a partir de programas de controle existentes, executados por um CP, com o objetivo de identificar se um determinado estado específico do programa modelado pode ser alcançado ou se tal estado específico não pode ser alcançado*". A partir deste capítulo, o termo "estado específico" estará associado ao termo "expressão proposicional". Pode-se dizer, concisamente, que um estado específico está relacionado a um conjunto pré-estabelecido de endereços de um programa de controle e seus respectivos valores, enquanto que a expressão proposicional está relacionada a este mesmo conjunto pré-estabelecido de endereços e seus respectivos valores encontrados na forma modelada em MEFEs originadas a partir do programa de controle em questão. A expressão proposicional, além do conjunto pré-estabelecido de endereços e seus respectivos valores, relaciona também os estados destas MEFEs, conforme será apresentado adiante.

4.2 A VERIFICAÇÃO DE MODELOS

Segundo Villani; Miyagi e Valette (2006), a verificação de modelos está associada à sua possível dicotomia em: prova através de teoremas e checagem de modelos. No primeiro caso, prova através de teoremas, o objetivo é inferir ou contradizer uma propriedade comportamental (do sistema modelado) através da utilização de provas lógicas e, no segundo caso, checagem de modelos, uma propriedade de interesse é verificada pela determinação de todos os conjuntos de estados alcançáveis do modelo do sistema e pela comprovação de ele (modelo) representa um modelo lógico da propriedade requerida. Ainda, conforme Villani; Miyagi e Valette (2006), a vantagem do primeiro caso é que o mesmo não se restringe a sistemas de estados finitos, enquanto que a vantagem do segundo caso, é que a checagem de modelos pode ser realizada automaticamente através de ferramentas computacionais para esta finalidade.

Alur; Courcoubetis e Dill (1990) propuseram que a checagem de modelos é um método de verificação de sistemas concorrentes no qual um modelo do comportamento deste sistema é comparado com uma proposição de lógica temporal. A lógica temporal é um formalismo utilizado no processo de checagem de modelos para descrever uma seqüência de transições entre estados, sendo que, neste tipo de lógica, o tempo é mencionado de uma forma hierárquica entre os estados, utilizando-se termos lingüísticos específicos como, por exemplo: *possivelmente*, *sempre*, *nunca* etc.

Clarke et al. (2000) reforçam a proposta de Alur; Courcoubetis e Dill (1990), afirmando que a checagem de modelos trata-se de um processo automático de verificação de modelos de sistemas concorrentes de estados finitos, onde são estipuladas especificações através de proposições de lógica temporal, as quais são checadas durante um processo de busca exaustiva pelo espaço de estados derivado dos modelos de sistemas sob análise. Neste sentido, um sistema concorrente de estados finitos pode ser entendido como sendo:

> Um conjunto de componentes que realizam execuções de forma simultânea. Normalmente, tais componentes apresentam algumas formas de se comunicarem. O modo de execução e o modo de comunicação podem ser diferentes de um sistema para outro. Assumem-se dois modos de execução: execução assíncrona/intercalada, na qual apenas um componente executa um passo por vez, e a síncrona, no qual todos os componentes executam um passo ao mesmo tempo.

> Os componentes, por sua vez, podem comunicar-se através da modificação dos valores de variáveis compartilhadas ou por troca de mensagens. (CLARKE; GRUMBER; PELED, 1999, p. 17)

Conforme Clarke; Grumber e Peled (1999), exemplos de sistemas concorrentes de estados finitos são os circuitos digitais síncronos e assíncronos, programas assíncronos de computadores etc. Conforme apresentado no Capítulo 3 e baseando-se na definição apresentada anteriormente, um modelo constituído por MEFEs possui as características de execução e de comunicação descritas nesta definição, o que permite considerá-lo como sendo um modelo de sistemas concorrentes de estados finitos.

Neste trabalho, a verificação de modelos que é abordada trata-se de uma checagem de modelos, de acordo com os conceitos apresentados. Entretanto, será mantido e utilizado o termo "verificação de modelos", uma vez que não provoca qualquer prejuízo conceitual. O processo de verificação de um determinado modelo[13] de sistemas concorrentes de estados finitos pode ser representado, basicamente, pelo modelo da Figura 4.1.

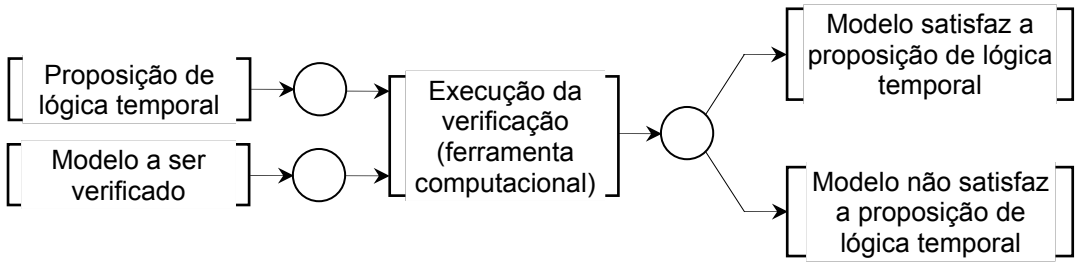


Figura 4.1 – Modelo que representa os elementos constituintes de um processo de verificação de modelos de sistemas concorrentes de estados finitos

A proposição de lógica temporal e o modelo (de sistemas concorrentes de estados finitos) a ser verificado constituem as informações de entrada da ferramenta computacional de

[13] O modelo apresentado na Figura 4.1 foi construído utilizando-se a técnica PFS (Production Flow Schema). Mais detalhes sobre esta técnica serão apresentados na seção inicial do Capítulo 5.

verificação. Para a verificação dos modelos propostos por este trabalho são utilizadas proposições que compõem uma versão simplificada da lógica temporal CTL (Computation Tree Logic) (BEHRMANN; DAVID; LARSEN, 2004; BENGTSSON; YI, 2004; ZOUBEK, 2004; ANAND et al., 2006), que é suportada pela ferramenta de modelagem e verificação UPPAAL.

Segundo Clarke; Emerson e Sistla (1986); Clarke et al. (2000); Hong et al. (2001); Hong et al. (2002), um modelo de sistemas concorrentes de estados finitos é uma condição necessária para permitir que uma ferramenta computacional realize o trabalho de verificação de forma automática, finalizando tal trabalho apresentando uma resposta na qual o modelo satisfaz (ou não) a proposição de lógica temporal estipulada. Entretanto, deve-se observar que, mesmo tratando-se da verificação de modelos de sistemas concorrentes de estados finitos, durante o processo de verificação destes modelos poderá ocorrer uma explosão combinatória de estados, e tal fato poderá implicar na não finalização deste processo (CLARKE; GRUMBERG; PELED, 1999), fato este que pode ocorrer, inclusive, com a ferramenta de modelagem e verificação UPPAAL, conforme foi constatado por Zoubek (2004)[14] e pelo presente trabalho.

4.2.1 Proposicões da lógica temporal CTL

Conforme já foi apresentado, é possível a utilização de proposições da lógica temporal CTL para se realizar a verificação de um determinado modelo de sistemas concorrentes de estados finitos através de ferramentas computacionais de verificação (HOLZMANN,

[14] Segundo Zoubek (2004), o problema da explosão combinatória de estados ocorre quando o modelo a ser verificado é relativamente complexo, de forma que o espaço combinatório de estados cresce exponensialmente com o número de processos paralelos deste modelo (CLARKE; GRUMBERG; PELED, 1999). Desta forma, poderá acontecer a situação em que o processo de busca realizado neste espaço de estados possa não ser realizado em um intervalo de tempo considerado aceitável, ou ainda, que tal processo de busca possa ser limitado devido a um computador (ou cluster de computadores) não possuir memória suficiente ou outros recursos computacionais para permitir a execução deste processo.

1997).

Um dos trabalhos introdutórios sobre a verificação de modelos com proposições da lógica temporal CTL é o de Clarke; Emerson (1981), onde mostrou-se que a verificação de um modelo de sistema concorrente de estados finitos pode ser realizada de forma automatizada a partir da investigação deste modelo com proposições da lógica temporal CTL. Depois deste trabalho, diversos outros foram sendo propostos, onde diferentes modelos de sistemas de estados finitos desenvolvidos foram submetidos à verificação com proposições da lógica temporal CTL (CLARKE; LONG; MCMILLAN, 1991; MOON, 1994; CORTÉS; ELES; PENG, 2000; BONFE; FANTUZZI, 2003; DAS; SARKAR; CHATTOPADHYAY, 2004; ZHAO; KROGH, 2006), sendo que nestes trabalhos também demonstra-se o aprimoramento das ferramentas computacionais de verificação de modelos. Aliado a isso, destaca-se o fato de que nos trabalhos de Hong et al. (2001); Hong et al. (2002); Aiguier et al. (2005) demonstra-se que proposições da lógica temporal CTL podem ser aplicadas de uma forma efetiva para a verificação de modelos desenvolvidos em MEFEs. Nos trabalhos de Blom et. al (2004); Godskesen; Nielsen e Skou (2004); Guerra (2004); Robinson-Mallett et al. (2005); Robinson-Mallett et al. (2006), observa-se a utilização da versão simplificada da lógica temporal CTL para a verificação de MEFEs utilizando-se a ferramenta de modelagem e verificação UPPAAL.

Nas subseções a seguir, serão apresentados os elementos que constituem as proposições da versão simplificada da lógica temporal CTL, ou seja, as expressões proposicionais e os operadores de lógica temporal.

4.2.2 Sintaxe das expressões proposicionais

As expressões proposicionais que constituem as proposições da versão simplificada da lógica temporal CTL são expressões compostas por variáveis de dados e/ou por estados do modelo a ser verificado, sendo tais expressões utilizadas para determinar se as mesmas podem ser satisfeitas por um ou mais estados simbólicos pertencentes à estrutura combinatória de estados derivada do modelo verificado. Estabelece-se a composição das expressões proposicionais da seguinte forma: nome_do_modelo.estado_do_modelo ou nome_do_modelo.variável_do_modelo==*x*.

Onde:

nome_do_modelo é o nome do modelo (MEFE) de interesse.

estado_do_modelo é o nome de um estado pertencente ao modelo (MEFE) de interesse.

variável_do_modelo é uma variável de dados de formato inteiro ou booleano pertencente ao modelo (MEFE) de interesse.

x refere-se a um determinado valor que está contido no conjunto dos possíveis valores que podem ser associados à variável_do_modelo.

Alguns exemplos destas proposições serão apresentados depois da abordagem sobre os operadores de lógica temporal que ocorrerá na próxima subseção.

Na literatura acadêmica, encontram-se diferentes nomenclaturas que têm sido adotadas para a representação simbólica de expressões proposicionais da lógica temporal CTL. A utilização de símbolos para representar tais expressões proposicionais tem o objetivo de simplificar as diferentes situações nas quais tais expressões proposicionais possam estar relacionadas e, assim, exemplos da representação simbólica de expressões proposicionais são: f_1 e f_2, f e g, φ e ψ e p e q (BROWNE et al., 1986; LINDAHL; PETTERSSON; YI, 1998; HONG et al., 2002). No presente trabalho, conforme poderá ser observado nas seções seguintes, são adotados os símbolos p e q.

4.2.3 Operadores de lógica temporal

Os operadores de lógica temporal formam, em conjunto com as expressões proposicionais, as proposições da versão simplificada da lógica temporal CTL. Tais operadores de lógica temporal podem ser classificados como: operadores lógicos e operadores temporais. Os operadores lógicos têm o objetivo de atribuir às expressões proposicionais determinadas características lógicas (BEHRMANN; DAVID; LARSEN, 2004), as quais podem ser de negação, disjunção, conjunção, implicação e equivalência, sendo tais operadores (acompanhados das representações simbólicas das expressões proposicionais) apresentados na Tabela 4.1.

Tabela 4.1 – Operadores lógicos

Operador lógico	**Interpretação**
$\neg p$	não p (negação)
$p \vee q$	p ou q (disjunção)
$p \wedge q$	p e q (conjunção)
$p \rightarrow q$	p implica q / se p então q (implicação)
$p \leftrightarrow q$	p se e somente se q (equivalência)

Por sua vez, os operadores temporais têm o objetivo de indicar à ferramenta computacional de verificação a forma pela qual será realizada a busca por estados simbólicos (na estrutura combinatória derivada do modelo sob verificação) que satisfaçam a expressão proposicional estipulada, sendo estes operadores temporais classificados, de acordo com Moon (1994), como: quantificadores de caminho, cuja descrição é apresentada na Tabela 4.2 e quantificadores de estado, cuja descrição é apresentada na Tabela 4.3.

Tabela 4.2 – Operadores temporais: quantificadores de caminho

Quantificador	**Representação Matemática**	**Interpretação**
A p	$\forall$	Todos (**A**ll): iniciando-se a partir de um estado simbólico atual p ocorre em todos os caminhos.
E p	$\exists$	Existe (**E**xists): iniciando-se a partir de um estado simbólico atual existe um caminho onde p ocorre.

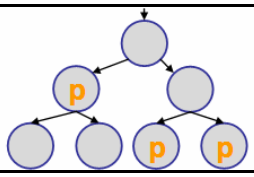
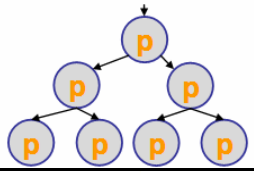
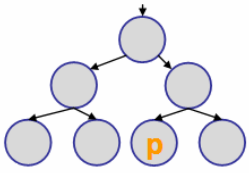
A Tabela 4.3 apresenta os quatro tipos de quantificadores de estado que são possíveis nas proposições da lógica temporal CTL. Dos quatro tipos de quantificadores de estado apresentados nesta tabela, somente dois destes podem ser utilizados na versão simplificada da lógica temporal CTL: Gp e Fp.

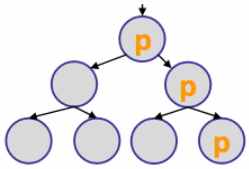
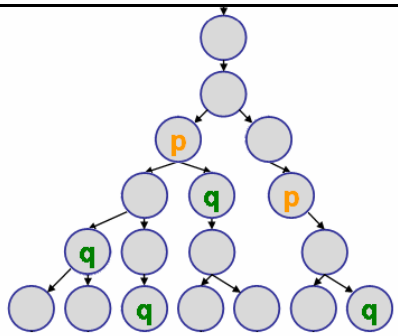
Tabela 4.3 – Operadores temporais: quantificadores de estado

Quantificador	Representação Matemática	Interpretação
X *p*	○ *p*	Próximo (ne**X**t): *p* deve ocorrer no próximo estado, a partir do estado atual.
G *p*	□ *p*	Globalmente (**G**lobally): *p* tem que ocorrer sempre.
F *p*	◊ *p*	Finalmente (**F**inally): *p*, eventualmente, tem que ocorrer.
p U *q*		Até que (**U**ntil): *p* tem que ocorrer até que em algum estado *q* ocorra.

Com a apresentação da Tabela 4.2 e da Tabela 4.3, é possível apresentar-se a Tabela 4.4, onde são vistas as combinações entre os operadores temporais (quantificadores de caminho e quantificadores de estado) e expressões proposicionais que são possíveis de serem estabelecidos na versão simplificada da lógica temporal CTL. Na Tabela 4.4, as proposições são acompanhadas por uma representação gráfica que ilustra a ocorrência das expressões proposicionais *p* e *q* em uma estrutura combinatória de estados.

Tabela 4.4 – Proposições na versão simplificada da lógica temporal CTL: exemplos da ocorrência de p e q em uma estrutura combinatória de estados simbólicos

Proposição	Exemplo da ocorrência das expressões proposicionais *p* e *q*
AF p	
AG p	
EF p	

Proposição	Exemplo da ocorrência das expressões proposicionais *p* e *q*
EG p	
AG (*p* imply *AF q*)	

Apesar das proposições apresentadas na Tabela 4.4 estarem relacionadas à versão simplificada da lógica temporal CTL utilizada no presente trabalho, tais proposições apresentadas, conforme destacam os trabalhos de Browne et al. (1986); Clarke et al. (2000); Zoubek (2004), são as mais freqüentemente utilizadas em lógica temporal CTL para a verificação de modelos, sendo que o significado de tais proposições é apresentado a seguir:

- *AF p* ≡ ¬*EG* ¬*p*: significa que para cada possível caminho de uma determinada estrutura combinatória de estados simbólicos existe, pelo menos, um estado simbólico neste caminho onde *p* ocorre;
- *EF p* ≡ ¬*AG* ¬*p*: significa que para algum caminho de uma determinada estrutura combinatória de estados simbólicos existe, pelo menos, um estado simbólico neste caminho no qual *p* ocorre;
- *AG p* ≡ ¬*EF* ¬*p*: significa que para cada possível caminho de uma determinada estrutura combinatória de estados simbólicos, em cada estado simbólico deste caminho, *p* ocorre;
- *EG p* ≡ ¬*AF* ¬*p*: significa que para algum caminho de uma determinada estrutura combinatória de estados simbólicos, a cada estado simbólico deste caminho *p* sempre ocorre;
- *AG* (*p* → *AF q*): significa que se *p* ocorrer tal fato implica que para cada

possível caminho de uma estrutura combinatória de estados simbólicos a partir de *p* existe, pelo menos, um estado simbólico neste caminho onde *q* ocorre;

Apresenta-se a Tabela 4.5, a qual ilustra as diferentes formas de representação das proposições que são possíveis na versão simplificada da lógica temporal CTL, sendo apresentada, inclusive, a representação de tais proposições conforme a sintaxe suportada pela ferramenta UPPAAL (BEHRMANN; DAVID; LARSEN, 2004).

Tabela 4.5 – Proposições da versão simplificada da lógica temporal CTL e suas diferentes formas de representação

Representação da proposição CTL	**Representação matemática**	**Representação na ferramenta UPPAAL**
$EF\ p$	$\exists \Diamond\ p$	E<> p
$AG\ p$	$\forall \Box\ p$	A[] p
$EG\ p$	$\exists \Box\ p$	E[] p
$AF\ p$	$\forall \Diamond\ p$	A<> p
$AG\ (p \rightarrow AF\ q)$	$p \rightarrow q$	$p \rightarrow q$

4.2.4 Exemplos de proposições

Na ferramenta UPPAAL, as expressões proposicionais da versão simplificada da lógica temporal CTL, que são compostas somente por variáveis de dados, estão associadas à possibilidade de uma variável ou um conjunto de variáveis pertencentes ao modelo sob verificação alcançar determinados valores estipulados (nome_do_modelo.variável_do_modelo==*x*), enquanto que as expressões proposicionais que são compostas por estados estão associadas à possibilidade de um estado ou um conjunto de estados pertencentes ao modelo sob verificação serem alcançados (nome_do_modelo.estado_do_modelo). Para exemplificar esta abordagem, apresentam-se na Tabela 4.6 dois exemplos de proposições da versão simplificada da lógica temporal CTL, os quais podem ser estipulados utilizando-se a ferramenta UPPAAL.

Tabela 4.6 – Exemplos de proposições

Proposição	Interpretação
E<> p1.cs and p2.cs and p1.x==2	Existe, pelo menos, um estado simbólico onde o estado ativo da MEFE p1 seja cs e o estado ativo da MEFE p2 seja cs e o valor da variável de dados x pertencente ao modelo p1 seja igual ao valor "2"
A[] p1.cs imply not p2.cs	Sempre que o estado cs da MEFE p1 for alcançado implica que o estado cs da MEFE p2 não será alcançado

4.3 A FERRAMENTA DE MODELAGEM E VERIFICAÇÃO UPPAAL

O nome UPPAAL surgiu da junção das três letras iniciais dos nomes Uppsala e Aalborg que são os nomes das universidades as quais pertencem os pesquisadores que tem desenvolvido a ferramenta. Trata-se de uma ferramenta computacional integrada para modelagem, simulação e verificação de modelos de sistemas concorrentes de estados finitos, desenvolvida em conjunto pelo Basic Research in Computer Science (Universidade de Aalborg, na Dinamarca) e pelo Department of Information Technology (Universidade de Uppsala, na Suécia).

Conforme Bengtsson; Yi (2004), esta ferramenta é a apropriada para sistemas que possam ser modelados através de modelos de estados finitos e que utilizem variáveis de relógio[15] (ALUR; DILL, 1994) e/ou variáveis de dados, sendo possível que os modelos desenvolvidos comuniquem-se entre si através de canais de comunicação (YI; PETTERSSON; DANIELS, 1994; LARSEN; PETTERSSON; YI, 1997; HESSEL; PETTERSSON, 2004). A ferramenta é composta por três ambientes:

- desenvolvimento;
- simulação;
- verificação.

O ambiente de desenvolvimento permite a construção de modelos de forma gráfica, sendo que a programação em linguagem textual também pode ser utilizada para algumas funções auxiliares dos modelos (GEBREMICHAEL, VAANDRAGER, ZHANG, 2006). Neste ambiente de desenvolvimento é possível serem vistos alertas que são gerados pela ferramenta quando ocorrem erros na construção dos modelos, erros estes que podem ser, entre outros, de sintaxe, referência, declaração etc. O ambiente de simulação torna-se fortemente importante quando se deseja acompanhar a evolução passo a passo do modelo, o que também torna possível a utilização deste ambiente para se realizar a procura de erros em modelos, fato este que foi discutido no Capítulo 1 e que será visto aplicado em estudo de casos no Capítulo 6. Neste ambiente é possível, por exemplo, depois de finalizada a verificação do modelo, observar um histórico de estados simbólicos gerado pela ferramenta que permite observar a forma pela qual foi alcançado o estado simbólico que determinou que a verificação do modelo fosse (ou não) satisfeita. Finalmente, o ambiente de verificação permite que proposições da versão simplificada da lógica temporal CTL possam ser especificadas para que a ferramenta as utilize na verificação do modelo que foi desenvolvido no ambiente de desenvolvimento.

4.3.1 Justificativas para a utilização da ferramenta UPPAAL

Em virtude dos diferentes tipos de modelos de sistemas existentes, encontram-se diversas ferramentas que são aplicadas à verificação destes modelos. Em geral, os critérios apresentados a seguir são os que têm sido considerados para a consolidação da seleção de uma ferramenta computacional de modelagem e verificação:

---

[15] Ver o Anexo C para mais informações sobre as variáveis de relógio.

- a compatibilidade entre a ferramenta de verificação e o modelo a ser verificado;
- a facilidade para o usuário desenvolver o modelo do sistema de interesse e verificá-lo;
- o tempo de processamento durante a execução da verificação do modelo;
- suporte técnico dado aos usuários da ferramenta, o qual deveria incluir, pelo menos: publicações técnicas, softwares adicionais para correções de eventuais erros de programação da ferramenta (sempre que estes forem detectados), aprimoramento contínuo do ambiente de interação com os usuários e dos algoritmos de verificação, manuais, web-site, fóruns de discussão (na Internet), compatibilidade com diferentes sistemas operacionais etc.

Durante a fase de pesquisa deste trabalho sobre qual ferramenta computacional seria utilizada para a modelagem e verificação dos modelos em MEFEs gerados a partir de programas de controle desenvolvidos em LD, constatou-se que a ferramenta UPPAAL poderia ser utilizada, tendo sido escolhida por apresentar, entre outras, as características descritas a seguir:

- sua linguagem para criação de modelos é gráfica, o que pode constribuir para um tempo menor de desenvolvimento dos modelos (LARSEN; PETTERSSON; YI, 1997; BURNS, 2003). O fato de se criar os modelos graficamente também poderá contribuir para a redução de eventuais erros de digitação, os quais podem ocorrer ao se utilizar ferramentas de modelagem e verificação que possuem linguagens textuais de programação como, por exemplo, as ferramentas SPIN (GIANNAKOPOULOU; PASAREANU, 2006) e SMV (MCMILLAN, 1999);
- permite a utilização de variáveis de dados do tipo inteiro ou booleano para a criação de condições de restrição e expressões de atualização do valor de variáveis sobre uma transição entre os estados de uma MEFE (LARSSON, 2000);
- o sistema modelado é armazenado num arquivo de padrão "*eXtendeded Markup Language*" (XML), o que torna possível a reutilização do sistema modelado por outros aplicativos que possam interpretar este padrão

(BEHRMANN; DAVID; LARSEN, 2004);

- o ambiente de simulação possibilita a observação e a interação passo a passo em relação à evolução do modelo, modelo este que poderá ser verificado no ambiente de verificação. Desta forma, torna-se possível a realização de testes manuais, os quais também podem auxiliar na tarefa de identificar erros nos modelos, conforme a discussão realizada no Capítulo 1;
- é gerado um histórico que apresenta a forma pela qual o algoritmo de verificação de modelos da ferramenta alcançou um determinado estado simbólico que resultou na violação da proposição de lógica temporal estipulada;
- possui um web-site com uma grande quantidade de material de apoio à pesquisa e um grupo de discussão composto por pesquisadores de diversas universidades do mundo;
- tem sido continuamente aprimorada por pesquisadores, os quais têm demonstrado forte comprometimento para a correção de eventuais erros de programação que possam surgir na ferramenta, além de também estarem comprometidos com o aumento da velocidade de processamento da mesma durante as verificações realizadas;
- é obtida gratuitamente na Internet para ser utilizada para fins acadêmicos.

4.4 CONSIDERAÇÕES SOBRE AS PROPRIEDADES DE ALCANÇABILIDADE E DE SEGURANÇA

Em Villani (2004) são abordados dois tipos de questões referentes à verificação de um sistema supervisório híbrido, as quais também se aplicam aos modelos de sistemas abordados no presente trabalho: uma referente à alcançabilidade (propriedade de alcançabilidade), a qual visa provar que um determinado estado sempre será alcançado, e a outra questão referente à segurança (propriedade de segurança), a qual visa provar que um determinado estado jamais será alcançado.

Nesta direção, em relação à propriedade de alcançabilidade, é investigado se dada uma determinada expressão proposicional p a mesma pode ser satisfeita por pelo menos um estado simbólico da estrutura combinatória de estados derivada do modelo verificado.

Complementando-se esta abordagem realizada por Villani (2004), outra possível forma de se apresentar a questão da alcançabilidade, conforme Bengtsson; Yi (2004), é através da investigação se existe algum caminho na estrutura combinatória de estados, partindo-se do estado simbólico inicial desta estrutura, no qual a expressão proposicional *p* pode ser satisfeita por algum estado simbólico ao longo deste caminho. A verificação da propriedade de alcançabilidade utiliza-se de uma proposição de lógica temporal dada por *EF p*, onde *EF* é a abreviação de *Exist Finally*.

O processo de verificação de modelos, em relação à questão da alcançabilidade, poderá ser composto por dois passos:

1. a criação de uma estrutura combinatória de estados derivada do modelo verificado;

2. a busca e identificação por, pelo menos, um estado simbólico nesta estrutura combinatória de estados que satisfaça a proposição (de alcançabilidade) estipulada.

O primeiro passo é, normalmente, realizado antes do segundo, porém, ambos podem ser realizados de forma simultânea, através da técnica conhecida como verificação *on-the-fly* (VARDI; WOLPER, 1986), a qual é utilizada pela ferramenta UPPAAL (PETTERSSON, 1999). Uma vantagem desta técnica pode estar na situação onde uma violação (da expressão proposicional) é encontrada antes mesmo de ter sido gerada toda a estrutura combinatória de estados, gerando-se ganhos como, por exemplo, um tempo de verificação de modelos comparativamente menor. A técnica *on-the-fly* permite que o processo de verificação possa ser interrompido tão logo a expressão proposicional estipulada seja satisfeita. Entretanto, caso a proposição estipulada estabeleça que todos os estados simbólicos da estrutura combinatória de estados devam ser investigados, como ocorre, por exemplo, com a proposição de lógica temporal composta pelos operadores temporais *AG*, será necessária a construção de toda a estrutura combinatória de estados, mesmo utilizando-se a técnica *on-the-fly*.

Quanto à propriedade de segurança, a mesma investiga se dada uma expressão proposicional *p*, tal expressão sempre ocorrerá em todos os estados simbólicos da estrutura combinatória de estados derivada do modelo verificado, sendo a tarefa da ferramenta

computacional de verificação identificar aquele(s) estado(s) simbólico(s) que leverá(ão) à violação da expressão proposicional em questão. Neste sentido, de acordo com Matoušek (2005), um sistema pode ser considerado seguro se todos os seus estados alcançáveis satisfizerem a uma determinada expressão proposicional *p* estipulada.

A verificação da propriedade de segurança utiliza-se de uma proposição de lógica temporal dada por *AG p*, onde os operadores temporais *AG* (abreviação de *All Globally*) investigam se todos os estados simbólicos da estrutura combinatória sob análise sempre satisfarão a expressão proposicional *p*.

Ainda em relação ao trabalho de Villani (2004), encontra-se uma afirmação onde se diz que verificar a propriedade de segurança de um determinado modelo está relacionado ao fato de se garantir que um determinado estado deste modelo jamais seja alcançado. Tal afirmação é encontrada no presente trabalho sob a forma de uma proposição de lógica temporal que é composta pelos operadores temporais *AG* e pela negação da expressão proposicional *p* considerada, ou seja, *AG* ¬*p*. Isto significa dizer que a expressão proposicional *p* jamais deverá ser satisfeita durante a investigação de todos os estados simbólicos que compõem uma determinada estrutura combinatória de estados derivada do modelo verificado, sendo que um exemplo de tal proposição poderá ser visto no segundo estudo de caso apresentado no Capítulo 6.

CAPÍTULO 5

5. PROCEDIMENTO PARA A MODELAGEM DE MEFES A PARTIR DE PROGRAMAS DE CONTROLE DESENVOLVIDOS EM LD

Neste capítulo será apresentado o procedimento para a modelagem de MEFEs a partir de programas de controle desenvolvidos em LD e executados por um CP. Itens que foram discutidos no Capítulo 2 como, por exemplo, o ciclo operacional de um CP, o tempo de varredura e os elementos do programa de controle, e no Capítulo 3 como, por exemplo, as MEFEs, os canais de comunicação e os estados compromissados, serão abordados neste capítulo, considerando-se a representação de tais itens nos modelos que serão apresentados.

5.1 CONSIDERAÇÕES INICIAIS

O procedimento proposto tem por objetivo descrever um procedimento para se realizar a modelagem de MEFEs a partir de programas de controle desenvolvidos em LD e executados por um CP, sendo que, para a realização deste procedimento, foi utilizada a ferramenta computacional UPPAAL. Utilizando-se tal ferramenta, testes foram conduzidos pelo presente trabalho para a verificação dos modelos desenvolvidos, e constatou-se que para a verificação de modelos de maior complexidade contendo diversos estados e canais de comunicação externos, a ferramenta UPPAAL não finalizava o processo de verificação,

gerando-se um erro relacionado à quantidade de memória insuficiente do computador utilizado para se executar tal processo. Utilizando-se, então, o recurso dos estados compromissados que está disponível na ferramenta, a verificação destes modelos mais complexos pôde, então, ser finalizada. Desta forma, os modelos que serão apresentados neste capítulo estarão disponíveis no Anexo B, devidamente documentados e adequados para a aplicação na ferramenta UPPAAL, isto é, utilizando-se estados compromissados. É importante observar que os modelos que serão apresentados nos dois casos estudados no Capítulo 6 já estarão contemplando a utilização de estados compromissados.

Antes de ser apresentado o procedimento de modelagem em questão, é importante destacar que tal procedimento considera a simbologia/codificação utilizada em CPs da família SLC 500, para endereços e elementos do programa de controle, conforme abordagem previamente realizada no Capítulo 2, de forma que a simbologia/codificação utilizada em outros CPs não constitui uma restrição do presente trabalho.

Para ilustrar a seqüência de execução do procedimento de modelagem proposto foi utilizada a técnica PFS (Production Flow Schema) (MIYAGI, 1996), a qual pode ser utilizada para descrever, graficamente e conceitualmente, processos relacionados com a produção de diversos itens (peças, produtos, informações etc.) sob a forma de seqüências de etapas de atividades e de distribuição. O PFS consiste, basicamente, de nós de *elementos-atividade*, nós de *elementos-distribuidor* e *arcos de fluxo*, sendo que os *arcos de fluxo* conectam, seqüencialmente, um tipo de nó ao outro. Na Figura 5.1 estão sendo representados os itens em questão.

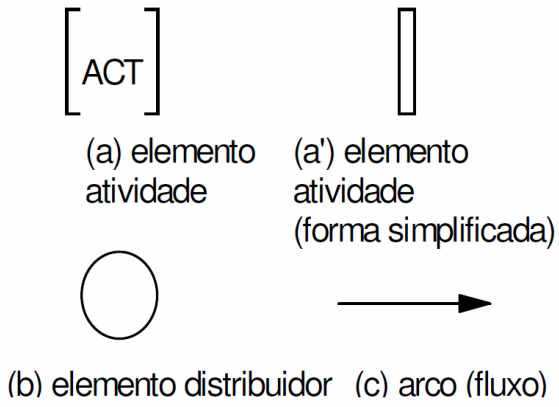


Figura 5.1 – Itens que compõem o PFS

A seguir, são descritos tais itens:

- *Elemento-atividade*: representa um componente ativo do sistema que é capaz de realizar ações, de forma que tais elementos podem agregar uma ou mais operações. Corresponde a um macro-elemento que, graficamente, é delimitado por colchetes e que possui inscrições em linguagem natural ou formal para especificá-lo;
- *Elemento-distribuidor*: representa um componente passivo do sistema capaz de armazenar, permanecer em certos estados e tornar visíveis os itens. Um elemento-distribuidor deve existir sempre entre dois elementos-atividade que estejam relacionados e, graficamente, tal elemento é representado por um círculo;
- *Arcos de fluxo*: conectam *elementos-atividade* a *elementos-distribuidor*, e vice-versa, representando o fluxo de informações ou de itens. Graficamente, tais itens são formados por arcos orientados. Os arcos conectados na parte externa do *elemento-atividade* indicam o fluxo principal, e os arcos conectados na parte interna do *elemento-atividade* indicam o fluxo secundário.

Ressalta-se que elementos de um mesmo tipo não podem ser conectados diretamente uns aos outros, e que cada um dos componentes recebe inscrições em linguagem natural, indicando sua interpretação específica para o modelo elaborado.

A partir destas considerações, na Figura 5.2 apresenta-se o procedimento proposto pelo presente trabalho, estruturado na forma de um modelo desenvolvido utilizando-se a técnica PFS.

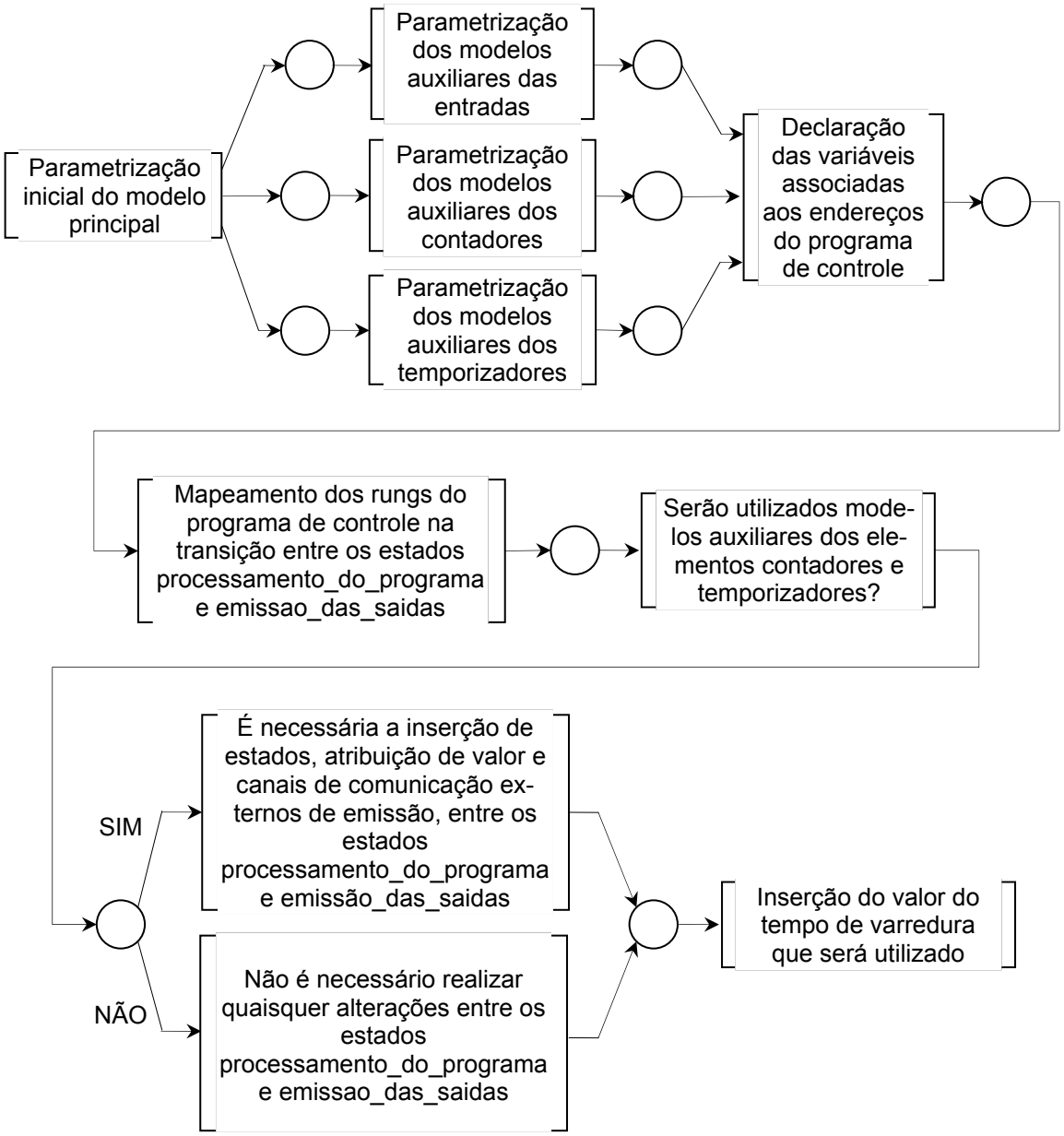


Figura 5.2 – Modelo PFS que representa o procedimento de modelagem proposto

Na próxima seção (5.2), cada um dos *elementos-atividade* apresentados na Figura 5.2 serão discutidos detalhadamente, explicitando-se, desta maneira, o procedimento em questão.

5.2 PASSOS DO PROCEDIMENTO PARA A MODELAGEM DE MEFES A PARTIR DE PROGRAMAS DE CONTROLE DESENVOLVIDOS EM LD

5.2.1 Parametrização inicial do modelo principal

Para as questões de modelagem que serão apresentadas a seguir, de uma maneira introdutória, apresenta-se a Figura 5.3, onde é ilustrado, como um exemplo inicial, um *rung* de um determinado programa de controle desenvolvido em LD.

Figura 5.3 – Exemplo de um rung de um determinado programa de controle em LD

No exemplo ilustrado na Figura 5.3, ao elemento bobina é associado o endereço de saída O:2.0/0 e, para que se tenha um nível lógico verdadeiro nesta bobina, é necessário que os dois elementos contatos do tipo NA conectados em série e associados aos endereços de entrada I:1.0/0 e I:1.0/1 também tenham nível lógico verdadeiro. Se a representação deste *rung* fosse realizada através de uma expressão booleana, uma possível representação seria: O:2.0/0 = I:1.0/0 & I:1.0/1. Na formatação da ferramenta de modelagem e verificação UPPAAL, este tipo de expressão pode ser escrita da seguinte forma: O2_0_0 = I1_0_0 && I1_0_1 (O2_0_0, I1_0_0 e I1_0_1 $\in V_L$). Este *rung* em questão (Figura 5.3) pode ser mapeado, utilizando-se a expressão booleana apresentada, em uma MEFE chamada modelo principal, de forma que um exemplo de modelo principal onde foi realizado este mapeamento é ilustrado na Figura 5.4. É possível notar que a expressão booleana em questão aparece mapeada na transição entre os estados processamento_do_programa e emissao_das_saidas do modelo principal ilustrado na Figura 5.4.

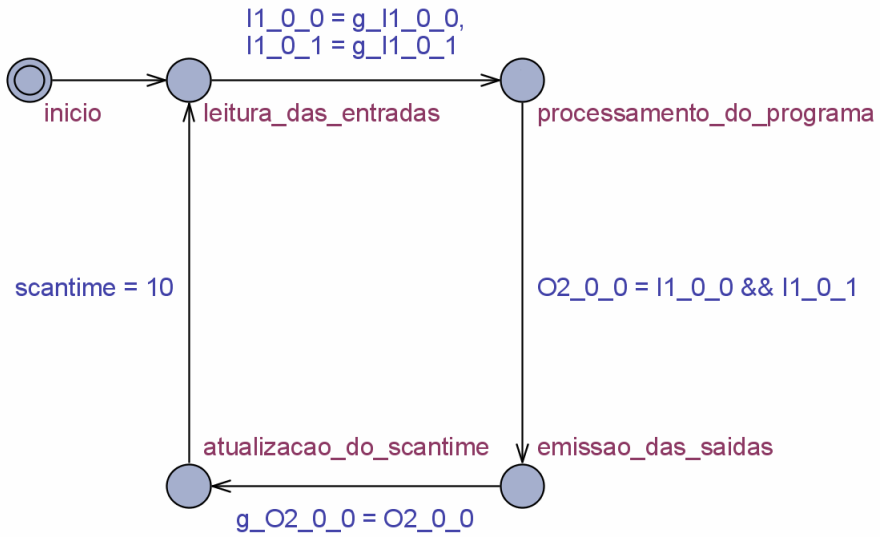


Figura 5.4 – Exemplo de um modelo principal

O modelo principal diferencia-se dos outros modelos que serão apresentados adiante em virtude de realizar, por exemplo, a modelagem do ciclo operacional do CP e de atualizar o valor do tempo de varredura deste ciclo operacional. É constituído, em sua forma elementar, por cinco estados, os quais são ilustrados na Figura 5.4. São estes estados:

- inicio;
- leitura_das_entradas;
- processamento_do_programa;
- emissao_das_saidas;
- atualizacao_do_scantime.

Conforme descrito, a modelagem do ciclo operacional do CP está contemplada no modelo principal: entre os estados leitura_das_entradas e processamento_do_programa é modelada a etapa de atualização do valor dos endereços de entrada do CP; entre os estados processamento_do_programa e emissao_das_saidas modela-se a etapa de processamento do programa de controle; finalmente, entre os estados emissao_das_saidas e atualizacao_do_scantime, modela-se a etapa de atualização do valor dos endereços de saída do CP.

O estado denominado inicio é o estado inicial do modelo principal. Este estado é executado somente uma vez em virtude do início do processo de verificação (ou simulação) do modelo global, modelo este que é constituído pelo modelo principal e seus modelos

auxiliares. Ainda de uma forma introdutória, neste trabalho considera-se que os modelos auxiliares são todos os modelos associados ao modelo principal, de forma que ambos são necessários para a realização do procedimento de modelagem dos programas de controle considerados.

A partir do estado inicio, o próximo estado a ser ativado no modelo principal é o estado leitura_das_entradas. Pelo fato de este estado ser ativado antes da primeira transição onde há a execução de expressões de atribuição de valores às variáveis locais do modelo principal, é neste estado que o algoritmo de verificação da ferramenta UPPAAL realiza um processo de geração de diferentes cenários que contemplarão todas as possíveis combinações de valores que podem ser associados aos endereços de entrada pertencentes ao programa de controle modelado, de forma que possa ser construída uma estrutura combinatória de estados resultante da verificação do modelo global[16].

O estado leitura_das_entradas permanecerá ativado enquanto o algoritmo de verificação da ferramenta UPPAAL estiver gerando as combinações dos valores em questão. Depois de finalizado este processo de geração dos diferentes cenários, o algoritmo de verificação habilitará a execução da transição[17] entre os estados leitura_das_entradas e processamento_do_programa e, ao se executar a transição em questão, os valores das variáveis globais de entrada serão copiados para as variáveis locais, utilizadas para o processamento do programa de controle mapeado no modelo principal.

No exemplo da Figura 5.4, as duas variáveis g_I1_0_0 e g_I1_0_1 (g_I1_0_0 e g_I1_0_1 $\in V_I$) que modelam, respectivamente, os dois endereços de entrada I:1.0/0 e I:1.0/1, têm

[16] Mais detalhes serão apresentados na abordagem dos modelos auxiliares das entradas, os quais auxiliam o algoritmo de verificação da ferramenta UPPAAL neste processo de geração de cenários que contemplem todas as possíveis combinações de valores que podem ser associados aos endereços de entrada.

[17] É importante destacar que no modelo principal não existem quaisquer transições com expressões de restrição. Neste sentido, considera-se que a única "restrição" existente entre os estados "leitura_das_entradas" e "processamento_do_programa" é a que é imposta pelo algoritmo de verificação da ferramenta UPPAAL durante o processo de geração dos valores dos endereços de entrada.

seus valores modificados pelo algoritmo de verificação da ferramenta UPPAAL[18]. Quando este algoritmo de verificação determinar o valor para cada uma destas variáveis, será possível, então, a execução da transição entre os estados leitura_das_entradas e processamento_do_programa, e os valores das variáveis g_I1_0_0 e g_I1_0_1 serão transferidos para as variáveis I1_0_0 e I1_0_1 (I1_0_0 e I1_0_1 $\in V_L$), as quais não são modificadas pelo algoritmo de verificação da ferramenta UPPAAL.

Neste contexto, o presente trabalho estabelece como premissa para a questão do procedimento de modelagem proposto, que o programa de controle a ser mapeado no modelo principal e, posteriormente verificado, seja mapeado entre os estados processamento_do_programa e emissao_das_saidas.

Quando é executada a transição entre os estados processamento_do_programa e emissao_das_saidas, a única forma pela qual as transições dos modelos auxiliares dos elementos contadores e/ou temporizadores podem ser executadas é quando tais transições forem habilitadas (sincronizadas) pelo modelo principal. Tal ativação é obtida, no presente trabalho, com a utilização de canais de comunicação externos (e interno), sendo que, os de emissão devem ser contemplados no modelo principal, enquanto que os de recepção, devem estar nos modelos auxiliares dos elementos contadores e/ou temporizadores.

Observa-se no *rung* apresentado na Figura 5.3 que não há elementos contadores e/ou temporizadores sendo ativados. Neste caso, conforme este procedimento apresentará com mais detalhes, pode-se dizer que a não utilização de elementos contadores e/ou temporizadores no programa de controle modelado implica na não utilização de canais de comunicação externos para se realizar a comunicação do modelo principal com os modelos auxiliares destes elementos. Como conseqüência deste fato, o modelo principal resultante

[18] Na Figura 5.5 serão apresentados os dois modelos auxiliares destas entradas consideradas, o que permite ao algoritmo de verificação da ferramenta UPPAAL realizar a seleção dos valores que considera necessários para a realização do processo de verificação.

do procedimento de modelagem será constituído em sua forma elementar, ou seja, por apenas cinco estados, conforme é ilustrado na Figura 5.4, independentemente da quantidade de *rungs* existentes no programa de controle a ser modelado.

Ainda em relação ao modelo principal, o estado emissao_das_saidas é o estado a partir do qual modela-se a execução da etapa de atualização do valor dos endereços de saída do CP. É possível observar no exemplo ilustrado na Figura 5.4 que a variável O2_0_0 (O2_0_0 $\in$ V_L), a qual modela o endereço de saída O:2.0/0 (ver *rung* da Figura 5.3), é utilizada para o processamento do programa de controle mapeado na transição do modelo principal. Depois da execução desta expressão mapeada, o resultado armazenado em O2_0_0 é transferido para a variável g_O2_0_0 (g_O2_0_0 $\in$ V_O), fato este que remete à modelagem da etapa de atualização dos valores dos endereços de saída do CP. Esta etapa da modelagem é importante para o estabelecimento de proposições da versão simplificada da lógica temporal CTL, conforme será observado no primeiro estudo de caso do Capítulo 6, pois considera o resultado final do processamento do programa de controle que é atualizado nos de saída do CP.

Finalmente, depois da ativação do estado emissao_das_saidas, o próximo estado do modelo principal a ser ativado é o estado atualizacao_do_scantime. Conforme, ainda, o exemplo da Figura 5.4, é possível observar que a atualização do valor do tempo de varredura (variável scantime) é realizada somente ao término da execução deste estado.

5.2.2 Parametrização dos modelos auxiliares das entradas

Na subseção anterior, a qual apresentou o passo de parametrização inicial do modelo principal, foi apresentado um exemplo de mapeamento de um *rung* de um determinado programa de controle a ser modelado, o que se realizou com o estabelecimento de uma expressão booleana relacionada ao *rung* em questão, e que foi escrita na transição entre os estados processamento_do_programa e emissao_das_saidas.

Entretanto, o comportamento combinatório das entradas e o funcionamento de elementos contadores e temporizadores não podem ser mapeados entre os estados em questão, simplesmente, como expressões booleanas. Por este motivo, foram criados modelos auxiliares do modelo principal, os quais promovem a modelagem do comportamento

combinatório das entradas e do funcionamento dos elementos contadores e temporizadores encontrados no programa de controle que será modelado. Assim, iniciando-se a discussão sobre os modelos auxiliares, nesta subseção serão abordados os modelos auxiliares dos endereços de entrada ou, simplesmente, modelos auxiliares das entradas.

Os modelos auxiliares das entradas são necessários porque permitem que o algoritmo de verificação da ferramenta UPPAAL realize um processo de geração de cenários que contemplarão todas as possíveis combinações de valores que podem ser associados aos endereços de entrada pertencentes ao programa de controle modelado, de forma que possa ser construída uma estrutura combinatória de estados resultante da verificação do modelo global[19].

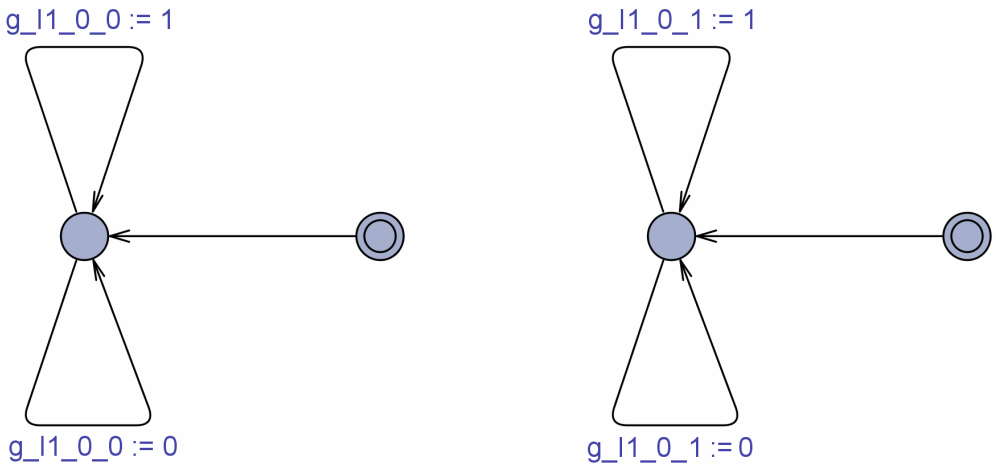
Conforme abordagem iniciada no Capítulo 1, os valores lógicos das entradas de um CP, em um determinado instante, podem ser diferentes dos esperados, em decorrência, por exemplo, de acionamentos indevidos ou de anomalias associadas aos dispositivos de detecção e de comando ou, ainda, anomalias dos próprios dispositivos de entrada do CP (PARR, 2003). Portanto, para garantir-se que todas as possíveis combinações dos valores que podem ser associados aos endereços de entrada sejam testadas, utiliza-se modelos auxiliares das entradas.

Dois exemplos de modelos auxiliares das entradas[20] são apresentados na Figura 5.5, os quais referem-se às entradas utilizadas no *rung* da Figura 5.3, e cujos endereços são I:1.0/0 (Figura 5.5 (a)) e I:1.0/1 (Figura 5.5 (b)). É importante observar que estas variáveis também são contempladas no modelo principal ilustrado na Figura 5.4 e, desta forma, estes modelos auxiliares das entradas e o modelo principal da Figura 5.4 formam um modelo

[19] Conforme Zoubek (2004), as entradas de um programa de controle devem ser modeladas como variáveis que mudam os seus valores de uma forma não determinística, garantindo-se que cada possível estado do modelo será alcançado durante o processo de verificação. Consideração semelhante também foi realizada por Moon (1994).

[20] A concepção do modelo auxiliar das entradas adotada neste trabalho é proveniente de Zoubek (2004).

chamado global, o qual pode ser submetido ao processo de verificação, de forma que este modelo global representa a modelagem completa do *rung* apresentado na Figura 5.3.



(a) Modelo auxiliar da entrada I:1.0/0 (b) Modelo auxiliar da entrada I:1.0/1

Figura 5.5 – Exemplos de modelos auxiliares das entradas

No exemplo da Figura 5.5, os endereços I:1.0/0 e I:1.0/1 são modelados através das variáveis booleanas g_I1_0_0 e g_I1_0_1. Os valores que poderão ser associados a estas variáveis são "0" ou "1", de acordo com a transição que vier a ser executada, o que será determinado pelo algoritmo de verificação da ferramenta UPPAAL.

5.2.3 Parametrização dos modelos auxiliares dos elementos contadores do tipo CTU

Outro tipo de modelo auxiliar que está sendo proposto pelo presente trabalho é o do elemento contador do tipo CTU. A seguir, são descritos os componentes que consitutem o modelo auxiliar do elemento contador, conforme o exemplo da Figura 5.6, componentes estes que, com exceção do canal de comunicação externo, estão associados aos componentes do elemento contador do tipo CTU que foram apresentados no Capítulo 2:

- C5_0_CU: representa a variável que está associada ao endereço do elemento contador que indica a sua ativação (*Count Up Bit*);
- C5_0_DN: representa a variável que está associada ao endereço do elemento contador responsável por indicar o término de seu incremento (*Count Done Bit*);
- C5_0_RES: representa a variável que está associada ao elemento bobina do

tipo *reset* (Tabela 2.4) com o endereço do elemento contador ao qual está relacionada, para que seja executado o *reset* do valor acumulado em *Accum*;

- "4": representa um valor que foi estipulado para o componente *Preset*;
- C5_0_ACC: representa o componente *Accum*;
- CTU_5_0?: é um canal de comunicação externo de recepção do modelo auxiliar do elemento contador do tipo CTU.

Na Figura 5.6 apresenta-se um exemplo do modelo em questão, contemplando-se o endereçamento C5:0[21].

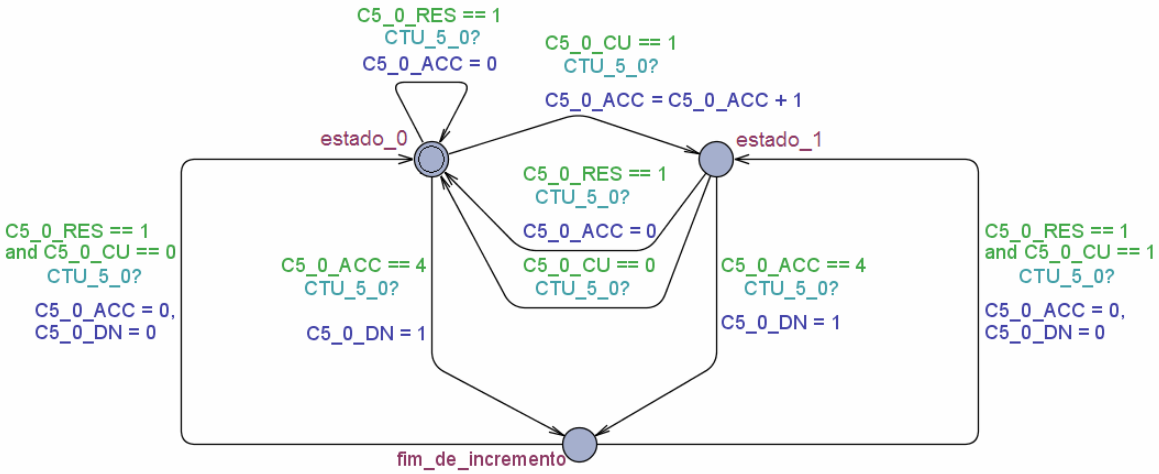


Figura 5.6 – Modelo auxiliar do elemento contador do tipo CTU de endereço C5:0

Para explicar o funcionamento do modelo auxiliar do elemento contador do tipo CTU será apresentado um exemplo: inicialmente, o estado ativo do modelo auxiliar do elemento contador do tipo CTU apresentado na Figura 5.6 é o estado estado_0. Supondo-se, então, que este elemento seja ativado pelo modelo principal, a variável C5_0_CU estará com

[21] Nos CPs da família SLC 500 é possível utilizar até 256 elementos contadores do tipo CTU, ou seja, de C5:0 até C5:255. Para utilizar-se, por exemplo, um modelo do elemento contador de endereço C5:1, utiliza-se o mesmo modelo da Figura 5.6, substituindo-se nas variáveis o nome 5_0 por 5_1, e assim por diante. Tal substituição também deverá ser contemplada no modelo principal, na ocasião da ativação do modelo de tal elemento.

valor “1”, e permitirá que o modelo auxiliar em questão, ao também receber uma habilitação via canal de comunicação externo CTU_5_0?, possa executar a transição entre os estados estado_0 e estado_1. Executando-se tal transição, o valor de C5_0_ACC será incrementado em uma unidade (C5_0_ACC = C5_0_ACC + 1), e passará a ter valor “1”, considerando-se que a variável C5_0_ACC é inicializada com valor “0”. Ao ativar o estado_1, como o valor de C5_0_ACC é menor que “4”, o modelo ficará, então, no estado_1, até que o valor de C5_0_CU seja “0” e que haja uma habilitação via canal de comunicação CTU_5_0?, o que resultará, novamente, do estado_0. Esta seqüência descrita se sucederá até que o valor de C5_0_ACC seja “4” quando, então, o modelo em questão, estando no estado_1 e recebendo uma habilitação via canal CTU_5_0?, irá para o estado fim_de_incremento, o que fará com que o valor de C5_0_DN seja igual a “1”, indicando, desta forma, o término do processo de contagem.

Em qualquer estado deste modelo auxiliar é prevista a situação na qual se o valor de C5_0_RES for igual a “1”, haverá uma evolução entre os estados do modelo, de forma que o valor de C5_0_ACC será “0” e, se for este o caso, o valor de C5_0_DN também será “0”. Isto representa a modelagem da ação de *reset* a que está sujeito o elemento contador do tipo CTU. É importante ressaltar que a variável C5_0_DN tem seu valor modificado para “1” somente quando o modelo auxiliar em questão ativar o estado fim_de_incremento e, devido a este fato, é que foi prevista a condição de *reset* para esta variável somente na execução de uma das duas transições que partem deste estado.

A evolução do modelo objeto dessa discussão sempre se dará com a satisfação das condições de restrição impostas às transições, restrições estas formadas por expressões compostas por variáveis de dados e por um canal de comunicação externo de recepção, de forma que ambos devem ser satisfeitos para que ocorra a execução da transição em questão. A maneira pela qual a ocorre a evolução do modelo é resultante do processo de modelagem do comportamento do elemento contador do tipo CTU, o que foi apresentado no Capítulo 2 (ver um exemplo na Figura 2.5).

5.2.4 Parametrização dos modelos auxiliares dos elementos temporizadores do tipo TON

O próximo modelo auxiliar que será apresentado está relacionado à modelagem do

elemento temporizador do tipo TON. A seguir, são descritos os componentes que consitutem o modelo auxiliar do elemento temporizador, conforme o exemplo da Figura 5.7, componentes estes que, com exceção do canal de comunicação externo, estão associados aos componentes do elemento temporizador do tipo TON que foram apresentados no Capítulo 2:

- T4_0_EN: representa a variável que está associada ao endereço do elemento temporizador que indica a sua habilitação (*Timer Enable Bit*);
- T4_0_DN: representa a variável que está associada ao endereço do elemento temporizador que indica o fim de seu incremento de tempo (*Timer Done Bit*);
- "180000" (180.000ms): representa um valor que foi estipulado para o componente *Preset* apresentado no Capítulo 2;
- T4_0_ACC: representa o componente *Accum* apresentado no Capítulo 2;
- scantime: representa o valor do tempo de varredura atualizado pelo modelo principal ao término da execução do estado atualizacao_do_scantime;
- TON_4_0?: é um canal de comunicação externo de recepção do modelo auxiliar do elemento temporizador do tipo TON.

Na Figura 5.7 apresenta-se um exemplo do modelo em questão, contemplando-se o endereçamento T4:0[22].

[22] Nos CPs da família SLC 500 é possível utilizar até 256 elementos temporizadores do tipo TON, ou seja, de T4:0 até T4:255. Para utilizar-se, por exemplo, um modelo do elemento contador de endereço T4:1, utiliza-se o mesmo modelo da Figura 5.7, substituindo-se nas variáveis o nome 4_0 por 4_1, e assim por diante. Tal substituição também deverá ser contemplada no modelo principal, na ocasião da ativação do modelo de tal elemento.

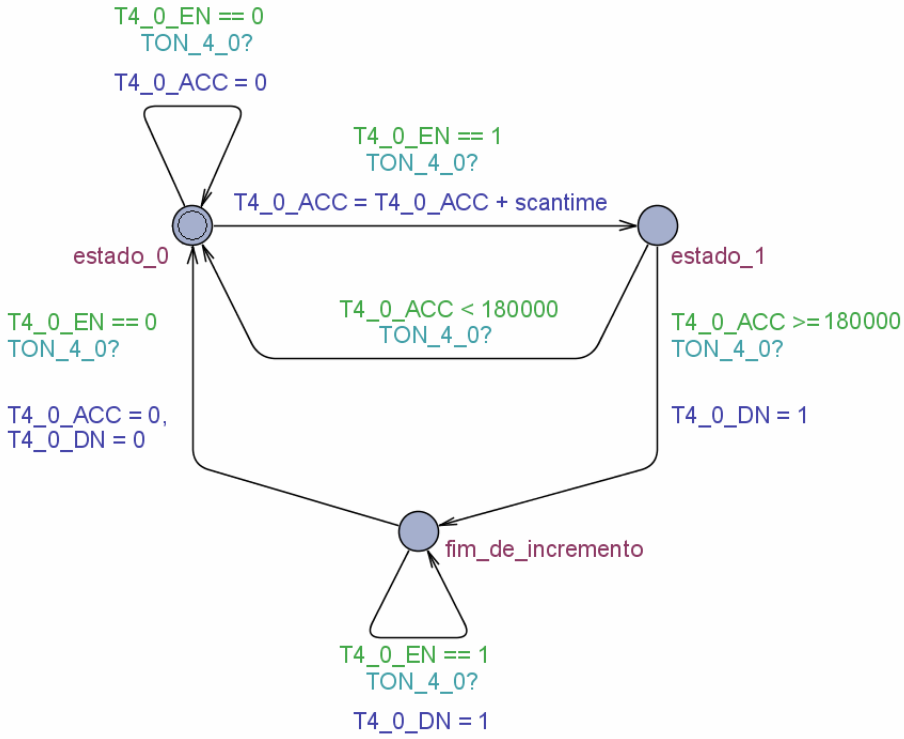


Figura 5.7 – Modelo auxiliar do elemento temporizador do tipo TON de endereço T4:0

Da mesma forma como ocorreu para o modelo auxiliar do elemento contador do tipo CTU, o funcionamento do modelo do elemento temporizador do tipo TON será descrito utilizando-se o modelo da Figura 5.7. Assim, inicialmente, o estado ativo do modelo em questão, é o estado estado_0. Supondo-se que este modelo auxiliar seja ativado pelo modelo principal, para se realizar o incremento de tempo, a variável T4_0_EN estará com valor "1" e permitirá que o modelo auxiliar em questão, ao receber uma habilitação via canal de comunicação externo de recepção TON_4_0?, possa executar a transição entre os estados estado_0 e estado_1. Executando-se tal transição, o valor de T4_0_ACC será incrementado com o valor da variável scantime (T4_0_ACC = T4_0_ACC + scantime). Ao se ativar o estado estado_1, se neste instante o valor de T4_0_ACC for menor que "180000", ao receber uma nova habilitação via canal de comunicação externo de recepção TON_4_0?, o modelo retornará ao estado estado_0 e ficará alternando entre estes dois estados (estado_0 e estado_1) enquanto o valor de T4_0_ACC for menor que "180000" e houver a habilitação para T4_0_EN e TON_4_0?.

Estando ativo o estado_1, sendo o valor de T4_0_ACC maior ou igual a "180000" e sendo o valor de T4_0_EN igual a "1", ao receber uma habilitação via TON_4_0?, o modelo auxiliar em questão ativará o estado fim_de_incremento, fazendo com que o valor de

T4_0_DN seja igual a "1", o que indicará o fim do processo de incremento do tempo. Ao se ativar fim_de_incremento, se durante as próximas execuções do modelo principal o valor de T4_0_EN se mantiver em "1", o modelo auxiliar se manterá neste estado (sem incremento no valor do tempo), e manterá o valor de T4_0_DN em "1" (executando-se a expressão T4_0_DN=1).

Em qualquer um dos três estados do modelo em questão (estado_0, estado_1 e fim_de_incremento), se não houver mais a ativação sendo realizada pelo modelo principal através de T4_0_EN (T4_0_EN igual a "0"), será ativado o estado estado_0, e o valor de T4_0_ACC passará a ser "0". Como o valor de T4_0_DN somente é modificado para "1" ao se ativar o estado fim_de_incremento, quando, neste estado, T4_0_EN for igual a "0", então, além do valor de T4_0_ACC, também será modificado para "0" o valor de T4_0_DN.

A evolução do modelo objeto dessa discussão sempre se dará com a satisfação das condições de restrição impostas às transições, restrições estas formadas por expressões compostas por variáveis de dados e por um canal de comunicação externo de recepção, de forma que ambos devem ser satisfeitos para que ocorra a execução da transição em questão. A maneira pela qual a ocorre a evolução do modelo é resultante do processo de modelagem do comportamento do elemento temporizador do tipo TON, o que foi apresentado no Capítulo 2 (ver um exemplo na Figura 2.7).

5.2.5 Declaração das variáveis associadas aos endereços do programa de controle

Para a maioria das variáveis utilizadas nos modelos em MEFEs propostos pelo presente trabalho (utilizando-se a ferramenta UPPAAL) podem ser associadas até três tipos de informações por variável:

- abrangência: global ou local;
- tipo: inteiro, booleano ou de comunicação;
- valor de inicialização.

Quanto à abrangência, as variáveis classificadas como globais são as que podem ser compartilhadas por todos os modelos que constituem o modelo global. Este é o caso, por exemplo, da variável scantime. Tal variável é atualizada no modelo principal a cada

execução da transição entre os estados atualizacao_do_scantime e leitura_das_entradas, e é utilizada pelo modelo auxiliar do elemento temporizador do tipo TON para a realização do acúmulo de tempo deste modelo. A variável global também é utilizada para se realizar a função de um canal de comunicação interno, conforme apresentado no Capítulo 3.

Por sua vez, as variáveis classificadas como locais são utilizadas somente nos modelos a que pertencem, ou seja, ou no modelo principal ou em um de seus modelos auxiliares, sendo este o caso, por exemplo, das variáveis I1_0_0 e I1_0_1, as quais são utilizadas para o processamento do programa de controle mapeado no modelo principal (Figura 5.4).

As variáveis utilizadas nos modelos podem ser definidas como sendo de um dos seguintes tipos: booleanas, inteiras e de comunicação. Todas as variáveis presentes nas MEFEs que modelam os endereços do tipo booleano do programa de controle em questão são definidas como sendo booleanas, sendo que essas variáveis são mais comumente encontradas no modelo principal. Já as variáveis inteiras são mais comumente encontradas nos modelos auxiliares dos elementos contadores e temporizadores. Os canais de comunicação externos de emissão são encontrados no modelo principal, enquanto que os canais de comunicação externos de recepção são encontrados nos modelos auxiliares dos elementos contadores e temporizadores.

As variáveis booleanas e as variáveis inteiras podem ser inicializadas com valores diferentes de “0”. Isto é necessário, por exemplo, nos modelos auxiliares dos elementos contadores e temporizadores, onde os valores das variáveis associadas ao componente *Preset* devem ser diferentes de “0”. Assim, conforme os exemplos apresentados nas Figuras 5.4 e 5.5, tem-se que:

- g_I1_0_0: variável global do tipo booleano inicializada em zero;
- g_I1_0_1: variável global do tipo booleano inicializada em zero;
- I1_0_0: variável local do tipo booleano inicializada em zero;
- I1_0_1: variável local do tipo booleano inicializada em zero;
- O2_0_0: variável local do tipo booleano inicializada em zero;
- g_O2_0_0: variável global do tipo booleano inicializada em zero;
- scantime: variável global do tipo inteiro inicializada em zero.

No caso do modelo auxiliar do elemento contador do tipo CTU, conforme o modelo

apresentado na Figura 5.6, tem-se:

- C5_0_CU: variável global do tipo booleano inicializada em zero;
- C5_0_DN: variável global do tipo booleano inicializada em zero;
- C5_0_RES: variável global do tipo booleano inicializada em zero;
- C5_0_ACC: variável global do tipo inteiro inicializada em zero;
- CTU_5_0?: é um canal de comunicação externo de recepção.

Para o modelo auxiliar do elemento temporizador do tipo TON, conforme o exemplo apresentado na Figura 5.7, tem-se:

- T4_0_EN: variável global do tipo booleano inicializada em zero;
- T4_0_DN: variável global do tipo booleano inicializada em zero;
- T4_0_ACC: variável global do tipo inteiro inicializada em zero;
- TON_4_0?: é um canal de comunicação externo de recepção;
- scantime: variável global do tipo inteiro inicializada em zero.

5.2.6 Mapeamento dos rungs

O *rung* do exemplo apresentado na Figura 5.4 poderia ser novamente discutido, mas o presente trabalho optou por apresentar outros exemplos de *rungs* pertencentes a programas de controle para que diferentes situações que envolvam estes mapeamentos de *rungs* possam ser contempladas. De uma forma inicial, na Figura 5.8, apresenta-se um exemplo de um programa de controle desenvolvido em LD e composto por dois *rungs*, de maneira que o objetivo deste programa é fazer com que o elemento bobina associado ao endereço de saída O:2.0/0 seja ativado quando os endereços de entrada I:1.0/0 e I:1.0/1 tiverem nível lógico verdadeiro.

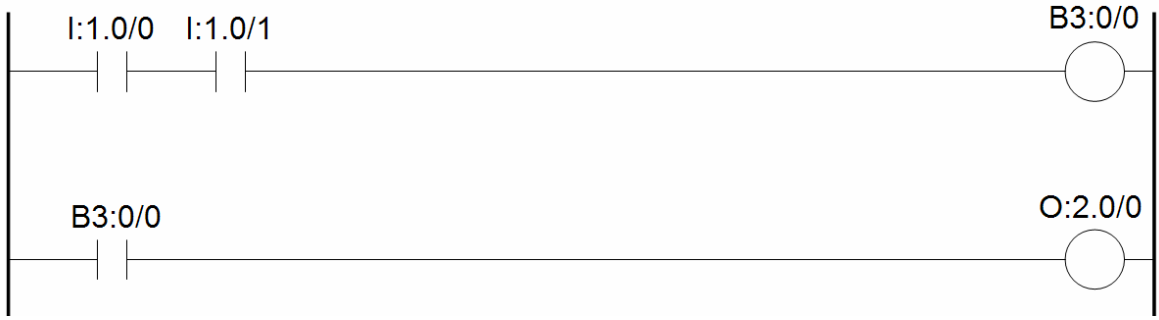


Figura 5.8 – Exemplo de um determinado programa de controle em LD com dois rungs

Para que a ativação de O:2.0/0 seja possível, de acordo com o exemplo da Figura 5.8, é necessário que no *rung* inferior o elemento contato do tipo NA associado ao endereço B3:0/0 seja ativado, o que é obtido quando o elemento bobina associado a B3:0/0 for ativado, o que depende da ativação simultânea[23] dos dois elementos contatos do tipo NA que estão associados aos endereços I:1.0/0 e I:1.0/1 (*rung* superior). A partir deste exemplo, na Figura 5.9 é ilustrado um modelo onde há um exemplo de mapeamento dos *rungs* apresentados na Figura 5.8, mapeamento este que deve ser realizado, conforme já estabelecido por este procedimento, na transição entre os estados processamento_do_programa e emissao_das_saidas.

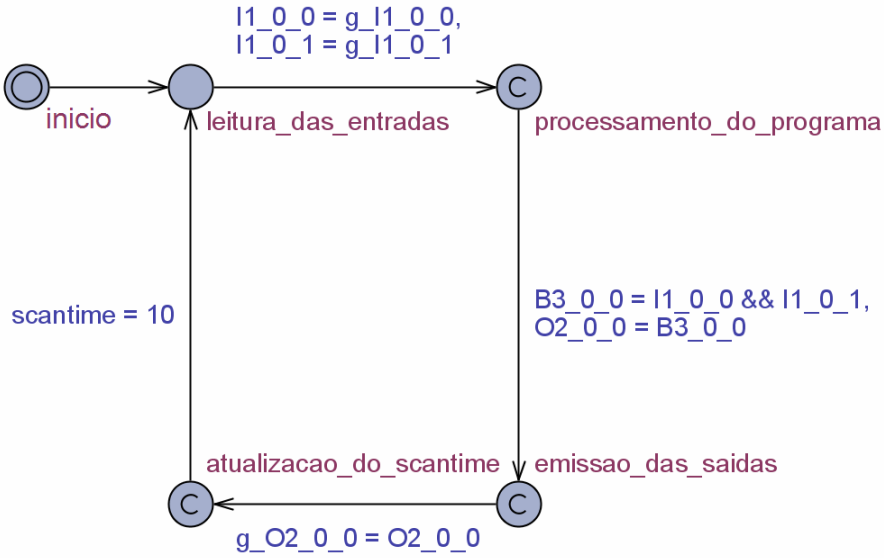


Figura 5.9 – Exemplo de um modelo principal contendo os dois rungs apresentados na Figura 5.8

No próximo exemplo, mostra-se que também é possível a realização do mapeamento de *rungs* que contêm elementos bobinas do tipo *latch* e do tipo *unlatch*. Um exemplo que

[23] Quando houver uma relação lógica do tipo OR entre as variáveis, esta relação será representada na ferramenta UPPAAL com a utilização do símbolo ||, de maneira que diversos exemplos da ocorrência da utilização deste símbolo serão vistos nos modelos principais apresentados no Capítulo 6.

onde tais elementos são utilizados é apresentado na Figura 5.10.

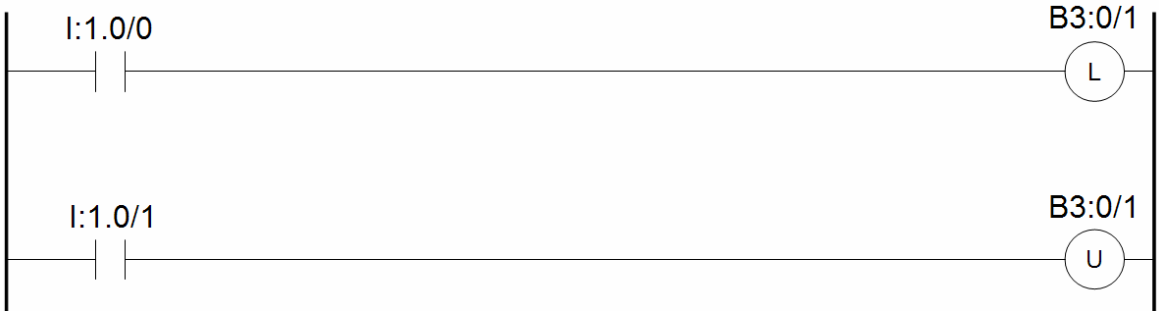


Figura 5.10 – Exemplo de um determinado programa de controle em LD com dois rungs

Na Figura 5.11 apresenta-se um modelo que ilustra um exemplo de mapeamento dos dois *rungs* apresentados na Figura 5.10.

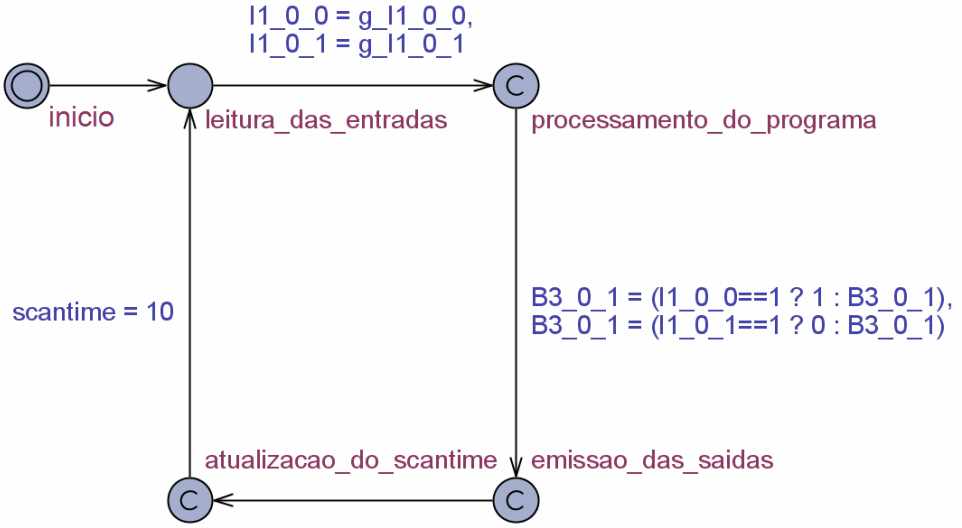


Figura 5.11 – Exemplo de um modelo principal que apresenta o mapeamento dos dois rungs apresentados na Figura 5.10

No modelo da Figura 5.11, a expressão que está vinculada ao elemento bobina do tipo *latch* de endereço B3:0/1 é a seguinte: B3_0_1 = (I1_0_0==1 ? 1 : B3_0_1), onde a variável B3_0_1 refere-se a modelagem do endereço B3:0/1 do CP. Na expressão em questão, compara-se o valor da variável I1_0_0 (que modela o endereço de entrada I:1.0/0, o qual está associado a um elemento contato do tipo NA) com o valor "1" (nível lógico verdadeiro); se esta comparação for satisfeita, atribuí-se o valor "1" à variável B3_0_1, caso contrário, atribui-se à variável B3_0_1 o valor atual dela própria. O símbolo "?" presente na expressão faz parte da sintaxe para este tipo de expressão na ferramenta

UPPAAL, e não tem, neste caso, qualquer relação com os canais de comunicação externos de recepção.

Na expressão vinculada ao elemento bobina do tipo *unlatch* (*rung* inferior da Figura 5.10), a sintaxe da expressão utilizada é semelhante a que foi vista para o elemento bobina do tipo *latch*, sendo que a diferença está no valor do nível lógico que poderá ser atribuído à variável B3_0_1 e na variável utilizada no processo de comparação. Assim, a expressão vinculada ao elemento bobina do tipo *unlatch* representado na Figura 5.10 é a seguinte: B3_0_1 = (I1_0_1==1 ? 0 : B3_0_1), onde compara-se o valor da variável I1_0_1 (que modela o endereço de entrada I:1.0/1, o qual está associado a um elemento contato do tipo NA) com o valor "1". Se a comparação for satisfeita, atribuí-se o valor "0" à variável B3_0_1, caso contrário, atribui-se à variável B3_0_1 o valor atual dela própria.

5.2.7 Inserção de estados devido aos modelos auxiliares dos elementos contadores do tipo CTU

Uma vez executada a parametrização dos modelos auxiliares dos elementos contadores do tipo CTU é possível, finalmente, ativá-los a partir do modelo principal. Isso é obtido (considerando-se que o endereço do elemento contador seja C5:0) com a ativação da variável C5_0_CU e a execução consecutiva de dois canais de comunicação externos de emissão dados por CTU_5_0!. A variável C5_0_CU e o canal de comunicação CTU_5_0? (que sincroniza com os dois canais CTU_5_0! que serão necessários) são encontrados na Figura 5.6.

Os dois canais de comunicação consecutivos CTU_5_0! são necessários no modelo principal porque esta foi a forma encontrada pelo presente trabalho para fazer com que as transições dos modelos auxiliares dos elementos contadores somente fossem habilitadas quando forem executadas as transições entre os estados processamento_do_programa e emissao_das_saidas do modelo principal. Desta maneira, não se permite que tais modelos auxiliares executem quaisquer de suas transições quando o modelo principal estiver executando diferentes transições, isto é, transições que não estejam compreendidas entre os estados processamento_do_programa e emissao_das_saidas.

Na Figura 5.12 é apresentado um modelo principal onde há o mapeamento do *rung*

apresentado na Figura 2.4. Observa-se que na Figura 5.12 não estão sendo representados os modelos auxiliares da entrada I1_0_0 (Figura 5.5 (a)) e do elemento contador do tipo CTU de endereço C5:0 (Figura 5.6), modelos que constituem, junto com o modelo principal da Figura 5.12, o modelo global, representando a modelagem completa do *rung* da Figura 2.4, o qual é possível de ser submetido à verificação.

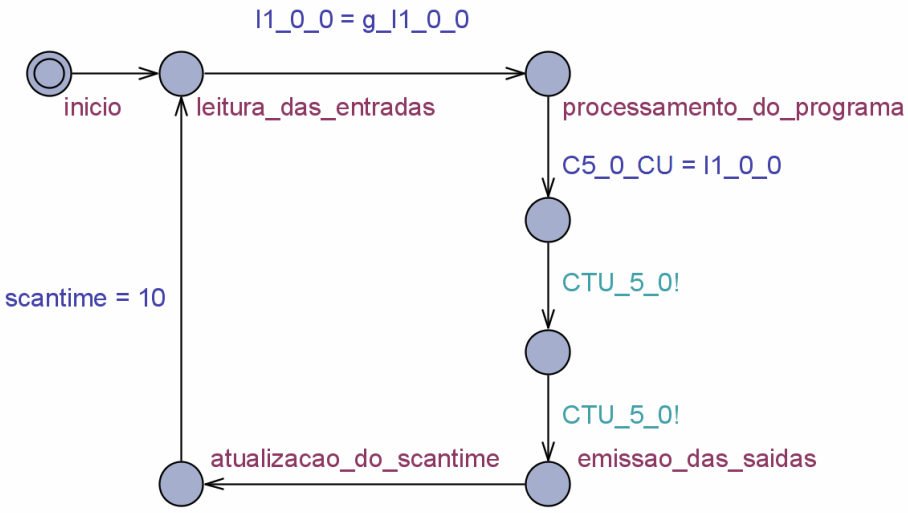


Figura 5.12 – Exemplo de um modelo principal que modela o rung da Figura 2.4

No modelo visto na Figura 5.12 observa-se a utilização de dois canais de comunicação externos consecutivos, o que se deve ao fato de que no modelo auxiliar do elemento contador do tipo CTU ser necessária a execução de duas transições consecutivas em virtude da forma pela qual este elemento do programa de controle foi modelado. Convencionou-se, por uma questão de organização da execução do programa de controle modelado, que nas transições onde são utilizados os dois canais de comunicação externos consecutivos não se realiza a modelagem de expressões booleanas. Por este motivo, a ativação de C5_0_CU foi feita anteriormente em relação aos dois canais de comunicação externos consecutivos. Finalmente, deve ser considerada a inclusão de dois estados no modelo principal, entre os estados processamento_do_programa e emissao_das_sadias, para permitir a execução dos dois canais de comunicação externos de emissão.

5.2.8 Inserção de estados devido aos modelos auxiliares dos elementos temporizadores do tipo TON

Uma vez executada a parametrização dos modelos auxiliares dos elementos

temporizadores do tipo TON é possível, finalmente, ativá-los a partir do modelo principal. Isso é obtido (considerando-se que o endereço do elemento temporizador modelado seja T4:0) com a ativação da variável T4_0_EN e a execução de dois canais de comunicação externos de emissão dados por TON_4_0!. A variável T4_0_EN e o canal de comunicação TON_4_0? (que sincroniza com os dois canais TON_4_0! que serão necessários) são encontrados na Figura 5.7.

De forma semelhante à apresentada para o elemento contador do tipo CTU, os dois canais de comunicação consecutivos TON_4_0! são necessários no modelo principal porque esta foi a forma encontrada pelo presente trabalho para fazer com que as transições dos modelos auxiliares dos elementos temporizadores somente fossem habilitadas quando forem executadas as transições entre os estados processamento_do_programa e emissao_das_saidas do modelo principal. Desta maneira, não se permite que tais modelos auxiliares executem quaisquer de suas transições quando o modelo principal estiver executando diferentes transições, isto é, transições que não estejam compreendidas entre os estados processamento_do_programa e emissao_das_saidas.

Desta forma, na Figura 5.13 apresenta-se um modelo principal onde há o mapeamento do *rung* visto na Figura 2.6. Observa-se que na Figura 5.13 não estão sendo representados os modelos auxiliares da entrada I1_0_0 (Figura 5.5 (a)) e do elemento temporizador do tipo TON de endereço T4:0 (Figura 5.7), modelos que constituem, junto com o modelo principal da Figura 5.13, o modelo global, o qual representa a modelagem completa do *rung* da Figura 2.6.

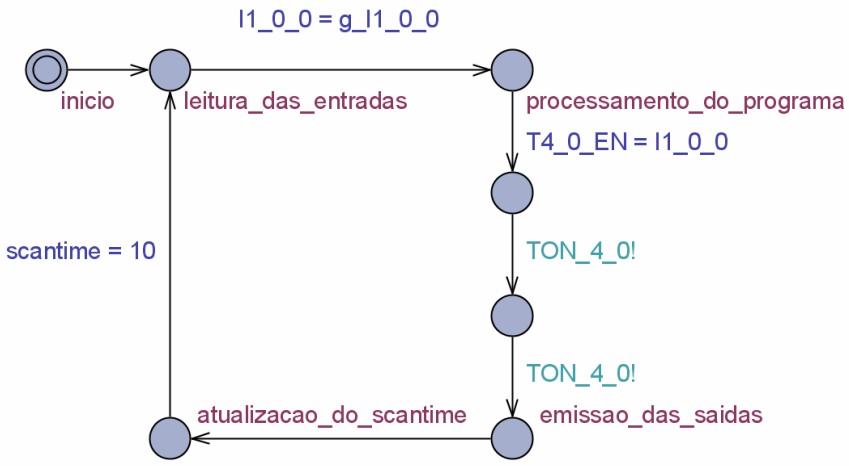


Figura 5.13 – Exemplo de um modelo principal que modela o rung da Figura 2.6

No modelo visto na Figura 5.13 observa-se a utilização de dois canais de comunicação externos consecutivos, o que se deve ao fato de que no modelo auxiliar do elemento temporizador do tipo TON ser necessária a execução de duas transições consecutivas em virtude da forma pela qual este elemento do programa de controle foi modelado. Convencionou-se, por uma questão de organização da execução do programa de controle modelado, que nas transições onde são utilizados os dois canais de comunicação externos consecutivos não se realiza a modelagem de expressões booleanas. Por este motivo, a ativação de T4_0_EN foi feita anteriormente em relação aos dois canais de comunicação externos consecutivos. Finalmente, deve ser considerada a inclusão de dois estados no modelo principal, entre os estados processamento_do_programa e emissao_das_sadias, para permitir a execução dos dois canais de comunicação externos de emissão.

5.2.9 Inclusão do tempo de varredura considerado

O menor valor inteiro (maior que zero) que pode ser associado à variável scantime presente no modelo principal é o valor "1". Desta forma, convencionou-se que tal valor corresponde a um valor de tempo de varredura equivalente a 1ms. Devido a isso, conforme apresentado anteriormente, se um valor igual a "10" for atribuído à variável scantime ao término da execução do estado atualizacao_do_scantime, isto significará que estará sendo considerado um tempo de varredura de 10ms.

Desta forma, foram apresentados os passos propostos à execução do procedimento de modelagem de MEFEs a partir de programas de controle escritos em LD. A seguir, no Capítulo 6, serão apresentados os dois estudos de casos que são abordados por este trabalho.

CAPÍTULO 6

6. ESTUDO DE CASOS

Neste capítulo serão abordados dois trabalhos onde há a modelagem e a verificação de modelos que são obtidos a partir de programas de controle desenvolvidos em LD. Estes trabalhos são o de Moon (1994) e o de Zoubek (2004). Os programas de controle apresentados em tais trabalhos serão modelados de acordo com o procedimento de modelagem apresentado no Capítulo 5, e os resultados obtidos pelo presente trabalho serão comparados com os resultados obtidos por Moon (1994) e Zoubek (2004).

Para a realização do estudo dos trabalhos em questão, foi observada a necessidade de estes trabalhos contemplarem os seguintes requisitos:

- a documentação dos programas de controle desenvolvidos em LD;
- a documentação das proposições utilizadas para a verificação dos modelos gerados a partir dos programas de controle em questão;
- a documentação dos resultados das diversas verificações realizadas nestes modelos.

6.1 O TRABALHO DE MOON

Moon (1994) pode ser considerado como sendo um dos primeiros trabalhos a utilizar uma ferramenta computacional para a verificação de modelos gerados a partir de programas de controle escritos em LD. Na modelagem realizada por Moon (1994), cada *rung* pertencente ao programa de controle considerado foi convertido em uma expressão booleana, de maneira que esta informação fosse possível de ser interpretada pela ferramenta computacional de verificação de modelos utilizada, que foi a SMV (*Symbolic Model Verifier*). O trabalho de Moon (1994) considerou que o programa de controle a ser modelado não deveria possuir elementos temporizadores e/ou contadores, e não foi

considerado o tempo de varredura do CP, o que caracterizou uma limitação em termos de aplicações.

Uma ilustração simplificada deste sistema de alarme é apresentada na Figura 6.1, onde o dispositivo de realização de controle é um CP, cujas especificações não foram apresentadas em Moon (1994).

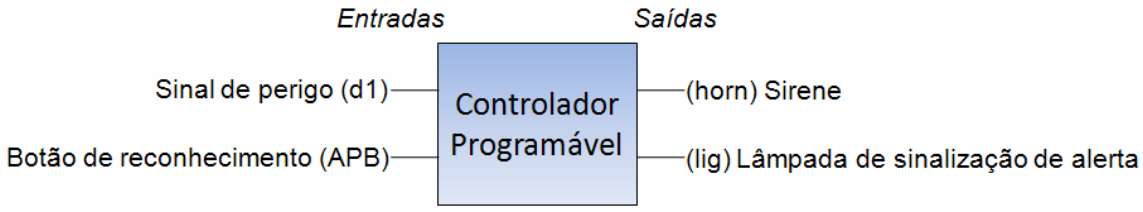


Figura 6.1 – Ilustração simplificada do sistema de alarme proposto por Moon (1994)

Na ilustração simplificada da Figura 6.1, do lado esquerdo observam-se dois endereços de entrada: o primeiro endereço é d1 (que representa um sinal de perigo), o qual está associado a um dispositivo de detecção; o segundo endereço é APB (que está associado a um botão para o reconhecimento do alarme que foi ativado), o qual está associado a um dispositivo de comando (botão). Do lado direito da mesma ilustração podem ser observados dois endereços de saída que estão associados a dispositivos de monitoração: horn, que é o endereço de uma saída que aciona uma sirene e lig, que é o outro endereço de saída que aciona uma lâmpada de sinalização de alerta.

A descrição do funcionamento deste sistema é a seguinte: se um sinal de perigo é detectado (d1=1), o sistema de alarme deverá ativar, imediatamente, a sirene (horn=1). Então, a partir deste fato, o operador que estiver trabalhando com este sistema de alarme poderá apertar o botão de reconhecimento (APB=1) para que a sirene seja desligada. Nesta condição, ao apertar o botão de reconhecimento, imediatamente a lâmpada de sinalização de alerta deverá ser acesa (lig=1), devendo permanecer acesa enquanto o sinal de perigo estiver ativo (d1=1). Se neste intervalo de tempo, devido à correção do problema, não houver mais o sinal de perigo (d1=0), então a lâmpada de sinalização de alerta deverá ser apagada imediatamente (lig=0), fazendo com que o sistema de alarme retorne ao seu estado inicial.

6.1.1 Programa de controle do sistema de alarme e proposta de modelagem

O programa de controle apresentado por Moon (1994) para este sistema de alarme é ilustrado na Figura 6.2.

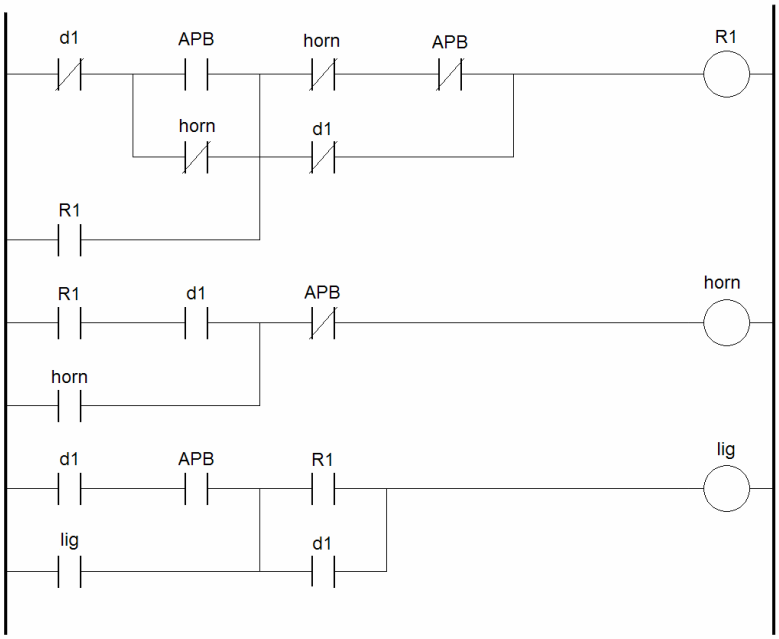


Figura 6.2 – Programa de controle do sistema de alarme (MOON, 1994)

A modelagem deste programa de controle (apresentado na Figura 6.2) em MEFEs, foi obtida a partir do procedimento de modelagem do Capítulo 5, de forma que o modelo principal resultante é ilustrado na Figura 6.3, o qual é denominado Principal. Os *rungs* convertidos em expressões booleanas foram mapeados na transição entre os estados processamento_do_programa e emissao_das_saidas. É importante destacar que os estados compromissados foram utilizados em virtude das considerações apresentadas no Capítulo 3 e Capítulo 5 sobre a ferramenta UPPAAL.

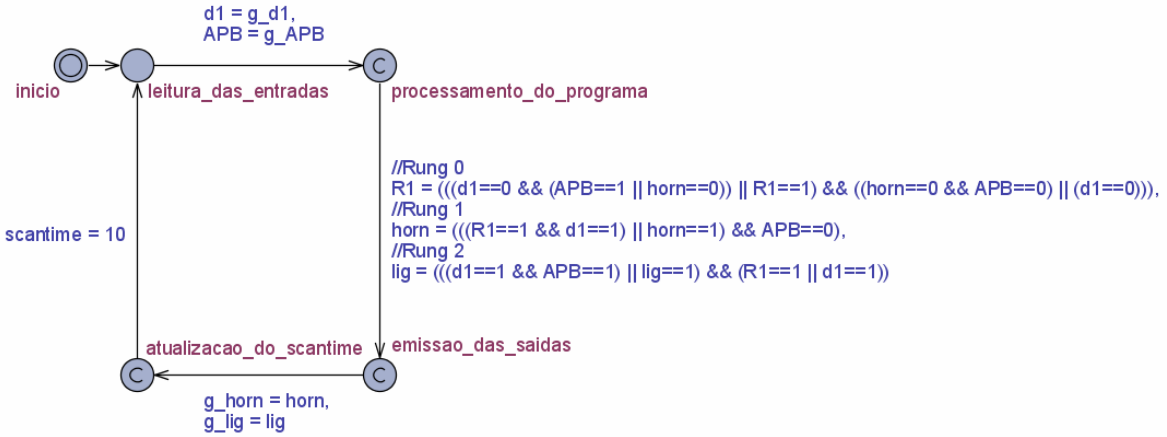
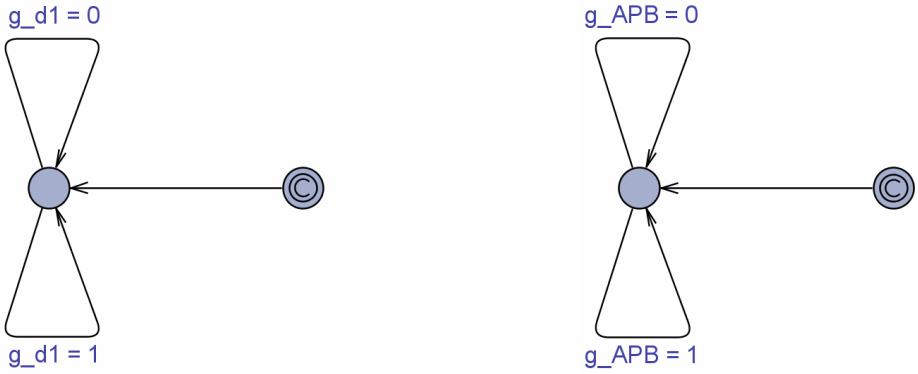


Figura 6.3 – Modelo principal criado a partir do programa de controle apresentado na Figura 6.2

No trabalho de Moon (1994) argumenta-se que os valores que podem ser atribuídos aos dois endereços de entrada existentes, d1 e APB, podem ser imprevisíveis, em virtude, por exemplo, da possibilidade de ocorrência de anomalias a que estão sujeitos os dispositivos de detecção e de comando conectados a essas entradas do CP, assunto que foi abordado nos Capítulos 1 e 5. Portanto, para que seja realizada a verificação do modelo do programa de controle, Moon (1994) assume que em cada execução do modelo pela ferramenta SMV podem ocorrer de forma não determinística quatro situações combinatórias dos valores lógicos associados a estas duas entradas:

- d1==0 e APB==0;
- d1==0 e APB==1;
- d1==1 e APB==0;
- d1==1 e APB==1.

Neste sentido, no presente trabalho, dois modelos auxiliares das entradas foram necessários para se realizar a modelagem do comportamento das duas entradas utilizadas pelo programa de controle do sistema de alarme, conforme apresentado nas Figuras 6.4 (a) e 6.4 (b), as quais constituem, junto com o modelo principal (Figura 6.3), o modelo global, o qual representa a modelagem completa do programa de controle do sistema de alarme em discussão.



a) Modelo do endereço de entrada d1 b) Modelo do endereço de entrada APB

Figura 6.4 – Modelos auxiliares dos endereços de entrada

6.1.2 Verificação do modelo proposto

O presente trabalho considerou três situações contempladas em Moon (1994) e utilizará a versão simplificada da lógica temporal CTL para a verificação do modelo global proposto, tendo-se por base tais situações (Tabela 6.1).

No Capítulo 4 foram apresentadas as duas possíveis formas de composição de expressões proposicionais na versão simplificada da lógica temporal CTL. Para a verificação do modelo do programa de controle do sistema de alarme será utilizada somente a expressão proposicional[24] do tipo nome_do_modelo.variável_do_modelo==*x*.

[24] Na ferramenta UPPAAL quando uma variável é do tipo global, para a estipulação de uma expressão proposicional envolvendo tal variável, não é especificado o nome_do_modelo, pois, sendo do tipo global, tal variável poderá ser utilizada em mais de um modelo.

Tabela 6.1 – Descrição das situações a serem verificadas no modelo do programa do sistema de alarme de Moon (1994)

Número da verificação	Descrição formal da situação que deve ser verificada apresentada por Moon (1994)	Proposição apresentada por Moon (1994) (ferramenta SMV)	Proposição apresentada pelo presente trabalho (ferramenta UPPAAL)
1	Sempre que o botão de reconhecimento for pressionado a sirene será desligada?	AG(APB $\rightarrow AF \neg$horn)	Principal.APB==1 $\rightarrow$ g_horn==0
2	Sempre que não houver o sinal de perigo a lâmpada de sinalização de alerta estará apagada?	AG($\neg$d1 $\rightarrow AF \neg$lig)	Principal.d1==0 $\rightarrow$ g_lig==0
3	Jamais ocorrerá a situação em que a sirene toca e que a lâmpada está acesa (simultaneamente)?	$AG \neg$(horn $\wedge$ lig)	***A[] not*** (g_horn==1 and g_lig==1)

O resultado obtido com a realização da verificação de número "1" foi que tal verificação foi satisfeita, resultado este também obtido por Moon (1994). Este era o resultado desejado, conforme a especificação do funcionamento do programa de controle que foi apresentada no início da seção 6.1.

Quanto à verificação de número "2", o resultado obtido foi que a mesma não foi satisfeita, indicando haver alguma situação na qual a lâmpada de sinalização acende (g_lig==1) sem haver o sinal de perigo (Principal.d1==0), o que não era o resultado desejado. Novamente, este resultado também foi o obtido por Moon (1994).

Executando-se uma análise do modelo principal utilizando-se o histórico de estados simbólicos gerado pela ferramenta UPPAAL, identificou-se que era possível ocorrer, simultâneamente, Principal.d1==0 e g_lig==1. Neste histórico foi possível observar que em uma determinada situação, ao término da execução do estado atualizacao_do_scantime, as variáveis lig e R1 tinham nível lógico verdadeiro (lig==1 e R1==1). Na execução seguinte do modelo principal, o algoritmo de verificação da ferramenta UPPAAL modificou o valor de g_d1 para nível lógico falso (g_d1==0), e este valor foi transferido para a variável d1 (d1==0), utilizada no processamento do programa de controle. Quando iniciou-se a execução do programa de controle modelado no modelo principal, as variáveis lig e R1 apresentavam nível lógico verdadeiro (lig==1 e R1==1), dando condições (no terceiro *rung* mapeado) para que a variável lig se mantivesse em nível lógico verdadeiro. Desta forma,

mesmo se a variável d1 tivesse nível lógico falso (que era o caso), o valor da variável lig seria mantido em nível lógico verdadeiro através das variáveis lig e R1. Depois da execução do estado emissao_das_saidas, o nível lógico verdadeiro armazenado na variável lig foi transferido para a variável g_lig (g_lig==1).

Conforme sugere Moon (1994), uma possível solução para se resolver este problema estaria na remoção do elemento contato do tipo NA associado ao endereço R1, o qual está em paralelo com o elemento contato do tipo NA associado ao endereço d1, conforme é ilustrado na Figura 6.5.

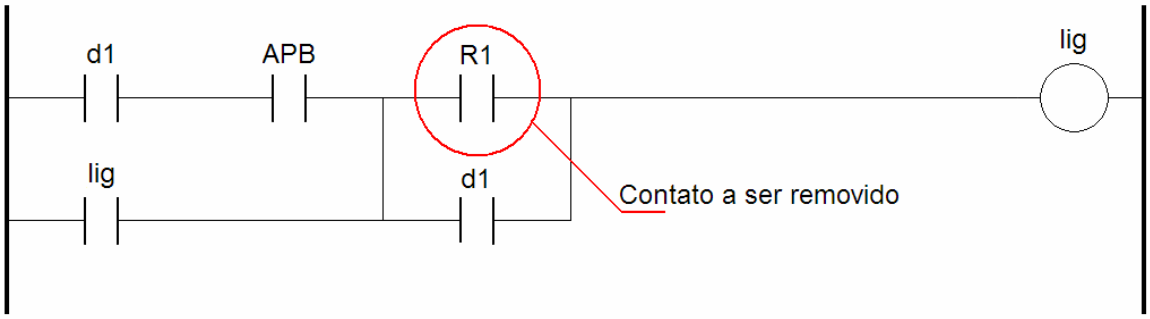


Figura 6.5 – Rung a ser corrigido do programa de controle do sistema de alarme (MOON, 1994)

A correção sugerida foi aplicada ao modelo principal do programa de controle em questão e, depois da realização de uma nova verificação, o resultado obtido foi que tal verificação foi satisfeita, sendo o mesmo resultado também obtido por Moon (1994). A Figura 6.6 apresenta o modelo principal com a correção sugerida.

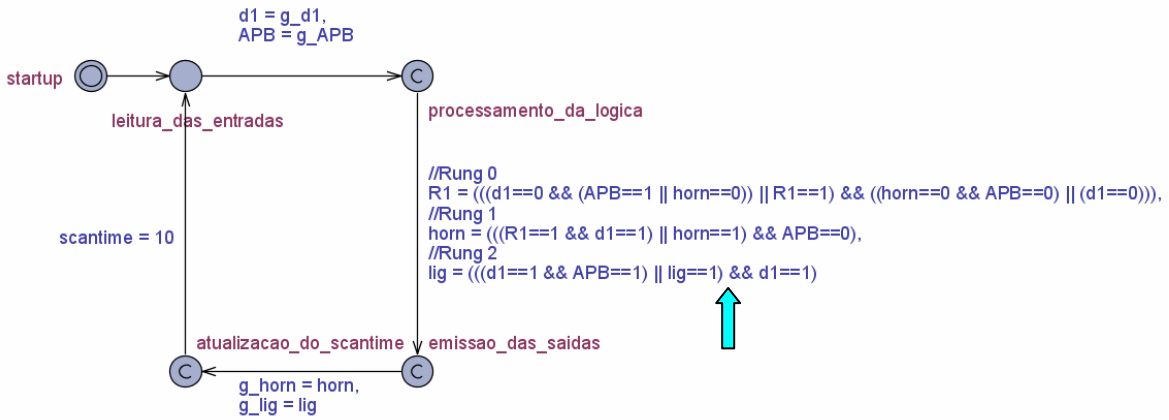


Figura 6.6 – Modelo principal corrigido do programa de controle do sistema de alarme

Para a realização da verificação de número “3”, o modelo utilizado pelo presente trabalho é

o modelo da Figura 6.6. Moon (1994), para a verificação em questão, também utiliza o modelo corrigido. O resultado obtido em ambos os trabalhos foi que a verificação é satisfeita, atendendo-se à especificação do programa de controle, onde não deve haver qualquer situação (estado simbólico) onde a lâmpada de sinalização de alerta esteja acesa (g_lig==1) e a sirene ligada (g_horn==1) simultaneamente.

6.2 O TRABALHO DE ZOUBEK

O trabalho de Zoubek (2004) pode ser considerado como sendo uma evolução do trabalho proposto por Moon (1994) porque contempla, no processo de criação de modelos a partir de programa de controle em LD[25], o tempo de varredura do CP e o elemento temporizador do tipo TON.

Tal trabalho tem por objetivo realizar a modelagem de autômatos temporizados estendidos com variáveis de dados a partir de programas de controle escritos em LD, para posterior verificação destes modelos com a ferramenta UPPAAL.

Em um dos dois estudos de casos do trabalho de Zoubek (2004), foram apresentadas três situações para as quais três verificações deveriam ser realizadas, verificações que tinham o objetivo de identificar erros em modelos gerados a partir de um programa de controle (de um equipamento queimador de gás) desenvolvido em LD, programa este que foi modelado utilizando-se autômatos temporizados estendidos com variáveis de dados. Para se realizar as três verificações, foi necessário o desenvolvimento de três modelos diferentes, os quais foram desenvolvidos sem considerar todos os *rungs* que constituiam o programa de controle em questão. O motivo pelo qual foi construído um modelo diferente para cada verificação é porque o modelo resultante da modelagem de todo o programa de controle, ao ser submetido ao processo de verificação, gerava uma explosão combinatória de

[25] Zoubek (2004) utiliza a simbologia/codificação empregada em CPs da família SLC 500.

estados, impedindo que a ferramenta UPPAAL determinasse o resultado da verificação. Desta maneira, de acordo com a proposição de lógica temporal estipulada em cada verificação, Zoubek (2004) considera no desenvolvimento de cada modelo somente os *rungs* que julga influenciar no processo de verificação em questão.

Para se ter uma noção da complexidade dos modelos obtidos por Zoubek (2004), na Figura 6.7 são apresentados dois *rungs*, de um total de dezoito, que constituem o programa de controle do equipamento queimador de gás, que foram considerados para a construção de um modelo que é utilizado para a realização da verificação de número "3", cuja proposição será apresentada na Tabela 6.3.

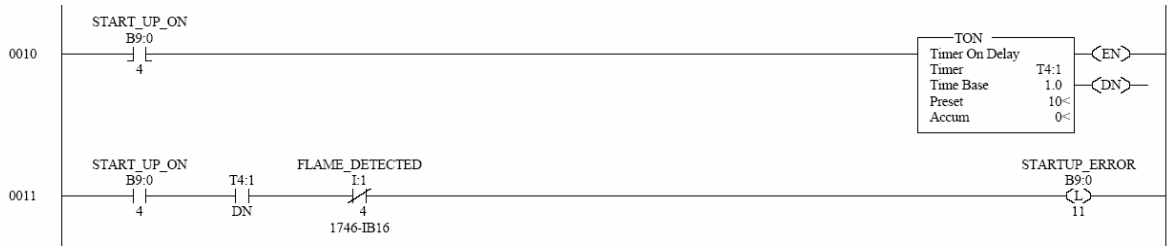


Figura 6.7 – Rungs 0010 e 0011 do programa de controle do equipamento queimador de gás (ZOUBEK, 2004)

O modelo (principal) em autômato temporizado estendido com variáveis de dados resultante da modelagem dos *rungs* apresentados na Figura 6.7 é ilustrado na Figura 6.8.

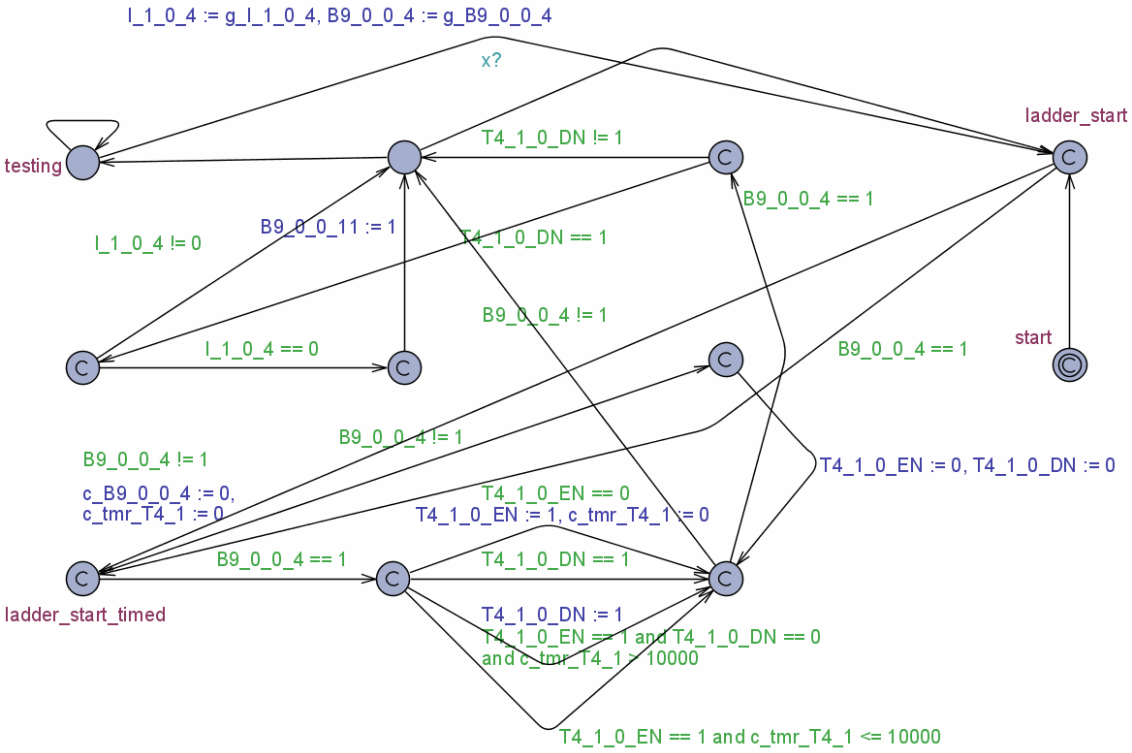


Figura 6.8 – Modelo dos dois rungs da Figura 6.7 (ZOUBEK, 2004)

6.2.1 Programa de controle do equipamento queimador de gás

No estudo de caso do equipamento queimador de gás, Zoubek (2004) apresenta um esquema simplificado onde são ilustrados os dispositivos de atuação (válvulas e ignitor) e de detecção (sensor de pressão e detector de chamas) do equipamento em questão. Este esquema é apresentado na Figura 6.9.

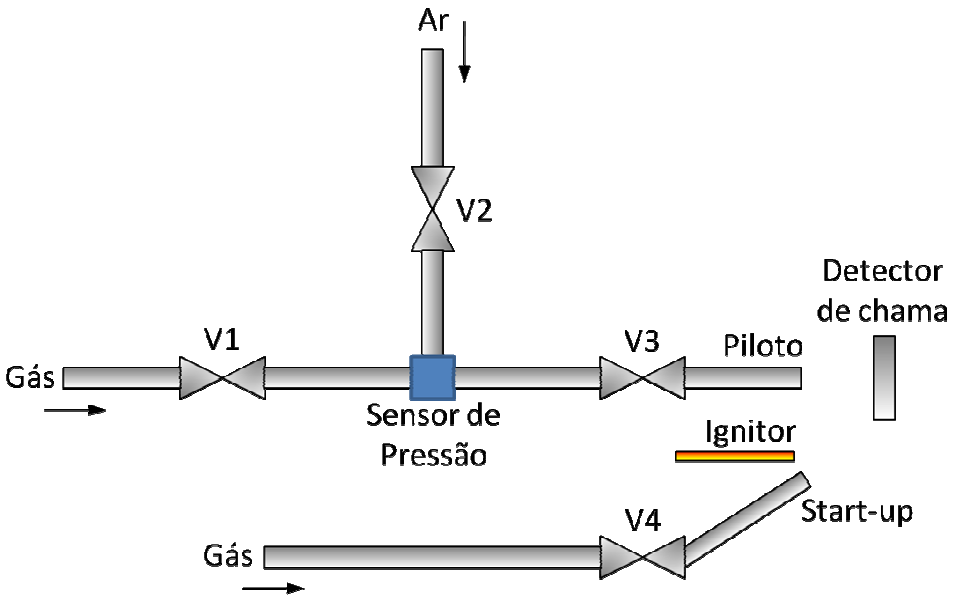


Figura 6.9 – Esquema simplificado do equipamento queimador de gás (ZOUBEK, 2004)

O programa de controle do equipamento queimador de gás em questão foi desenvolvido para ser executado por um CP da família SLC 500, sendo tal programa dividido em duas figuras para adequar-se à estruturação deste texto. A primeira parte do programa de controle em questão, que compeende do *rung* 0000 ao *rung* 0007, é ilustrada na Figura 6.10, e a segunda parte do mesmo programa de controle, que compeende do *rung* 0008 ao *rung* 0015, é apresentada na Figura 6.11.

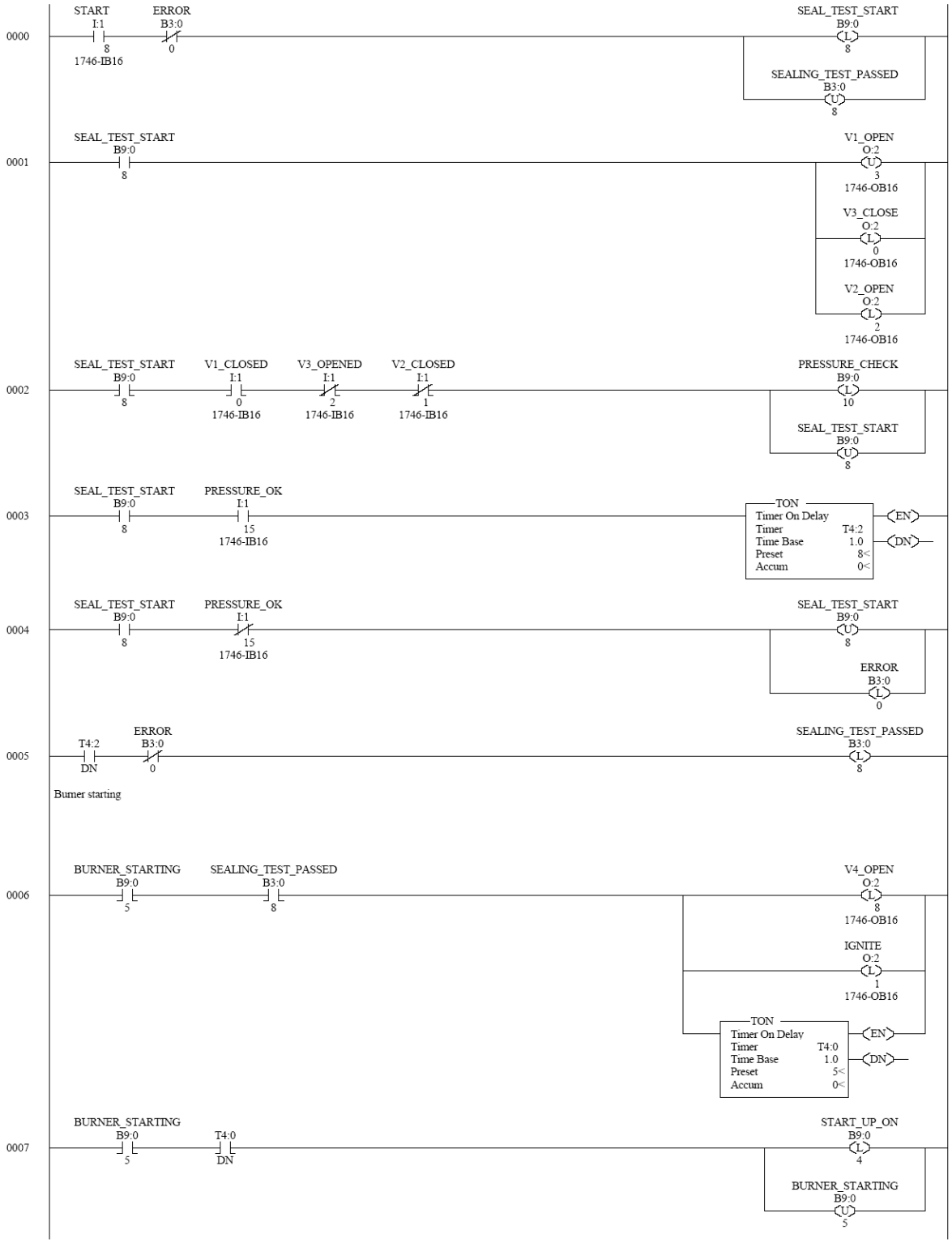


Figura 6.10 – Programa de controle do queimador de gás – Parte 1 (ZOUBEK, 2004)

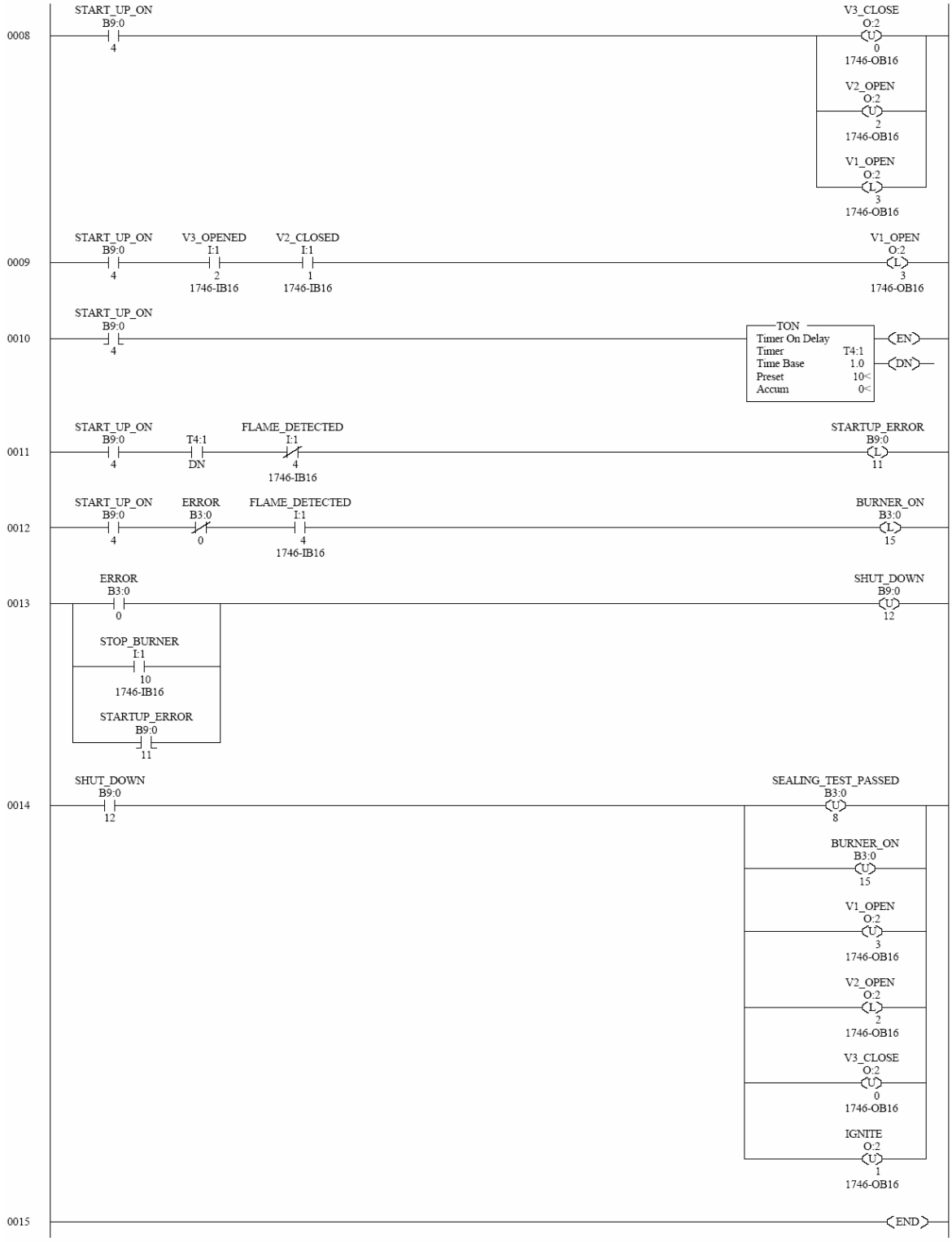


Figura 6.11 – Programa de controle do queimador de gás – Parte 2 (ZOUBEK, 2004)

Observa-se nas Figuras 6.10 e 6.11 que os endereços do CP utilizados no programa de controle em questão estão associados a nomes simbólicos. Pode-se dizer que o objetivo de tal associação seja o de facilitar o desenvolvimento e a interpretação do programa de

controle. A Tabela 6.2 contempla a relação de todos os endereços utilizados no programa de controle em questão, seus respectivos nomes simbólicos, suas descrições e suas representações na modelagem realizada por Zoubek (2004) e pelo presente trabalho.

Tabela 6.2 – Endereços do programa de controle do equipamento queimador de gás

Endereço no programa de controle	Nome simbólico associado ao endereço no programa de controle	Descrição do endereço	Endereço modelado no trabalho de Zoubek (2004)	Endereço modelado pelo presente trabalho
I:1.0/0	V1_CLOSED	A válvula V1 está fechada	I_1_0_0	I1_0_0
I:1.0/1	V2_CLOSED	A válvula V2 está fechada	I_1_0_1	I1_0_1
I:1.0/2	V3_CLOSED	A válvula V3 está fechada	I_1_0_2	I1_0_2
I:1.0/4	FLAME_DETECTED	A chama principal está acesa	I_1_0_4	I1_0_4
I:1.0/8	START	Botão para início da queima	I_1_0_8	I1_0_8
I:1.0/10	STOP_BURNER	Botão para parar o queimador	I_1_0_10	I1_0_10
I:1.0/15	PRESSURE_OK	A pressão está correta	I_1_0_15	I1_0_15
O:2.0/0	V3_CLOSE	Fechar a válvula V3	O_2_0_0	O2_0_0
O:2.0/1	IGNITE	Iniciar o ignitor	O_2_0_1	O2_0_1
O:2.0/2	V2_OPEN	Abrir a válvula V2	O_2_0_2	O2_0_2
O:2.0/3	V1_OPEN	Abrir a válvula V1	O_2_0_3	O2_0_3
O:2.0/8	V4_OPEN	Abrir a válvula V4	O_2_0_8	O2_0_8
B3:0/0	ERROR	Erro ocorrido durante a inicialização	B_3_0_0_0	B3_0_0
B3:0/8	SEALING_TEST_PASSED	O teste de selagem foi executado com sucesso	B_3_0_0_8	B3_0_8
B3:0/15	BURNER_ON	O queimador principal está ligado	B_3_0_15	B3_0_15
B9:0/4	STARTUP_ON	A inicialização foi finalizada	B9_0_0_4	B9_0_4
B9:0/5	BURNER_STARTING	O queimador está iniciando o funcionamento	B9_0_0_5	B9_0_5
B9:0/8	SEAL_TEST_START	Iniciar o teste de selagem	B9_0_0_8	B9_0_8
B9:0/10	PRESSURE_CHECK	Checar a pressão	B9_0_0_10	B9_0_10
B9:0/11	STARTUP_ERROR	Erro de inicialização do queimador	B9_0_11	B9_0_11
B9:0/12	SHUT_DOWN	Desligar o queimador	B9_0_12	B9_0_12

6.2.2 Proposta de modelagem

O presente trabalho apresenta um modelo global do programa de controle (do equipamento queimador de gás) encontrado em Zoubek (2004), modelo este que foi desenvolvido seguindo-se o procedimento de modelagem proposto no Capítulo 5. O modelo global obtido é constituído pelo modelo principal apresentado na Figura 6.12, pelos modelos auxiliares das entradas (Figura 6.13) e pelos modelos auxiliares dos elementos temporizadores do tipo TON (Figura 6.14). A partir deste único modelo global e que não necessita de simplificações, realizam-se três verificações no modelo, e o resultado obtido

em cada uma destas verificações é comparado com o resultado obtido em cada verificação realizada por Zoubek (2004).

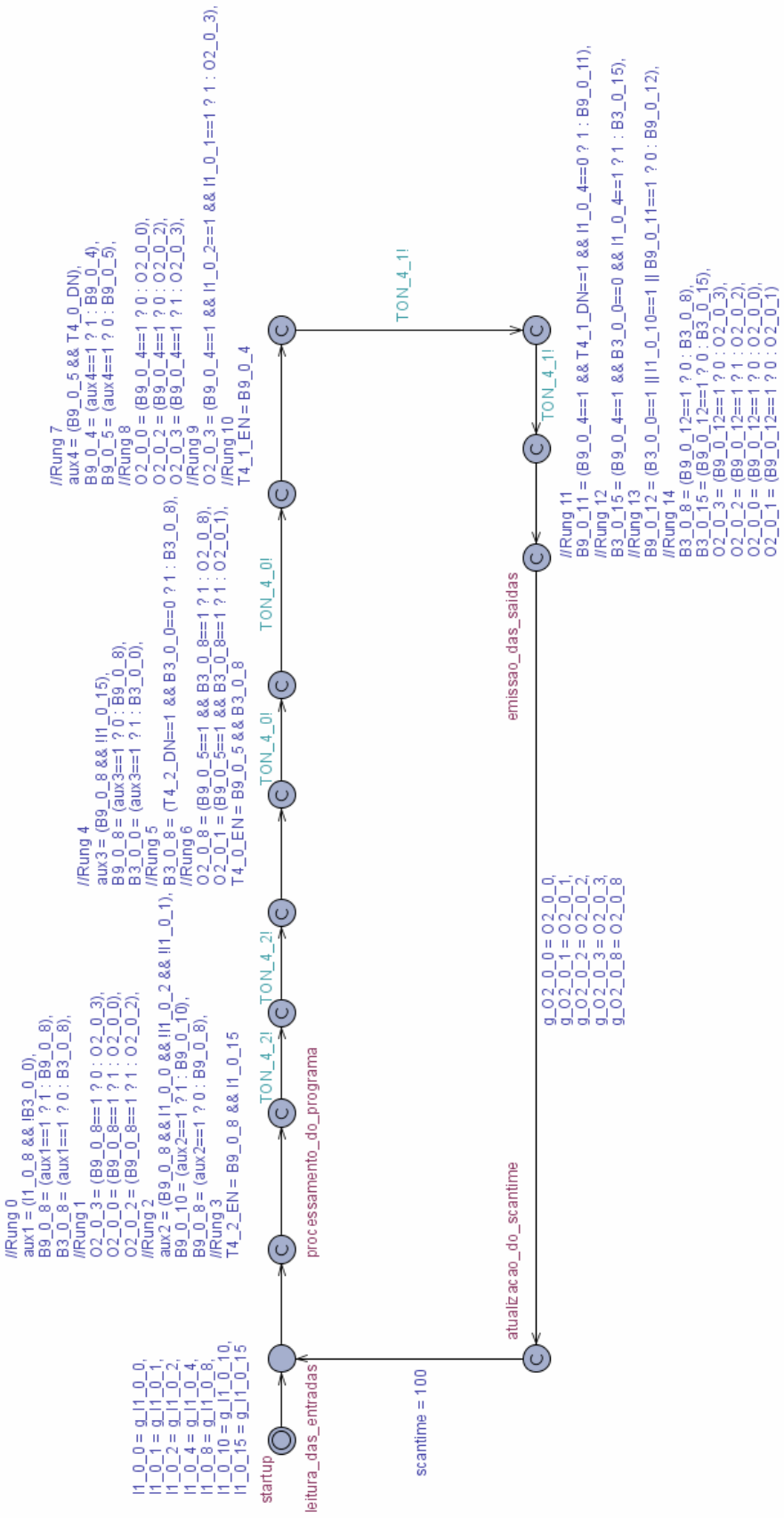
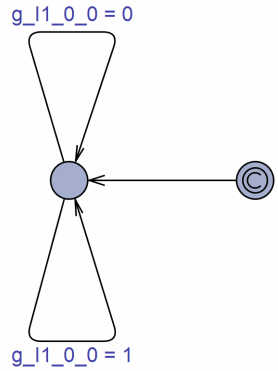
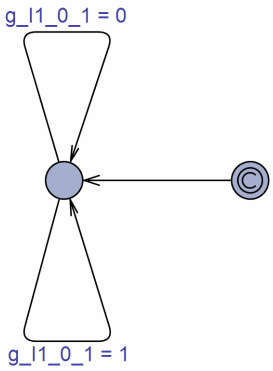


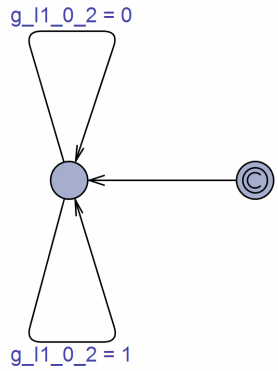
Figura 6.12 – Modelo principal proposto pelo presente trabalho do programa de controle do equipamento queimador de gás



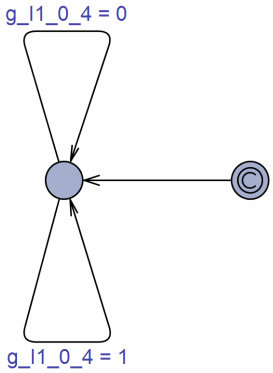
(a) Modelo auxiliar de entrada de I:1.0/0



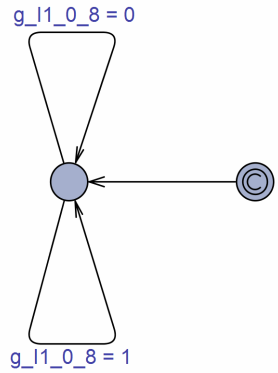
(b) Modelo auxiliar de entrada de I:1.0/1



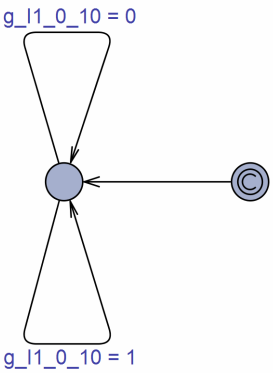
(c) Modelo auxiliar de entrada de I:1.0/2



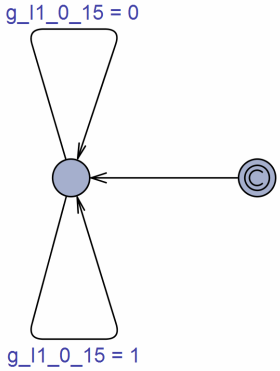
(d) Modelo auxiliar de entrada de I:1.0/4



(e) Modelo auxiliar de entrada de I:1.0/8

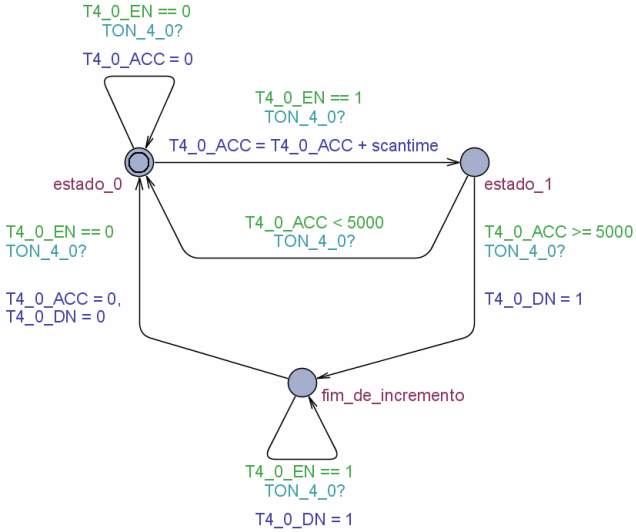


(f) Modelo auxiliar de entrada de I:1.0/10

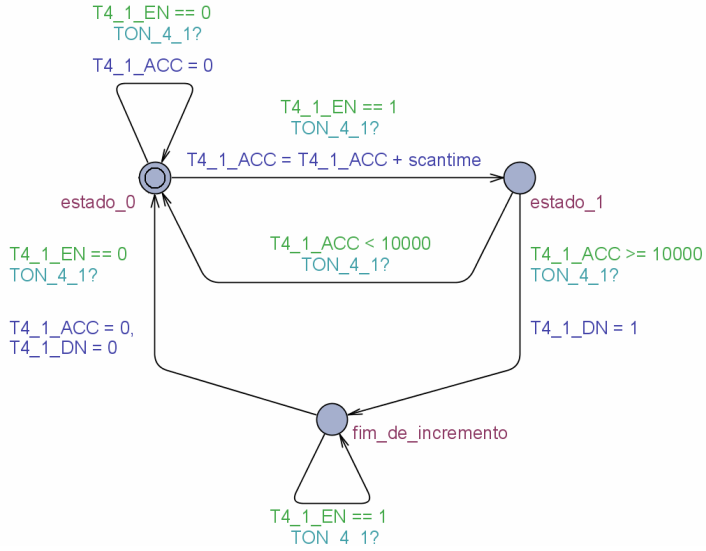


(g) Modelo auxiliar de entrada de I:1.0/15

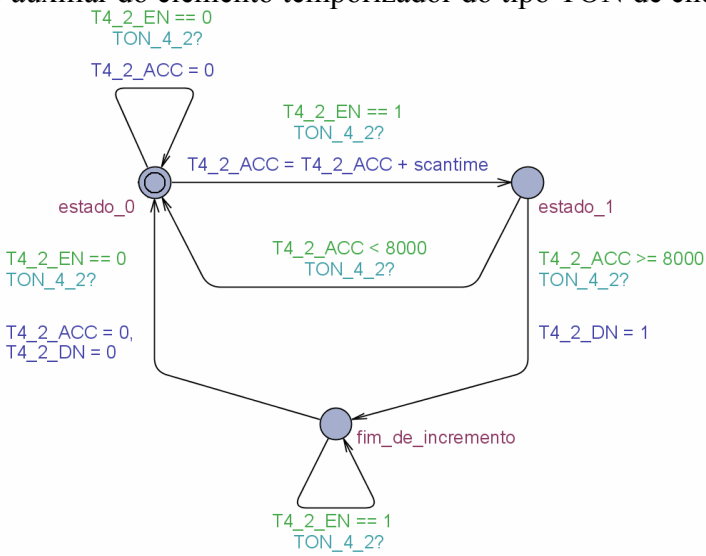
Figura 6.13 – Modelos auxiliares dos endereços de entrada



(a) Modelo auxiliar do elemento temporizador do tipo TON de endereço T4:0



(b) Modelo auxiliar do elemento temporizador do tipo TON de endereço T4:1



(c) Modelo auxiliar do elemento temporizador do tipo TON de endereço T4:2

Figura 6.14 – Modelos auxiliares dos elementos temporizadores do tipo TON

6.2.3 Verificação do modelo proposto

Assim, tendo-se o modelo global (Figuras 6.12, 6.13 e 6.14) do programa de controle do equipamento queimador de gás e a descrição formal de cada uma das três situações que devem ser verificadas, as proposições da versão simplificada da lógica temporal CTL necessárias para a verificação das três situações apresentadas por Zoubek (2004) são descritas na Tabela 6.3, sendo tais proposições formadas pelas duas possíveis formas de expressões proposicionais apresentadas no Capítulo 4, ou seja, nome_do_modelo.estado_do_modelo e nome_do_modelo.variável_do_modelo==valor.

Tabela 6.3 – Descrição das situações a serem verificadas no modelo do programa do queimador de gás de Zoubek (2004)

Número da verificação	Descrição formal da situação que deve ser verificada apresentada por Zoubek (2004)	Proposição apresentada por Zoubek (2004)	Proposição apresentada pelo presente trabalho
1	Se ocorrer um erro no sistema, o teste de selagem jamais pode ser indicado como bem sucedido.	***A[]*** not (P8.testing and P8.B3_0_0_0==1 and P8.B3_0_0_8==1)	***A[]*** not (Principal.leitura_das_entradas and Principal.B3_0_0==1 and Principal.B3_0_8==1)
2	Se ocorrer um erro o queimador se desligará.	***E<>*** (P9.testing and P9.B3_0_0_0==1 and P9.B9_0_0_12==0)	***E<>*** (Principal.leitura_das_entradas and Principal.B3_0_0==1 and Principal.B9_0_12==0)
3	Durante a inicialização do queimador, se depois de um período de 10 segundos uma chama não for detectada será indicado o erro de inicialização do queimador.	***A[]*** ((P3.testing and P3.I_1_0_4==0 and P3.c_B9_0_0_4>10040) imply P3.B9_0_0_11==1)	***A[]*** ((Principal.leitura_das_entradas and Principal.I1_0_4==0 and T4_1_DN==1) imply Principal.B9_0_11==1)

A inserção da expressão proposicional Principal.leitura_das_entradas nas proposições propostas pelo presente trabalho deve-se ao fato de que deseja-se saber o valor das variáveis pertencentes às expressões proposicionais depois da execução do programa de controle do CP. Isto faz-se necessário, conforme apresenta Zoubek (2004), porque durante a execução do programa de controle determinados endereços poderão ter seus valores modificados mais de uma vez, em virtude, por exemplo, da utilização de elemento bobinas do tipo *latch* e *unlatch*. Portanto, desta forma, consideram-se somente os resultados do processamento do programa de controle efetivamente enviados aos dispositivos de saída do

CP. No trabalho de Zoubek (2004), este tipo de consideração é realizada estabelecendo-se a seguinte expressão proposicional: nome_do_modelo.testing. Nesta expressão, nome_do_modelo é dado por três diferentes nomes, P3, P8 e P9, pois, conforme foi apresentado, Zoubek (2004) utiliza-se de um modelo diferente para cada verificação realizada, e testing representa um estado no qual os valores resultantes do processamento do programa de controle já foram enviados aos endereços de saída e não serão, neste caso, modificados.

Conforme a Tabela 6.3, a verificação de número "1" a ser realizada no modelo global (do programa de controle do equipamento queimador de gás) é feita tendo-se por base uma proposição dada por: ***A[]*** not (Principal.leitura_das_entradas and Principal.B3_0_0==1 and Principal.B3_0_8==1), proposição esta que refere-se à verificação de uma propriedade de segurança, significando que para todos os possíveis caminhos existentes na estrutura combinatória de estados derivada do modelo global verificado não deverá haver, pelo menos, um estado simbólico onde (Principal.leitura_das_entradas e Principal.B3_0_0==1 e Principal.B3_0_8==1) ocorra simultaneamente, ressaltando-se que o nome do modelo principal é Principal, o qual se faz necessário, em virtude das variáveis em questão serem locais.

A verificação de número "1" não foi satisfeita, indicando que a expressão proposicional em questão é possível de ser obtida, de forma que o mesmo resultado foi obtido por Zoubek (2004). O presente trabalho, utilizando o histórico de estados simbólicos gerado pela ferramenta UPPAAL, o qual armazenou a seqüência de execução do modelo global que resultou na obtenção do estado simbólico onde ocorreu a expressão proposicional em questão, e da análise adotada por Zoubek (2004), constatou que a verificação não foi satisfeita, inicialmente, em virtude de três erros no programa de controle (modelado), os quais foram identificados nos *rungs* 0003 e 0004. Estes dois *rungs* são apresentados na Figura 6.15 com as correções que foram consideradas necessárias, as quais também foram sugeridas por Zoubek (2004). Observar que os círculos existentes na Figura 6.15 indicam os endereços corrigidos.

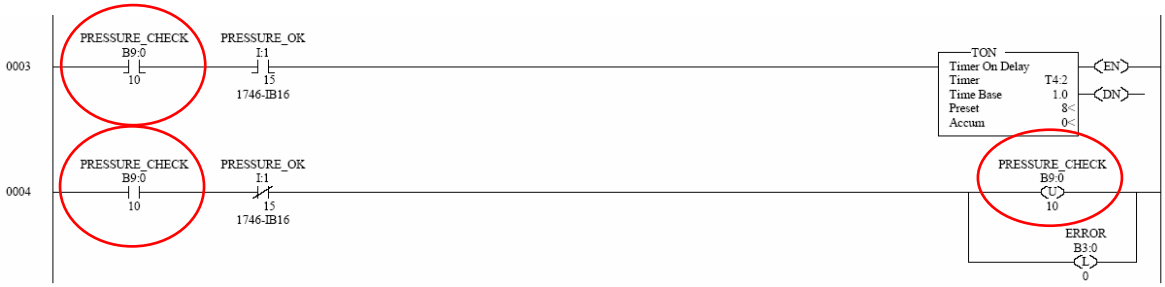


Figura 6.15 – Endereços corrigidos depois da verificação de número “1”

No *rung* 0003, o endereço do primeiro elemento contato do tipo NA deveria ser B9:0/10, e não B9:0/8, e no *rung* 0004, o elemento contato do tipo NA deveria ser B9:0/10, e não B9:0/8. Finalmente, ainda no *rung* 0004, o elemento bobina do tipo *unlatch* que foi associado ao endereço B9:0/8 deveria ter sido associado ao endereço B9:0/10 (observar na Figura 6.10 o programa de controle original). Ressalta-se que estas correções dependeram da compreensão da especificação do comportamento do programa de controle, de forma que tais correções foram sugeridas porque observou-se que o programa de controle, da forma como estava, era capaz de cumprir somente parcialmente as especificações de seu funcionamento. Ou seja, havia uma programação incorreta nos *rungs* 0003 e 0004, pois em tais *rungs* não era observada a confirmação da ativação das válvulas V1, V2 e V3, o que era um pré-requisito para se iniciar a supervisão da ativação do sensor de pressão (I:1/15).

Depois de executadas estas modificações no modelo global (proposto pelo presente trabalho), a verificação de número “1” foi novamente realizada, e continuou sendo não satisfeita, resultado este que divergiu do obtido por Zoubek (2004) ao aplicar as mesmas correções ao seu modelo. Devido a esta divergência, utilizando-se um CP da família SLC 500, o presente trabalho realizou um teste no programa de controle do equipamento queimador de gás (Figuras 6.10 e 6.11). Neste teste, utilizou-se a forma histórica e seqüencial pela qual a ferramenta UPPAAL realizou as alterações dos valores associados aos endereços de entrada modelados, e identificou-se que, mesmo com a realização das três correções em questão, no programa de controle em questão executado pelo CP foi possível obter B3:0/0==1 e B3:0/8==1, o que veio a confirmar o resultado obtido pelo presente trabalho.

A partir deste resultado, realizando-se uma análise mais profunda no modelo global, identificou-se um erro no *rung* 0013, o qual pode ter sido ocasionado por um engano cometido pelo programador ao inserir um elemento bobina do tipo *unlatch* no lugar de um

elemento bobina que deveria ser do tipo *latch*. A correção deste *rung* está sendo apresentada na Figura 6.16.

Figura 6.16 – Rung 0013 corrigido depois da realização da verificação de número “1”

Depois de esta correção ser aplicada ao modelo principal, a verificação de número “1” foi, finalmente, satisfeita. Com a correção do *rung* 0013, o modelo principal com as quatro correções realizadas pelo presente trabalho é ilustrado na Figura 6.17.

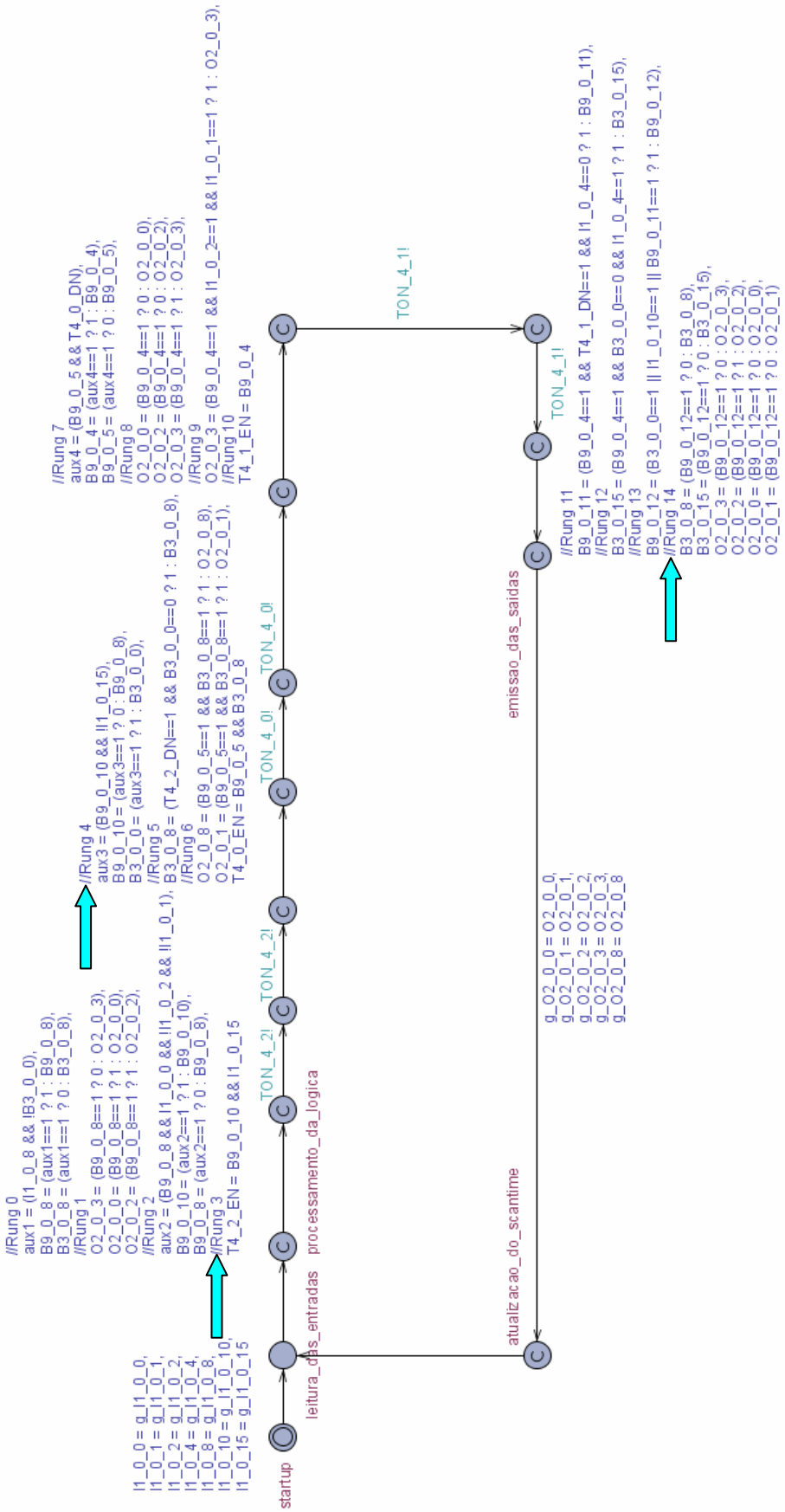


Figura 6.17 – Modelo principal resultante da aplicação de quatro correções

Para a verificação de número “2”, o presente trabalho utilizou-se do modelo principal da Figura 6.17 (com as quatro correções realizadas). Neste segundo caso, verifica-se a propriedade de alcançabilidade dada pela proposição ***E<>*** (Principal.leitura_das_entradas and Principal.B3_0_0==1 and Principal.B9_0_12==0). O resultado esperado para esta verificação de número “2” é que ela não seja satisfeita, ou seja, que a expressão proposicional (Principal.leitura_das_entradas and Principal.B3_0_0==1 and Principal.B9_0_12==0) não ocorra. Este resultado esperado foi o obtido pelo presente trabalho, isto é, não existe qualquer estado simbólico onde a expressão proposicional (Principal.leitura_das_entradas and Principal.B3_0_0==1 and Principal.B9_0_12==0) ocorre.

No trabalho de Zoubek (2004), tal verificação foi satisfeita, demonstrando que o estado simbólico em questão foi possível de ser obtido, indicando a existência de um erro no programa de controle modelado. Zoubek (2004) identifica este erro no *rung* 0013 do programa de controle modelado e aplica a correção ao seu modelo. Este erro identificado por Zoubek (2004) no *rung* 0013 por ocasião da verificação de número “2” é o mesmo que foi identificado pelo presente trabalho por ocasião da verificação de número “1”, cuja correção foi apresentada na Figura 6.16. Depois da correção aplicada ao modelo a ser verificado, o resultado obtido por Zoubek (2004) é que a verificação não foi satisfeita, ou seja, o resultado esperado.

Pode-se dizer que o erro corrigido pelo presente trabalho no *rung* 0013, por ocasião da verificação de número “1”, foi fundamental para se determinar o resultado obtido na verificação de número “2”. Analisando-se este resultado de Zoubek (2004), o presente trabalho realizou um teste onde foi executada a verificação de número “2” sem a correção do *rung* 0013 que havia sido realizada por ocasião da verificação de número “1”, e o resultado obtido foi o mesmo de Zoubek (2004). Desta forma, é possível dizer que, em virtude do procedimento de eliminação de *rungs* ou de algum erro cometido durante o processo de modelagem devido à complexidade dos modelos, o resultado da verificação de número “1” obtido por Zoubek (2004), o qual não correspondeu ao resultado esperado, segundo o teste realizado com o CP da família SLC 500, teve influência no resultado da verificação de número “2”.

Antes de iniciar a dicussão sobre a verificação de número “3”, faz-se necessária uma

observação quanto à proposição de lógica temporal apresentada por Zoubek (2004): conforme é apresentado no *rung* 0010 da Figura 6.11, o temporizador T4:1 ativará o bit de *Timer Done Bit* (T4:1/DN) somente depois que o endereço B9:0/4 permanecer ativado por 10s. Ao término deste período, se a chama não for detectada (I:1.0/4==0), será gerado um erro (B9:0/11=1). Caso contrário, isto é, se a chama for detectada (I:1.0/4==1) ao término deste período, será gerado um sinal de queimador ligado (B3:0/15=1).

Zoubek (2004) poderia ter utilizado a variável T4_1_0_DN (que modela o bit de *Timer Done Bit* (T4:1/DN)) para o estabelecimento da proposição da verificação de número "3", mas optou por utilizar um recurso no qual associou uma variável do tipo relógio c_B9_0_0_4 à variável B9_0_0_4 (do tipo booleana), medindo, dessa forma, o tempo em que B9_0_0_4 permanece ativada. Em virtude da utilização de uma variável do tipo relógio, Zoubek (2004) estabeleceu uma proposição utilizando a lógica temporal TCTL (Timed-Computation Tree Logic). Ressalta-se que, de acordo com o modelo da Figura 6.8 (cujo nome é P3), quando o valor de c_B9_0_0_4 for maior que "10040", o valor da variável T4_1_0_DN será "1", o que viabilizaria a utilização da variável T4_1_0_DN por parte de Zoubek (2004).

No presente trabalho, a proposição estabelecida na versão simplificada da lógica temporal CTL para a verificação de número "3" utilizou o bit *Timer Done Bit*, e é dada pela seguinte expressão: ***A[]*** ((Principal.leitura_das_entradas and Principal.I1_0_4==0 and T4_1_DN==1) imply Principal.B9_0_11==1). Tal verificação foi satisfeita, e o mesmo resultado foi obtido por Zoubek (2004), o que era o resultado desejado, antendendo-se às especificações da Tabela 6.3.

CAPÍTULO 7

7. CONCLUSÕES

O presente trabalho apresentou um procedimento para a modelagem de MEFEs, tendo-se em vista, a criação de tais modelos a partir de programas de controle desenvolvidos utilizando-se a linguagem de programação chamada LD.

A partir de um determinado programa de controle de interesse, modelos em MEFEs puderam ser gerados, de forma que tais modelos contemplaram em sua estrutura o mapeamento de todos os *rungs* pertencentes ao programa de controle em questão. Para a construção destes modelos, também foi considerado o ciclo operacional do CP, conforme foi apresentado no Capítulo 2. Uma vez gerados os modelos em MEFEs, tornou-se possível a utilização de uma ferramenta computacional de verificação, própria para estabelecer se os modelos verificados satisfariam (ou não) determinadas proposições estabelecidas utilizando-se a versão simplificada da lógica temporal CTL.

O processo de verificação realizado teve o objetivo de identificar se um determinado "estado específico" do programa modelado poderia ser (ou não) alcançado. Se este estado específico fosse, por exemplo, um estado indesejado do programa de controle modelado, isto impactaria na não conformidade de tal programa em relação às especificações comportamentais estipuladas na forma de proposição em lógica temporal, indicando haver, por conseguinte, um erro no programa de controle modelado. Com o recurso do histórico de estados simbólicos gerado pela ferramenta de verificação UPPAAL, foi possível realizar uma análise do modelo global verificado, buscando-se, a partir deste cenário, alternativas para se corrigir o problema.

Diversos outros estudos de casos que não puderam constar neste trabalho foram realizados adotando-se o procedimento proposto, utilizando-se, para isso, programas de controle existentes e executados por CPs da família SLC 500. A partir dos modelos gerados, erros puderam ser identificados e, posteriormente, corrigidos, tanto nos modelos, quanto nos

programas de controle originais. Depois da correção dos erros que foram identificados, testes foram conduzidos nos programas de controle corrigidos executados pelos CPs, e não se alcançaram os "estados específicos" considerados indesejados.

7.1 OBSERVAÇÕES COMPLEMENTARES

Analisando-se o conteúdo apresentado nos capítulos anteriores, é possível destacar dois aspectos importantes quando o presente trabalho é comparado ao que foi proposto por Moon (1994):

- o presente trabalho considerou, nas questões de modelagem, elementos contadores do tipo CTU e elementos temporizadores do tipo TON, permitindo-se que programas de controle em LD que contemplem tais elementos possam ser considerados para serem modelados e, posteriormente, verificados;
- considerou o tempo de varredura do CP.

Nesta mesma direção, alguns aspectos considerados positivos podem ser destacados quando o presente trabalho é comparado ao que foi proposto por Zoubek (2004):

- ausência da utilização das variáveis do tipo relógio, o que contribui para a redução da complexidade no desenvolvimento e verificação dos modelos;
- a modelagem proposta pelo presente trabalho, em virtude do item anterior, poderá ser mais facilmente compreendida por programadores de CPs que desejarem realizar a verificação de programas de controle desenvolvidos em LD;
- não há eliminação de *rungs* do programa de controle modelado, fato este que elimina as incertezas (neste aspecto) quanto aos resultados obtidos em cada processo de verificação;
- faz-se necessária somente a criação de um modelo global, gerado a partir dos programas de controle de interesse, de forma que, as diversas proposições em lógica temporal que se fizerem necessárias poderão ser estipuladas para a verificação deste único modelo global;
- apresenta-se um modelo para o elemento contador do tipo CTU e;
- o ciclo de operação do CP é tratado no modelo principal de uma forma

sistemática, facilitando o processo de análises dos resultados obtidos em virtude da verificação realizada.

Portanto, uma importante contribuição do presente trabalho é o estabelecimento de um procedimento de modelagem de MEFEs que são geradas a partir de programas de controle desenvolvidos em LD, de forma que tal procedimento pode ser utilizado por programadores de CPs de forma sistemática, permitindo-os realizar a verificação de programas de controle já existentes. A utilização da ferramenta de verificação UPPAAL contribui fortemente para simplificar o processo de desenvolvimento dos modelos, ressaltando-se o fato de que o procedimento proposto pode ser aplicado a outras ferramentas computacionais que permitam a modelagem e a verificação deste tipo de máquinas de estado.

7.2 TRABALHOS FUTUROS

Em relação aos trabalhos futuros que podem ser realizados a partir do presente trabalho, seguem algumas propostas a serem consideradas.

Há uma recomendação das normas IEC 61511, IEC 61508 e ANSI/ISA-84.00.01 na qual as linguagens de programação LD ou FBD devem ser as linguagens adotadas para o desenvolvimento de blocos de função que são relacionados à segurança[26]. Tais blocos de função são executados por uma família de CPs que também são relacionados à segurança (*Programmable Electronic Safety-Related Systems*), CPs estes que gerenciam os eventos associados a dispositivos de comando, de detecção, de monitoração e de atuação que exercem funções de segurança no processo onde estão inseridos.

[26] Existe uma associação chamada *PLCopen Association* que tem tratado do desenvolvimento e da padronização destes blocos de função. Em alguns documentos obtidos junto a esta associação observou-se que a técnica de MEF tem sido utilizada para a realização de testes em modelos que, posteriormente, são convertidos em blocos de função relacionados à segurança, os quais são escritos em LD ou FBD.

Como exemplos destes dispositvos descritos podem ser citados: botões de emergência, botões do tipo bi-manual, botões do tipo *dead-man switch*, detectores de gás e de chama, barreiras óticas de proteção, monitoração de nível de reservatórios, iluminação de emergência, válvulas com funções críticas relacionadas à segurança etc. Neste sentido, um futuro trabalho poderia ser conduzido utilizando-se as contribuições que foram apresentadas nesta dissertação como forma efetiva para se verificar estes blocos de função relacionados à segurança.

Outra possível proposta poderia estar relacionada ao aprimoramento do presente trabalho, no sentido de fazer com que o valor do tempo de execução de cada *rung* pertencente ao programa de controle modelado seja considerado para efeito de incremento do valor do tempo de varredura. Desta forma, o valor do tempo necessário para a execução de cada *rung* seria adicionado ao valor do tempo de varredura, não sendo mais necessário utilizar valores críticos desta medida de tempo para tratar o ciclo de operação do CP no modelo principal em MEFE.

Finalmente, outro aspecto que pode ser explorado é o desenvolvimento de um software para a realização da conversão de programas de controle desenvolvidos em LD (de diversos fabricantes de CPs) para MEFEs, de forma automática. Na ferramenta UPPAAL, os modelos desenvolvidos em MEFEs são armazenados no computador em formato XML, fato este que pode ser considerado como algo que venha a colaborar com a dimunição da complexidade para se desenvolver tal software de conversão. Este software contribuiria para que os eventuais erros decorrentes da execução manual do procedimento apresentado pelo presente trabalho possam ser evitados.

ANEXO A - DADOS COMPLEMENTARES SOBRE A LINGUAGEM LD

As regras de construção de programas de controle em linguagem LD, de acordo com Miyagi (1996), são:

- Os elementos contatos e os elementos bobinas devem ficar na intersecção das linhas e colunas de uma matriz e os elementos bobinas devem ocupar somente a última coluna à direita;
- As linhas verticais das extremidades à direita e à esquerda chamam-se linhas mãe; na da esquerda são conectados os elementos contatos e na da direita são conectadas somente os elementos bobinas;
- Os elementos contatos e os elementos bobinas são conectados através de linhas horizontais e não é permitida mais de uma linha em uma única "linha" da matriz; as linhas horizontais são interligadas através de linhas verticais e não se permitem várias linhas em uma única coluna; a intersecção entre uma linha horizontal e uma linha vertical pode ser uma conexão ou, apenas, um cruzamento sem conexão.

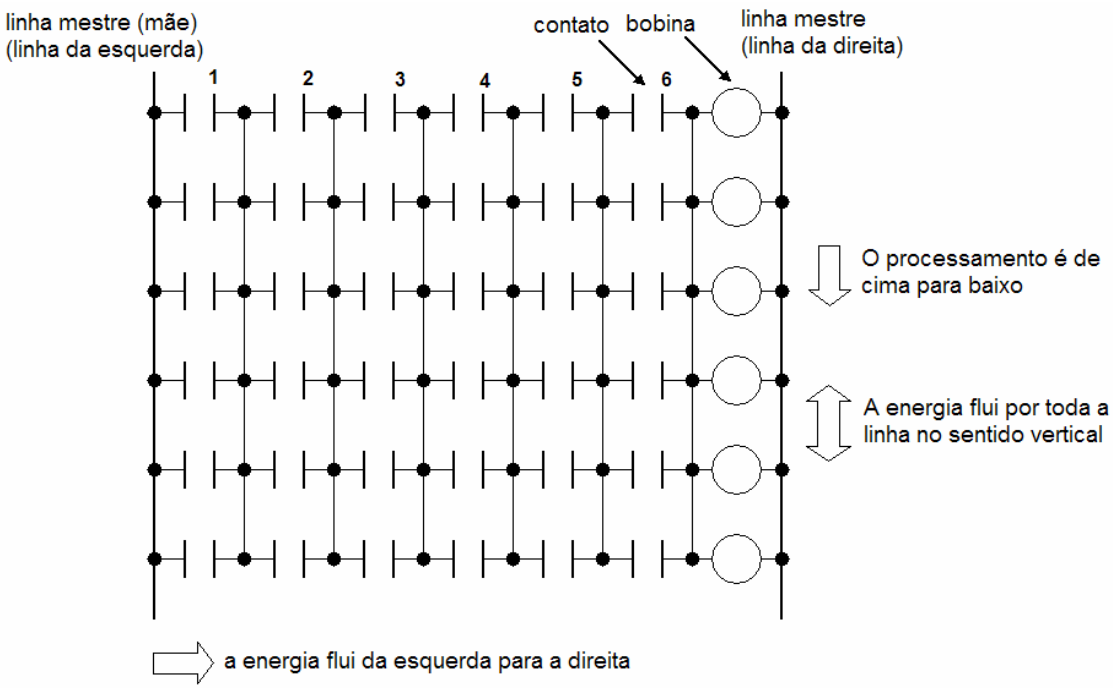


Figura A.1 – Regras de construção de programa de controle em linguagem LD

A energia que flui na matriz apresentada na Figura A.1 é uma energia hipotética e está relacionada à varredura do programa de controle, isto é, à seqüência de execução do programa de controle que é realizado pelo CP. As regras de varredura e processamento do programa de controle apresentadas por Miyagi (1996) e que também podem ser encontradas resumidamente em Chmiel; Hrynkiewicz e Muszyński (2002) são:

- A energia (varredura) flui através das linhas horizontais da esqueda para a direita, e de acordo com os estados aberto/fechado dos elementos contatos, executando-se a função lógica AND;
- A energia (varredura) flui através das linhas verticais de cima para baixo, executando-se a função lógica OR dos estados das linhas horizontais que estão à sua esquerda, transmitindo o resultado para a(s) linha(s) horizontal(is) à sua direita (isto é, se pelo menos uma das linhas à sua esquerda tiver um resultado lógico verdadeiro, o sinal transmitido à sua direita será verdadeiro);
- O acionamento (análogo à energização elétrica) dos elementos bobinas depende da existência de fluxo de energia da linha horizontal à sua esquerda;

- O processamento do programa de controle é realizado de cima para baixo.

ANEXO B – ESTADOS COMPROMISSADOS

Os modelos principais e modelos auxiliares das entradas que foram apresentados no procedimento de modelagem do Capítulo 5 foram desenvolvidos sem utilizar-se estados compromissados. Porém, observou-se que na utilização da ferramenta UPPAAL, para se evitar a não finalização do processo de verificação de determinados modelos, necessita-se designar determinados estados do modelo principal e dos modelos auxiliares das entradas como sendo compromissados. Desta forma, o presente trabaho propõe uma maneira pela qual os modelos apresentados no Capítulo 5 podem ser utilizados para verificação com a ferramenta UPPAAL.

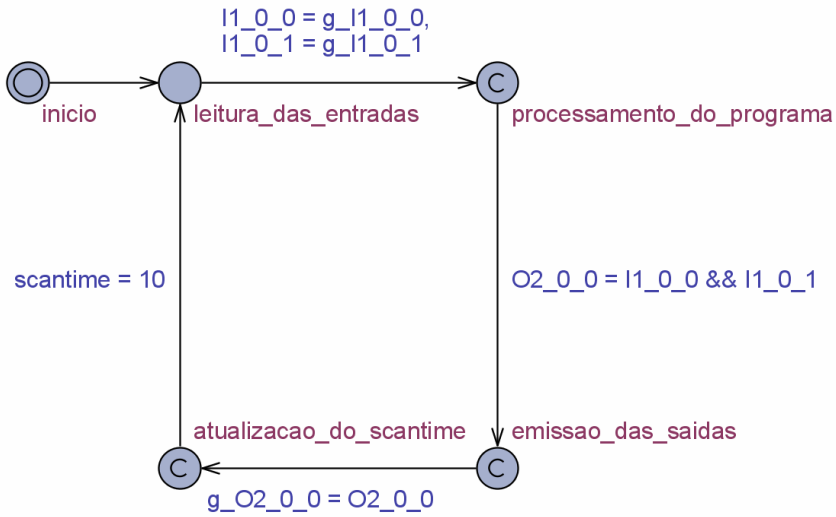
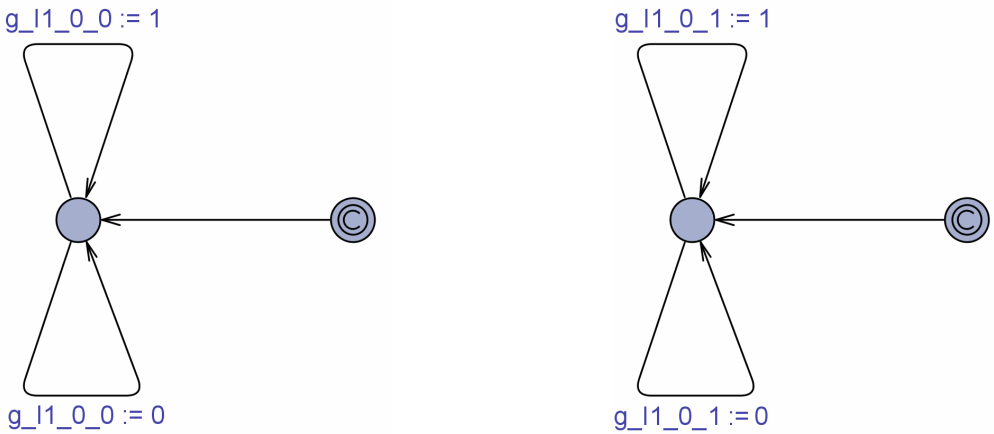


Figura B.1 – Modelo da Figura 5.4



(a) Modelo auxiliar de entrada de I:1.0/0 (b) Modelo auxiliar de entrada de I:1.0/1

Figura B.2 – Modelos da Figura 5.5

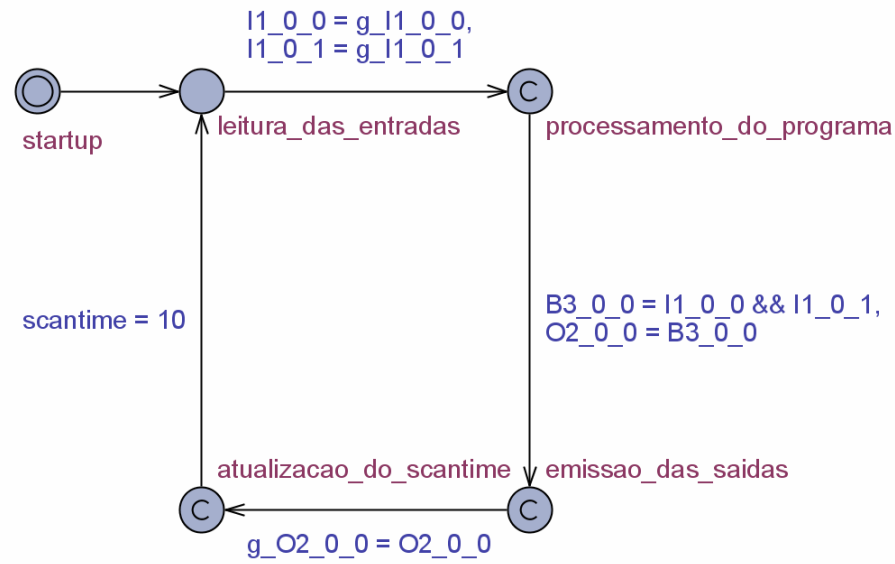


Figura B.3 – Modelo da Figura 5.9

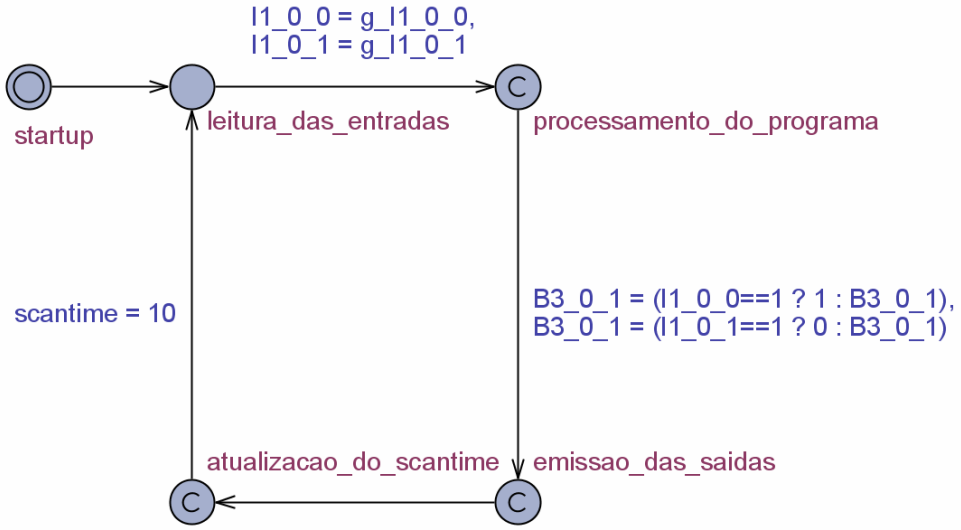


Figura B.4 – Modelo da Figura 5.11

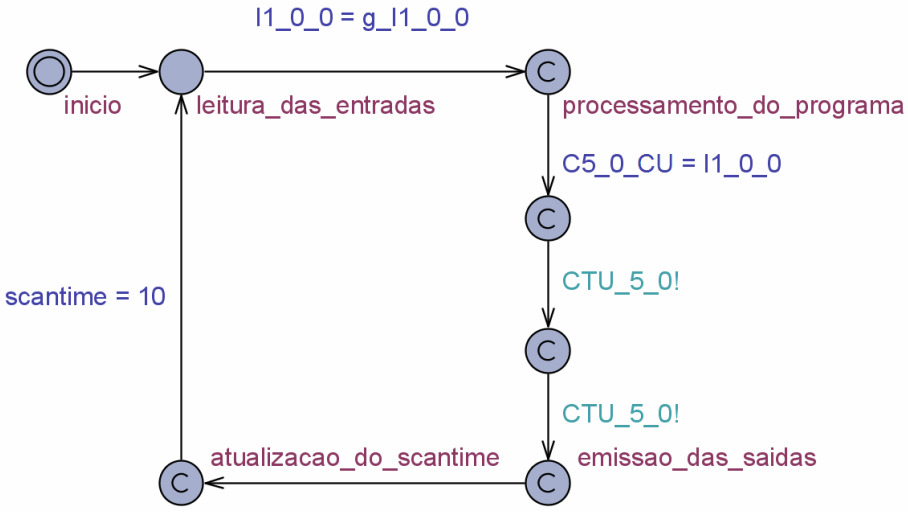


Figura B.5 – Modelo da Figura 5.12

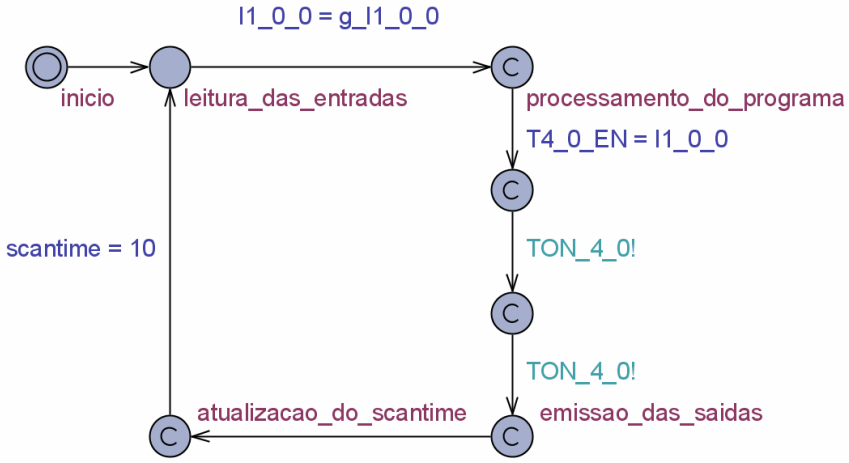


Figura B.6 – Modelo da Figura 5.13

ANEXO C – AUTÔMATOS TEMPORIZADOS

Um autômato temporizado é um modelo gerado a partir de um autômato finito determinístico (AFD), sendo estendido com um conjunto finito de variáveis chamadas de variáveis do tipo relógio, cujos valores pertencem ao conjunto dos números reais não-negativos. Tais variáveis são utilizadas para se medir o progresso do tempo entre os possíveis estados que este autômato temporizado poderá ativar[27] (ALUR; DILL, 1990; WOŹNA; ZBRZEZNY; PENCZEK, 2002; BURNS, 2003; QUOTTRUP; BAK; ZAMANABADI, 2004; RODRIGUEZ-NAVAS; PROENZA; HANSSON, 2005).

Na Figura C.1 é apresentado um exemplo de autômato temporizado que foi adaptado de Alur, Dill (1996).

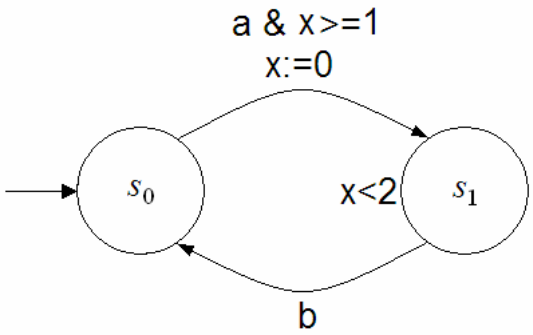


Figura C.1 – Exemplo de um autômato temporizado com uma variável do tipo relógio.

No autômato temporizado apresentado na Figura C.1, os nós s_0 e s_1 são chamados de estados e os arcos orientados que conectam os estados são chamados de transições. Durante a execução de uma transição de um autômato temporizado o tempo que está sendo medido pelas variáveis do tipo relógio não é incrementado. Quando um estado torna-se

[27] Em um autômato temporizado (LAROUSSINIE; LARSEN; WEISE, 1995) todas as suas variáveis do tipo relógio (caso haja duas ou mais variáveis) têm sua progressão ocorrendo de forma síncrona.

ativo é, então, possível que o tempo seja incrementado, sendo que esta condição é restrita quando são utilizados os estados compromissados, onde o incremento de tempo nestes estados não é permitido.

Sobre cada transição de um autômato temporizado é possível associar-se restrições à quaisquer variáveis do tipo relógio encontradas neste autômato, o que significa dizer que a execução de uma determinada transição pode estar condicionada à satisfação de determinadas condições impostas pelas restrições às variáveis do tipo relógio. Tais restrições em um autômato temporizado recebem o nome de *guarda* (BORDBAR; OKANO, 2004). Na Figura C.1, a restrição de *guarda* imposta à execução da transição entre s_0 e s_1 é dada pela expressão x>1.

As variáveis do tipo relógio podem sofrer um *reset* durante a execução de uma determinada transição entre estados, como pode ser visto na Figura C.1 com a variável x, na execução da transição entre os estados s_0 e s_1. Em qualquer instante, a leitura do valor das variáveis do tipo relógio indica o valor de tempo incrementado nestas variáveis, desde a última vez em que sofreram um *reset*.

Também é possível haver restrições às variáveis do tipo relógio de um autômato temporizado nos próprios estados deste autômato. Tais restrições são conhecidas por restrições invariantes e têm a função de delimitar o tempo de ativação de um determinado estado de um autômato temporizado, sob a condição de que se o tempo de ativação deste estado exceder a restrição imposta não será possível que este estado seja deixado. Na Figura C.1 observa-se uma restrição invariante no estado s_1, onde x deve ser menor que "2" para que seja possível a transição de s_1 para s_0.

Tomando-se como exemplo o autômato temporizado da Figura C.1, é possível obter do mesmo as seguintes informações (ALUR; DILL, 1996):

- Há apenas uma variável do tipo relógio, a qual foi definida pela letra x;
- As letras a e b são símbolos de um alfabeto de entrada que são reconhecidos pelo autômato temporizado. No exemplo da Figura C.1, depois de reconhecer o símbolo a e se o valor do tempo medido pela variável x for maior ou igual a "1", o autômato temporizado tem um tempo menor que "2" unidades de tempo para ler o símbolo b, significando que o atraso máximo

entre o reconhecimento dos símbolos a e b deve ser menor que “2” unidades de tempo;

- O estado inicial é dado pelo nó com um arco orientado sem origem que aponta para si (nó) e, no caso da Figura C.1, tal estado é dado por s_0;
- O estado inicial (s_0) não possui qualquer restrição invariante, o que significa dizer que o autômato temporizado poderia permanecer com este estado ativado por tempo indeterminado. Porém, a transição de s_0 para s_1 possui uma restrição de guarda que determina que s_0 só pode ser deixado se x>=1;
- Quando o sistema evolui do estado inicial s_0 para o estado s_1, a variável x sofre um *reset*. Enquanto o autômato temporizado estiver com o estado s_1 ativado, a variável x indicará o valor do tempo decorrido desde a execução da última transição;
- Uma vez estando em s_1, para que seja possível para o autômato temporizado ir para s_0 é necessário que o valor de x seja menor que “2”, restrição esta que é imposta pela restrição invariante dada por x<2;

Conforme apresentado por Alur; Dill (1996) um autômato temporizado é formado por uma tupla do tipo $\mathcal{A}=(\sum, S, S_0, C, E)$.

Onde:

$\sum$ representa um alfabeto de entrada (conjunto finito e não-vazio de símbolos).

S representa um conjunto finito de estados.

S_0 representa o estado inicial, onde $S_0 \subseteq S$.

C representa um conjunto finito de variáveis do tipo relógio.

E representa um conjunto de transições onde $E \subseteq S \times S \times \sum \times 2^C \times \Phi(C)$. Uma transição do tipo (s, s', a, λ, δ) representa uma transição que é executada a partir do estado s para o estado s', através da leitura do símbolo a e da satisfação de δ, que é um conjunto de restrições impostas às variáveis de C. $\lambda \subseteq C$ é um conjunto de variáveis do tipo relógio que sofrem *reset* com a execução da transição.

REFERÊNCIAS BIBLIOGRÁFICAS

AIGUIER, M.; GASTON, C.; LE GALL, P.; LONGUET, D.; TOUIL, A. **A Temporal Logic for Input Output Symbolic Transition Systems**. Proceedings of the 12th Asia-Pacific Software Engineering Conference APSEC'05, 2005.

ALUR, R.; DILL, D. **Automata for Modelling Real-Time Systems**. In Proc. 17th Int. Conf. on Automata, Languages and Programming (ICALP '90'), Vol. 443 of LNCS. pp. 322-335, Springer, 1990.

ALUR, R.; COURCOUBETIS, C.; DILL, D. **Model-Checking for Real-Time Systems**. In Proceedings of the 5th IEEE Symposium on Logic in Computer Science, pp. 414-425, IEEE Computer Society Press, 1990.

ALUR, R.; DILL, D. **A Theory of Timed Automata**. Theoretical Computer Science archive. Volume 126 , Issue 2, pages: 183 – 235, Elsevier Science Publishers Ltd., April, 1994.

ALUR, R.; DILL, D. **Automata Theoretic Verification of Real-Time Systems**. Formal Methods for Real-Time Computing, Trends in Software Series, John Wiley & Sons Publishers, pages: 55-82, 1996.

ANAND, M.; VESTAL, S.; DAJANI-BROWN, S.; LEE, I. **Formal Modeling and Analysis of the AFDX Frame Management Design**. 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, ISORC 2006, Gyeongju, South Korea, 2006.

ANDERSON, C.; RUNESON, P. **Verification and validation in industry - a qualitative survey on the state of practice**. Proceedings of the ISESE´02 International Symposium on Empirical Software Engineering, pp. 37-47, 2002.

BEHRMANN, G.; DAVID, A.; LARSEN, K. G. **A Tutorial on UPPAAL**. In Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04), LNCS, Vol. 3185/2004, Springer-Berlin, 2004.

BENGTSSON, J.; YI, W. **Timed Automata: Semantics, Algorithms and Tools.**

In Lecture Notes on Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.

BLOM, J.; HESSEL, A.; JONSSON, B.; PETTERSSON, P. **Specifying and Generating Test Cases Using Observer Automata.** Proceedings of the 4th International Workshop on Formal Approaches to Testing of Software (FATES'04), 2004.

BOLTON, W. **Programmable Logic Controllers**. Ed. Newnes, 4th Edition, 304 pages, 2006.

BONFE, M.; FANTUZZI, C. **Design and Verification of Industrial Logic Controllers with UML and Statecharts**. Proceedings of the 2003 IEEE Conference on Control Applications - CCA 2003, Vol. 2, pp. 1029-1034, 2003.

BOOTH, T. L. **Sequential Machines and Automata Theory**. Ed. John Wiley & Sons Inc, 592 pages, 1967.

BORDBAR, B.; OKANO, K. **Testing Deadlock-freeness in Real-time Systems: A Formal Approach**. 4th International Workshop on Formal Approaches to Testing of Software - FATES 2004, LNCS 3395, pp. 95-109, September, 2004.

BROWNE, M. C.; CLARKE, E. M.; DILL, D. L.; MISHRA, B. **Automatic Verification of Sequential Circuits Using Temporal Logic**. IEEE Transaction on Computers, Vol. C-35, N. 12, December, 1986.

BURNS, A. **How to Verify a Safe Real-Time System: The Application of Model Checking and Timed Automata to the Production Cell Case Study**. Journal of Real-Time Systems. Kluwer Academic Publishers. Vol. 24, pp. 135-151, 2003.

CASSANDRAS, C. **Discrete Event Systems - Modeling and Performance Analysis** - Aksen Associates Incorporated Publishers, New Jersey, 1993.

CHENG, K. T.; KRISHNAKUMAR, A. S. **Automatic Functional Test Generation Using the Extended Finite State Machine Model**. 30th ACM/IEEE Conference on Design Automation, pp. 86-91, June, 1993.

CHIRN, J-L.; MCFARLANE, D. C. **Petri Nets based design of ladder logic diagrams**. Proceedings of Control 2000, Cambridge, UK, 2000.

CHMIEL, M.; HRYNKIEWICZ, E.; MUSZYŃSKI, M. **The way of ladder diagram analysis for small compact programmable controller**. Proc. of the 6th KORUS-2002 - Russian-Korean International Symposium on Science and Technology, pp. 169-173, 2002.

CLARKE, E. M.; EMERSON, E. A. **Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic**. In Logic of Programs: Workshop, Yorktown Heights, NY, May 1981, LNCS 131. Springer, 1981.

CLARKE, E. M.; EMERSON, E. A.; SISTLA, A. P. **Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification**. ACM Transactions on Programming Languages and Systems, Vol. 8, N.2, pp. 244-263, April, 1986.

CLARKE, E. M.; LONG, D. E.; MCMILLAN, K. L. **A language for compositional specification and verification of finite state hardware controllers**. Proceedings of the IEEE, 79 (9): 1293-1292, 1991.

CLARKE, E. M.; GRUMBERG, O.; PELED, D. A. **Model Checking**. The MIT Press, Cabridge, Massachusetts, 1999.

CLARKE, E. M.; GRUMBERG, O.; JHA, S.; LU, Y.; VEITH, H. **Progress on the State Explosion Problem in Model Checking**. In Lecture Notes in Computer Science, Berlin. Springer-Verlag, 2000.

COHEN, D. I. A. **Introduction to computer theory**. John Wiley & Sons, Inc., 838 pages, 2nd Edition, USA, 1990.

CORTÉS, L. A.; ELES, P.; PENG, Z. **Formal Coverification of Embedded Systems using Model Checking**. Proceedings of the 26th Euromicro Conference, Vol. 1, pp. 106-113, Maastricht, Netherlands, 2000.

DAS, B.; SARKAR, D.; CHATTOPADHYAY, S. **Model Checking on State Transition Diagram**. Proceedings of the ASP-DAC'04 - Asia and South Pacific Design Automation Conference, pp. 412-417, 2004.

DENNING, P. J.; DENNIS, J. B.; QUALITZ, J. E. **Machines, Languages and Computation**. Prentice Hall Ed., Englewood Cliffs, N. J., 1978.

DE SMET, O.; COUFFIN, S.; ROSSI, O.; CANET, G.; LESAGE, J.-J.; SCHNOEBELEN, P.; PAPINI, H. **Safe Programming of PLC Using Formal Verification Methods**. Proceedings of 4th International PLCOpen Conference on Industrial Control Programming (ICP '2000), Utrecht, Netherlands, October, 2000.

DE SMET, O.; ROSSI, O. **Verification of a Controller for a Flexible Manufacturing Line written in Ladder Diagram via Model Checking**. Proceedings of the American Control Conference, Anchorage, May, 2002.

DEVADAS, S.; KEUTZER, K.; KRISHNAKUMAR, A. S. **Design verification and reachability analysis using algebraic manipulation**. Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors ICCD'91, pp. 250-258, October, 1991.

ERICKSON, K. T. **Programmable logic controllers – the workhorse of factory automation keeps things on track**. IEEE Potentials (Publications). 15 (1): 14-17, 1996.

GIANNAKOPOULOU, D.; PASAREANU, C. **Towards a Compositional SPIN**. Proc. SPIN' 06: 13th International Workshop on Model Checking of Software, Vienna, Austria, 2006.

GODSKESEN, J. C.; NIELSEN, B.; SKOU, A. **Connectivity Testing through Model-Checking**. International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), Madrid, Spain, September, 2004.

GROOVER, M. P. **Automation, Production Systems, and Computer-Integrated Manufacturing**. 2nd ed., 856 pages, Prentice Hall Ed., New York, 2000.

GUERRA, P. A. C. **Uma Abordagem Arquitetural para Tolerância a Falhas em Sistemas de Software Baseados em Componentes**. Tese (Doutorado) - Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2004.

HACKWORTH, J. R.; HACKWORTH, F. D. **Programmable Logic Controllers: Programming Methods and Applications**. Prentice Hall, 320 pages, 2003.

HELLGREN, A.; FABIAN, M.; LENNARTSON, B. **Modular Implementation of Discrete Event Systems as Sequential Function Charts Applied to an Assembly Cell**. Proceedings of the 2001 IEEE International Conference on control Applications, pp. 453-458, México City, México, 2001.

HESSEL, A.; PETTERSSON, P. **A Test Case Generation Algorithm for Real-Time Systems**. In proceedings of the 4th International Conference on Quality Software (QSIC'04), pp. 268-273, Hans-Dieter Ehrich and Klaus-Dieter Schewe (eds.), IEEE Computer Society, 2004.

HESSEL, A. **Model-Based Test Case Generation for Real-Time Systems**. PhD Dissertation, Department of Computer Systems, Uppsala University, Sweden, 2007.

HOLZMANN, G. J. **The Model Checker SPIN**. IEEE Transaction on Software Engineering. Vol. 23, No. 5, May, 1997.

HONG, H. S.; LEE, I.; SOKOLSKY, O.; CHA, S. D. **Automatic Test Generation from Statecharts Using Model Checking**. Proceedings of the 1st International Workshop on Formal Approaches to Testing of Software (FATES '01), pages 15-30, Aalborg, Denmark, August, 2001.

HONG, H. S.; LEE, I.; SOKOLSKY, O.; URAL, H. **A Temporal Logic Based Theory of Test Coverage and Generation**. Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS 2002, Lecture Notes in Computer Science, Vol. 2280, pages 327-341, 2002.

HOPCROFT, J. E.; ULLMAN, J. D.; MOTWANI, R. **Introduction to Automata Theory, Languages, and Computation**. Addison-Wesley Ed., Second Edition, 2001.

HOPPE, T.; MESEGUER, P. **VVT Terminology: A Proposal**. IEEE Expert: Intelligent Systems and Their Applications. Volume 8, Issue 3, pages: 48–55, June, 1993.

HUANG, S-Y.; LIU, C-J. **A Low-Power Architecture for Extended Finite State Machines Using Input Gating**. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 87 (12): 3109-3115, 2004.

IEC 61508 - **Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems**, 1998.

KURFESS, T. R. **Robotics and Automation Handbook**. CRC Press, 608 pages, USA, 2005.

LAROUSSINIE, F.; LARSEN, K. G.; WEISE, C. **From Timed Automata to Logic - and Back**. In Proc. of the 20th Int. Symp. on Math. Found. Comp. Sci. (MFCS'95), Vol. 969 of LNCS, pp. 529−539, Springer-Verlag, 1995.

LARSEN, K. G.; PETTERSSON, P.; YI, W. **Uppaal in a Nutshell**. In International Journal of Software Tools for Technology Transfer 1(1+2), Springer, 1997.

LARSSON, F. **Efficient Implementation of Model-Checkers for Networks of Timed Automata**. Licentiate Thesis 2000-003, Department of Information Technology, Uppsala University, 2000.

LEE, J–S.; HSU, P–L. **An Improved Evaluation of Ladder Logic Diagrams and Petri Nets for the Sequence Controller Design in Manufacturing Systems**. The International Journal of Advanced Manufacturing Technology, Vol. 24, Numbers 3-4, pages 279-287, Springer-London, August, 2004.

LEE, G. B.; ZANDONG, H.; LEE, J. S. **Automatic generation of ladder diagram with control Petri net**. Journal of Intelligent Manufacturing, Vol. 15, Number 2, pp. 245-252, 2004.

LINDAHL, M.; PETTERSSON, P.; YI, W. **Formal Design and Analysis of a Gear Controller**. Lecture Notes In Computer Science. Vol. 1384 - Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems, pp. 281–297, Springer-Verlag, 1998.

LUCAS, M. R.; TILBURY, D. M. **Comparing Industrial Logic Design Methods**

Used in the Automotive Industry. IEEE International Conference on Systems, Man and Cybernetics. Vol. 1, pp. 530-537, 2003.

LUCAS, M. R.; TILBURY, D. M. **Methods of Measuring the Size and Complexity of PLC Programs in Different Logic Control Design Methodologies**. International Journal of Advanced Manufacturing Technology, Vol. 26(5–6): 436–447, September, 2005.

MATOUŠEK, P. **Symbolic Data Structures for Parametric Verification**. PhD Thesis, Faculty of Information Technology, Brno University of Technology, 2005.

MATSUSAKI, C. T. M. **Modelagem de sistemas de controle distribuídos e colaborativos de sistemas produtivos**. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2004.

MENEZES, P. B. **Linguagens Formais e Autômatos**. Editora Sagra Luzzatto, 5ª Edição, 232 Páginas, Porto Alegre, 2005.

MIYAGI, P. E. **Controle Programável - Fundamentos do controle de sistemas a eventos discretos**. Ed. Edgard Blücher Ltda, São Paulo, 1996.

MCMILLAN, K. L. **Getting Started with SMV**. Cadence Berkeley Labs, 91 pages, Berkeley, USA, 1999.

MOON, I. **Modeling programmable logic controllers for logic verification**. IEEE Control Systems Magazine, 14(2), pp. 53-59, 1994.

NAKAMOTO, F. Y. **Sistematização do projeto do controle de sistemas produtivos**. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2002.

PARR, E. A. **Programmable Controllers: An Engineer's Guide**. Newnes Publishers, 448 pages, Great Britain, 2003.

PENG, S. S.; ZHOU, M. C. **Ladder diagram and Petri-Net-based discrete-event control design methods**. IEEE Transaction on Systems, Man, and Cybernetics - Part C: Applications and Reviews, 34 (4): 523-531, November, 2004.

PETTERSSON, P. **Modeling and verification of real-time systems using timed automata: theory and practice**. PhD Dissertation, Department of Computer Systems, Uppsala University, Sweden, 1999.

QUOTTRUP, M. M.; BAK, T.; ZAMANABADI, R. I. **Multi-Robot planning: a timed automata approach**. Proceedings of the 2004 IEEE International Conference on Robotics and Automation. New Orleans, USA, April, 2004.

ROBINSON-MALLETT, C.; LIGGESMEYER, P.; MÜCKE, T.; GOLTZ, U. **Generating optimal distinguishing sequences with a model checker**. ACM SIGSOFT Software Engineering Notes. SESSION: Advances in Model-Based Testing (A-MOST 2005), 30 (4), July, 2005.

ROBINSON-MALLETT, C.; LIGGESMEYER, P.; MÜCKE, T.; GOLTZ, U. **Extended state identification and verification using a model checker**. ScienceDirect: Advances in Model-Based Testing - Information and Software Technology, 48 (10): 981-992, October, 2006.

RODRIGUEZ-NAVAS, G.; PROENZA, J.; HANSSON, H. **Using UPPAAL to model and verify a clock synchronization protocol for the Controller Area Network**. 10th IEEE Conference on Emerging Technologies and Factory Automation - ETFA'2005, Vol. 2, 2005.

ROSSI, O.; SCHNOEBELEN, P. **Formal modeling of timed function blocks for the automatic verification of Ladder Diagram programs**. Proc. 4th Int. Conf. Automation of Mixed Processes: Hybrid Dynamic Systems (ADPM'2000), Dortmund, Germany, September, 2000.

SANTOS FILHO, D.J. **Controle de Sistemas Antropocêntricos de Produção Baseado em Redes de Petri Interpretadas**. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 1998.

SANTOS FILHO, D.J. **Aspectos do Projeto de Sistemas Produtivos**. Tese (Livre docência) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2000.

SIEMENS - **System Software for S7-300/400 System and Standard Functions - Volume 1/2**, Reference Manual, 2006.

SMITH, D. J.; SIMPSON, K. G. L. **Functional Safety – A straightfoward Guide to Applying IEC 61508 and Related Standards**. Elsevier Butterworth-Heinemann Ed., United Kingdom, 2005.

SUESUT, T.; INBAN, P.; NILAS, P.; RERNGREUN, P.; GULPHANICH, S. **Interpretation Petri net model to IEC 1131-3: LD for programmable logic controller**. Proceedings of the 2004 IEEE Conference on Robotics, Automation and Mechatronics. Vol. 2, pp. 1107-1111, Singapore, 2004.

ŠUSTA, R. **Verification of PLC Programs**. Doctoral Dissertation, Department of Cybernetics of the Faculty of Electrotechical Engineering. Prague, Czech Republic, 2003.

UZAM, M.; JONES, A. H.; AJLOUNI, N. **Conversion of Petri Net Controllers for Manufacturing Systems into Ladder Logic Diagrams**. Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation - ETFA, Vol. 2, pp. 649-655, Kauai, HI, USA, 1996.

UZAM, M.; JONES, A. H. **Discrete Event Control System Design Using Automation Petri Nets and their Ladder Diagram Implementation**. International Journal of Advanced Manufacturing Technology 14 (10): 716-728, Springer-Verlag, 1998.

VARDI, M.; WOLPER, P. **An Automata-Theoretic Approach to Automatic Program Verification**. In Proceedings of the 1st IEEE Symposium on Logic in Computer Science, pp. 322-331, 1986.

VILLANI, E. **Modelagem e Análise de Sistemas Supervisórios Híbridos**. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2004.

VILLANI, E.; PASCAL, J. C.; MIYAGI, P. E.; VALETTE, R. **A Petri Net-based Object-Oriented Approach for the Modelling of Hybrid Productive Systems**, Journal of Nonlinear Analysis, Oxford, V. 62, N. 8, pages 1394-1418, 2005.

VILLANI, E.; MIYAGI, P. E.; VALETTE, R. **Landing System Verification Based on Petri Nets and a Hybrid Approach**. IEEE Transactions on Aero-

space and Electronic Systems, Vol. 42, N.4, October, 2006.

WAGNER, F.; SCHMUKI, R.; WAGNER, T.; WOLSTENHOLME, P. **Modeling Software with Finite State Machines**. Auerbach Publications, New York, 2006.

WANG, L., TAN, K. C. **Modern Industrial Automation Software Design – Principles and Real-World Examples**. John Wiley & Sons, Inc., Hoboken, New Jersey, 2006.

WOŹNA, B., ZBRZEZNY, A., PENCZEK, W. **Checking Reachability Properties for Timed Automata via SAT**. Report ICS PAS, No. 949, October, 2002.

ZANMA, T.; MIYABAYASHI, T.; ISHIDA, M. **Sequence generation of discrete event system and logic controller and its applicability to fault detection**. The 8th IEEE International Workshop on Advanced Motion Control AMC'04 - pp. 601-606, 2004.

ZHAO, Q.; KROGH, B. H. **Formal Verification of Statecharts Using Finite-State Model Checkers**. IEEE Transactions on Control Systems Technology, 14 (5): 943-950, September, 2006.

ZHOU, M. C.; VENKATESH, K. **Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach**. World Scientific Publishing, Series in Intelligent Control and Intelligent Automation, Vol. 6, 409 pages, London, 1999.

ZOUBEK, B.; ROUSSEL, J-M.; KWIATKOWSKA, M. **Towards Automatic Verification of Ladder Logic Programs**. Proceedings of IMACS-IEEE "CESA'03": "Computational Engineering in Systems Applications", Cd Rom Paper S2-I-04-0169, 6 pages, Lille, France, 9-12 July, 2003.

ZOUBEK, B. **Automatic verification of temporal and timed properties of control programs**. PhD Thesis, School of Computer Science, University of Birmingham, United Kingdom, September, 2004.

YI, W.; PETTERSSON, P.; DANIELS, M. **Automatic Verification of Real-Time Communicating Systems by Constraint Solving**. In Proceedings of the 7th

International Conference on Formal Description Techniques, Berne, Switzerland, October, 1994.