INSTITUTO MILITAR DE ENGENHARIA

ALEXANDRE DE MELLO SILVA

APLICAÇÃO DE VERIFICAÇÃO DE MODELOS A PROGRAMAS DE CLP: EXPLORANDO A TEMPORIZAÇÃO

Dissertação de Mestrado apresentada ao Curso de Mestrado em Engenharia Elétrica do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Ciências em Engenharia Elétrica.

Orientador: Prof. Antonio Eduardo Carrilho da Cunha, Dr. Eng.

Rio de Janeiro
2008

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80-Praia Vermelha
Rio de Janeiro-RJ CEP 22290-270

629.8 Silva, Alexandre de Mello
S586a Aplicação de Verificação de Modelos a Programas de CLP: Explorando a Temporização / Alexandre de Mello Silva. - Rio de Janeiro: Instituto Militar de Engenharia, 2008.
121 p.: il.

Dissertação (mestrado) - Instituto Militar de Engenharia - Rio de Janeiro, 2008.

1. Sistemas de Controle. 2. Verificação de Modelos. 3. CLP. 4. Tempo Real. 5. Autômatos Temporizados. I. Título. II. Instituto Militar de Engenharia.

CDD 629.8

INSTITUTO MILITAR DE ENGENHARIA

ALEXANDRE DE MELLO SILVA

APLICAÇÃO DE VERIFICAÇÃO DE MODELOS A PROGRAMAS DE CLP: EXPLORANDO A TEMPORIZAÇÃO

Dissertação de Mestrado apresentada ao Curso de Mestrado em Engenharia Elétrica do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Ciências em Engenharia Elétrica.

Orientador: Prof. Antonio Eduardo Carrilho da Cunha, Dr. Eng. Aprovada em 28 de Janeiro de 2008 pela seguinte Banca Examinadora:

Prof. Antonio Eduardo Carrilho da Cunha, Dr. Eng. do IME - Presidente

Prof. Antonio Marcus Nogueira Lima, Dr. da UFCG

Prof. Jean-Marie Alexandre Farines, Dr. da UFSC

Prof. Jorge Audrin Morgado de Gois, Dr.-Ing do IME

Rio de Janeiro
2008

Eu dedico este trabalho a *DEUS*; aos anjos que *ELE* permitiu que me auxiliassem, em especial Fernanda, Audimar, Luiz Carlos, Márcia e a linda Luisa; e a todos os amigos dos quais tive agradáveis e preciosos momentos de convívio subtraídos para que esse resultado aparecesse.

AGRADECIMENTOS

A Deus, eu sou eternamente grato, por tudo.

Agradeço a minha família, Fernanda, minha querida esposa, Audimar, minha mãe, Luiz Carlos meu pai, e Márcia, minha irmã, por perceberem a diminuição dos meus sorrisos, e retribuírem com mais e mais carinho e preocupação comigo.

A nossa casa, o Instituto Militar de Engenharia, e à Seção de Engenharia Elétrica, que me acolheram, mais uma vez, tendo participação expressiva nas minhas realizações.

Ao Prof. Antonio Eduardo Carrilho da Cunha que escolhi como meu orientador e me aceitou como orientando antes mesmo de me matricular neste curso. É um desbravador na pesquisa de controle de sistemas a eventos discretos e sistemas híbridos no IME, arcando com todos os ônus e bônus dessa empreitada. E, em meio a toda sorte de turbulências dos novos caminhos, mantém o otimismo indispensável para seguir e conduzir seus colegas à frente. Agradeço também ao seu esforço em equilibrar o professor que precisa corrigir e o amigo que precisa dar a força.

A todos os professores do curso de mestrado da Seção de Engenharia Elétrica que com dedicação e paciência latentes, partilharam seus conhecimentos ao longo do ano de 2006, nos cobrando noites sem dormir, mas nos oferecendo o prazer de andar com as próprias pernas.

À equipe da Universidade Federal de Campina Grande, Kyller, Leandro e Luiz Paulo que se preocuparam em transmitir críticas construtivas. E em especial ao Prof. Antônio Marcus Nogueira Lima, que permitiu e incentivou esse trabalho em conjunto, e por ter aceito o convite para participar da banca examinadora deste trabalho.

Ao CENPES/Petrobrás na pessoa do Eng Marcelo Lima, por ter dado suporte a este trabalho.

Ao Prof. Ney Bruno e ao Técnico João Carlos, do Laboratório de Automação e Sistemas de Controle do IME, por me cederem um recanto para a escrita da maior parte dessa dissertação em detrimento de suas privacidades, além de estarem sempre disponíveis em ajudar, seja sanando uma dúvida de português ou inglês, com o reparo do computador ou com seus exemplos de vida carregados de persistência.

Aos Profs. Amit Bhaya, Jean-Marie Alexandre Farines e Jorge Audrin Morgado de Gois, por também aceitarem o convite para participar da banca examinadora deste

trabalho.

Aos meus companheiros de mestrado, pela amizade e pela união de sucesso que não acabam aqui.

Aos colegas de outras turmas, nossos "irmãos mais velhos".

Enfim, agradeço aos amigos e parentes que também auxiliaram neste projeto, direta ou indiretamente.

RESUMO

Esta dissertação aborda a verificação de modelos aplicada a programas de Controladores Lógicos Programáveis (CLPs) com ênfase naqueles que constituem sistemas de tempo real, procurando dar seqüência ao trabalho de OLIVEIRA (2006). Este trabalho, também desempenha um papel no projeto de melhoria do processo de engenharia de programas de CLPs de Sistemas Instrumentados de Segurança (SIS). Este fato impõe dois objetivos a serem perseguidos: modelagem e especificação formais e automatizadas (para facilitar a utilização pelos futuros engenheiros usuários), e tratamento de programas com um número de entradas e saídas em escala industrial.

Os programas são considerados escritos em diagramas lógicos. O documento básico para extração das especificações formais é a Matriz de Causa e Efeito (MCE).

Após uma parte introdutória que apresenta o trabalho e descreve sua seqüência principal, justifica-se a escolha do arcabouço teórico: autômatos temporizados, Matrizes de Limites de Diferenças (DBM) e o verificador UPPAAL; e um breve embasamento do mesmo é exposto.

Em seguida, são expostos trabalhos que têm como objeto de estudo a verificação de modelos de programas de CLPs por meio do uso da ferramenta UPPAAL, sendo pinçados seus conceitos principais, para comparação com o modelo adotado e auxílio na formulação das soluções.

Dessa análise emerge um registro sobre a verificação do modelo adotado utilizando-se a ferramenta UPPAAL e seus limites. Uma proposta de especificação de propriedades de tempo real é feita, aperfeiçoando a utilização de autômatos auxiliares chamados de observadores. Duas ferramentas, MCE2ctl e XML2SMV, são desenvolvidas perfazendo as contribuições, em resposta aos objetivos colocados anteriormente. MCE2ctl gera, a partir dum arquivo txt contendo uma MCE convertida de um arquivo xls, um arquivo txt com especificações em CTL próprias ao verificador SMV. XML2SMV converte um modelo xml num modelo codificado também para o verificador SMV.

ABSTRACT

This work covers model checking applied to Programmable Logic Controllers (PLC) programs with emphasis on those which are considered real-time systems. It is a sequence of OLIVEIRA (2006). It also takes part of an improvement project of engineering process of security instrumentation systems (SIS) PLC programs. This fact pose us two aims: automatic formal modeling and specification, and dealing with programs in which the numbers of inputs and outputs are on industrial scale.

The programs are considered to be written in Logic Diagrams. The basic document from which the specifications are extracted is the Cause and Effect Matrix (CEM).

After an introduction, where this work is presented and its main sequence is described, the theoretical framework adopted is justified : timed automata, Difference Bounded Matrix (DBM) and UPPAAL, the verification tool. This part gives an overview of this background, too.

The following chapter exposes the published works related with model checking applied to PLC programs and the UPPAAL tool. Their main concepts are outlined to be compared against the adopted model and to aid the solutions formulation.

A register about model checking with UPPAAL and the limitations that arises in such a process emerge from this analysis. One proposal of real-time properties specification is done, betting the use of auxiliar automatas called observers. Two tools, MCE2tctl and XML2SMV, are developed completing the contributions that answers the objectives put before. MCE2ctl produces, from a txt file containing a CEM converted from a xls file, a txt file with Computation Tree Logic (CTL) specifications for SMV tool. XML2SMV parser a xml model to a SMV model.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SÍMBOLOS

ABREVIATURAS

CENPES	-	Centro de Pesquisa e Desenvolvimento (da PETROBRAS)
CLP	-	Controlador Lógico Programável
CTL	-	Lógica da Árvore de Computação ou Lógica de Tempo Ramificante (*Computation Tree Logic*)
TCTL	-	Lógica Temporal da Árvore de Computação (*Timed Computation Tree Logic*)
DBM	-	Matriz de Limites de Diferenças (*Difference Bounded Matrix*)
FBD	-	Diagrama de Blocos de Funções (*Function Block Diagram*)
IL	-	Lista de Instruções (*Instruction List*)
IME	-	Instituto Militar de Engenharia
LD	-	Diagrama Ladder (*Ladder Diagram*)
LTL	-	Lógica de Tempo Linear (*Linear Tree Logic*)
MCE	-	Matriz de Causa e Efeito
P&ID	-	Diagrama de Processo e Instrumentação (*Process and Instrumentation Diagram*)
ST	-	Texto Estruturado (*Structured Text*)
SFC	-	Cartas de Funções Seqüenciais (*Sequential Function Chart*)
SIS	-	Sistemas Instrumentados de Segurança
SMV	-	*Symbolic Model Verifier*, verificador de modelos desenvolvido na Carnegie Mellon University (MCMILLAN, 2001).
UFCG	-	Universidade Federal de Campina Grande

SÍMBOLOS

A	-	Operador de caminho que significa: *para todos os caminhos*.
E	-	Operador de caminho que significa: *existe um caminho*.
G	-	Operador temporal que significa: *sempre* ou *globalmente*.
F	-	Operador temporal que significa: *no futuro*.

U	-	Operador temporal que significa: *até*.
W	-	Operador temporal que significa: *a menos que*.
X	-	Operador temporal que significa: *próximo*.
R	-	Operador temporal que significa: *habilita*.
$\rightarrow$	-	Conector *implica* da lógica.
$\rightsquigarrow$	-	Operador *leads to* do UPPAAL. $p \rightsquigarrow q$ equivale a $AG(p \rightarrow AF\ q)$
&	-	Conector lógico "e".
&&	-	Conector lógico "e" quando escrito na guarda de um arco no UPPAAL.
==	-	Operador da igualdade quando escrito na guarda de um arco no UPPAAL, não é usado para atribuição, só para teste.
|	-	Conector lógico "ou".
!	-	Símbolo lógico da negação quando antecede uma variável lógica e símbolo do arco emissor do sincronismo numa rede de autômatos temporizados do UPPAAL, quando aparece depois do nome do canal.
?	-	símbolo do arco receptor do sincronismo numa rede de autômatos temporizados do UPPAAL.
||	-	símbolo de cardinalidade de conjunto. Exemplo: Seja C = {a, b, c}, então |C|=3.

1 INTRODUÇÃO

1.1 OBJETO DE ESTUDO

Este trabalho está inserido na área de verificação e validação formal de sistemas de tempo real, em especial sistemas automatizados por Controladores Lógicos Programáveis - CLPs. Por verificação formal entende-se a rigorosa exploração de incorreções do projeto de um sistema, expresso em um modelo matemático (WANG, 2004). A verificação formal existe em função da confiabilidade exigida em sistemas dos quais dependem vidas humanas ou valorosos recursos materiais.

As duas variações de verificação formal mais em voga são: a prova por teorema e a verificação de modelos (LAMPÉRIÈRE-COUFFIN et al., 1999) e (WANG, 2004). Este trabalho se utiliza da verificação de modelos, tendo esta opção obedecido, principalmente, à seqüência de trabalhos anteriores (OLIVEIRA, 2006) e à oportunidade de inserção no projeto de Sistemas Instrumentados de Segurança (SIS) do CENPES/PETROBRAS junto à Universidade Federal de Campina Grande (UFCG). É importante ressaltar que, independente do peso desse alinhamento, a forte presença da verificação de modelos em meio a publicações científicas na área de verificação formal, confere lastro suficiente à aludida opção (LAMPÉRIÈRE-COUFFIN et al., 1999; WANG, 2004).

A verificação formal de sistemas de tempo real tem se mostrado aquecida em função de três fatores (WANG, 2004):

- O sucesso e consolidação da verificação formal na indústria de sistemas integrados em larga escala (VLSI) tornou natural a expectativa da expansão da aplicação dessa teoria para outras áreas, entre elas, a verificação formal de sistemas de tempo real. Destaca-se a participação majoritária da verificação de modelos dentro deste sucesso;

- A disponibilidade de arcabouços teóricos preparados para aplicações práticas; e

- A crescente complexidade e abrangência dos ditos Sistemas Embarcados (*Embedded Systems*) aumentou o número de projetos ao mesmo tempo em que seus cronogramas ficaram mais apertados, isso fez transparecer ainda mais a in-

suficiência da técnica de simulações para alcançar confiabilidade. Atentando-se para o fato de que a simulação não é uma técnica exaustiva.

Aqui no Instituto Militar de Engenharia (IME), o trabalho de OLIVEIRA (2006) foi o primeiro nesta área. Como resultado da busca por aplicações práticas, este trabalho não só encontrou nos projetos de SIS um caso de estudo, como também despertou o interesse do CENPES/PETROBRAS em participar ativamente desta pesquisa. Um ponto importante do trabalho, foi a indicação da necessidade de se explorar a abordagem a sistemas de tempo real, para tratamento dos projetos de SIS.

Esta situação configurou uma oportunidade ímpar de se construir uma base teórica, aliada à vivência de aplicação a um projeto real, combinação esta que sem dúvida será de grande valia para os projetos de *hardware* e sistemas embarcados que existem e porventura venham a existir no âmbito do Exército Brasileiro.

1.2 DESENVOLVIMENTO

A exploração da verificação de modelos aplicada a sistemas de tempo real, tema deste trabalho, é norteada por três aspectos:

- Os sistemas de tempo real considerados são programas de CLP, em função do estudo estar inserido no projeto SIS do CENPES/PETROBRAS o qual será detalhado na seção 2.1 e do mesmo representar uma seqüência do trabalho de OLIVEIRA (2006).

- A participação neste projeto é validar o modelo utilizado para gerar testes automáticos por meio da ferramenta UPPAAL-Tron (UPPAAL, 2008), com isso o verificador UPPAAL (UPPAAL, 2008) despontou como uma opção natural de ferramenta para este trabalho.

- Como preceito do citado projeto, a verificação de modelos deve se dar com a mínima interferência humana possível. Sendo assim, não se leva em conta o recurso à abstração, feita por especialistas, para tratar dos grandes sistemas normalmente resultantes da modelagem de programas de CLP.

FREY e LITZ (2000) e WANG (2004) são dois destacados trabalhos que discorrem sobre as vantagens e desvantagens dos vários fatores envolvidos na verificação de modelos de sistemas de tempo real: abordagem baseada ou não no modelo do processo a

ser controlado, formalismo de descrição do sistema (autômatos temporizados, lógicas, equações algébricas, entre outros), representação do espaço de estados (BDD, DBM, etc). A ferramenta é um dos fatores analisados e, segundo essas referências, o UPPAAL se mostra como um verificador viável, o que, aliado ao fato deste utilizar o mesmo modelo utilizado pela ferramenta de geração automática de testes, o credenciou como a escolha desta dissertação (mais alguns detalhes sobre esta escolha constam da seção 3.1).

Foram levantados trabalhos que se utilizam do UPPAAL para verificação de modelos aplicada a programas de CLP. Este material auxiliou muito a análise realizada sobre a proposta de GORGÔNIO et al. (2006), adotada neste trabalho, e sobre as possíveis alterações na **modelagem** que poderiam contribuir para a redução da explosão do espaço de estados. Daí surge a primeira contribuição desta dissertação:

> Uma proposta de procedimento para especificação envolvendo propriedades de tempo real e verificação de modelos aplicada a programas de CLP, utilizando um autômato auxiliar.

A despeito das tentativas de alterações na modelagem, a investigação realizada indicou que a verificação de programas de CLP com mais de onze entradas, envolvendo um temporizador, é inviável. Isto considerando o uso do verificador UPPAAL e sem se recorrer à abstração, a qual normalmente é feita por um especialista, sendo difícil o estabelecimento de um tratamento automático.

Para validar o modelo a ser utilizado com o UPPAAL-Tron na geração de testes automáticos, lançou-se mão do verificador SMV (*Symbolic Model Verifier*) (MCMILLAN, 2001) o qual não contempla propriedades de tempo real. Desta situação surge mais uma contribuição:

> A confecção de duas ferramentas computacionais: o MCE2ctl, para a extração de especificações em lógica CTL a partir de MCEs, e o XML2SMV, para conversão de um modelo próximo ao diagrama lógico, num modelo para uso com o SMV.

O modelo para uso com o SMV foi adaptado de OLIVEIRA (2006) e o XML2SMV foi elaborado com base numa ferramenta desenvolvida na UFCG.

1.3 GUIA DE LEITURA À DISSERTAÇÃO

Este trabalho está organizado da seguinte forma:

- O Capítulo 2 apresenta a formulação do problema tratado nesta dissertação, apresentando onde este trabalho se encaixa no Projeto SIS, juntamente com um embasamento sobre os Controladores Lógicos Programáveis.

- O Capítulo 3 fornece uma visão geral sobre a Verificação de Modelos, Lógicas Temporais e o trabalho de OLIVEIRA (2006).

- O Capítulo 4 traz uma visão geral sobre os autômatos temporizados, que são elementos básicos da linguagem formal para modelagem de Sistemas de Tempo Real utilizada neste trabalho. Também faz um breve comentário sobre a Matriz de Limites de Diferenças (*Difference Bounded Matrix*, DBM), a estrutura de dados utilizada pelo verificador UPPAAL.

- O Capítulo 5 apresenta uma revisão das principais abordagens encontradas na literatura de utilização da Verificação de Modelos para validar programas de CLPs.

- O Capítulo 6 apresenta o modelo de um SIS desenvolvido por GORGÔNIO et al. (2006), as tentativas de mudança na modelagem para tornar viável a verificação de um sistema tomado como exemplo e os procedimentos para a verificação do modelo adotado, utilizando a ferramenta UPPAAL.

- O Capítulo 7 aborda a verificação lógica do modelo de SIS desenvolvido por GORGÔNIO et al. (2006) utilizando a abordagem de OLIVEIRA (2006), apresentando ferramentas de automatização do procedimento de verificação.

- O Capítulo 8 traz considerações finais sobre a abordagem deste trabalho.

- A Anexo 10.1 exibe o modelo do UPPAAL utilizado na seção 6.5. O Anexo 10.2 contém, os modelos e a indicação dos nomes dos arquivos utilizados na verificação tratada no Capítulo 7.

- O Apêndice 11.1 transcreve a descrição do exemplo do Apêndice A da Norma ISA 5.2 (ISA, 1992), modelado por GORGÔNIO et al. (2006) e objeto de validação no presente trabalho.

2 FORMULAÇÃO DO PROBLEMA

Este capítulo coloca em linhas gerais o que é o Projeto SIS e como a dissertação nele se encaixa. O CLP e a sua programação representam o cerne do projeto, e como tal, seus principais conceitos são destacados na seqüência.

2.1 PROJETO SIS

A FIG. 2.1 ilustra o contexto deste trabalho. O modelo atual adotado pela PETROBRAS para o projeto de um SIS inclui a seqüência à esquerda dividida em três etapas, a saber: Projeto Básico, Projeto Detalhado e Implementação.

- No Projeto Básico as especificações do sistema são listadas na forma de alguns documentos: Matriz de Causa e Efeito, Diagramas de Processo e Instrumentação (*Process and Instrumentation Diagram*, P&ID), Especificações textuais, Fluxogramas e Diagramas Lógicos.

- Na etapa do Projeto Detalhado, com base nos documentos gerados na etapa anterior, a própria empresa ou uma empresa terceirizada desenvolve o projeto do SIS. Os documentos de referência produzidos nesta etapa são o programa de aplicação a ser carregado no CLP e os diversos diagramas elétricos e de instrumentação do sistema. Este programa segue a simbologia da norma IEC 61131-3 (IEC, 2003). A maioria dos programas de CLP gerados atualmente são escritos na linguagem de diagramas de contatos, entretanto no caso de SIS emprega-se a linguagem de blocos funcionais.

- Na etapa de Implementação são executados testes para validação do sistema. Tipicamente, testes de bancada ou os ditos *loop-tests* (testes feitos com o CLP ligado à malha de controle) são realizados, envolvendo representantes das equipes que elaboraram a automação e o cliente usuário do sistema. Como estes testes não exaurem as possibilidades de falhas, testes mais abrangentes e automáticos estão sendo propostos pela UFCG para aumentar a confiabilidade do sistema. Assim, os seguintes papéis são distinguidos.

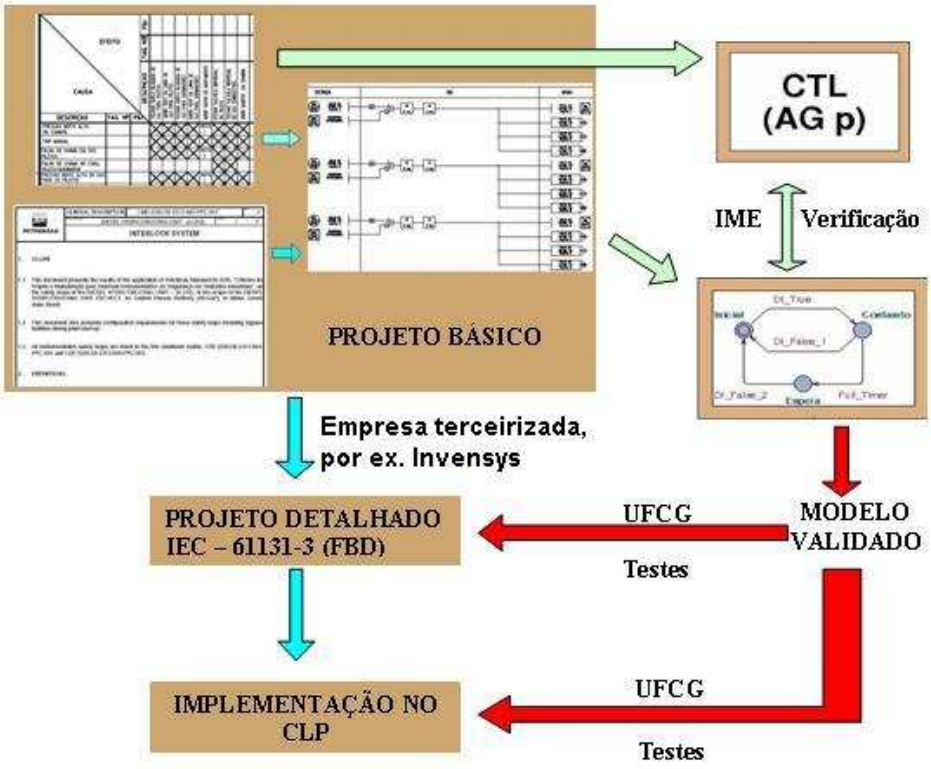


FIG.2.1: Verificação para validação do modelo.

O objetivo da equipe da UFCG é, com base no modelo obtido a partir dos documentos produzidos na fase de Projeto Básico, gerar um conjunto de testes, automaticamente, para validar tanto o programa do CLP fruto da fase de Projeto Detalhado quanto o programa rodando no CLP na fase de Implementação.

O objetivo da participação do IME no projeto SIS, contida neste trabalho, é validar o modelo a ser usado para geração de testes automáticos, usando para isto a técnica de verificação de modelos. Um objetivo concomitante é que essa verificação de modelos se dê de forma a mais automática possível. Para isso, a geração do modelo e a extração das especificações no formato de entrada do verificador escolhido devem ser automáticas ou fortemente apoiadas por ferramentas computacionais.

2.2 CONTROLADORES LÓGICOS PROGRAMÁVEIS

Um CLP, na sua forma mais simples, é um controlador baseado em um microprocessador que amostra sinais de entrada binários, em instantes discretos de tempo, executa um programa de controle, durante um certo período de tempo, e depois atualiza o valor dos sinais de saída binários. A FIG. 2.2 ilustra a sua estrutura.

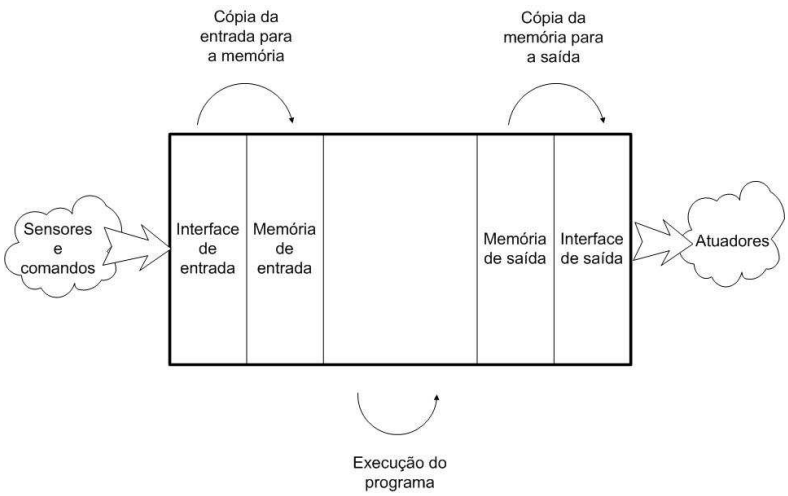


FIG.2.2: Estrutura de um CLP

O CLP é aplicado a controle de processos industriais. Os sinais de entrada são, via de regra, sinais de sensores e comandos. A informação dos sensores e comandos está disponível a todo momento na interface de entrada, todavia, somente durante a operação de leitura, os sinais são copiados para a memória de entrada. Essa informação, congelada na memória de entrada, é usada durante a execução do programa de controle. Os sinais de saída, computados na operação precedente, são copiados da memória de saída para a interface de saída, operação esta chamada de escrita. A interface de saída é um conjunto de portas (chaves) com valores nominais de operação (tensão, corrente entre outros) adequados ao comando de atuadores.

As operações de leitura, execução e escrita são realizadas repetida e indefinidamente, de acordo com o ciclo exibido na FIG. 2.3, chamado de *ciclo de varredura*. O tempo que cada ciclo leva para ser completado é o mesmo, caracterizando um ciclo periódico, cujo período é chamado de *tempo de varredura* (ou de *scan*).

O modo de operação descrito acima, chamado de mono-tarefa, atende a maior parte das aplicações. Para algumas outras, no entanto, é necessário um recurso adicional chamado de execução multitarefa (MOKADEM, 2006).

No funcionamento normal (execução do programa principal), o tempo de resposta mínimo a um evento (mudança de valor de uma variável de entrada) está entre 1 ciclo de varredura, caso o evento ocorra no exato instante que antecede a sua leitura pelo CLP (Entrada 1 na FIG. 2.4), e 2 ciclos de varreduras, caso o evento ocorra imediatamente após a conclusão da leitura de entradas do CLP (Entrada 2 na FIG. 2.4) (MOKADEM, 2006). A execução multitarefa visa reduzir esse tempo para menos de um ciclo. A

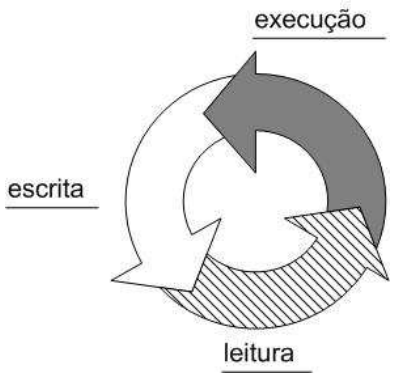


FIG.2.3: Ciclo de varredura de um CLP.

seqüência de operação é a seguinte: a execução do programa principal é interrompida, ocorre um desvio para tratamento de uma rotina específica, que atualiza a saída (resposta a uma possível alteração) e depois retorna à execução normal do programa principal.

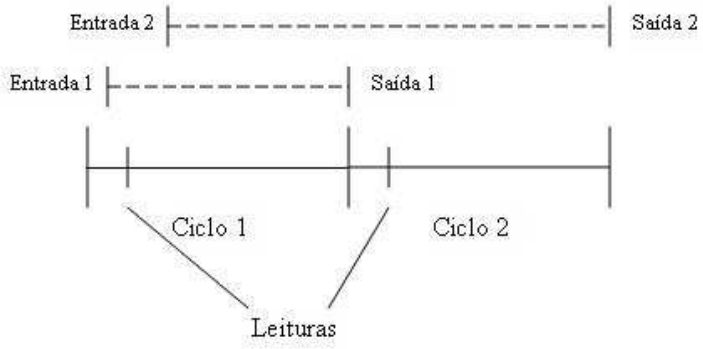


FIG.2.4: Tempo de resposta num CLP sem execução multitarefa.

A norma IEC 61131-3 (IEC, 2003) estabelece que a interrupção possa se dar em função da alteração de uma determinada entrada ou ocorrer periodicamente. O número de tarefas que podem ser executadas e o tempo de execução de cada uma delas são soluções particulares de cada fabricante de CLP.

A norma IEC 61131-3 (IEC, 2003) define 5 linguagens básicas de programação de CLPs:

a) Lista de Instruções (*Instruction List* - IL),

b) Texto Estruturado (*Structure Text* - ST),

c) Diagrama de Contatos (*Ladder Diagram* - LD),

d) Diagrama de Blocos Funcionais (*Function Block Diagram* - FBD) e

e) Cartas de Funções Seqüenciais (*Sequential Function Chart* - SFC).

Na realidade, a norma não considera SFC como uma linguagem de programação propriamente dita, e sim como um linguagem de alto nível para estruturar e organizar (modelar) unidades de um programa caracteristicamente seqüencial. Como expresso na mesma norma, as definições encontradas na seção sobre SFC são derivadas do GRAFCET, cuja norma de referência é a IEC 60848 (IEC, 1999).

A norma ISA 5.2 (ISA, 1992) define símbolos para confecção de diagramas lógicos binários para representar operações de processos. Tais símbolos não são exatamente padronizados como símbolos da linguagem FBD da norma IEC 61131-3 (IEC, 2003), mas, a maioria, possui equivalente direto dentro da mesma.

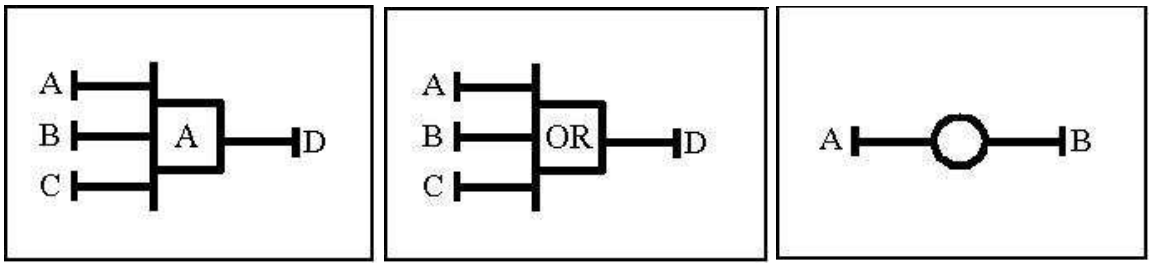
O modelo a ser verificado pode ter como base o modelo extraído do programa escrito numa das linguagens da IEC 61131-3 (IEC, 2003) ou escrito como diagramas lógicos conforme a norma norma ISA 5.2 (ISA, 1992). Neste trabalho, o modelo adotado é extraído de um programa escrito como diagrama lógico, em virtude deste ser um documento padrão do projeto básico de um SIS.

O modelo a ser verificado poderá, opcionalmente, ser composto ainda pelo modelo do comportamento do CLP e/ou pelo modelo do ambiente. No caso do modelo ser verificado contra propriedades de tempo real, a modelagem do comportamento do CLP tem sido a escolha unânime na literatura.

A seguir serão apresentados os elementos básicos da norma ISA 5.2 (ISA, 1992).

A FIG. 2.5 mostra os elementos lógicos básicos. Do lado esquerdo de cada símbolo estão as entradas e do lado direito estão as saídas. A FIG. 2.5(a) representa o bloco lógico E (*AND*) onde a variável de saída D só fica ativa (torna-se verdadeira) se e somente se as variáveis de entrada A, B e C estiverem ativas (forem verdadeiras) e escreve-se $D = A \ \& \ B \ \& \ C$. A FIG. 2.5(b) representa o bloco lógico OU (*OR*) onde a variável de saída D só fica ativa se e somente se uma ou mais variáveis de entrada A, B ou C estiverem ativas e escreve-se $D = A \mid B \mid C$. A FIG. 2.5(c) representa o bloco lógico NÃO (*NOT*) onde a variável de saída B só fica ativa se e somente se a variável de entrada A não estiver ativa e escreve-se $B = !A$.

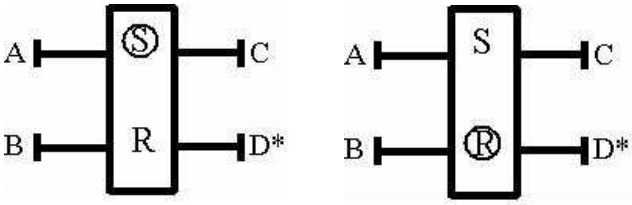
A FIG. 2.6 exibe os elementos de memória básicos da norma ISA 5.2 (ISA, 1992). nos quais S representa *SET* e R representa *RESET*. A saída lógica C fica ativa assim que a entrada lógica A se tornar ativa. C continua ativa, independente do estado



(a) "E" lógico (b) "OU" lógico (c) "NÃO" lógico

FIG.2.5: Elementos lógicos básicos (ISA, 1992).

subseqüente de *A*, até que a entrada lógica *B* se torne ativa. *C* então permanece desativada independente do estado subseqüente de *B*, até que *A* volte a ficar ativa. A saída *D** é opcional e é a negação da saída *C*. Se ambas as entradas de *S* e de *R* estiverem, simultaneamente, ativas a saída *C* ficará ativa ou não em função da prioridade identificada pelo elemento circulado: na FIG. 2.6(a) *S* tem prioridade sobre *R*, tornando *C* ativa; já na FIG. 2.6(b) a prioridade é para o *R*, então neste caso a saída *C* fica desativada.



(a) elemento *set/reset* com prioridade para *set*

(b) elemento *set/reset* com prioridade para *reset*

FIG.2.6: Elementos de memória básicos (ISA, 1992).

O temporizador é o divisor de águas em relação a necessidade e a indicação do sistema ser de tempo real ou não. Sua ausência do sistema implica que o mesmo não é de tempo real. A entrada lógica (IN na FIG. 2.7) dispara um relógio de formas variadas, formas essas que determinam os tipos de temporizadores. A maneira mais fácil de visualizá-lo é com o símbolo identificando suas entradas e saídas, acompanhado do gráfico que caracteriza sua operação. Na FIG. 2.7 são exibidos os três tipos de temporizadores básicos definidos na norma ISA 5.2 (ISA, 1992) e a caracterização de seus comportamentos. Nesta figura, *t* é uma constante definida no diagrama.

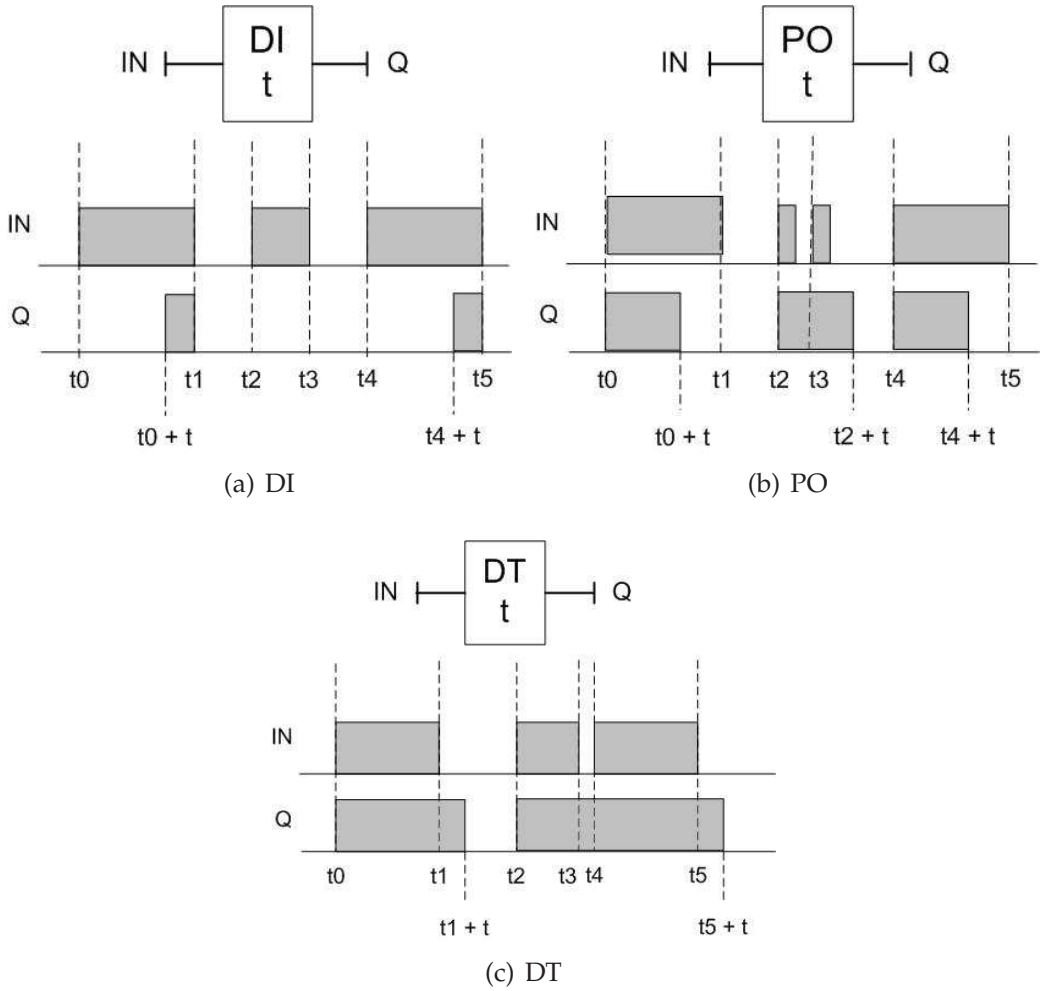


FIG.2.7: Tipos de temporizadores: a) Temporizador com atraso na inicialização da saída (*Delay Initiation of output*) - DI. b) Temporizador por pulso (*Pulse Output*) - PO. c) Temporizador com atraso no desligamento da saída (*Delay Termination of output*) - DT.

2.3 RESUMO DO CAPÍTULO

Neste capítulo foi apresentado o projeto SIS, onde este trabalho se encaixa e conceitos de seus elementos principais: o CLP, seu funcionamento e suas linguagens de programação. Foi dado ênfase aos símbolos da norma ISA 5.2 (ISA, 1992), pois o diagrama lógico em consonância com esta norma, foi escolhido entre os documentos do projeto básico para ser o ponto de partida para modelagem do programa de controle. Este modelo será a referência contra a qual os documentos das fases posteriores do projeto serão confrontados.

3 VERIFICAÇÃO DE MODELOS

Este capítulo apresenta a verificação de modelos, mostra seu funcionamento e as variações existentes na sua implementação, com foco no tratamento de sistemas de tempo real. Justificam-se a escolha da linguagem de descrição do sistema e da ferramenta de verificação utilizados nesta dissertação.

Em seguida, são introduzidos os conceitos de lógica temporal necessários para a escrita de especificações formais. Também são apresentados padrões para execução dessa escrita.

Neste ponto, o suporte teórico abordado permite expor um resumo das contribuições de OLIVEIRA (2006), as quais motivaram a investigação e o desenvolvimento realizados nesta dissertação.

3.1 VISÃO GERAL

A verificação de modelos é uma técnica automática para verificação e validação formal de sistemas concorrentes de estados finitos (CLARKE et al., 1999). O termo em inglês é *model checking*, cuja tradução mais próxima seria inspeção de modelos por evitar ambigüidade na inclusão dos conceitos de verificação e validação. Apesar disso, é comum usar somente a denominação verificação de modelos, englobando também o conceito de validação, o que será feito doravante.

Dentre as técnicas para verificar e validar, formalmente, um sistema, a verificação de modelos e a prova por teorema são as duas mais utilizadas (LAMPÉRIÈRE-COUFFIN et al., 1999).

A verificação e validação formal nasceram e se desenvolveram a partir da preocupação da Ciência da Computação em atestar o funcionamento dos seus sistemas - *softwares* ou *hardwares* - pois a simulação e os testes se mostraram insuficientes em inúmeros casos: acidente do foguete Ariane 5 (SPARACO, 1996), aplicação de doses fatais de radiação em pacientes com câncer pelo Therac-25 (LEVESON e TURNER, 1993), um equipamento de radioterapia controlado por computador, etc. No início dos anos 80, Clarke e Emerson (CLARKE e EMERSON, 1981), e de forma independente, Quielle e Sifakis (QUIELLE e SIFAKIS, 1982), introduziram os algoritmos de verificação de modelos usando lógica temporal, lançando assim, a base do método.

Tanto a prova por teorema quanto a verificação de modelos têm a grande vantagem de cobrir todos os possíveis estados do sistema. No entanto, a primeira necessita da intervenção de especialistas de enorme capacidade técnica, além de ser um processo demorado. A verificação de modelos, em contrapartida, é menos complicada de ser executada além de ser mais rápida. Os seus maiores obstáculos são a aplicação restrita a sistemas de estados finitos e a explosão do espaço de estados.

Para se efetuar a verificação é necessário o cumprimento de algumas etapas: a primeira delas é a escrita do sistema em um modelo formal que o verificador possa ler. Também é preciso que as propriedades esperadas para o sistema sejam igualmente formalizadas em adequação ao verificador. Isto é feito colocando-se as propriedades na forma de autômatos, expressões em lógica temporal ou condições algébricas. De posse destas duas formalidades o verificador então testa as propriedades, através de todas as possibilidades de realização do sistema, gerando o sucesso ou o contra-exemplo (ver Fig.3.1).

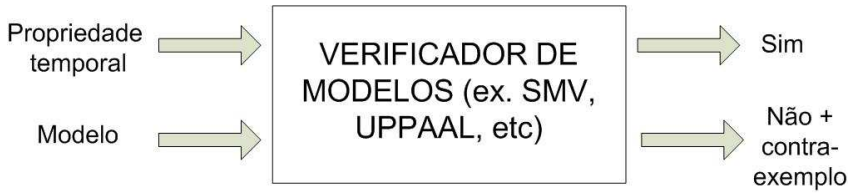


FIG.3.1: Processo de verificação de modelos

A verificação de modelos foi empregada em sistemas baseados em Controladores Lógicos Programáveis (CLPs) primeiramente por MOON (1994). Essa técnica oferece reconhecida contribuição nos projetos industriais o que pode ser atestado pela considerável produção científica despertada pelo assunto (MOON, 1994), (GOURCUFF et al., 2006), (RAUSCH e KROGH, 1998), (DE SMET e ROSSI, 2002), (LAMPÉRIÈRE-COUFFIN et al., 1999), (MADER e WUPPER, 1999), entre outros. Porém os obstáculos apontados por MOON (1994) ainda permanecem impedindo o seu uso natural (GOURCUFF et al., 2006), quais sejam:

1. A especificação formal de propriedades em lógica temporal ou na forma de autômatos ainda é uma tarefa árdua para a maioria dos engenheiros, o que poderia ser minimizado pela existência de uma linguagem de alto nível que descrevesse essas especificações de uma maneira mais fácil ou uma ferramenta de extração de especificações padronizadas a partir de documentos e descritivos

do processo, como a Matriz de Causa e Efeito.

2. Os fabricantes de CLP não disponibilizam softwares capazes de traduzir automaticamente seus programas em modelos formais. Neste contexto, a padronização de modelos (bibliotecas) para temporizadores, contadores, computação aritmética, manipulação de arquivos e outras funções de alto nível, representa um importante avanço.

3. E finalmente o principal empecilho: a explosão do espaço de estados, que nasceu com a própria técnica de verificação de modelos. Este empecilho não permite sua utilização direta em cima de sistemas industriais os quais, em geral, são de grande porte, além de conterem especificações que envolvem propriedades de tempo real. Nestes casos há que se recorrer a técnicas tais como a abstração, a decomposição, entre outras.

Existem algumas variações que emergem das etapas do processo de verificação de modelos.

O sistema a ser descrito pode levar em conta, ou não, o processo a ser controlado (FREY e LITZ, 2000). A modelagem do processo restringe o modelo geral, diminuindo seu espaço de estados. Um bom exemplo desta abordagem é a modelagem da Estação 2 da plataforma MSS encontrada em MOKADEM (2006) a qual será mais detalhada na seção 5.5 na pág.61. Todavia, os inúmeros processos existentes dificultam o estabelecimento de um procedimento para este passo. Como faz parte dos objetivos deste trabalho diminuir a interferência humana no processo de verificação de modelos, aqui escolheu-se a modelagem mais adequada a uma sistematização: o processo a ser controlado é visto como um ambiente livre ou, segundo FREY e LITZ (2000), a descrição do sistema é não baseada em modelo (**tipo N**). Cada entrada lógica pode ser lida como ativa (verdadeira) ou não ativa (falsa) a qualquer tempo, independente das outras e do passado.

Algumas linguagens de descrição e especificação de sistemas, utilizadas para verificação de programas de CLPs são (FREY e LITZ, 2000):

- Redes Petri;
- Sistemas condição / evento;
- Autômatos;

- Lógicas (de altas ordens);
- Linguagens síncronas;
- Sistemas de Transição Geral e
- Equações (algébricas).

Em se tratando de sistemas de tempo real (no caso de programas de CLP, são geralmente aqueles que possuem temporizadores), um formalismo amplamente usado para descrição é o *autômato temporizado* (MOKADEM, 2006). O que o torna interessante é o fato de um dos problemas de verificação de modelos, o problema de acessibilidade, ser PSPACE, isto é, o consumo de memória é polinomial em função do tamanho da entrada em número de bits (WANG, 2004). Algumas importantes ferramentas se utilizam deste formalismo, quais sejam: UPPAAL (BENGTSSON et al., 1996), RED (WANG, 2004) e Kronos (YOVINE, 1997), sendo que outra, o HyTech (HENZINGER et al., 1997), se utiliza dos autômatos híbridos lineares que é uma superclasse com relação aos autômatos temporizados (WANG, 2004).

A escolha da linguagem de descrição do sistema foi feita em conjunto com a escolha da ferramenta adotada. Apoiado nos fatos citados no parágrafo anterior e somando-se a isso: o fato do modelo a ser validado já estar escrito no formato de autômatos temporizados do UPPAAL (pois a equipe da UFCG se utiliza do UPPAAL-Tron para geração de testes automáticos) e o fato de existirem interessantes trabalhos publicados na área, por exemplo o de MOKADEM (2006) e o de ZOUBEK (2004), escolheu-se a ferramenta UPPAAL e a rede de autômatos temporizados como linguagem de descrição do sistema.

3.2 LÓGICAS TEMPORAIS

Lógica temporal é um formalismo para descrever seqüências de transições entre estados, num sistema reativo. O tempo não aparece explicitamente, em vez disso, usam-se operadores temporais para descrever características tais como:

- No futuro, uma dada proposição será verdadeira.
- Um determinado estado nunca será alcançado.

Esses operadores temporais podem ser combinados aos operadores da lógica booleana ("e" = & , "ou" = | , "não" = ! , "se então" = → , "se e somente se" = ↔). Existem várias lógicas temporais: CTL*, CTL, LTL, ACTL, ATL, TCTL, etc.

A CTL* É um tipo de lógica temporal, que descreve propriedades de árvores de computação desenvolvidas a partir de sistemas de transição denominados *Estruturas de Kripke* (CLARKE et al., 1999). Na FIG.3.2, *a, b e c* são proposições atômicas. A árvore é formada designando-se um estado de uma estrutura de Kripke como estado inicial e desmembrando (através das transições) a estrutura numa árvore infinita contendo o estado inicial na raiz. A árvore mostra todas as possíveis execuções partindo do estado inicial, veja a FIG.3.2 e a FIG.3.3 (CLARKE et al., 1999).

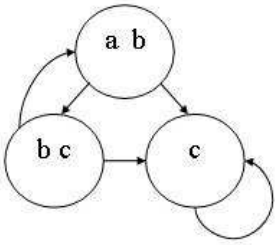


FIG.3.2: Gráfico de transição de estados ou modelo Kripke

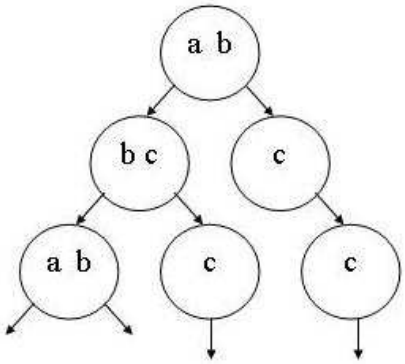


FIG.3.3: Árvore de computação

Os operadores temporais usados em CTL* são (CLARKE et al., 1999):

- *F* = no futuro (*in the future*),
- *G* = sempre (*globally*),
- *X* = na próxima vez (*next time*),
- *U* = até (*until*), *a U b*, a propriedade *a* vale até valer *b*, onde *b* é garantido valer no futuro;
- *R* = libera (*release*), *a R b*, a propriedade *b* vale enquanto não valer *a* incluindo o primeiro estado em que *a* valer.

Além dos operadores temporais e dos conectores booleanos, também são usados os quantificadores de caminho (*path quantifiers*) (CLARKE et al., 1999):

- *A* = para todo caminho (*always*) e

- E = existe pelo menos um caminho (*exist*).

Como exemplo, no estado inicial da FIG.3.3 é válida a fórmula $EG\ b$.

A CTL, também chamada de lógica de tempo ramificante, é um subconjunto da CTL*, assim como a LTL (*Linear Time Logic*) ou lógica de tempo linear. Na CTL, os operadores temporais atuam sobre os possíveis caminhos a partir de um dado estado. Desta forma, cada operador temporal deve ser imediatamente precedido por um quantificador de caminho.

Sejam AP o conjunto de nomes de proposições atômicas e P_i proposições atômicas pertencentes a AP, com $i = 1, 2, \ldots$ A sintaxe da lógica CTL é dada pela gramática a seguir, onde φ e ψ são fórmulas CTL:

$$\varphi, \psi \ ::= \ P_1 \mid P_2 \mid \ldots \mid !\varphi \mid \varphi \ \& \ \psi \mid EX\ \varphi \mid AX\ \varphi \mid E\ \varphi\ U\ \psi \mid A\ \varphi\ U\ \psi \tag{3.1}$$

Deve-se notar que a partir desse núcleo base pode-se derivar outras fórmulas:

- $EF\varphi \stackrel{def}{=\!=} E\ (True\ U\ \varphi)$
- $AF\ \varphi \stackrel{def}{=\!=} A\ (True\ U\ \varphi)$
- $EG\ \varphi \stackrel{def}{=\!=} !AF\ !\varphi$
- $AG\ \varphi \stackrel{def}{=\!=} !EF\ !\varphi$
- $E\ (\varphi\ R\ \psi) \stackrel{def}{=\!=} !A\ (!\varphi\ U\ !\psi)$
- $A\ (\varphi\ R\ \psi) \stackrel{def}{=\!=} !E\ (!\varphi\ U\ !\psi)$

Essas fórmulas derivadas são tidas como as mais comuns. A FIG.3.4 mostra exemplos de árvores de computação que ilustram esse operadores sendo usados relacionados aos estados iniciais.

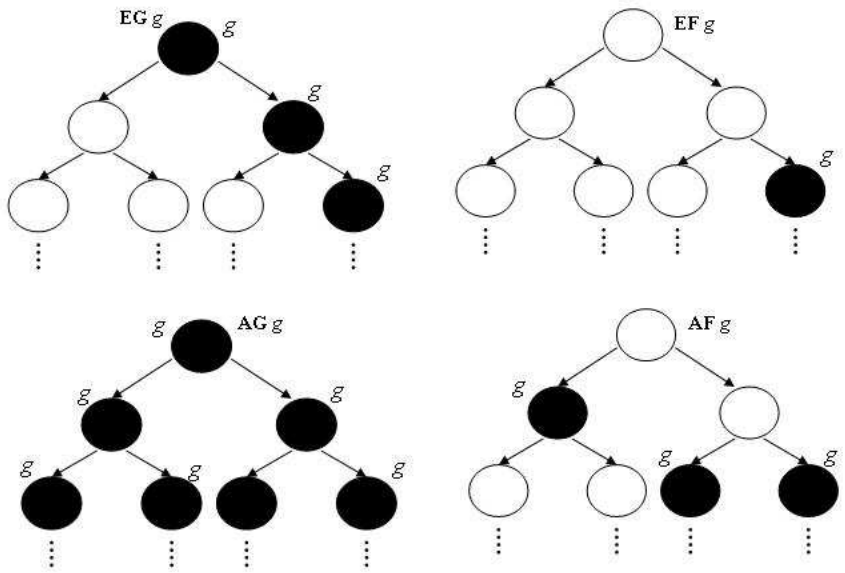


FIG.3.4: Operadores CTL mais comuns

A FIG.3.5 ilustra um exemplo de fórmula LTL, não pertencente a CTL.

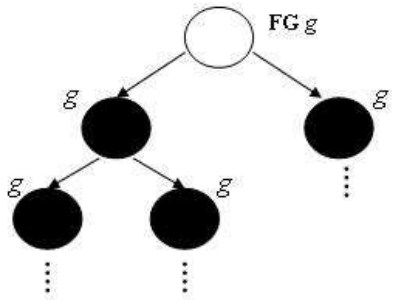


FIG.3.5: Exemplo de fórmula não CTL

A lógica TCTL (*Timed Computation Tree Logic*), proposta por ALUR et al. (1993), é uma extensão à lógica CTL, afim de expressar propriedades de tempo real, do tipo:

A cancela é aberta em menos de 5s após detectada a aproximação de um veículo.

A sintaxe da lógica TCTL é dada pela gramática a seguir, onde φ *e* ψ agora são fórmulas TCTL:

$$\varphi, \psi \quad ::= \quad P_1 \mid P_2 \mid \ldots \mid !\varphi \mid \varphi \,\&\, \psi \mid E\ \varphi\ U_{\sim c}\ \psi \mid A\ \varphi\ U_{\sim c}\ \psi \tag{3.2}$$

onde, $P_i \in AP, \sim \in \{<, >, \leq, \geq, =\}$ *e* $c \in N$.

Os operadores $E\_U\_$ e $A\_U\_$ da CTL correspondem, respectivamente, aos operadores $E\_U_{\geq 0}\_$ e $A\_U_{\geq 0}\_$ (sem restrição) da TCTL. Pode-se definir da mesma maneira que na lógica CTL, as seguintes derivações:

- $EF_{\sim c}\ \varphi \stackrel{def}{=\!=} E\ (True\ U_{\sim c}\ \varphi)$
- $AF_{\sim c}\ \varphi \stackrel{def}{=\!=} A\ (True\ U_{\sim c}\ \varphi)$
- $EG_{\sim c}\ \varphi \stackrel{def}{=\!=} !AF_{\sim c}\ !\varphi)$
- $AG_{\sim c}\ \varphi \stackrel{def}{=\!=} !EF_{\sim c}\ !\varphi)$
- $E\ (\varphi\ R_{\sim c}\ \psi) \stackrel{def}{=\!=} !A\ (!\varphi\ U_{\sim c}\ !\psi)$
- $A\ (\varphi\ R_{\sim c}\ \psi) \stackrel{def}{=\!=} !E\ (!\varphi\ U_{\sim c}\ !\psi)$

Por exemplo, a propriedade citada anteriormente, pode ser enunciada como:

$$AG(\ veículo \rightarrow\ AF_{\leq 5}\ cancela\ aberta)$$

Para maiores detalhes e referências no assunto, consultar CLARKE et al. (1999) e ALUR et al. (1993).

3.3 PADRÕES DE ESPECIFICAÇÃO DE PROPRIEDADES ESCRITAS EM CTL

A passagem da especificação informal para a formal tem sido um dos maiores obstáculos para a adoção da verificação de estados finitos (ex. verificação de modelos) na prática (DWYER et al., 1998). Neste sentido, DWYER et al. (1998) propõem um sistema de padrões de especificações que auxilia a vencer esta dificuldade. Este sistema consiste num conjunto de padrões de propriedades organizados em uma hierarquia, conforme a FIG. 3.6 abaixo, e de padrões de intervalos (limitados por eventos ou estados) dentro dos quais os padrões de propriedades podem ser válidos (veja FIG. 3.7). O sistema é proposto para formalismos baseados em estados, a saber, CTL e LTL (*Linear Time Logic*), e baseados em eventos, a saber, Expressões Regulares Quantificadas (*Quantified Regular Expressions*, QRE). Neste trabalho interessa a lógica CTL cuja abordagem por DWYER et al. (1998) é resumida na seqüência. Primeiramente, relaciona-se definições sucintas de cada padrão de propriedade, exibidos na FIG. 3.6.

Ausência : um dado estado / evento não ocorre dentro de certo intervalo.

Existência : um dado estado / evento deve ocorrer dentro de certo intervalo.

Existência limitada : um dado estado / evento deve ocorrer k vezes dentro de certo intervalo. Variações deste padrão especificam o número mínimo e / ou máximo de ocorrências.

Universalidade : Um dado estado / evento ocorre ao longo de todo um intervalo.

Precedência : Um estado / evento P deve ser sempre precedido pelo estado / evento S dentro de certo intervalo.

Resposta : Um estado / evento S deve sempre ser seguido por um evento P dentro de certo intervalo.

Cadeia de precedência : Uma seqüência de estados / eventos P1, P2, ... , Pn deve sempre ser precedida por uma seqüência de estados / eventos S1, S2, ... , Sn dentro de certo intervalo. Este padrão é uma generalização do padrão Precedência.

Cadeia de resposta : Uma seqüência de estados / eventos P1, P2, ... , Pn deve sempre ser seguida por uma seqüência de estados / eventos S1, S2, ... , Sn dentro de certo intervalo. Este padrão é uma generalização do padrão Resposta.

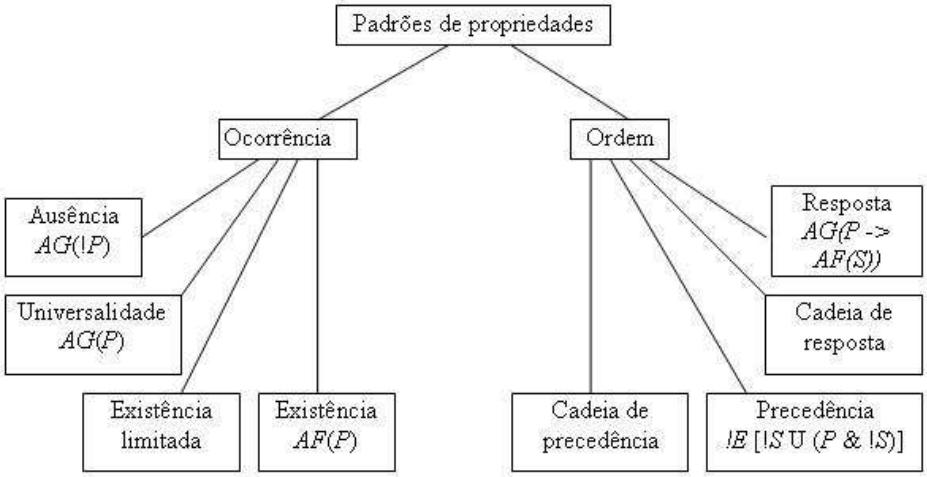


FIG.3.6: Hierarquia dos padrões de propriedade

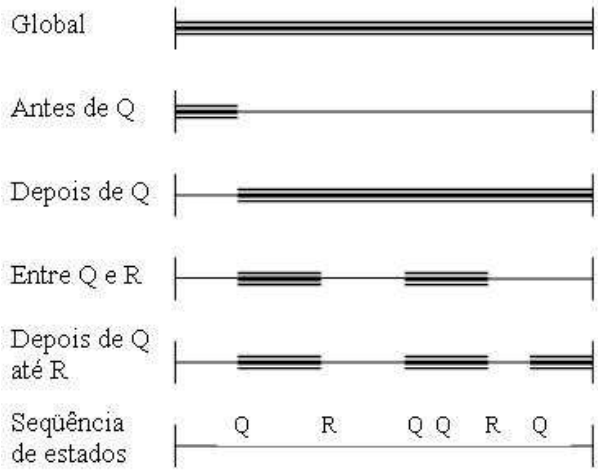


FIG.3.7: Padrões de intervalo

Nas TAB. 3.1 e TAB. 3.2 são exemplificadas algumas das combinações de padrões que dizem respeito a especificações em CTL, para melhor situar a abrangência de cada um deles. Sugere-se acompanhar a análise das citadas tabelas, observando-se a FIG. 3.7.

A abordagem proposta por DWYER et al. (1998) não pode ser aplicada na íntegra, diretamente, em conjunto com a ferramenta UPPAAL. Esta ferramenta utiliza um subconjunto da lógica CTL que possui dois limitantes em relação à lógica CTL propriamente dita:

- não permite a utilização de operadores em cascata. Por exemplo, a expressão

TAB.3.1: Padrão ausência e escopos.

Ausência	
Intento	Descrever a porção da execução de um sistema que é livre de certos eventos ou estados. Também conhecida como "Nunca".
Ex de mapeamento	P *é falso*.
Globalmente	$AG\ (!P)$
Antes de Q	$A\ [!P\ U\ (Q \mid AG\ (!Q))]$
Depois de Q	$AG\ (Q \rightarrow\ AG\ (!P))$
Entre Q e R	$AG\ (Q \rightarrow\ A\ [!P\ U\ (R \mid AG\ (!R))])$
Depois de Q até R	$AG\ (Q \rightarrow !E\ [!R\ U\ (P\ \&\ !R)])$

TAB.3.2: Padrão precedência e escopos.

Precedência	
Intento	Descrever relações entre um par de eventos/estados onde a ocorrência do primeiro é uma pré-condição para a ocorrência do segundo. Também conhecida como "Habilitação".
Ex de mapeamento	*S precede P*
Globalmente	$!E\ [!S\ U\ (P\ \&\ !S)]$
Antes de Q	$!E\ [(!S\ \&\ !Q)\ U\ (P\ \&\ !S\ \&\ !Q\ \&\ EF\ (Q))]$
Depois de Q	$!E\ [!Q\ U\ (Q\ \&\ !E\ [!S\ U\ (P\ \&\ !S)])]$
Entre Q e R	$AG(Q \rightarrow !E\ [(!S\ \&\ !R)\ U\ (P\ \&\ !S\ \&\ !R\ \&\ EF\ (R))])$
Depois de Q até R	$AG\ (Q \rightarrow !E\ [(!S\ \&\ !R)\ U\ (P\ \&\ !S\ \&\ !R)]).$

para *depois de Q não existe P*, $AG(Q \rightarrow AG\ !P)$, não é aceita. O operador $\rightsquigarrow$ (*leads to*) é uma exceção por significar: $P \rightsquigarrow Q \ \Leftrightarrow AG\ (P \rightarrow AF\ Q)$; e

- só permite dois operadores temporais: *globalmente* - G (*globally*) e no futuro - F (*in the future*). Não são permitidos os operadores *próximo* - X (*next*), o qual não faz parte da lógica TCTL, *até* - U (*until*), *a menos que* - W (*unless*) ou *habilita* - R (*release*).

3.4 ABORDAGEM DE OLIVEIRA (2006)

OLIVEIRA (2006) oferece uma visão de como a verificação de modelos se aplica à programação de CLPs. Em especial, são tratados os programas que correspondem a sistemas puramente discretos (que não de tempo real), não sendo imprescindível a modelagem do comportamento do CLP. O verificador de modelos utilizado é o SMV (MCMILLAN, 2001) e os trabalhos referenciados (MOON, 1994; GOURCUFF et al., 2006; RAUSCH e KROGH, 1998; DE SMET e ROSSI, 2002; LAMPÉRIÈRE-COUFFIN et al., 1999) se enquadram nesta perspectiva, utilizando por vezes, diferentes versões do SMV (Cadence SMV e NuSMV). As contribuições do trabalho residem numa proposta de tradução de programas escritos em FBD para um sistema escrito na linguagem da ferramenta SMV e noutra proposta para sistematizar a extração de especificações escritas em CTL a partir de um formalismo de especificação, denominado Matriz de Causa e Efeito (MCE). A escolha da linguagem a partir da qual os programas seriam modelados e a escolha da MCE como documento de extração foram orientadas pelo resultado da procura de exemplos reais junto ao CENPES/Petrobrás.

O modelo proposto em OLIVEIRA (2006) é baseado em módulos padrões (ou *carimbos* como é citado originalmente no texto) de memórias e temporizadores. Por exemplo, o código abaixo corresponde ao módulo padrão do temporizador *DI* (vide FIG. 2.7(a)).

```
MODULE DI(entrada) VAR
  estado : {parado,contando,terminado};
ASSIGN
  init(estado) := parado;
  next(estado) := case
              estado=parado & !entrada: {parado};
              estado=parado & entrada : {contando};
```

```
              estado=contando & entrada: {terminado,contando};
              estado=contando & !entrada: {parado};
              estado=terminado & entrada : {terminado};
              estado=terminado & !entrada : {parado};
              1 : estado;
             esac;
DEFINE
   saida := estado = terminado;
```

Cabe observar, que o modelo do temporizador abstrai seu comportamento real, não sendo possível a verificação de propriedades que envolvam o valor do tempo. Da mesma forma são definidos módulos padrões para os temporizadores *DT* e *PO*.

Já o código a seguir corresponde ao módulo padrão do elemento de memória *set/reset* com prioridade para *set* (vide FIG. 2.6(a)).

```
MODULE setreset(entrada_set,entrada_reset) VAR
  estado : boolean;
ASSIGN
  init(estado) := 0;
  next(estado) := case
               entrada_set & !entrada_reset: {1};
               entrada_reset & !entrada_set: {0};
               entrada_set & entrada_reset : {1};
               1 : estado;
               esac;
DEFINE
   saida := estado;
```

Cada elemento (memória ou temporizador) do programa é associado a uma instância do módulo padrão na declaração principal. No programa do SMV do ANEXO 2 (Modelo3) pode-se ver como os padrões são instanciados. Há algumas diferenças nos módulos padrões do programa as quais são discutidas na seção 7.1, o que não prejudica a visualização dos módulos apresentados.

Os procedimentos para especificação elaborados por OLIVEIRA (2006) tomam como base a MCE. Esta matriz, por sua construção, não exprime padrões de intervalos complexos e sim propriedades da forma *o evento x ocorreu, então evento y vai ocorrer*

as quais são suficientes e adequadas a operações de desligamento em SIS. Não são adequadas a propriedades com encadeamento mais complexo no tempo, como por exemplo:

> *Se o motor da bomba não estiver sobrecarregado e se, tendo a sobrecarga ocorrido, a indicação deste fato estiver* resetada*, então o mesmo poderá ser ligado.*

A primeira especificação genérica sugerida por OLIVEIRA (2006) é: sempre que a condição para ativar uma variável *VAR* for verdadeira (C_{ON}) e a condição para desátivá-la for falsa ($!C_{OFF}$) então, para todos os futuros caminhos, *VAR* será verdadeira (ativada), ou em lógica CTL:

$$AG\ ((C_{ON}\ \&\ !C_{OFF}) \quad \rightarrow \quad AF\ VAR) \tag{3.3}$$

Onde C_{ON} significa a condição ou conjunto de condições, para a variável *VAR* estar ativada e C_{OFF} significa a condição para a mesma estar desativada. A escrita na ferramenta UPPAAL ficaria como: $(C_{ON}\ \&\ !C_{OFF}) \rightsquigarrow VAR$, e equivale ao padrão resposta com escopo global de acordo com DWYER et al. (1998), vide seção 3.3.

Deve-se observar que a expressão (C_{ON} & $!C_{OFF}$) reflete a existência de uma prioridade para $!C_{OFF}$ o que nem sempre acontece.

A segunda especificação sugerida por OLIVEIRA (2006) é: a variável VAR não passa de falsa a verdadeira sem que, neste caminho, a condição de ativação seja satisfeita, ou em lógica CTL:

$$AG\ (!VAR \quad \rightarrow \quad !E\ [!C_{ON}\ U\ (VAR\ \&\ !C_{ON})]) \tag{3.4}$$

Essa condição não pode ser transcrita diretamente para a ferramenta UPPAAL em função das restrições comentadas anteriormente. Pode existir um caminho indireto, utilizando-se autômatos auxiliares, e sistematizado para contornar tais limites, porém este não foi alcançado ao longo deste trabalho. Além disso, esta especificação não representa uma propriedade expressa diretamente por uma MCE. Por outro lado, ela pode ser bastante útil para explorar possíveis falhas e / ou erros de projeto. Isso porque quando não for satisfeita irá mostrar um traço (caminho) que não estava previsto, pelo qual chega-se ao mesmo efeito.

As outras 2 especificações apresentadas são referentes à desativação de uma variável, e podem ser diretamente escritas no padrão das duas primeiras, a saber:

$$AG\ ((C_{OFF}\ \&\ !C_{ON}) \quad \rightarrow \quad AF\ !VAR) \tag{3.5}$$

$$AG\ (VAR \quad \rightarrow \quad !E\ [!C_{OFF}\ U\ (!VAR\ \&\ !C_{OFF})]) \tag{3.6}$$

Não se pode deixar de destacar as limitações da abordagem de OLIVEIRA (2006) que motivaram este trabalho, quais sejam, o não tratamento de sistemas de tempo real e a automatização dos procedimentos de modelagem e escrita das especificações.

3.5 RESUMO DO CAPÍTULO

Neste capítulo foi dada uma visão geral da verificação de modelos e detalhada a parte da especificação formal. Após, apresentou-se o resumo das contribuições de OLIVEIRA (2006), que motivaram este trabalho.

No próximo capítulo será tratada a outra parte concernente a verificação de modelos aplicada a sistemas de tempo real, a linguagem da descrição dos modelos.

4 AUTÔMATOS TEMPORIZADOS

Este capítulo introduz a base teórica dos autômatos temporizados e as variações utilizadas pelo verificador adotado. Com base nos estados simbólicos apresentados dentro dessa teoria, expõe a estrutura de dados usada para armazená-los, concluindo assim a descrição da linguagem formal para modelagem. Leitores familiarizados com o tema podem pular este capítulo.

4.1 INTRODUÇÃO

Os autômatos temporizados, introduzidos por ALUR e DILL (1994), constituem um formalismo clássico para modelar sistemas de tempo real. Será utilizada a variação introduzida por T. A. HENZINGER et al. (1992), denominada de *Autômatos Seguros Temporizados* (*Timed Safety Automata*), a qual contempla a noção de progresso de uma forma mais intuitiva (por intermédio de invariantes, apresentados na seção 4.2).

Na sua essência, um autômato (finito) temporizado é um grafo contendo um conjunto finito de nós, ou lugares de controle, e um conjunto finito de arcos (transições entre os lugares de controle) rotulados. Essa estrutura é estendida por variáveis reais as quais modelam os relógios do sistema. Todos os relógios do sistema são inicializados por zero e evoluem sincronamente a uma mesma taxa. Os relógios podem ser zerados quando uma transição é realizada.

4.2 SINTAXE

Sejam C o conjunto de variáveis reais tidas como relógios; $B(C)$ o conjunto de restrições de relógio as quais são fórmulas conjuntivas de restrições singulares da forma $x \sim y$ ou $x - y \sim n$ com $\sim \in \{\leq, <, =, >, \geq\}$, $x, y \in C$ e $n \in \mathbb{N}$; e Σ o alfabeto finito de ações, que são atribuições de valor a uma variável lógica do sistema.

Definição 4.1. *(BENGTSSON e YI, 2004) Um autômato temporizado A é uma tupla $\langle N, l_0, E, I \rangle$ onde*

- *N é um conjunto finito de lugares (ou nós) de controle;*
- *$l_0 \in N$ é o lugar de controle inicial;*

- *$E \subset N \times B(C) \times \Sigma \times 2^C \times N$ é o conjunto de arcos (transições entre lugares de controle) rotulados;*
- *$I : N \longrightarrow B(C)$ associa invariantes (restrições de relógios) aos lugares de controle.*
- *Será escrito $l \xrightarrow{g,a,r} l'$ quando $\langle l, g, a, r, l' \rangle \in E$, onde*
  - *l e $l' \in N$ são lugares de controle;*
  - *$g \in B(C)$ é uma guarda associada a um arco;*
  - *$a \in \Sigma$ é uma ação; e*
  - *r é um subconjunto de C cujos relógios são zerados quando da realização da transição.*

4.3 SEMÂNTICA

Uma avaliação (momentânea) dos relógios é uma aplicação $v : C \longrightarrow \mathbb{R}_+^{|C|}$. Por exemplo, sejam x e y variáveis relógios disparadas em instantes distintos as quais não são zeradas ao longo do tempo e u e v avaliações distintas. Se $u(x) = 2,1$ e $u(y) = 3,4$, então $v(x) = 4,0 \rightarrow v(y) = 5,3$, pois a diferença entre x e y se mantém constante até que um deles comece nova contagem. Ao escrever $v \in g$, entenda-se que os valores dos relógios sob a avaliação (momentânea) v satisfazem à guarda $g \in B(C)$. Para $d \in \mathbb{R}_+$, $v + d$ denotará o mapeamento $v(x) + d$ para todos os relógios $x \in C$. Para $r \subset C$, $[r \mapsto 0]v$ denota uma atribuição de relógios que faz o mapeamento de todos os relógios pertencentes a r para 0 e que mantém a avaliação v para o conjunto de relógios C/r.

A semântica de um autômato temporizado é definida como um sistema de transição, onde um estado ou configuração consiste no lugar de controle corrente (l) e nos valores correntes dos relógios (v), isto é, no par (l, v). Existem dois tipos de transição entre estados: uma transição por ação, quando um arco habilitado é executado, e uma transição por tempo decorrido, quando o lugar de controle permanece o mesmo e o valor dos relógios é incrementado.

Definição 4.2. *(BENGTSSON e YI, 2004) A semântica de um autômato temporizado é um sistema de transição (também conhecido como sistema de transição temporizado) onde estados são pares (l,u), e as transições são definidas pelas regras:*

- *Transição por tempo decorrido: $(l, v) \xrightarrow{d} (l, v + d)$ se $v \in I(l)$ e $v + d \in I(l)$ para um real não negativo $d \in \mathbb{R}_+$.*

- *Transição por ação:* $(l, v) \xrightarrow{a} (l', v')$ *se* $l \xrightarrow{g,a,r} l'$, $v \in g$, $v' = [r \mapsto 0]v$ *e* $v' \in I(l)$.

Exemplo 4.1. *A FIG 4.1 ilustra a modelagem do comportamento de uma luminária sob a intervenção de um usuário qualquer. Sua lâmpada se acende com baixa ou alta luminosidade. Existe um único botão que comanda o comportamento da luminária. Estando a luminária desligada, ao apertar o botão uma vez, a luz acenderá com baixa luminosidade, a partir daí, apertando novamente o botão em menos de 5 s, a luz tornar-se-á brilhante, caso contrário, o segundo aperto implicará no desligamento da luminária.*

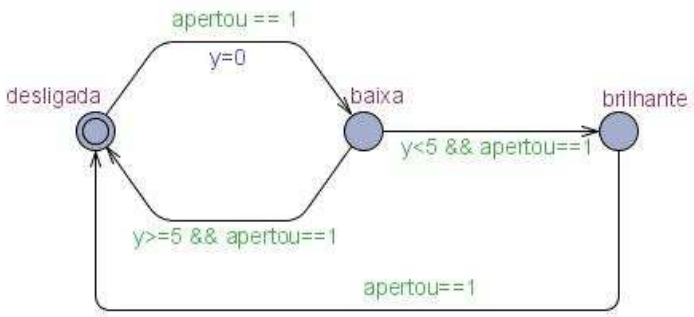


FIG.4.1: Modelo do comportamento da luminária

Neste exemplo tem-se:

- $N = \{desligada, baixa, brilhante\}$;
- $l_0 = desligada$, indicado pelo duplo círculo;
- $I =$ não existe invariante neste exemplo;
- Exemplo de dois possíveis estados: *(desligada, y=0) e (baixa, y=10)*;
- Exemplo de um arco: $baixa \xrightarrow{y \leq 5 \&\& apertou==1,\ \emptyset,\ \emptyset} brilhante$;
- Exemplo de transição por tempo decorrido: $(baixa, y = 0) \xrightarrow{5} (baixa, y = 5)$; e
- Exemplo de transição por ação: $(desligada, y = 2) \xrightarrow{apertou==1,\ \emptyset, y=0} (baixa, y = 0)$.

4.4 REDE DE AUTÔMATOS TEMPORIZADOS EM PARALELO

No exemplo 4.1, percebe-se que a luminária, sendo um objeto, é inerte e para se observar uma evolução no comportamento da mesma, o sistema precisa ter, em

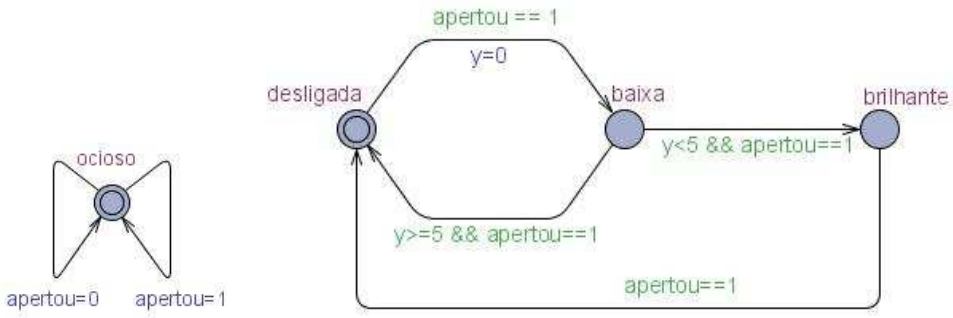


FIG.4.2: Modelo do comportamento do usuário em paralelo à luminária

paralelo, o modelo do usuário comunicando-lhe sua ação FIG 4.2. Escolheu-se realizar tal comunicação por intermédio da variável lógica *apertou*.

Os dois autômatos da FIG. 4.2 evoluindo em paralelo, constituem um exemplo de uma rede de autômatos temporizados cuja definição formal é colocada a seguir.

Definição 4.3. *Uma rede de n autômatos temporizados é uma coleção da forma* $A_i = \langle N_i, l_i^0, E_i, I_i \rangle, 0 \leq i \leq n$, *com um conjunto comum de relógios e ações. Um vetor de lugares de controle é um vetor* $\bar{l} = (l_1, ..., l_n)$. *O vetor de lugares de controle inicial fica* $\bar{l}_0 = (l_1^0, ..., l_n^0)$.

As funções invariantes de cada autômato são compostas em uma função comum sobre os vetores de lugares de controle $I(\bar{l}) = \wedge_i, I_i(l_i)$. *Será escrito* $\bar{l}[l_i'/l_i]$ *para denotar o vetor cujo i-ésimo elemento* l_i *de* $\bar{l}$ *é substituído por* l_i'.

A semântica da rede de autômatos temporizados é dada em função das seguintes regras de transição:

- $(\bar{l}, v) \longrightarrow (\bar{l}, v + d)$ *se* $\forall\ d' : 0 \leq d' \leq d \Rightarrow v + d' \in I(\bar{l})$, *para um real não negativo* $d \in \mathbb{R}_+$.
- $(\bar{l}, v) \longrightarrow (\bar{l}[l_i'/l_i], v')$ *se existe* $l_i \xrightarrow{\tau,\ g,\ r} l_i'$ *tal que* $v \in g$, $v' = [r \mapsto 0]v$ *e* $v' \in I(\bar{l})$.

Considerando agora o exemplo 4.1 e sua modelagem de acordo com a FIG 4.2, tem-se:

- $S = \{$(*Luminaria.desligada*, *Usuario.ocioso*), (*Luminaria.baixa*, *Usuario.ocioso*), (*Luminaria.brilhante*, *Usuario.ocioso*)$\}$;
- $s_0 =$ (*desligada*, *ocioso*), omitindo-se o nome do autômato na designação de cada lugar de controle que compõe o par;

- Alguns estados: (*desligada*, *ocioso*, $y = 0$), (*baixa*, *ocioso*, $y = 10$); e
- Uma seqüência possível de estados (omitindo-se o lugar de controle do *usuário*): (*desligada*, $y = 0$) $\rightarrow$ (*desligada*, $y = 10$) $\rightarrow$ (*baixa*, $y = 0$) $\rightarrow$ (*baixa*, $y = 4$) $\rightarrow$ (*brilhante*, $y = 4$) $\rightarrow$ (*brilhante*, $y = 100$).

4.5 REGIÕES, ZONAS E SEMÂNTICA SIMBÓLICA

Como os relógios são variáveis que assumem valores reais, o sistema de transição de um autômato temporizado é infinito, não sendo um modelo adequado para verificação automática. A fundação para torná-lo adequado vem da noção de equivalência de região sobre as atribuições de relógios (ALUR e DILL, 1994).

Definição 4.4. *(BENGTSSON e YI, 2004) Seja k uma função, chamada de majoração de relógios, que mapeia cada relógio x para um número natural $k(x)$ (isto é, o limite superior de x). Para um número real d, seja $\{d\}$ a notação para a sua parte fracionária e $\lfloor d \rfloor$ a notação para a parte inteira. Duas avaliações de relógio u, v são regiões equivalentes, o que se denota por $u \cong_{k(x)} v$, se*

1. *para todo x, ou $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$ ou ambos $u(x) > k(x)$ e $v(x) > k(x)$,*
2. *para todo x, se $u(x) \leq k(x)$ então $\{u(x)\} = 0$ se $\{v(x)\} = 0$ e*
3. *para todo x e y, se $u(x) \leq k(x)$ e $u(y) \leq k(y)$ então $\{u(x)\} \leq \{u(y)\}$ se $\{v(x)\} \leq \{v(y)\}$.*

A equivalência de regiões é indexada pela máxima constante k. Uma vez que essas constantes são estabelecidas pelas restrições de relógios de um determinado autômato, pode-se omitir o índice escrevendo somente $\cong$. Uma classe de equivalência $[u]$, induzida por $\cong$ é chamada de *região*, onde $[u]$ denota o conjunto de avaliações de relógios região equivalente a u.

Assim, a FIG. 4.3 identifica as regiões de equivalência para o exemplo 4.1. As avaliações $v(y) = 2,3456$ e $u(y) = 2,1978$ são regiões equivalentes, ou $v \cong_5 u$, pois pertencem a mesma região F. Da mesma forma, sendo $v'(y) = 6,3456$ e $u'(y) = 9,1978$, $v' \cong_5 u'$, pois ambas pertencem a região M, exemplificando o primeiro item da Definição 4.4.

E, se u e v são regiões equivalentes e $u(y) \leq k(y) = 3$ então $(\{u(x)\} = 0 \rightarrow \{v(x)\} = 0)$. Por outro lado, se $u(y) \geq k(y) = 5$ nada se pode afirmar, pois se $u(y) = 6,0$ (com

$\{u(x)\} = 0$), pode-se ter $v(y) = 9,1234$ (destacando-se que u e v pertencem a região M), mas $\{v(x)\} \neq 0$. Isto esclarece o que foi apontado no segundo item da Definição 4.4.

O terceiro item não pode ser ilustrado pela FIG. 4.3 pois o sistema contemplado por esta só possui um relógio.

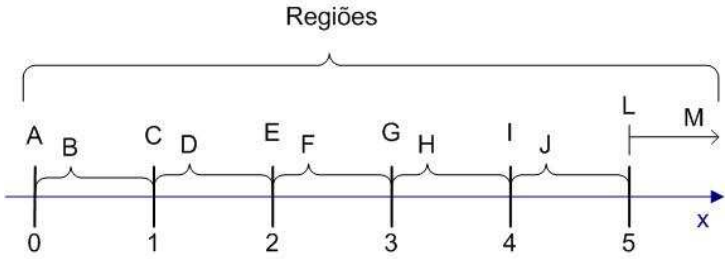


FIG.4.3: Conjunto de regiões definidas pelo relógio y e a máxima constante 5.

Um modelo de estados finitos, chamado de grafo de regiões ou autômato de regiões (BENGTSSON e YI, 2004), é construído com base no conceito de regiões de equivalência, a partir do autômato temporizado original. A relação de transição entre os estados simbólicos deste modelo é definida a seguir.

Definição 4.5. (BENGTSSON e YI, 2004) *Relação de transição* $\Rightarrow$ *entre os estados simbólicos (as classes):*

- $(l, [u]) \Rightarrow (l, [v])$ *se* $(l, u) \xrightarrow{d} (l, v)$ *para um real positivo d, e*
- $(l, [u]) \Rightarrow (l', [v])$ *se* $(l, u) \xrightarrow{a} (l', v)$ *para uma ação a.*

A idéia básica que advém desta teoria é que, dado um autômato A, dois estados simbólicos $(l, [u])$ e $(l, [v])$ podem realizar as mesmas transições $\Rightarrow$.

Deve-se notar que a relação de transição $\Rightarrow$ é finita, assim como o conjunto de *regiões*. Porém, mostra-se que o número de regiões cresce exponencialmente com o número de relógios e com o valor da(s) máxima(s) constante(s) (ALUR e DILL, 1994). A FIG.4.3 e o exemplo da FIG 4.4 dão uma idéia dessa relação. Na primeira, encontram-se um relógio e a sua máxima constante igual a 5, gerando 12 regiões. Na segunda, encontram-se dois relógios, sendo a máxima constante do primeiro igual a 2 e a do segundo igual a 1, gerando 28 regiões (6 pontos, ex. (0,0); 14 segmentos e semi-retas abertos, ex. (0 < x = y < 1); 8 regiões abertas, ex. (0 < x < y < 1).

Uma melhoria na representação do espaço de estados segundo esta teoria foi desenvolvida com base na representação de *zonas* e *grafo de zonas* (T. A. HENZINGER

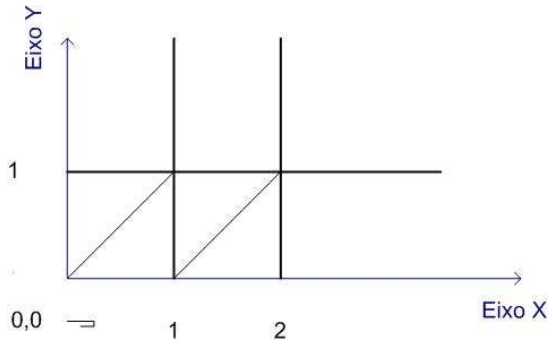


FIG.4.4: Grafo de regiões para dois relógios x e y, com máximas constantes iguais a 2 e 1, respectivamente.

et al., 1992). No grafo de zonas, ao invés de regiões, zonas são usadas para denotar os estados simbólicos.

A FIG 4.5 ilustra a adequação do exemplo 4.1, página 45, a este conceito, omitindo-se mais uma vez o usuário. Deve-se destacar que nesta situação, a quantidade de estados simbólicos não depende do valor da máxima constante (limite superior) como ocorre usando-se grafo de regiões. Se a máxima constante (limite superior) para y fosse 10, ter-se-iam os mesmos 8 estados simbólicos enquanto a representação por grafo de regiões implicaria mais de 50 estados.

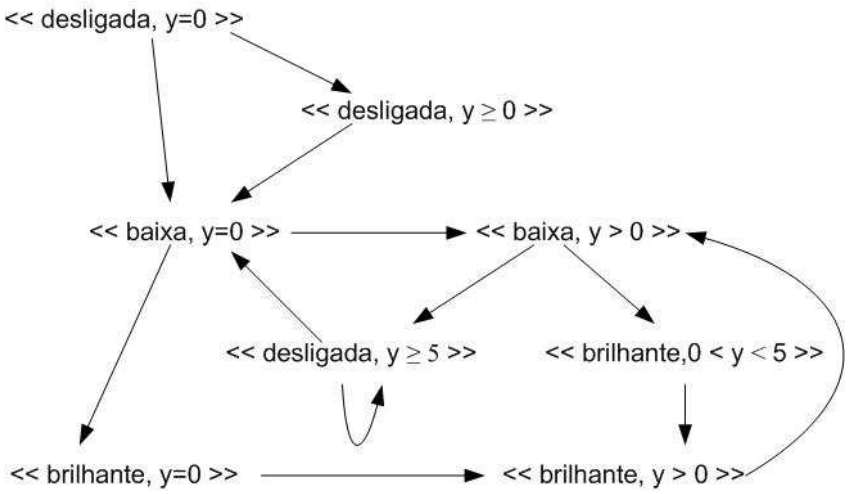


FIG.4.5: Grafo de zonas para o exemplo da luminária.

Para se obter maiores detalhes acerca desta representação, seus formalismos e algoritmos, indica-se o artigo de BENGTSSON e YI (2004).

4.6 AUTÔMATOS TEMPORIZADOS DO UPPAAL

Algumas particularidades são adicionadas às definições anteriores, na modelagem do autômato temporizado em conformidade com a ferramenta UPPAAL (BENGTSSON et al., 1996):

- As restrições de relógios usadas como invariantes estão limitadas às formas $x < n$ ou $x \leq n$, onde n é um número natural. A notação do UPPAAL é um pouco diferente da usada na seção 3.2. O exemplo:

  $$AG(\ veículo \rightarrow\ AF_{\leq 5}\ cancela\ aberta)$$

  na notação do UPPAAL fica:

  $$AG(\ veículo \rightarrow\ AF\ ((x \leq 5)\ \&\ cancela\ aberta))$$

  onde x é uma variável relógio usada para contar o tempo a partir da detecção da aproximação do veículo.

- Os arcos entre lugares de controle são rotulados ainda por etiquetas de sincronismo "m!" e "m?" designadas por canais (de sincronização). O sinal "!" identifica o canal emissor da mensagem enquanto "?" identifica o receptor e "m" é o nome do canal. Quando o arco rotulado pelo canal emissor ("!") é executado, todos os arcos rotulados pelo canal receptor são executados sincronamente. Quando só existe uma dupla emissor/receptor tem-se um canal binário. Quando existem um emissor e dois ou mais receptores, tem-se um canal múltiplo ou de *broadcast*. Com isso um arco entre lugares de controle passa a ser rotulado por $l \xrightarrow{g,\ a,\ r,\ canal} l'$

- Três tipos possíveis de lugares de controle estão disponíveis e são exibidos na FIG. 4.6:

  1. Urgentes, FIG. 4.6(a), nos quais o tempo não é permitido passar. De acordo com a semântica, são equivalentes a adicionar-se um relógio extra x, o qual é levado a zero em todos os arcos de chegada aos lugares deste tipo, e nestes lugares são adicionados os invariantes $x \leq 0$.

2. Compromissados (ou *commited*), FIG. 4.6(b), os quais possuem restrições mais severas que as dos estados *urgentes*. Lugares de controle compromissados também não permitem a passagem do tempo, além disso, têm prioridade de execução sobre os demais.
3. Comuns, FIG. 4.6(c) são aqueles que não são classificados em nenhum dos dois tipos anteriores.

- Para o canal de sincronismo há a declaração *urgent synchronization*. O canal declarado desta forma diferencia-se do declarado normalmente, pela imposição da execução do arco que o contém, assim que a guarda deste seja satisfeita.

(a) Lugar de controle urgente

(b) Lugar de controle compromissado

(c) Lugar de controle comum

(d) Lugar de controle inicial

FIG.4.6: Notação do UPPAAL para os lugares de controle.

A semântica da definição 4.3 fica expandida com a introdução da seguinte regra de transição:

$(\bar{l}, v) \longrightarrow (\bar{l}[l'_i/l_i, l'_j/l_j], v')$ se existem $l_i \xrightarrow{g_i,\, a,\, r_i,\, c?} l'_i$ e $l_j \xrightarrow{g_j,\, a,\, r_j,\, c!} l'_j$ tais que $v \in (g_i \wedge g_j)$, $v' = [r_i \cup r_j \mapsto 0]v$ e $v' \in I(\bar{l}[l'_i/l_i, l'_j/l_j])$.

Como exemplo de um emprego do canal de sincronismo, substitui-se a variável *apertou* na FIG. 4.2, pelo canal de sincronismo de mesmo nome, na FIG. 4.7.

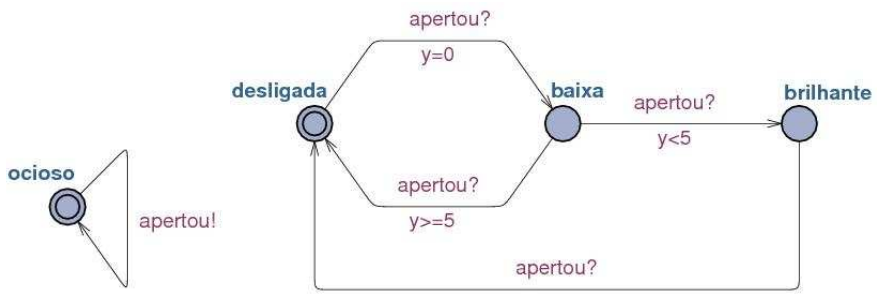


FIG.4.7: Modelo da FIG. 4.2 alterado pelo emprego do canal de sincronismo.

(BEHRMANN et al., 2004) é um manual e (BENGTSSON et al., 1996) é um documento os quais tratam da orientação da utilização da ferramenta UPPAAL e prestam esclarecimentos adicionais acerca da mesma.

4.7 ESTRUTURA DE DADOS DBM

A ferramenta UPPAAL se utiliza de Matriz de Limites de Diferenças (*Difference Bounded Matrix*) DBM como estrutura de dados para armazenar os relógios e suas evoluções com o tempo. Aliás, é a representação simbólica por zonas que tem a importante propriedade de permitir tal utilização. A seguir, introduz-se a descrição da sua estrutura básica de acordo com BENGTSSON e YI (2004).

Da seção 4.2, tem-se que uma restrição de relógio é uma fórmula conjuntiva de restrições singulares da forma $x \sim m$ ou $x - y \sim n$ com $\sim \in \{\leq, <, =, >, \geq\}$, x *e* $y \in C$ e m, $n \in \mathbb{N}$. Para definir uma forma mais abrangente e unificada de restrições de relógios, é introduzido um relógio de referência $\mathbf{0}$ cujo valor é constante e igual a 0. Seja $C_0 = C \cup \{\mathbf{0}\}$. Então, qualquer zona $D \in B(C)$ pode ser reescrita como uma conjunção de restrições singulares de relógio da forma $x - y \leq n$ com $\leq \in \{\leq, <\}$, x *e* $y \in C_0$ e $n \in \mathbb{Z}$ (BENGTSSON e YI, 2004). Ou seja, as duas formas de restrições singulares, colocadas no início da seção 4.2, ficam reduzidas a somente uma, $x - y \leq n$.

Como conseqüência, se a representação por zonas possui duas restrições de relógios com o mesmo par de variáveis, somente a intersecção das duas é necessária. Posto isto, uma representação por zonas pode ser feita usando somente $|C_0|^2$ (cardinalidade de C_0 ao quadrado) restrições atômicas da forma $x - y \leq n$, tais que cada par de relógios $x - y$ é mencionado uma única vez. Assim, é possível armazenar zonas usando matrizes de dimensão $|C_0| \times |C_0|$, onde cada elemento da matriz corresponde a uma restrição atômica, daí o nome *Matriz de Limites de Diferenças (DBM)*. D_{ij} é o elemento (i, j) da DBM ao qual corresponde a zona D.

Para construir a DBM da zona D, numeram-se todos os relógios de C_0 como $0, \ldots, n$, sendo o índice para $\mathbf{0}$ igual a 0. Cada linha é associada a um relógio e os limites inferiores em relação a diferença entre ele e os outros relógios. Desta forma, uma coluna corresponde aos limites superiores. Para as diferenças não limitadas, tem-se $D_{ij} = \infty$, onde ∞ é um valor específico para esta situação.

Como exemplo, considere a zona $D = x - \mathbf{0} < 20 \;\&\; y - \mathbf{0} \leq 20 \;\&\; y - x \leq 10 \;\&\; x - y \leq -10 \;\&\; \mathbf{0} - z < 5$. A numeração dos relógios é dada na seguinte ordem $\mathbf{0}, x, y, z$. A DBM correspondente fica (BENGTSSON e YI, 2004):

$$M(D) = \begin{pmatrix} (0,\leq) & (0,\leq) & (0,\leq) & (5,<) \\ (20,<) & (0,\leq) & (-10,\leq) & \infty \\ (20,\leq) & (10,\leq) & (0,\leq) & \infty \\ \infty & \infty & \infty & (0,\leq) \end{pmatrix}$$

Os algoritmos, tais como o de análise de alcançabilidade, são construídos com base em duas operações: comparação e adição. A comparação é definida da seguinte forma: $(n,\leq) < \infty$, $(n_1,\leq) < (n_2,\leq)$ *se* $n_1 < n_2$ *e* $(n,<) < (n,\leq)$. A adição, por sua vez, é definida como: $b_1 + \infty = \infty$, $(m,\leq) + (n,\leq) = (m+n,\leq)$ *e* $(m,\leq) + (n,\leq) = (m+n,<)$.

Existem outras estruturas de dados utilizadas na verificação de modelos de autômatos temporizados. Aliás, a verificação de modelos ganhou seu espaço (teve as primeiras ferramentas computacionais sendo usadas efetivamente) graças à utilização dos Diagramas de Decisão Binários Ordenados (*Ordered Binary Decision Diagrams* - OBDDs)(CLARKE et al., 1999). Entre as ferramentas que dele se utilizam está a pioneira SMV, e suas derivadas SMV Cadence (MCMILLAN, 1999) e NuSMV. Esta estrutura, no entanto, não é adequada ao armazenamento de zonas, e tratam exclusivamente de sistemas a eventos discretos. Têm sido desenvolvidas e apresentadas adaptações a OBDDs que suportam o armazenamento e o tratamento de modelos de sistemas de tempo real (WANG, 2004), resultando em variações de verificadores que não entraram no escopo deste trabalho.

4.8 RESUMO DO CAPÍTULO

Neste capítulo realizou-se uma explanação sobre a teoria dos autômatos temporizados, sobre a variação desta adotada pelo verificador de modelos UPPAAL e sobre a estrutura de dados que lhes é propícia.

No capítulo seguinte são apresentados resumos dos trabalhos selecionados, acompanhados de algumas críticas e comparações. Eles serão a referência durante a análise do modelo da equipe da UFCG (GORGÔNIO et al., 2006), assunto do capítulo 6.

5 MODELOS TEMPORIZADOS DE CLPS PARA VERIFICAÇÃO

Este capítulo contém um levantamento bibliográfico do que recentemente tem sido publicado em verificação de programas de CLPs considerando-se propriedades de tempo real. Esta pesquisa bibliográfica foi a primeira etapa desenvolvida neste trabalho e serviu para subsidiar as análises realizadas no andamento posterior do trabalho.

5.1 INTRODUÇÃO

Foi realizada uma pesquisa bibliográfica e levantados os trabalhos que lidam com verificação de modelos para validar programas de CLP, utilizando autômatos temporizados (ALUR e DILL, 1994). A pesquisa não se restringiu ao portal *Web of Science* [1], mas os resultados nele obtidos, expostos na TAB. 5.1 a seguir, oferecem uma boa medida da distribuição dos temas entre as publicações.

TAB.5.1: Resultado da pesquisa no *Web of Science realizada em 13/out/07*

N_o	Combinação de termos	Ocorrências
1	*validation*	52.602
2	*"formal validation"*	56
3	*"formal verification"*	482
4	*formal* & *validation* & *verification*	86
5	*"timed automata"*	295
6	*"PLC programs"*	4
7	*formal* & (*validation* | *verification*)	1492
8	*formal* & (*validation* | *verification*) & *"timed automata"*	32
9	*formal* & (*validation* | *verification*) & *PLC*	5

[1] http://scientific.thomson.com/index.html

Na pesquisa destacaram-se os seguintes trabalhos: MADER e WUPPER (1999), WILLEMS (1999), ZOUBEK (2004) e MOKADEM (2006). Para se chegar nos sistemas modelados por autômatos temporizados (do UPPAAL), estes trabalhos consideraram como fontes os programas de controle escritos utilizando-se as seguintes linguagens da IEC 61131-3:

- Mader (MADER e WUPPER, 1999) e Willems (WILLEMS, 1999): Lista de Instruções (*Instruction List* - IL);
- Mokadem (MOKADEM, 2006): Cartas de Funções Seqüenciais (*Sequential Function Charts*) - SFC e
- Zoubek (ZOUBEK, 2004): Diagrama de contatos (*Ladder Diagram* - LD).

A Figura 5.1 ilustra o posicionamento destes trabalhos.

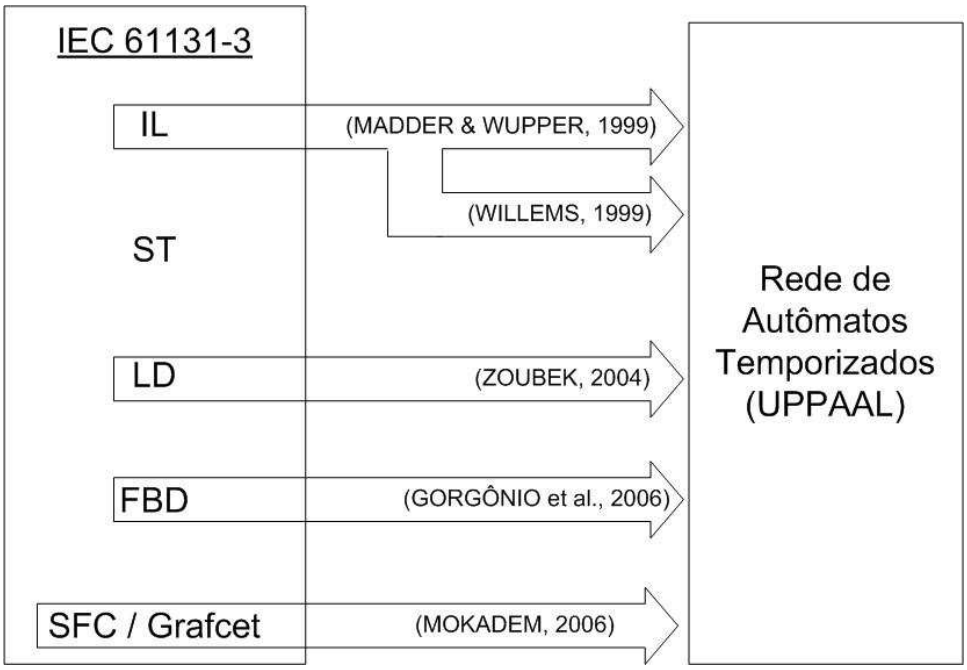


FIG.5.1: Abordagens para modelagem de programas de CLPs em autômatos temporizados.

O trabalho de GORGÔNIO et al. (2006) é um relatório técnico tratado no capítulo 6 o qual expõe a modelagem proposta pela equipe da UFCG dentro do Projeto SIS (vide Seção 2.1). Apesar do projeto não contemplar SFC e IL como linguagens fonte, a forma como os modelo são gerados e como é tratada a especificação, são interessantes alternativas como poderá ser visto nas seções seguintes. No caso da especificação,

esquemas alternativos de sua escrita, como o uso de autômatos observadores (MOKADEM, 2006 e BENGTSSON et al., 1996) são considerados para contornar os limites decorrentes do fato da ferramenta UPPAAL lidar com um subconjunto da lógica CTL (vide seção 3.3).

Nas próximas seções estão contidos os resumos dos trabalhos citados. A exceção de GORGÔNIO et al. (2006) que será analisado no próximo capítulo.

5.2 ABORDAGEM DE MADER E WUPPER (1999)

MADER e WUPPER (1999) é um dos primeiros trabalhos, senão o primeiro, a propor um procedimento para passar um programa de CLP escrito em IL para um autômato temporizado. A definição de autômato temporizado utilizada é uma variação descrita em MALER e YOVINE (1996) a qual é bem próxima da descrita por ALUR e DILL (1994) e por T. A. HENZINGER et al. (1992), vide seção 4.2. Em particular, os arcos possuem os mesmos elementos, só diferindo na ordem.

São apresentadas duas formas de tratar temporizadores da norma IEC 61131-3 (2003), levando em conta suas características de tempo real:

- Tratando o temporizador como uma chamada a uma função normal, podendo guardar valores anteriores em variáveis locais. Faz uso de variáveis do tipo inteiro para emular a contagem do tempo e para guardar valores de suas avaliações. Como a contagem é realizada indefinidamente, podem levar a autômatos infinitos; e

- Por intermédio de um autômato em paralelo. Neste caso a contagem se limita ao intervalo de contagem de cada temporizador.

A segunda forma, por intermédio de um autômato em paralelo, têm se mostrado a mais aceita no meio acadêmico (WILLEMS, 1999 e MOKADEM, 2006).

O procedimento baseia-se no estabelecimento de um mapeamento de instruções para arcos. Cada linha de instrução de um programa é então substituída pelo arco correspondente.

Vale assinalar que temas como a modelagem do ambiente ou das entradas e saídas, o tamanho do autômato para modelar sistemas reais, e a questão de como escrever as especificações ficaram abertos a partir dessa abordagem. Algumas melhorias têm sido propostas a esta modelagem, como pode ser visto nas seções seguintes.

5.3 ABORDAGEM DE WILLEMS (1999)

Em WILLEMS (1999) é desenvolvido um conjunto de ferramentas para converter automaticamente programas de PLC, escritos em IL, para autômatos temporizados, com base em MADER e WUPPER (1999). Os programas que compõem o conjunto são *il2i, i2ta,i2lotos e aut2ta* (veja FIG. 5.2), onde o *i* solitário denota um programa no formato de Lista de Instruções Intermediária. O caminho mais direto - *il2i e i2ta* - não contempla a redução do espaço de estados, por isso a necessidade do conjunto indireto. Este último é mais complexo e se baseia no tratamento da explosão de estados sobre o modelo com o tempo abstraído e subseqüente incorporação da informação temporal. Foi comentado que deveria ser verificado, formalmente, a equivalência entre os autômatos temporizados produzidos pelos dois caminhos. Além das ferramentas desenvolvidas, a apresentação de um exemplo de aplicação, a operação de ligar uma lâmpada acionando um botão, parecido com o EX. 4.1 da pág. 45, foi interessante.

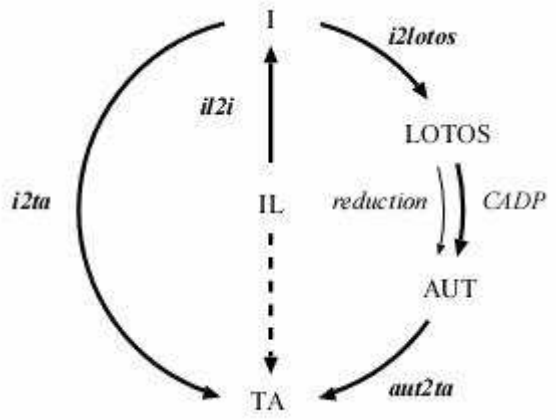


FIG.5.2: Conjunto de ferramentas para conversão de IL para autômatos temporizados (WILLEMS, 1999).

Outras observações podem ser feitas:

- a abordagem realizada não toca a questão da execução multitarefa.
- o tratamento da explosão de estados é realizado pelo pacote de desenvolvimento *Caesar / Aldebram Development Package* - CADP, o qual pode ser obtido a partir de *www.inrialpes.fr/vasy/cadp.html*.
- o CADP não leva em consideração aspectos quantitativos de tempo.

- O exemplo de aplicação foi bastante reduzido (uma entrada e uma saída), e deixou de lado o tamanho do autômato para modelar sistemas reais e também a modelagem do ambiente ou das E / S e a escrita das especificações.

5.4 ABORDAGEM DE ZOUBEK (2004)

Em ZOUBEK (2004) foi desenvolvido o *Checker tool*, programa que converte programas de CLP escritos em LD para uma rede de autômatos temporizados tratável pela ferramenta UPPAAL.

Assim como em MADER e WUPPER (1999) uma estrutura é associada a cada linha de um programa em IL, na abordagem de ZOUBEK (2004) também se constrói uma estrutura para cada degrau do diagrama de contatos. Primeiramente, cada elemento do degrau é associado a um nó, o degrau é então percorrido da esquerda para a direita, veja a FIG. 5.3.

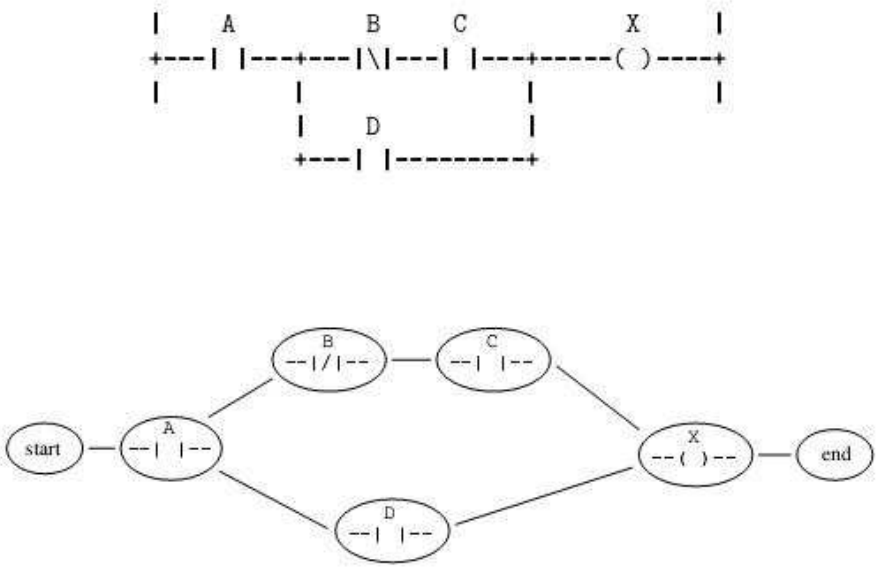


FIG.5.3: Um degrau e a estrutura do seu respectivo modelo (ZOUBEK, 2004).

Depois, é introduzido um algoritmo para conectar os nós (elementos) de um degrau num único bloco.

Os modelos obtidos usando somente a metodologia apresentada até aqui seriam muito grandes e complexos, comprometendo a viabilidade da verificação. Para alcançar modelos menores, possíveis de serem analisados, são propostos em ZOUBEK (2004) os seguintes algoritmos de abstração:

- Recorte de programa (program slice);
- Variável fixa;
- Variáveis comuns e
- Propriedades de tempo real e que não de tempo real.

Os algoritmos são utilizados para reduzir o espaço de estados antes de se chegar ao modelo convertido em autômatos do UPPAAL. Os algoritmos se baseiam na especificação e geram um modelo para cada uma delas (ver FIG. 5.4).

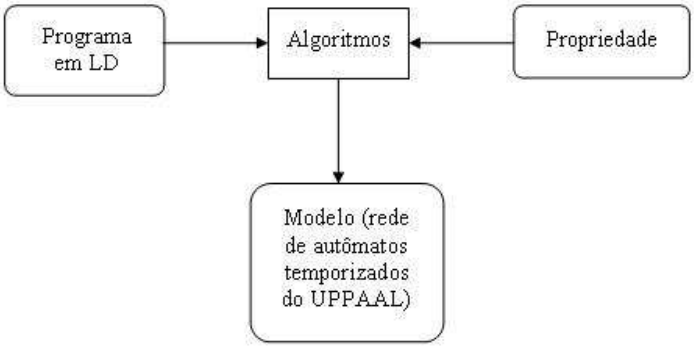


FIG.5.4: Esquema simplificado da geração do modelo pela ferramenta *Checker tool*.

A modelagem proposta adota modelos separados para o programa principal e o comportamento do CLP. Os modelos para o comportamento do CLP, para o programa principal e para as entrada são apresentados na FIG. 5.5.

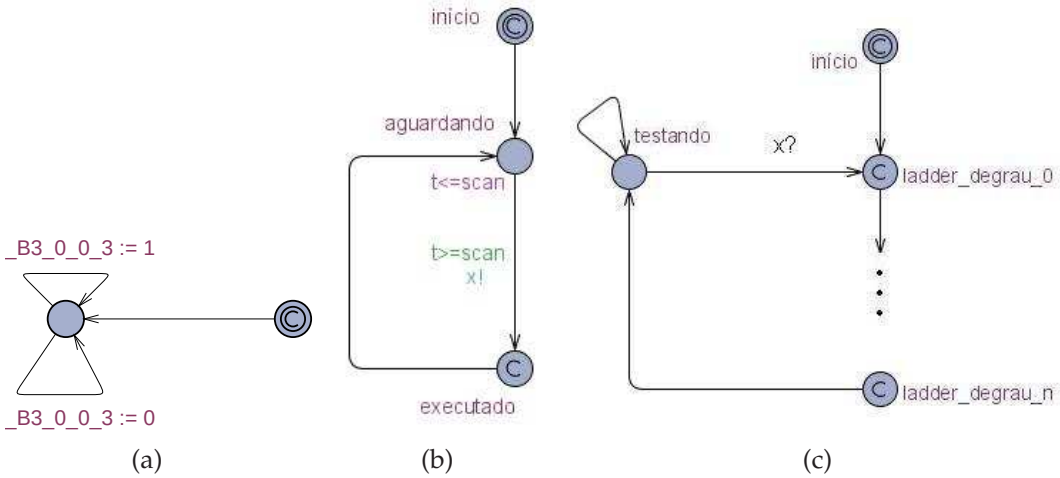


FIG.5.5: a)Modelo da entrada, b) modelo do ciclo de varredura e c) modelo do programa em LD.

O modelo do ciclo de varredura FIG. 5.5(b) é semelhante ao apresentado, posteriormente, por GORGÔNIO et al. (2006) (vide seção 6.3) na sua independência do programa e na coordenação que realiza sobre o modelo deste. O canal de sincronismo $x!$ na mesma figura sintetiza essa coordenação, disparando a seqüência de lugares de controle compromissados (*commited*) e arcos no modelo do programa, que representam as instruções em LD agrupadas como mencionado anteriormente.

Uma diferença para com o modelo do ciclo de varredura de GORGÔNIO et al. (2006) está em que, no lugar de controle *testando* do autômato do programa principal, as entradas podem ficar alternando indefinidamente. A implicação dessa diferença na variação do espaço de estados é explorada na seção 6.4.2.

Cumpre destacar os seguintes comentários:

- Esses algoritmos de redução do espaço de estados dependem das propriedades a serem verificadas, não sendo aplicadas ao programa por si só. Devido a isto, eles não podem ser usados para gerar modelos para testes uma vez que estes não são feitos contra propriedades e sim em cima de entradas e saídas. Contudo, eles são muito úteis para validar (contra especificações de projeto) o modelo que será testado e que foi gerado sem a preocupação com a explosão do espaço de estados.

- O verificador UPPAAL possui o recurso de não inserir no produto dos autômatos que compõem o sistema, aqueles que não se relacionam com a propriedade que se está verificando. Mas para isso, é necessário que o programa a ser verificado seja separado em autômatos em paralelo, o que não é o caso da modelagem proposta por ZOUBEK (2004) (veja a FIG. 5.5(c)). Porém, parece que o algoritmo de recorte de programa, lança mão do mesmo conceito, numa etapa anterior ao produto para confecção do sistema, a etapa da geração dos autômatos temporizados que modelam o sistema.

- O procedimento para modelagem do sistema de ZOUBEK (2004) é uma proposta original e até a escrita desta dissertação não foi percebido algum trabalho que contivesse uma comparação com a técnica proposta. Algumas questões orientaram o adiamento de uma análise mais aprofundada sobre esta técnica:

  - A técnica é apoiada numa modelagem instrução por instrução e degrau por degrau, o que a torna direcionada para programas escritos em LD.

- A questão de modelar instrução por instrução é suspeita de contribuir para o aumento da explosão do espaço de estados. MOKADEM (2006) fez uma comparação entre a modelagem de MADER e WUPPER (1999) a qual é feita instrução por instrução, e a sua a qual é apoiada no conceito de atomicidade, e confirmou essa suspeita para os modelos comparados.
- O primeiro caso de estudo de ZOUBEK (2004) contém 8 entradas e três temporizadores (um do sistema, outro do temporizador e outro para ser usado na especificação a ser verificada), e por meio de sua proposta de modelagem, somente após uma reescrita da especificação a qual permitiu o uso dos algoritmos da *Variável Fixa* e das *Variáveis Comuns*, a verificação foi tornada possível.
- Por fim, o adiamento do aprofundamento da análise se transformou em cancelamento devido à constatação dos limites da ferramenta UPPAAL descritos na seção 6.4.6.

- a modelagem de ZOUBEK (2004) proposta para o temporizador é dependente da sua proposta de modelagem do sistema, e neste trabalho não se vislumbrou como aproveitá-la para as outras modelagens.

5.5 ABORDAGEM DE MOKADEM (2006)

O trabalho de MOKADEM (2006) aborda a conversão de programas de CLP escritos em GRAFCET / SFC para autômatos temporizadosas.

- Melhora o modelo do temporizador de MADER e WUPPER (1999);
- Faz uso de hipóteses de atomicidade sobre o modelo de MADER e WUPPER (1999);
- Contempla a programação multitarefa; e
- Propõe uma solução para a abstração de *estados transitórios*.

5.5.1 AUTÔMATO DO TEMPORIZADOR

O modelo do temporizador TON da norma IEC 61131-3 (IEC, 2003), exposto na FIG. 5.6, proposto por MOKADEM (2006), incorpora as seguintes vantagens em relação ao proposto por MADER e WUPPER (1999) e tratado na seção 5.2:

- Só possui um canal de sincronismo contra os três propostos anteriormente;
- Não possui os *selfloops*, tornando o modelo mais limpo; e
- As atualizações das variáveis são feitas internamente, isolando seu comportamento do modelo do programa principal.
- Percebe-se que o temporizador em questão é bem próximo à versão apresentada em GORGÔNIO et al. (2006), como será visto na seção 6.3.

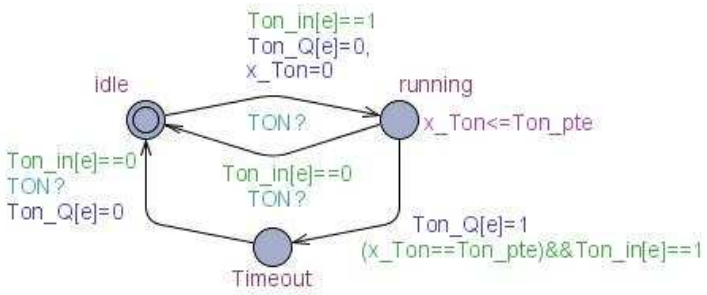


FIG.5.6: Temporizador do tipo TON proposto em MOKADEM (2006).

5.5.2 CONVERSÃO DE PROGRAMAS DE CLP ESCRITOS EM GRAFCET / SFC PARA AUTÔMATOS TEMPORIZADOS

A modelagem do controlador se baseia no programa escrito em LD, que por sua vez é estritamente orientado pelo modelo em GRAFCET do mesmo. Um programa em LD derivado do modelo em GRAFCET tem um primeiro bloco que atualiza as transições, depois vem um segundo bloco onde os estados são atualizados e por último, as saídas são atualizadas. Um autômato obtido segundo esta proposta é ilustrado na FIG. 5.7. Este autômato representa parcialmente o autômato que modela o programa principal do caso de estudo central apresentado por MOKADEM (2006).

Como pode ser visto, o programa só permite o tempo correr em três lugares de controle: leitura de entradas (*input_reading*), execução do programa (*computing*) e escrita das saídas (*output_emission*), abstraindo o comportamento real do CLP. Observa-se que aqui não há necessidade de um autômato auxiliar externo que modele o comportamento do ciclo de varredura do CLP como em ZOUBEK (2004) e em GORGÔNIO et al. (2006) (veja seção 6.3), pois a leitura só é feita uma vez e o tempo de varredura é respeitado em função dos invariantes de cada lugar de controle não *committed*.

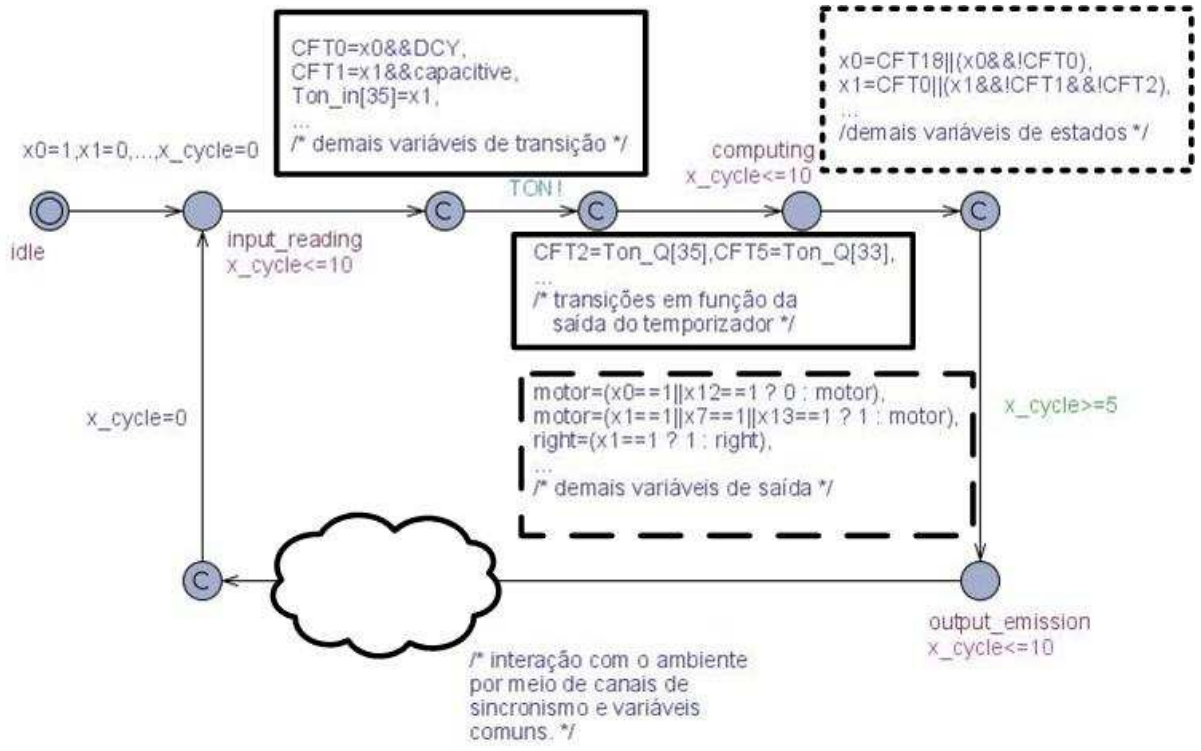


FIG.5.7: Modelo do programa principal (MOKADEM, 2006).

5.5.3 HIPÓTESES DE ATOMICIDADE

A hipótese de atomicidade é a que justamente permite o arranjo nos três lugares de controle mencionados na seção anterior, mais o estado inicial, sendo o resto *committed*, ao invés do que fora proposto por MADER e WUPPER (1999) anteriormente e que está resumido na FIG. 5.8, pelo qual cada atualização de variável seria modelada por um arco. MOKADEM (2006) exibe um quadro identificando que o modelo obtido da proposta por MADER e WUPPER (1999) não pôde ser verificado, enquanto o modelo de acordo com a sua proposta o fez.

Essa configuração de MOKADEM (2006) é bem interessante e bastante adequada a processos modelados em GRAFCET e aqueles que possuem uma clara ordenação. Três etapas podem ser distinguidas na FIG. 5.7 e correspondem diretamente a forma de escrever um programa baseado em GRAFCET: atualização das transições (do GRAFCET) entre o lugar de controle *input_reading* e o *computing*, identificado pela moldura em linha contínua; atualização dos estados (também do GRAFCET) entre o lugar de controle *computing* e o imediatamente a frente, identificado pela moldura em traços menores; e atualização das saídas entre o lugar de controle após a atualização dos esta-

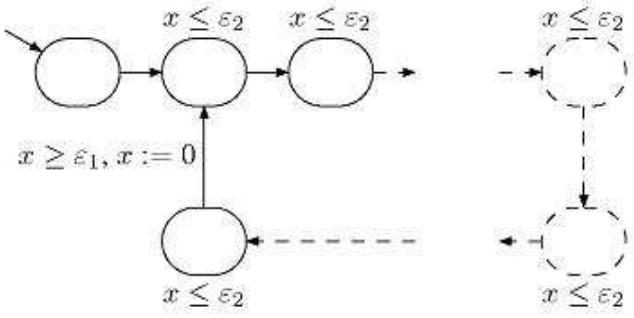


FIG.5.8: Modelo de MADER e WUPPER (1999).

dos e o lugar de controle *output_emission*, identificado pela moldura em traços maiores. Observa-se que para SIS que são, marcadamente, dirigidos a eventos, é possível que seja difícil ordenar a execução do mesmo.

As transições entre a escrita das saídas e a leitura das entradas (identificada por uma nuvem na FIG. 5.7) interagem com os modelos do ambiente, por meio de mensagens de sincronismo (*upjack, downjack, stop*, etc), de forma a restringir seu comportamento, diminuindo consideravelmente o espaço de estados. O ambiente do caso de estudo apresentado por MOKADEM (2006) foi modelado por seis autômatos, dos quais a esteira rolante da FIG. 5.9 é o exemplo mais complexo, por interagir com todos os outros autômatos do ambiente (os quatro sensores, mais o modelo de entrada das peças).

Percebe-se que o nível de refinamento do modelo do ambiente é muito alto para se pensar numa possível sistematização. O resultado da abstração utilizada nessa modelagem é a redução das possíveis combinações de entrada (são 9 entradas no programa principal) de tal forma que as verificações se dão em algumas dezenas de segundos.

5.5.4 ESPECIFICAÇÃO EM TCTL COM AUTÔMATOS AUXILIARES

MOKADEM (2006) faz uso de um autômato auxiliar o qual é denominado de autômato observador. A idéia é a mesma do autômato auxiliar denominado de autômato de teste por BEHRMANN et al. (2004).

No caso de estudo da plataforma MSS comentado ao longo desta seção a propriedade que precisa ser verificada é:

> *A partir do momento em que o sensor acusa que a base está na proximidade da posição de teste, a mesma é parada em menos de 5s (o motor que a aciona é desligado).*

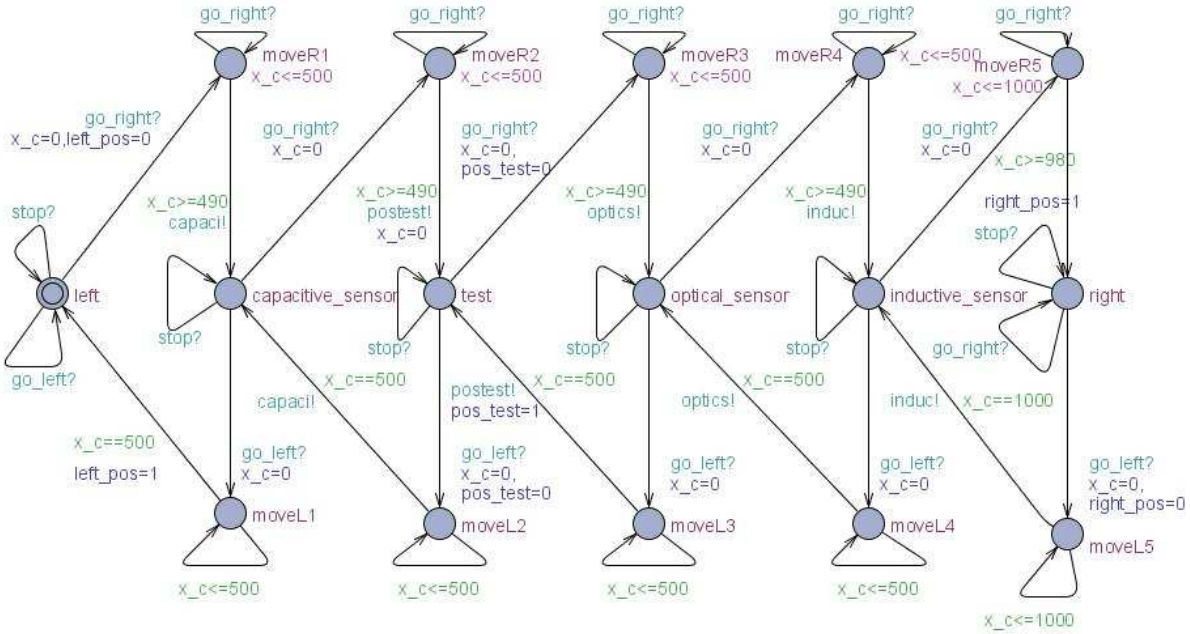


FIG.5.9: Modelo da esteira rolante.

A FIG. 5.10 explicita a estrutura do autômato observador.

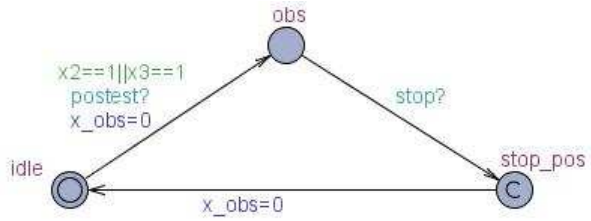


FIG.5.10: Autômato observador de MOKADEM (2006).

A especificação em TCTL fica:

AG obs.stop_pos & obs.x_obs < 5.

Observa-se que a contagem é disparada pela conjunção de duas condições: a base estar na posição (*postest?*) e o sistema se encontrar na etapa x2 ou x3 do GRAFCET que deu origem ao programa cujo modelo é parcialmente exibido pela FIG. 5.7. A mesma é encerrada quando se comanda a parada do motor (*stop?*).

Esta não é a única forma de se construir uma especificação em TCTL, mas é a forma adotada neste trabalho por separar nitidamente, o modelo do programa de controle, da estrutura adicionada para permitir a especificação TCTL. Esta característica favorece

possíveis procedimentos para extração de especificações formais e mantém a leitura do modelo final menos complexa.

5.6 RESUMO DO CAPÍTULO

Neste capítulo foram apresentados os trabalhos pesquisados atinentes à modelagem de programas de CLP por autômatos temporizados, com vistas à verificação de modelos.

No capítulo seguinte, o modelo proposto por GORGÔNIO et al. (2006) e adotado no projeto SIS, é explorado à luz dos conceitos destacados.

6 VALIDAÇÃO DO MODELO DESENVOLVIDO PELA UFCG UTILIZANDO O UPPAAL

6.1 INTRODUÇÃO

Este capítulo apresenta a modelagem de programas de CLP adotada no âmbito do projeto SIS e desenvolvida pela equipe da UFCG (GORGÔNIO et al., 2006). A premissa deste projeto, como citada na seção 2.1, estabelece que a modelagem seja feita por autômatos temporizados que possam ser tratados pela ferramenta UPPAAL-TRON (chamados neste trabalho por *modelo xml do UPPAAL*, em referência à extensão do arquivo), e seja feita, automaticamente, a partir de programas de CLP. Uma conseqüência disto, é a modelagem livre do ambiente e das variáveis de entrada correspondentes.

Por outro lado, o modelo gerado deve estar de acordo com os requisitos iniciais do projeto (descrições textuais e MCE). Isso significa que ele deve ser validado contra as especificações extraídas a partir dos requisitos. Logo, a preocupação final deste capítulo é a verificação do modelo xml do UPPAAL gerado.

A explosão do espaço de estados é uma preocupação constante desde o início do estudo. Os conceitos e propostas dos trabalhos vistos no capítulo anterior permitem explorar tentativas de alterações na **modelagem** para expandir o limite imposto por este obstáculo. Boa parte da seção 6.4 resume este esforço. Contudo, como será visto, as limitações do verificador UPPAL em relação à escala do sistema são severas.

Uma proposta de procedimento para escrita da especificação de propriedades de tempo real é colocada ao final do capítulo, porém sob o contexto de um novo exemplo, com um número bem menor de entradas e saídas.

A solução encontrada para a validação do modelo é fazê-la recorrendo-se à verificação de modelos de sistemas de eventos discretos por meio da ferramenta SMV, ignorando as propriedades de tempo real. Mas essa parte já é assunto do capítulo seguinte.

6.2 EXEMPLO APRESENTADO NA NORMA ISA 5.2

Os assuntos apontados na introdução deste capítulo são abordados, tomando como guia a modelagem proposta pela UFCG aplicada ao exemplo do APÊNDICE A

da norma ISA-S5.2, 1976 (R1992) cujo enunciado se encontra transcrito no APÊNDICE 1 deste trabalho.

O sistema do exemplo em questão trata de uma operação para encher dois tanques (um de cada vez) por intermédio de uma única bomba, conforme o diagrama de processo e instrumentação (P&ID) ilustrado na FIG. 6.1. Os instrumentos estão indicados por símbolos da norma ISA-S5.

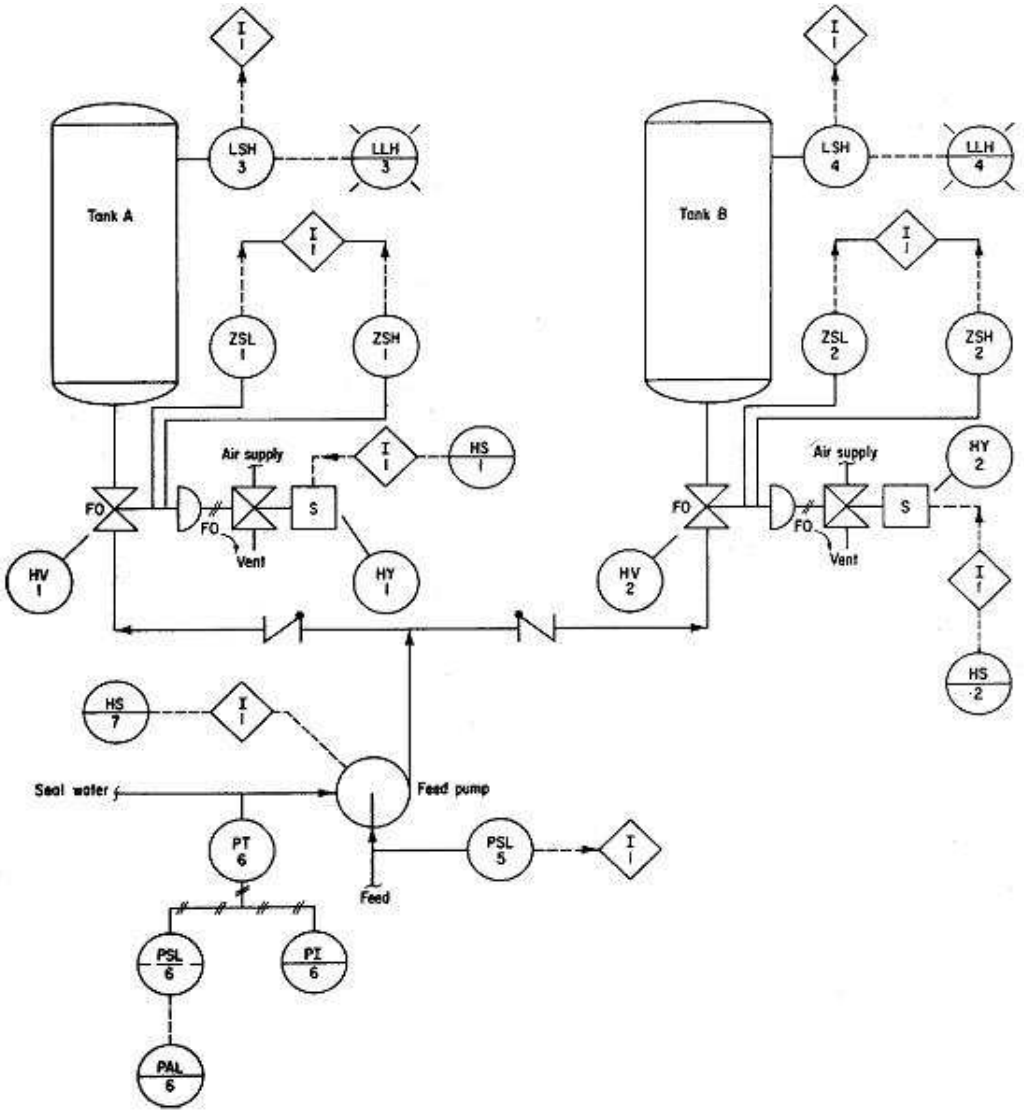


FIG.6.1: Diagrama de processo e instrumentação da operação para encher tanques.

A operação pode ser feita manualmente (posição ON) ou de forma automática (posição AUTO) de acordo com o selecionado na chave seletora HS-7 que ainda possui a posição OFF para desligamento manual. Quando em operação, a lâmpada piloto vermelha L-8A está ligada, quando não, é a lâmpada piloto verde L-8B que fica ligada.

Quando a bomba está ligada no modo manual, ela poderá ser desligada, por atuação sobre a própria chave HS-7 ou pela ação de algum sinal de parada de segurança:

1. A pressão de sucção ficar baixa por 5 s continuamente (PSL-5 > 5 s).

2. O motor da bomba não pode estar sobrecarregado e, ocorrido a sobrecarga, a indicação deste fato (o relé térmico) deve ser *resetada*.

Outro fator de parada de segurança é a pressão d'água de lastro estar baixa (PSL-6). Porém esta situação não se constitui num sinal intertravado, somente aciona o alarme de pressão baixa, PAL-6. O operador, ao perceber o alarme, deverá tomar os procedimentos necessários.

Quando a bomba está ligada no modo automático, ela poderá ser desligada pelas mesmas razões do desligamento no modo manual, mais as seguintes:

1. Enchendo um tanque, a válvula de controle correspondente (HV) deixa sua posição de totalmente aberta, ou a válvula correspondente ao outro tanque deixa sua posição de totalmente fechada. Isto aconteceria em resposta ao acionamento de HS-1, HS-2 ou por falha.

2. O tanque selecionado para encher atinge o nível desejado (LSH).

O diagrama lógico extraído da norma ISA 5.2 (ISA, 1992) é mostrado na FIG. 6.2 e concretiza o controle elaborado na fase de projeto básico. Ele deve conter as especificações escritas para o sistema, tais como:

A bomba será desligada se o sinal de sobrecarga do motor se tornar ativo.

6.3 MODELAGEM DESENVOLVIDA POR GORGÔNIO ET. AL. (2006)

A estrutura geral do modelo guarda algumas semelhanças à proposta por OLIVEIRA (2006), existem padrões para os temporizadores e para as memórias. Mais dois padrões são somados a esses: um padrão de entrada, uma vez que o UPPAAL não possui uma variável do tipo lógica, não determinística, e um padrão para emular o comportamento do CLP, como será visto à frente.

A FIG. 6.3 ilustra o autômato ENV_ , o padrão (*template*) para modelagem do comportamento de uma variável de entrada.

O canal *update* destina-se ao sincronismo do tipo *broadcast*, pelo qual o autômato SCAN (FIG. 6.8) se comunica com as instâncias de ENV_ Essa comunicação é unidirecional, SCAN emite (simbolizado pelo "!" após o nome do canal) e os outros

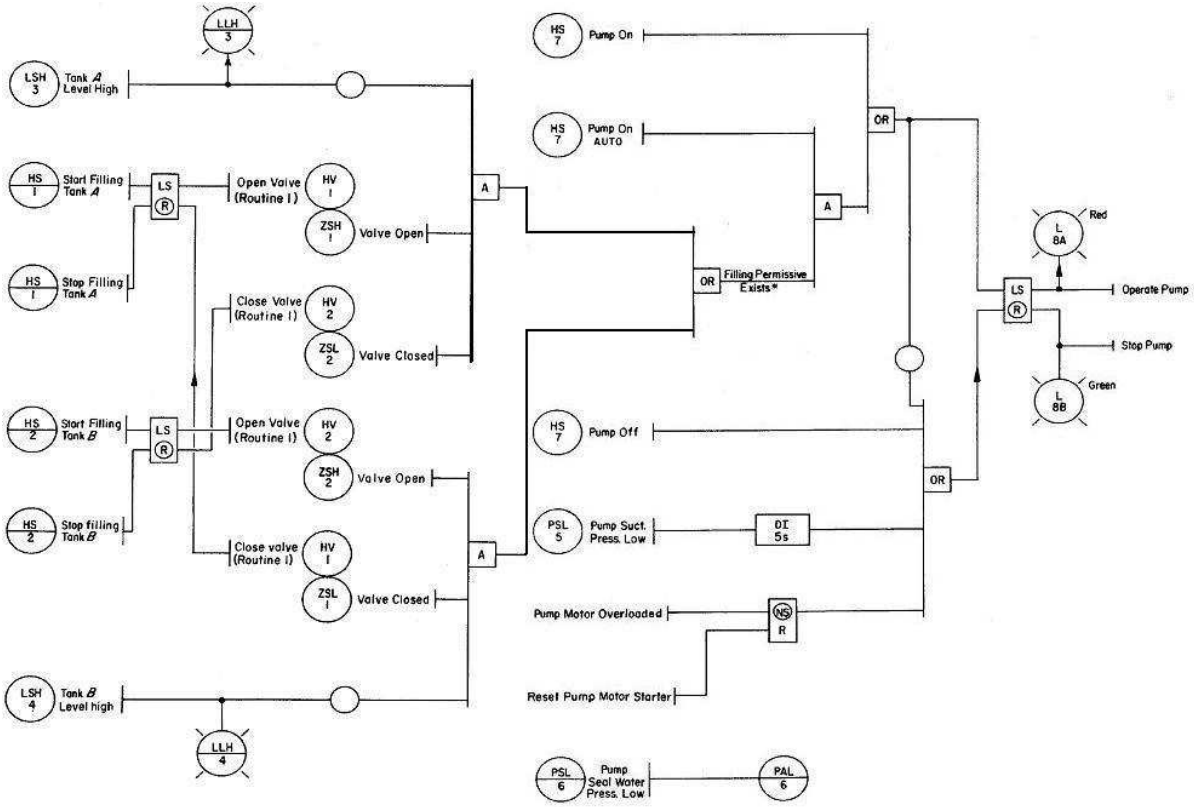


FIG.6.2: Diagrama lógico de operação e intertravamento do sistema.

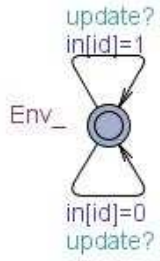


FIG.6.3: Padrão para a modelagem de entradas.

autômatos recebem (simbolizado pelo "?" após o nome do canal) e executam a transição sincronamente. O uso do canal *update* força o ambiente (conjunto de variáveis de entrada) a se alterar na transição associada. Isso abstrai mudanças no ambiente em outros momentos que não interessam à modelagem e que poderiam concorrer para o aumento da explosão do espaço de estados.

O vetor *in*[ ] de valores lógicos é declarado globalmente e representa as variáveis de entrada. A variável *id* é um parâmetro inteiro, passado aos autômatos, que identifica cada instância de um autômato padrão e também qual a variável em questão. Assim, cada instância de *Env_* representa o modelo do comportamento de uma variável de entrada, por exemplo: *Env_(9)* é o autômato que representa o sensor de nível do tanque B (LSH-4 nas FIG. 6.1 e 6.2), *in*[9] = 1 indica que o sensor está ativado, ou seja, que o tanque B está cheio. A instância é associada ao padrão no campo *Declarações do sistema* na forma *Tank_B_Level_High = Env_(9)*.

A modelagem das entradas é ilustrada, parcialmente, na FIG. 6.4.

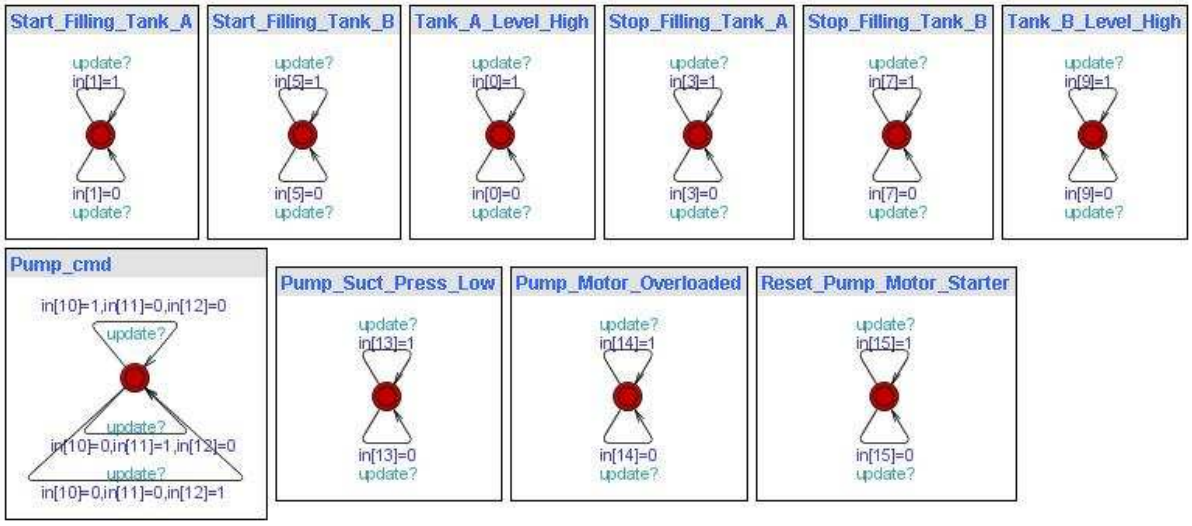


FIG.6.4: Modelagem das entradas.

A instância *Pump_cmd* percebida na FIG. 6.4 corresponde ao padrão ENV3_ ilustrado na FIG. 6.9. Este padrão foi acrescentado para modelar um tipo muito comum de entrada, as chaves de três posições (HS-7 nas FIG. 6.1 e 6.2), e auxiliar a modelagem em dois pontos: reduzir o espaço de estados, pois com ele as três entradas lógicas podem se combinar de 3 formas apenas, contra as 8 permitidas inicialmente; e evitar uma maior complexidade na especificação, pois não bastaria supor que *in*[11] == 1, ter-se-ia que completar para *in*[10] == 0 && *in*[11] == 1 && *in*[12] == 0. Aqui, fica mais fácil perceber o ponto de conflito: refinamento versus a sistematização da mode-

lagem. Para se introduzir esta mudança, há que se prever como identificar este padrão para a variável de entrada lida durante a conversão para o modelo xml do UPPAAL.

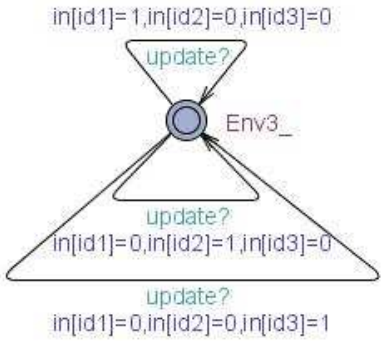


FIG.6.5: Padrão para a modelagem de entradas do tipo *chave de três posições*.

O autômato *Scan_Cycle* da FIG. 6.6 é o padrão para modelagem do ciclo de varredura do CLP. O ciclo é composto de três ações características, a leitura das variáveis de entrada, a execução do programa controlador e a escrita das variáveis de saída, executadas em seqüência e ininterruptamente enquanto o CLP está ligado.

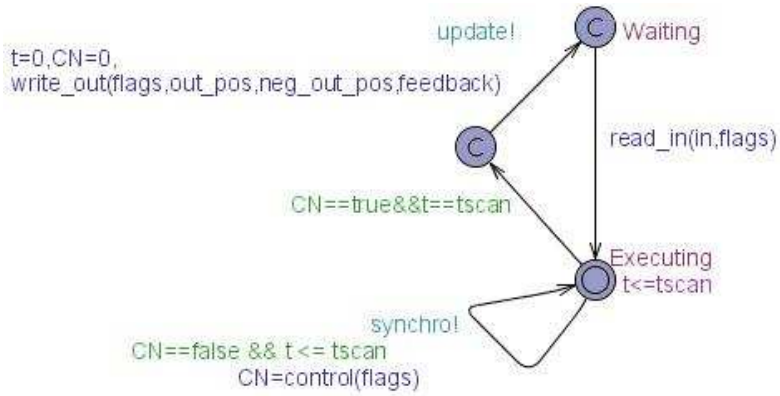


FIG.6.6: Padrão para a modelagem do ciclo de varredura.

A função local *read_in(in,flags)* faz a leitura das variáveis de entrada. *flags*[ ] é um vetor de valores lógicos, declarado globalmente, que contém a imagem congelada de *in*[ ] após a leitura e os valores de variáveis internas.

A variável inteira, declarada globalmente, *tscan* é o parâmetro que passa o valor do período de varredura do CLP. O relógio *t* também é declarado globalmente.

O canal de sincronismo *synchro* é utilizado por *Scan_Cycle* para sincronizar com os autômatos que modelam elementos de controle (memórias padrão *SR_* e temporizadores, neste caso em particular, o padrão *DI_*). Cada transição sincronizada pelo canal *synchro* é um trecho do programa de controle sendo executado. A função

control(flags) testa a ocorrência de alterações no vetor *flags*. Quando não mais acontece alteração de *flags*[ ], diz-se que houve a convergência (o programa de controle foi executado completamente) e a variável lógica *CN* vai para verdadeiro (*true*) indicando esta situação.

O autômato *SR_* é o padrão de modelagem para memórias (*flip-flops*), a prioridade é traduzida dentro da função global *evaluation( )*. *DI_* é o padrão para o temporizador com atraso na inicialização. Assinala-se que o relógio *c* é declarado localmente. Veja a FIG. 6.7.

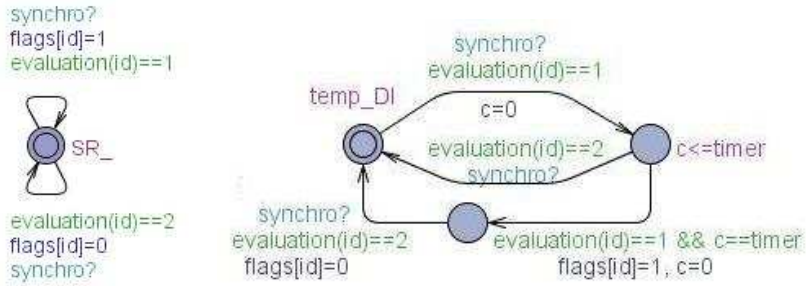


FIG.6.7: Padrões para a modelagem de memórias e do temporizador com atraso na inicialização.

Na FIG. 6.8 aparecem as instâncias das memórias, do temporizador e do modelo do ciclo de varredura para o exemplo do APÊNDICE A da norma ISA 5.2.

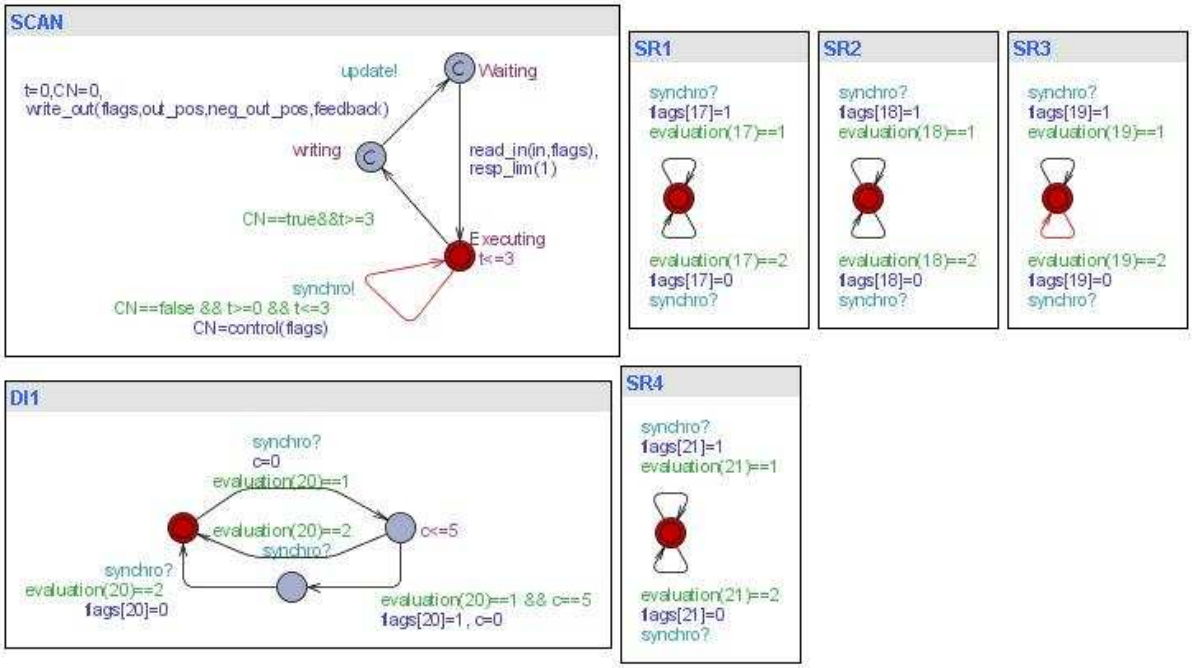


FIG.6.8: Instâncias dos modelos do ciclo de varredura, da memória e do temporizador.

Como exemplo, cita-se o trecho da função *evaluation( )* que corresponde ao SR4, *flip-flop set-reset* que indica a sobrecarga do motor. A prioridade é determinada dentro das expressões lógicas.

```
if (id == id_SR4) {

  if (flags[id_Pump_Motor_Overloaded]==1) return 1;
  if(flags[id_Reset_Pump_Motor_Starter]==1 && flags[id_Pump_Motor_
  Overloaded]==0) return 2;
  else return 0;
}
```

Destaca-se que:

```
...
int id_Pump_Motor_Overloaded = 14;
int id_Reset_Pump_Motor_Starter = 15;
...
int id_SR4 = 21;
```

são pseudônimos declarados globalmente para facilitar a escrita e leitura do programa.

Um canal similar ao canal *synchro* foi usado também nas abordagens de MADER e WUPPER (1999), WILLEMS (1999) e MOKADEM (2006), mas nestes casos, só para comunicação com o autômato do temporizador. A proposta do uso do canal *synchro*, contida em GORGÔNIO et al. (2006), expande essa comunicação para com as memórias, modeladas por autômatos independentes e acaba concentrando toda a lógica de controle. A ordem de execução do programa de controle e a convergência se dão pela escrita das entradas e saídas de cada elemento modelado. Contudo, são necessários algumas execuções repetidas do laço que emite o sincronismo, até se chegar a convergência. Isso contribui para o aumento do espaço de estados se comparado a modelagem de MOKADEM (2006) assinalada na seção 5.5.3.

Em contra partida, a modelagem das memórias por autômatos, pode favorecer a utilização da técnica do quociente da composição para redução da explosão do espaço de estados (LARSEN et al., 1995). Esta técnica se baseia na retirada de autômatos do produto que resultaria no sistema, com a contrapartida de uma alteração na especificação que capture essa retirada.

Por fim, a modelagem desta seção se mostrou bastante propícia a um processo de conversão automática. Maiores detalhes sobre as vantagens e outros pontos que surgem quando da realização da verificação serão explorados na seção seguinte.

6.4 VERIFICAÇÃO E LIMITAÇÕES

Esta seção dá continuidade ao estudo de caso feito sobre a operação para encher dois tanques (um de cada vez) por intermédio de uma única bomba (exemplo do APÊNDICE A da norma ISA-S5.2), tratado nas duas seções anteriores. As condições para desligamento da bomba foram consideradas como base para a especificação utilizada (vide TAB. 6.1):

TAB.6.1: Condições para o desligamento do sistema do exemplo da ISA 5.2 .

Item	Descrição
A.3.2.1	Enchendo o tanque A no modo automático, se a válvula A deixa sua posição de *totalmente aberta* ou a válvula B deixa sua posição de *totalmente fechada*, então a bomba será desligada.
A.3.2.1	Enchendo o tanque B no modo automático, se a válvula B deixa sua posição de "totalmente aberta" ou a válvula A deixa sua posição de "totalmente fechada", então a bomba será desligada.
A.3.2.2	Enchendo o tanque A no modo automático, o mesmo atinge o nível cheio, então a bomba será desligada.
A.3.2.2	Enchendo o tanque B no modo automático, o mesmo atinge o nível cheio, então a bomba será desligada.
A.3.2.3	Se a pressão de sucção estiver baixa por pelo menos 5 s então a bomba será desligada.
A.3.2.4	Se o motor da bomba ficar sobre-carregado então a mesma será desligada.
A.3.2.5	Enchendo o tanque A (ou B) no modo automático, se a seqüência é interrompida manualmente através de HS-1 (ou HS-2), a bomba será desligada.
A.3.2.6	Se HS-7 for chaveada para OFF, a bomba será desligada.

A MCE para o desligamento do sistema, é dada pela TAB. 6.2 a seguir. O item A.3.2.5, *Enchendo o tanque A (ou B) no modo automático, se a seqüência é interrompida manualmente através de HS-1 (ou HS-2), a bomba será desligada*, não é considerado uma causa da MCE, pois ele depende das válvulas executarem o comando, e este estado já foi descrito no item A.3.2.1 .

TAB.6.2: MCE para o desligamento do sistema do exemplo da ISA 5.2 .

Efeito / Causa		Desliga bomba out[7] (SP)
Item da norma	TAG do programa	
A.3.2.1	(Pump_On_AUTO & !Valve_A_Opened & Valve_B_Closed) | (Pump_On_AUTO & Valve_A_Opened & !Valve_B_Closed)	X
A.3.2.1	(Pump_On_AUTO & !Valve_B_Opened & Valve_A_Closed) | (Pump_On_AUTO & Valve_B_Opened & !Valve_A_Closed)	X
A.3.2.2	(Pump_On_AUTO & Valve_A_Opened & Valve_B_Closed & Tank_A_Level_High)	X
A.3.2.2	(Pump_On_AUTO & Valve_B_Opened & Valve_A_Closed & Tank_B_Level_High)	X
A.3.2.3	Pump_Motor_Overloaded	X
A.3.2.4	Pump_Suct_Press_Low & contador > 5	X
A.3.2.6	Pump_Off	X

O principal resultado alcançado foi a identificação da existência de um baixo limite superior para o número de entradas e de temporizadores que a estrutura da ferramenta UPPAAL impõe para execução da verificação de modelos. A simples verificação da especificação A.3.2.1 não foi possível. A constatação mais precisa deste resultado é tema da seção 6.4.6 a frente.

Porém, até se chegar a esta conclusão foram tentadas algumas mudanças na **modelagem**. Umas se mostraram ajustes necessários, outras foram percebidas como alterna-

tivas que tornaram o desempenho pior (aumento do número de estados armazenados e explorados) e outras não tiveram a relação com a explosão do espaço de estados analisada em detalhes pois a verificação com ou sem elas se mostrou inviável. Neste ponto do trabalho, o estudo detalhado sobre essas alternativas, foi abandonado para se priorizar a validação do modelo sem se levar em conta as propriedades de tempo real. Esta seção faz uma passagem sobre as principais tentativas e mudanças na modelagem.

Os testes foram realizados numa máquina rodando a plataforma LINUX (distribuição Mandriva), com um processador pentium 4 de 2,4 GHz e 512 MB de memória. O programa utilizado para medição do consumo de recursos (memória e tempo) foi o *memtime*, indicado na própria página do UPPAAL (UPPAAL, 2008).

Nas tentativas de se realizar a verificação de modelos sobre o sistema do exemplo do APÊNDICE A da norma ISA-S5.2, foram utilizadas várias combinações das opções disponíveis na ferramenta UPPAAL, sem contudo viabilizar a verificação. Nos casos apresentados nesta dissertação, as opções usadas para execução da verificação foram as opções padrão: nenhum traço, ordem de pesquisa -b primeiros criados são visitados primeiros (*breadth first*), -S1 Algoritmo conservativo para redução do espaço de estados (*conservative space optimisation*), representação do espaço de estados utilizando-se da opção -u, a qual muda o modo de comparação quando representando estados com gráficos de restrição mínima (minimal constraint graphs).

6.4.1 AJUSTE DO MODELO DO CICLO DE VARREDURA

Ao tentar usar especificações limitadas no tempo como a alternativa abaixo para o item A.3.2.1 da norma, a verificação resultou em verdadeira mesmo sendo o tempo de varredura igual a 0,3 s. Observar que na especificação assinalada abaixo, q é um relógio disparado por um observador externo o qual foi apresentado na seção 5.5.4 e é detalhado na seção 6.5

$$(in[1] \ \& \ (!\ in[2] \mid !\ in[4]) \ \& \ in[11]) \rightsquigarrow (!\ out[5] \ \& \ q < 3)$$

Ou seja, a resposta *not out[5]* (motor desligado) se verificava em todos os casos antes de 3ds. Percebeu-se então a necessidade de uma guarda que só permitisse a passagem para o próximo ciclo, considerando-se o tempo de varredura. Daí o uso de $t == tscan$ ou $t >= tscan$ na guarda saindo do lugar de controle *Executing*.

Esclarecendo um pouco mais, a guarda saindo do lugar de controle *Executing* proposta por GORGÔNIO et al. (2006) como $CN == true$ deve passar a $CN ==$

true && t == *tscan*. E a guarda do laço no mesmo lugar de controle deve passar de $CN == false$ && $t >= 0$ && $t < tscan$ para $CN == false$ && $t >= 0$ && $t <= tscan$ para ficar coerente com a alteração anterior na outra guarda.

6.4.2 USO DO CANAL DE SINCRONISMO *UPDATE*

Em ZOUBEK (2004) as entradas variam livremente, quando o lugar de controle corrente do modelo do ciclo de varredura não é *compromissado*. Isso levantou a questão se era mesmo necessário o uso do canal *update* restringindo o comportamento do modelo.

Para comparar os dois casos, considerou-se um modelo sem lógica de controle, cuja quantidade de entradas foi aumentada progressivamente. A especificação verificada foi *A[] not deadlock*. As tabelas TAB. 6.3 e TAB. 6.4 registram o resultado e indicam que o uso do canal *update* contribui para reduzir o número de estados armazenados e explorados.

A comparação deve ser feita observando-se o número de estados armazenados e/ou o número de estados explorados. As outras colunas foram acrescentadas para orientar e facilitar a repetição dos resultados. Em particular, *Max RSS* é a máxima quantidade de memória alocada na memória principal enquanto *Max VSize* é a máxima quantidade de memória endereçada na memória virtual (ZOUBEK, 2004).

TAB.6.3: Modelo com update

N.o de entradas	Estados armazenados	Estados explorados	tempo decorrido	Max VSize	Max RSS
1	8	12	0,41 s	2.856	1.696
2	32	48	0,10 s	1.420	124
3	128	192	0,10 s	1.420	120
4	512	768	0,10 s	1.420	120
5	2.048	3.072	0,10 s	1.420	120

Pode-se perceber que o modelo com o canal *update* diminui a explosão do espaço de estados quando comparado ao modelo alternativo sem um canal com a mesma

TAB.6.4: Modelo sem update

N.o de entradas	Estados armazenados	Estados explorados	tempo decorrido	Max VSize	Max RSS
1	10	30	0,11 s	1.416	120
2	64	172	0,11 s	1.420	124
3	512	1240	0,10 s	1.416	124
4	4.096	9.200	0,21 s	36.972	3.184
5	32.768	70.112	1,67 s	38.816	5.716

função do canal *update*.

6.4.3 ESPECIFICAÇÕES COM OPERADOR $\rightsquigarrow$ (*LEADS TO*) DO UPPAAL E O MODELO SEM CANAL *UPDATE*

Outro ponto contra o modelo alternativo sem o canal update é o fato da especificação que usa $\rightsquigarrow$ aceitar como cenários futuros, os laços sem sincronismo e sem guardas com relógios que forcem o passar do tempo, existentes nestes modelos. O problema reside na interpretação de "cenários futuros". Na realidade $p \rightsquigarrow q$ quer dizer *AG (p imply AF q)*, e *AF q* significa para todos os caminhos a partir daquele estado, num estado futuro, ou seja, para os caminhos enxergados pela árvore de computação e não, necessariamente, para um futuro temporal.

Exemplo: Considere a tentativa de verificação da propriedade A.3.2.1.

$$(in[1] \ \& \ (!in[2] \mid !in[4]) \ \& \ in[11]) \rightsquigarrow (!out[5])$$

A verificação acusaria falha e indicaria um traço em que ($in[1]$ & ($!in[2]$ | $!in[4]$) & $in[11]$) acontece e depois o sistema ficaria preso num laço de uma entrada executando indefinidamente, por exemplo $in[1] = 1$.

Ou seja, o modelo com o canal *update* acumulou mais uma vantagem frente à sua alternativa, permitindo a escrita de especificações com operador $\rightsquigarrow$, sobressaindo-se como a melhor opção.

6.4.4 MODELAGEM DO AMBIENTE (VÁLVULAS)

O problema da modelagem das válvulas decorre da especificação extraída do texto de descrição do problema não ser precisa e mesmo da dificuldade em se escrever uma especificação para a situação física existente (parecida com a questão da chave de três

posições FIG. 6.9). O texto menciona a seguinte afirmativa: *enquanto bombeando para um tanque*.

Traduzir para uma sentença em CTL, utilizando-se as variáveis de entrada normais, se mostrou difícil pela tendência em se querer usar a chave que escolhia o tanque para encher, quando na realidade, o estado citado era caracterizado pelas posições da válvulas.

Além disso, e principalmente, o fato de haver um sensor para indicar se a válvula estava fechada e outro para indicar se a mesma estava aberta, implicava em aceitar a possibilidade dos sensores indicarem que a válvula estava ao mesmo tempo aberta e fechada. Este fato complica a escrita das especificações pois não é suficiente escrever que ZSH1 (sensor que indica que a válvula 1 está aberta) está ativo, é preciso mencionar que ZSL1 (sensor que indica que a válvula 1 está fechada) não está ativo. Daí a tentativa de se ajustar o modelo das entradas relativas aos sensores das válvulas, conforme a FIG. 6.9, diminuindo ainda o espaço de estados (a combinação $in[id\_ZSH1] = 1, in[id\_ZSL1] = 1$ não aparece no modelo, não sendo possível de ocorrer). Porém, mais uma vez, o custo dessa mudança é passado para a geração do modelo xml do UPPAAL.

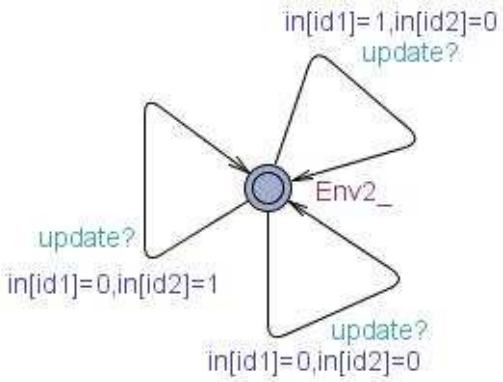


FIG.6.9: Possível padrão para a modelagem de dois sensores para a indicação da posição de uma válvula.

Como a validação do sistema em questão por meio da ferramenta UPPAAL não foi possível mesmo com o uso dessas alternativas em função da explosão do espaço de estados (vide seção 6.4.6) então, foi dada preferência em manter um único modelo para as entradas, facilitando a confecção dos programas de conversão (vide FIG. 7.1 na seção 7.1). Nos casos de chaves de três posições, ao se ignorar o modelo da FIG. 6.9, o próprio traço gerado vai auxiliar numa intervenção para melhorar a escrita da especificação a qual desprezará os estados considerados espúrios ou irrelevantes.

6.4.5 ADAPTAÇÕES BASEADAS EM ZOUBEK (2004) E O USO DO CONECTOR → NAS ESPECIFICAÇÕES

Em ZOUBEK (2004) foi utilizada uma forma de especificar que permite a utilização do → (*imply* na sintaxe do verificador UPPAAL). Esta especificação usa o último estado antes da atualização das variáveis de entrada como teste para a verificação do efeito destas após a execução do programa. Com relação ao modelo da UFCG, basta que se nomeie tal estado, como por exemplo, *writing*. Isso permite a seguinte variação de especificação:

Especificação anterior: $varA \rightsquigarrow varB$

Especificação alterada: *(varA and SCAN.writing) imply varB*

Pensou-se que esta especificação poderia diminuir a explosão do espaço de estados frente à especificação usando $\rightsquigarrow$.

Contudo, como esta alternativa não permitiu que a verificação obtivesse êxito, não foi feito um estudo mais preciso dessa questão.

6.4.6 LIMITES DA FERRAMENTA UPPAAL

A despeito da possibilidade de aumento de memória utilizada, a taxa de crescimento do número de estados armazenados e explorados indica a intransponibilidade desse limite na realidade atual.

As tabelas 6.5 e 6.6 são resultado de testes com sistemas descritos, respectivamente, pelas figuras 6.10 e 6.11 e testemunham a relação apontada acima. A especificação verificada nos testes foi: *A[] not deadlock*.

O primeiro sistema (FIG. 6.10) teve o número de entradas variado conforme as linhas da TAB. 6.5, ou seja, primeiro utilizou-se uma entrada ligada ao elemento lógico *e* (*A*) o qual por sua vez se conecta ao temporizador e em seguida a uma saída qualquer (FIG. 6.10(a)). Depois, acrescentou-se uma entrada por teste e todas foram conectadas à entrada do elemento lógico *e*, mantendo-se os elementos e as conexões depois do mesmo(figuras 6.10(b) e 6.10(c)).

Pode-se notar que a inserção de cada entrada faz o número de estados armazenados e visitados ser multiplicado por uma taxa que, a cada inserção, tende a se aproximar mais de 4.

No segundo sistema (FIG. 6.11), cada entrada se conecta a um temporizador formando um conjunto que se conecta a um elemento lógico *e*. O número de conjuntos

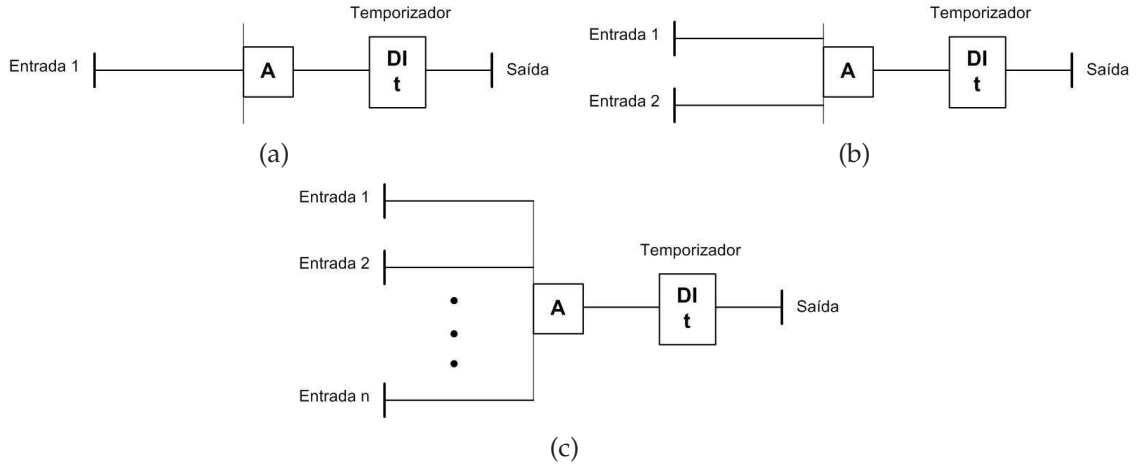


FIG.6.10: Sistema para avaliar a relação do número de entradas com o espaço de estados. a)Primeiro teste, com uma entrada; b) Segundo teste, com duas entradas; e c) n-ézimo teste, com "n" entradas.

TAB.6.5: Número de entradas X espaço de estados.

N.o de entradas	Estados armazenados	Estados explorados	Tempo decorrido	Max VSize	Max RSS
1	64	94	0,21s	36.548	2.836
2	188	236	0,12s	36.812	2.904
3	664	760	0,11s	2.852	1.620
4	2.480	2.672	0,11s	2.852	1.592
5	9.568	9.952	0,31s	37.208	3.448
6	37.568	38.336	1,04s	38.688	5.200
7	148.864	150.400	3,96s	44.976	12.412
8	592.640	595.712	17,25s	71.120	41.772
9	2.364.928	2.371.072	71,53s	191.316	161.168
10*	9.459.712	9.484.288	24.064s	689.244	476.136

* Interrompido por falha de memória

conectados à entrada deste elemento foi variado conforme as linhas da TAB. 6.6. O tempo de contagem é o mesmo em todos os temporizadores.

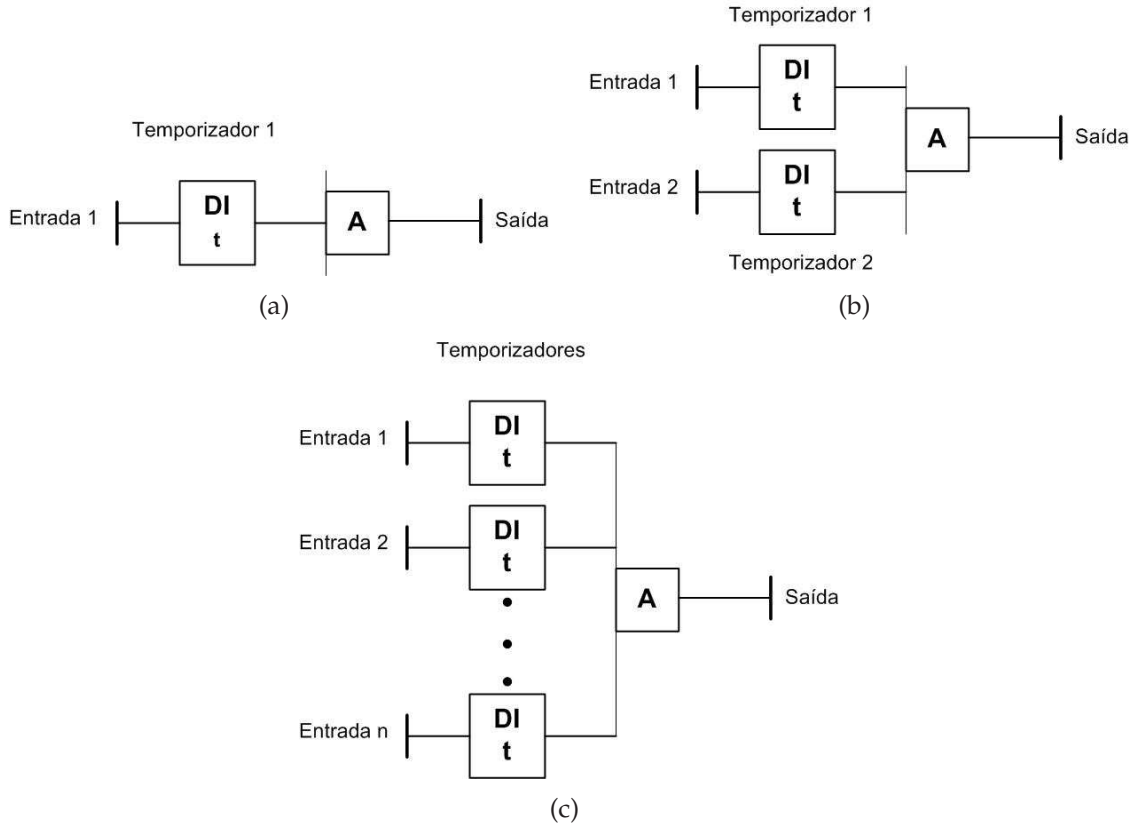


FIG.6.11: Sistema para avaliar a relação do número de temporizadores com o espaço de estados. a)Primeiro teste, com um temporizador; b) Segundo teste, com dois temporizadores; e c) n-ézimo teste, com "n" temporizadores.

Conclusão, a inserção de cada conjunto faz o número de estados armazenados e visitados ser multiplicado por uma taxa que tende a se aproximar de 30 na medida em que o número de conjuntos aumenta.

Com isso, fica constatada a severa limitação quanto ao número de entradas do sistema quando se utiliza o verificador UPPAAL. O que não ocorre quando se trabalha com ferramentas que tratam de sistemas a eventos discretos, como é o caso do verificador SMV.

6.5 UM EXEMPLO VIÁVEL

O exemplo da operação para encher tanques, tratado na seção anterior, não possibilitou o estabelecimento de um procedimento para verificação de propriedades de tempo real, visto que mesmo a verificação de propriedades que não de tempo real, não

TAB.6.6: Número de entradas e temporizadores X espaço de estados

N.o de entradas	N.o de temp.	Estados armazenados	Estados explorados	Tempo decorrido	Max VSize	Max RSS
1	1	83	124	0,31s	2.860	1.752
2	2	2.094	3.080	0,11s	2.852	1.624
3	3	60.591	94.627	3,12s	3.976	6.024
4	4	1.898.522	3.110.638	159,40s	135.380	104.208
5	5	-	-	371.106,90s	482.220	1.498.680

foi conseguida. Por esta razão, paralelamente aos testes realizados, foi feita a análise de um sistema menor (com apenas duas entradas e um temporizador) para adiantar como seria escrita, de forma sistematizada, a especificação de propriedades de tempo real.

Este tipo de verificação exige a medição do intervalo entre pelo menos dois eventos (ou estados) por intermédio de um relógio externo ao sistema. Existem duas maneiras de introduzir e disparar esse relógio ao se utilizar a ferramenta UPPAAL: adicionando-se uma função ou um autômato temporizado auxiliar.

Quando a propriedade de tempo real é verificada sobre um sistema automatizado por CLP, surgem alguns detalhes importantes a serem observados. Em se tratando de CLPs sem interrupção (o de uso mais comum), apesar de ser possível a alteração das entradas a qualquer tempo, a leitura desta alteração não é imediata, sendo realizada no início de cada ciclo de varredura. Assim, as alterações realizadas logo após a leitura, só serão percebidas pelo CLP no próximo ciclo, conforme discorrido na seção 2.2.

A modelagem proposta por GORGÔNIO et al. (2006) e adotada no projeto SIS não permite a alteração das entradas a qualquer tempo, pois todas se dão sincronamente. Essa modelagem pressupõe que as propriedades de tempo real a serem verificadas não envolvem precisão da ordem de um período de varredura, o que é satisfeito pela grande maioria dos casos. Além disso, quando o sistema em estudo fizer uso da programação multitarefa para reduzir o tempo de resposta a menos de um período de varredura, tanto a modelagem adotada, como a especificação que será proposta a seguir deverão ser adaptadas. A modelagem provavelmente deverá conter um autômato específico para tratamento da tarefa assíncrona (vide o exemplo de MOKADEM, 2006, explorado na seção **??**).

Alguns dos trabalhos levantados sobre especificação não tratam de TCTL, mas oferecem uma padronização abrangente que pode ser estendida para abordá-la.

O exemplo a seguir explora os detalhes comentados.

Exemplo 6.1. *O programa de CLP da FIG. 6.12 é proposto para efetuar o controle de uma prensa que é manejada por um operário. O controle deve prover segurança, só permitindo o acionamento da prensagem quando ambas as mãos do operário estiverem apertando os dois botões A e B existentes. A operação de prensagem é realizada quando se põe em marcha o motor comandado pelo contator R. Para isso, deve-se obedecer a seguinte seqüência de funcionamento:*

a. Com somente um botão sendo pressionado, o operador não coloca a prensa em funcionamento;

b. Somente com ambas as mãos atuando nos botões A e B, a prensa pode abaixar;

c. Se o operador atua com uma mão e a outra tarda mais do que três segundos a atuar, a prensa não deve operar e, é necessário repetir a manobra; e

d. Uma vez ativado o contator R, se qualquer uma das mãos for retirada de seu comando correspondente, R desativa e não volta a se ativar, se o operador demorar mais que três segundos para recolocar sua mão no comando, caso em que deverá retirar ambas as mãos e repetir a manobra.

O CLP não possui o recurso de interrupção e seu ciclo de varredura leva 0,3 s.

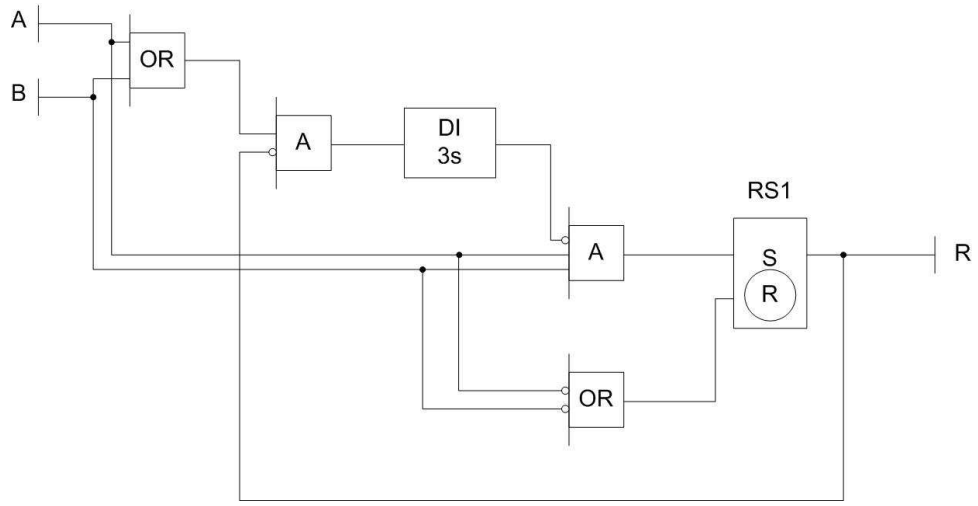


FIG.6.12: Diagrama lógico do programa de controle das prensas

Obs.: A e B são as entradas no CLP oriundas das duas botoeiras (uma para cada mão do operador) as quais podem ou não estar ligadas em série.

O modelo utilizado no UPPAAL está exposto no ANEXO 1.

6.5.1 ESPECIFICAÇÃO UTILIZANDO-SE UM AUTÔMATO AUXILIAR

Nesta seção o exemplo 6.1 é analisado com foco na regra "c" que expressa uma propriedade de tempo real.

Acrescenta-se ao sistema modelado o padrão de autômato temporizado observador da FIG. 6.13 o qual tem as seguintes características:

- No modelo de GORGÔNIO et al. (2006) toda alteração no ambiente, ou seja, todo evento ocorre sincronazado pelo canal *update*, então o relógio do autômato observador deve ser disparado nesse exato instante.

- A contagem pode ser disparada por uma condição, interrompida por outra e, após terminada, levada a zero por outra condição. A indicação das mesmas no autômato, torna sua edição simples.

- Uma vantagem dessa modelagem, é o fato de não ter que se preocupar em criar um canal de sincronismo exclusivamente para essa abordagem. Aliás, fica claro que o acoplamento do autômato observador é feito diretamente, sem interferência alguma na rede de autômatos que modela o sistema.

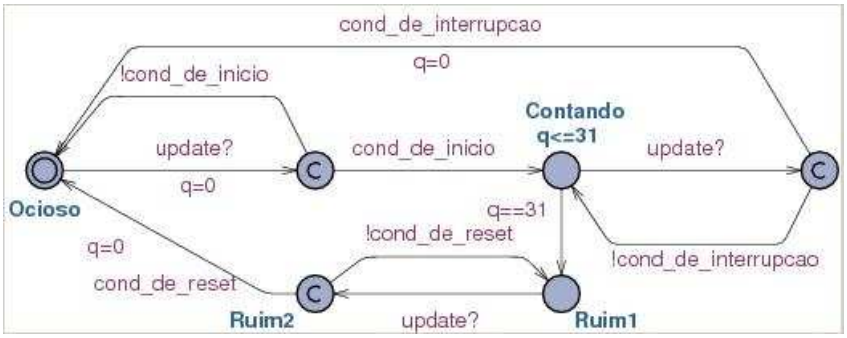


FIG.6.13: Autômato temporizado observador *Obs*.

O autômato observador usa um relógio declarado localmente q. Para facilitar a análise do exemplo, escolheu-se o décimo de segundo como unidade de contagem do tempo do temporizador. As três condições compõem as guardas que são expressões lógicas as quais, por sua vez, são combinações de proposições atômicas em função das entradas, ligadas por conectores booleanos. Elas precisam ser editadas para cada especificação envolvendo uma propriedade de tempo real:

- *cond_de_incio* é a expressão lógica que permite o início da contagem.

- *cond_de_interrupcao* é a expressão lógica que interrompe a contagem antes dela terminar, fazendo o relógio q retornar a zero e o autômato observador ao seu estado inicial.

- *cond_de_reset* é a expressão lógica que, após o autômato ter passado para os lugares de controle considerados ruins, permite que este volte ao seu lugar de controle inicial, tornando $q = 0$.

Deve-se tomar cuidado pois tanto o invariante do lugar de controle *Obs.Contando*, como a guarda da transição de saída desse lugar, devem incluir a igualdade, pois do contrário, um *deadlock* impediria o estado *Obs.Ruim1* de ser alcançado.

Analisando a regra "c", tanto a entrada 0 (botão A) como a 1 (botão B) podem disparar o relógio.

Para se editar as condições, é interessante procurar enquadrá-la num dos padrões de DWYER et al. (1998), porém nem todos podem ser escritos na lógica do UPPAAL. Numa primeira observação, o padrão de DWYER et al. (1998) que aparece como mais adequado é o *resposta*. A propriedade *o botão A e o botão B foram apertados simultaneamente com o contador Obs.q menor do que 31, então, para todos os caminhos futuros, a prensa será ligada* expressa a especificação neste padrão.

((!Obs.Ruim1 & !Obs.Ruim2) & in[0] & in[1]) ⇝ out[0]

A verificação resulta em falha pois o traço apresentado indica a possibilidade de ocorrer:

$DI1.c \in [27, 30]$, onde *DI1.c* é o relógio do temporizador, e $Obs.q = 30$ e $in[0] = 1 = in[1]$ em *Scan.Waiting*, para logo depois em *Scan.Executing* ocorrer $DI.c \in [27, 31]$ e $Obs.q \in [0, 3]$ e $in[0] = 1 = in[1]$ permitindo que o temporizador atinja o atraso antes da execução. A FIG. 6.14 expõe esta situação.

Para contornar esse problema, deve-se diminuir um período de varredura da guarda de saída do estado Obs.Contando: $q >= 31 - 3$.

Outra especificação possível envolve o padrão de ausência:

A[] !((Obs.Ruim1 | Obs.Ruim2) & out[0])

Efetuando-se a verificação, mantendo-se a guarda de saída do estado Obs.Contando ($Obs.q >= 31 - 3$), é gerado um traço no qual *Obs.q* pode ser 29s, assim como *DI.c*, permitindo tanto *out[0]=1* como *Obs.Ruim1*.

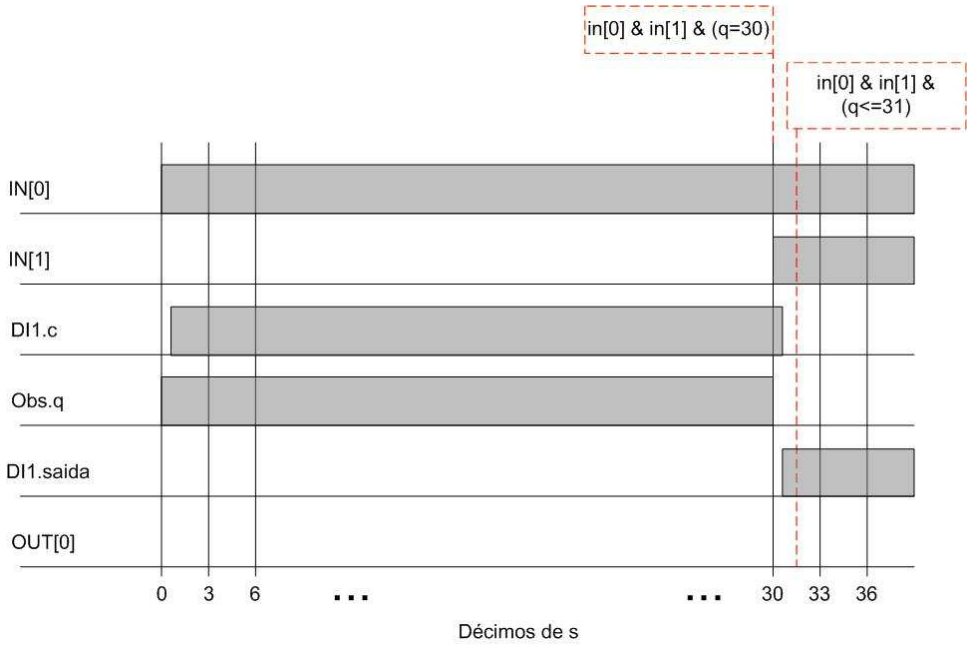


FIG.6.14: Traço de erro para guarda do arco saindo de *Obs.Contando* igual a $q == 31$.

Para essa situação, o ajuste se dá na direção contrária: deve-se somar um período de varredura à guarda, para que a propriedade seja atendida pelo sistema ($q >= 31+3$).

Resumindo e organizando:

- Se a propriedade deve ser verificada **antes** de determinado momento, deve-se **diminuir** um período de varredura da guarda de saída (e do invariante) do estado Obs.contando. Isto garante que os intervalos dos relógios estejam todos contidos no limite estabelecido.

- Se a propriedade deve ser verificada **depois** de determinado momento, deve-se **somar** um período de varredura à guarda de saída (e do invariante) do estado Obs.contando. Isto garante que os intervalos dos relógios estejam todos contidos no limite estabelecido (veja FIG. 6.15).

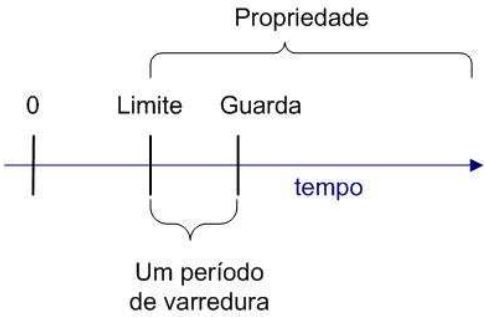


FIG.6.15: Ajuste da guarda de saída de Obs.Contando para propriedades válidas a posteriori.

Com isso, fica definido um procedimento para a verificação de propriedades de tempo real, de programas de CLP modelados de acordo com a proposta de GORGÔNIO et al. (2006).

6.6 RESUMO DO CAPÍTULO

Neste capítulo, a modelagem proposta por GORGÔNIO et al. (2006) e adotada no projeto SIS, foi apresentada. A verificação do modelo obtido segundo esta modelagem, utilizando-se a ferramenta UPPAAL não foi viável. Porém, antes de se chegar a esta conclusão, foram discutidas algumas alternativas baseadas nos trabalhos levantados no capítulo 5, para se contornar este obstáculo ou pelo menos diminuir a rigidez dos seus limites.

Foi feita ainda, uma proposta de procedimento da escrita de especificação para a modelagem de GORGÔNIO et al. (2006).

Uma vez que o verificador UPPAAL é bastante limitado em relação ao número de entradas e temporizadores, para cumprir com o compromisso assumido no Projeto SIS de validar o modelo a ser utilizado para geração de testes automáticos, recorreu-se ao verificador SMV. Isso é assunto do próximo capítulo, onde são apresentadas as ferramentas que automatizaram os procedimentos propostos por OLIVEIRA (2006) relativos a este verificador.

7 AUTOMATIZAÇÃO DOS PROCEDIMENTOS DE VERIFICAÇÃO DESENVOLVIDOS PELO IME

No capítulo 6 mostrou-se que o verificador UPPAAL impõe severos limites em relação ao número de entradas e temporizadores. O recurso de abstrair o sistema para permitir sua utilização, não pôde ser automatizado. Com isso, a solução encontrada para a questão da validação do modelo xml do UPPAAL foi efetuá-la desconsiderando as propriedades de tempo real, empregando o verificador SMV (versão 2.5.4).

Neste capítulo são apresentadas as ferramentas desenvolvidas para automatizar o processo de validação do modelo xml extraído do diagrama de lógica segundo GORGÔNIO et al. (2006). Este modelo é convertido para um modelo aceito pelo verificador SMV por meio da ferramenta *XML2SMV*. Juntamente com essa conversão, insere-se as especificações em CTL, obtidas de uma MCE por meio da ferramenta *MCE2ctl*. Finalmente, as ferramentas desenvolvidas são usadas sobre a modelagem proposta pela UFCG aplicada ao exemplo do APÊNDICE A da norma ISA-S5.2, 1976 (R1992) e tratada nas seções 6.2, 6.3 e 6.4, obtendo-se sucesso desta vez.

7.1 ALTERAÇÕES DO MODELO DE OLIVEIRA (2006)

Algumas adaptações aos procedimentos de conversão contidos em OLIVEIRA (2006) foram introduzidas para aproximar o *modelo smv* (modelo para verificação no SMV) ao modelo xml do UPPAAL, de tal maneira que a validação do primeiro possa corresponder a validação do segundo.

A principal mudança, foi a introdução do módulo *ciclo()*, exposto a seguir, o qual corresponde ao autômato *Scan_Cycle* da FIG. 6.6 (pág. 72).

```
MODULE ciclo() VAR
  estado : {lendo, executando, escrevendo};
ASSIGN
  init(estado) := lendo;
  next(estado) := case
              estado = lendo : {executando};
              estado = executando : {executando, escrevendo};
```

```
estado = escrevendo : {lendo};
1 : estado;
esac;
```

As demais alterações complementam a anterior.

1. Para cada entrada de OLIVEIRA (2006), no novo modelo, acrescenta-se uma variável com a inicial *f* correspondendo a *flag*. Estas variáveis internas são atualizadas quando a instância de ciclo() está no estado lendo, conforme o código escrito na declaração ASSIGN, comentado no ítem 5.

   Ex.: *Pump_On: boolean;* passa a *Pump_On, fPump_On: boolean;*

2. Acrescenta-se à declaração VAR a declaração da instância de módulo ciclo(): *clp: ciclo();* e as declarações das variáveis de saída: *out[0], ... , out[6], out[7], out[8] : boolean;*. Estas variáveis de saída são atualizadas quando a instância de ciclo() está no estado escrevendo (clp.estado = escrevendo é verdadeiro), também conforme o código escrito na declaração ASSIGN, comentado no ítem 5.

3. Os módulos passam a receber o estado da instância de ciclo(), e têm suas atualizações internas realizadas somente quando o estado passado é igual a executando.

   Ex.: *MODULE setreset(entrada_set, entrada_reset)* passa a *MODULE setreset(entrada_set, entrada_reset, momento)*.

4. A declaração DEFINE sofre os reflexos das alterações anteriores.

   Ex.: *SR1: resetset((Start_Filling_Tank_A), (Stop_Filling_Tank_A));* passa a *fSR1: resetset((fStart_Filling_Tank_A), (fStop_Filling_Tank_A), clp.estado);*

5. A declaração ASSIGN é acrescentada para acomodar as atualizações das variáveis internas (que correspondem a entradas) e das variáveis de saída.

   Ex.:

```
ASSIGN
next(fTank_A_Level_High):= case
        clp.estado = lendo : Tank_A_Level_High;
        1 : fTank_A_Level_High;
 esac;
...
```

```
next(out[6]):= case
        clp.estado = escrevendo : fout[6];
        1 : out[6];
 esac;
```

6. Adiciona-se a assertiva

```
FAIRNESS clp.estado = escrevendo
```

a significar que *AF clp.estado = escrevendo* e, pela construção de *ciclo()*, *AF clp.estado = lendo* e *AF clp.estado = executando*.

A equipe da UFCG desenvolveu uma ferramenta para conversão do modelo xml extraído do diagrama de lógica, para o modelo xml do UPPAAL (BARBOSA et al.). Baseada e integrada nesta ferramenta, como parte deste trabalho, foi confeccionada uma ferramenta (seta assinalada com "IME" na FIG. 7.1) para conversão do mesmo modelo xml de origem, para o modelo *smv*.

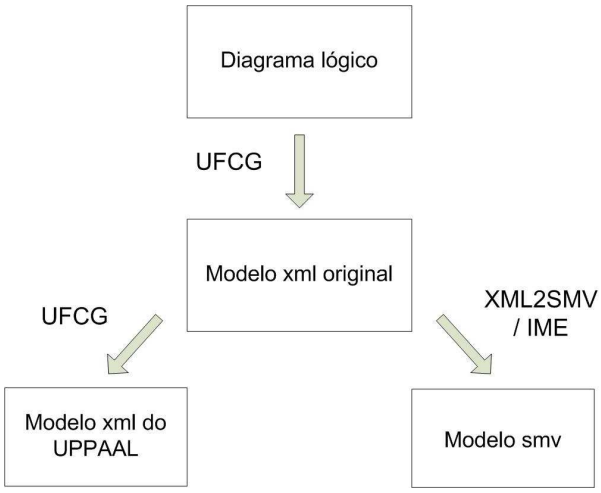


FIG.7.1: Ferramentas desenvolvidas.

As ferramentas da FIG. 7.1 foram desenvolvidas em Java.

7.1.1 EXTRAÇÃO AUTOMÁTICA DAS ESPECIFICAÇÕES

Para permitir a verificação proposta, falta obter as especificações escritas em CTL.

Neste trabalho, considerar-se-á a MCE para desligamento de um equipamento ou planta por segurança (SIS). E segundo esta interpretação, a prioridade é sempre para a ativação do sinal de parada.

O primeiro passo nessa direção é estabelecer um padrão de arquivo fonte da MCE. Neste trabalho foi considerado, que o mesmo estará disponível no formato *xls* do *Excel*, de acordo com o padrão abaixo.

TAB.7.1: Padrão de MCE no *Excel*.

Efeito	out[1]	out[2]	out[3]
Causa			
in[1]	x		
in[2]&in[3]	x		
in[4]		x	x

De posse desse arquivo, deve-se abrí-lo no *Excel* e então salvá-lo em *txt*, escolhendo-se a tabulação como separação das células. O resultado deverá ser parecido com o colocado abaixo:

```
Efeito  out[1]  out[2]  out[3]
Causa
in[1]   x
in[2]&in[3] x
in[4]       x   x
```

A ferramenta desenvolvida, *MCE2ctl*, produz para cada coluna da MCE, as especificações 3.3 e 3.4 (página 41) correspondentes. Tendo como entrada o arquivo acima, a seguinte saída será produzida (num arquivo texto):

```
SPEC AG (((in[1])|(in[2]&in[3])) -> AF(out[1]))
SPEC AG ( !(out[1])-> !E(!((in[1])|(in[2]&in[3])) U ((out[1]) & ((in[1])
|(in[2]&in[3])))))
SPEC AG (((in[4])) -> AF(out[2]))
SPEC AG ( !(out[2])-> !E(!((in[4])) U ((out[2]) & ((in[4])))))
SPEC AG (((in[4])) -> AF(out[3]))
SPEC AG ( !(out[3])-> !E(!((in[4])) U ((out[3]) & ((in[4])))))
```

A ferramenta *XML2SMV* (FIG. 7.1) pode agora ser exibida na sua forma completa (FIG. 7.2), tendo como entrada, também o arquivo contendo as especificações e como saída o modelo do sistema somado às especificações.

A próxima seção faz uma breve descrição da *XML2SMV*. A *MCE2ctl* não é descrita em função da sua maior simplicidade.

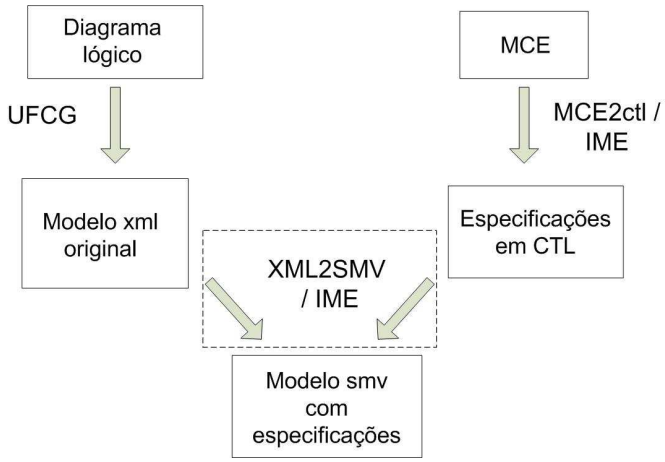


FIG.7.2: Ferramentas desenvolvidas.

7.1.2 DESCRIÇÃO DA FERRAMENTA *XML2SMV*

A *XML2SMV* é completamente apoiada na ferramenta desenvolvida pela UFCG, mostrada na FIG. 7.1. Até o momento da publicação desta dissertação ainda não havia um documento descrevendo a citada ferramenta. A estrutura da *XML2SMV* se baseia em seis classes principais, cujas funcionalidades são indicadas na TAB. 7.2 a seguir:

TAB.7.2: Principais classes que compõem a ferramenta XML2SMV e algumas de suas funcionalidades.

Classe	**Funcionalidade**
XML2SMV	É a classe principal que chama as demais rotinas.
XMLProcessor	Carrega o documento xml numa estrutura a qual permite a utilizacao da interface SAX para lidar com documentos xml.
GlobalDeclarationGenerator	Preenche o objeto *ofg.data* da classe *OutputFileGenerator* com o código gerado e chama o método *public void seekAndReplace()* também da classe *OutputFileGenerator*.
SpecDeclarationGenerator	Preenche o objeto *ofg.data* com as especificações em lógica CTL e chama o método *public void seekAndReplace()*.
CodeGenerator	Possui os métodos que geram partes do código: *public String handlingString(String str)*, *public String handlingInputs(int ind, Stack stack)*
OutputFileGenerator	Possui os objeto *ofg.data* e *ofg.auxiliarVector* do tipo *Vector<String>*, os quais armazenam strings, possui o método *public void seekAndReplace()* o qual compõe o código em *ofg.auxiliarVector*.

Parte da classe *XML2SMV* é exposta a seguir para se visualizar a seqüência dos principais passos realizados.

```
public class Principal {

    // A qui vem algumas declaracoes de objetos.

    public static void main(String[] args) {

        // Efetua-se o tratamento da entrada (o nome do arquivo com o
        // modelo xml e o do arquivo txt com a(s) especificacao(oes)).

            // Inicia-se o 1.o passo: carregar o documento xml numa
            // estrutura a qual permite a utilizacao da interface SAX
            // para lidar com documentos xml.

            XMLProcessor xmlProcessor = new XMLProcessor();

            doc = xmlProcessor.init(args[0]);
            root = doc.getDocumentElement();

        // Faz-se a instanciacao

        // O 1.o passo eh terminado, com a extracao de algumas informacoes
        // gerais, por exemplo, numero de entradas.

        ofg.setOutputFile(OUT_XML_FILE);

        gdg.createGlobalDeclarations();  // Este eh o 2.o passo, onde todo
                                         // o modelo smv   gerado.

        scdg.createSpecDeclarations();   // Neste 3.o passo, a(s)
                                         // especificacao(oes) e(sao)
                                         // inserida(s).

        ofg.generateOutput(); // 4.o passo, o vetor com todo as strings
```

```
                                //(ofg.auxiliarVector) eh escrito no arquivo
                                // de saida.

        System.out.println("Modelo Gerado com Sucesso!!!
        \nArquivo de saida ==> SMV_XML_out.xml");
    }

}
```

O segundo passo é detalhado um pouco mais por meio da FIG. 7.3.

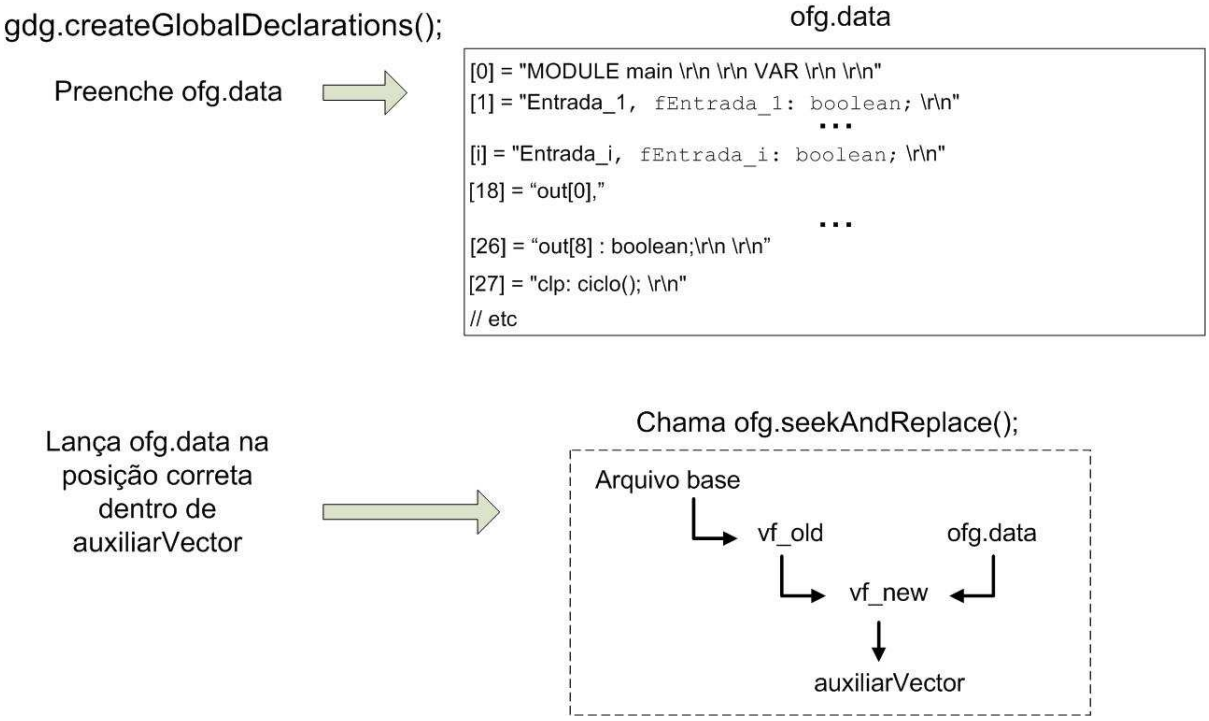


FIG.7.3: 2º passo da ferramenta *XML2SMV*.

O objeto *ofg.data* é oriundo da classe *OutputFileGenerator*, do tipo *Vector<String>*, o qual irá armazenar todo o código gerado.

O arquivo base é o arquivo txt que contém o esqueleto do arquivo smv, ou seja, as marcas e declarações comuns a todos os arquivos, exemplo: *Main*, *SPEC*, e todos os módulos, *ciclo()*, etc (vide ANEXO 2). O conteúdo deste arquivo é passado ao objeto do tipo *Vector<String>*, *vf_old*, do método *public void seekAndReplace()*.

O objeto *vf_new* é oriundo do método *public void seekAndReplace()*, do tipo *Vector<String>*, o qual irá armazenar todo o código gerado a partir da composição entre *vf_old* e *ofg.data*.

O terceiro passo, exibido na FIG. 7.4, conclui a montagem do código no *ofg.auxiliarVector*.

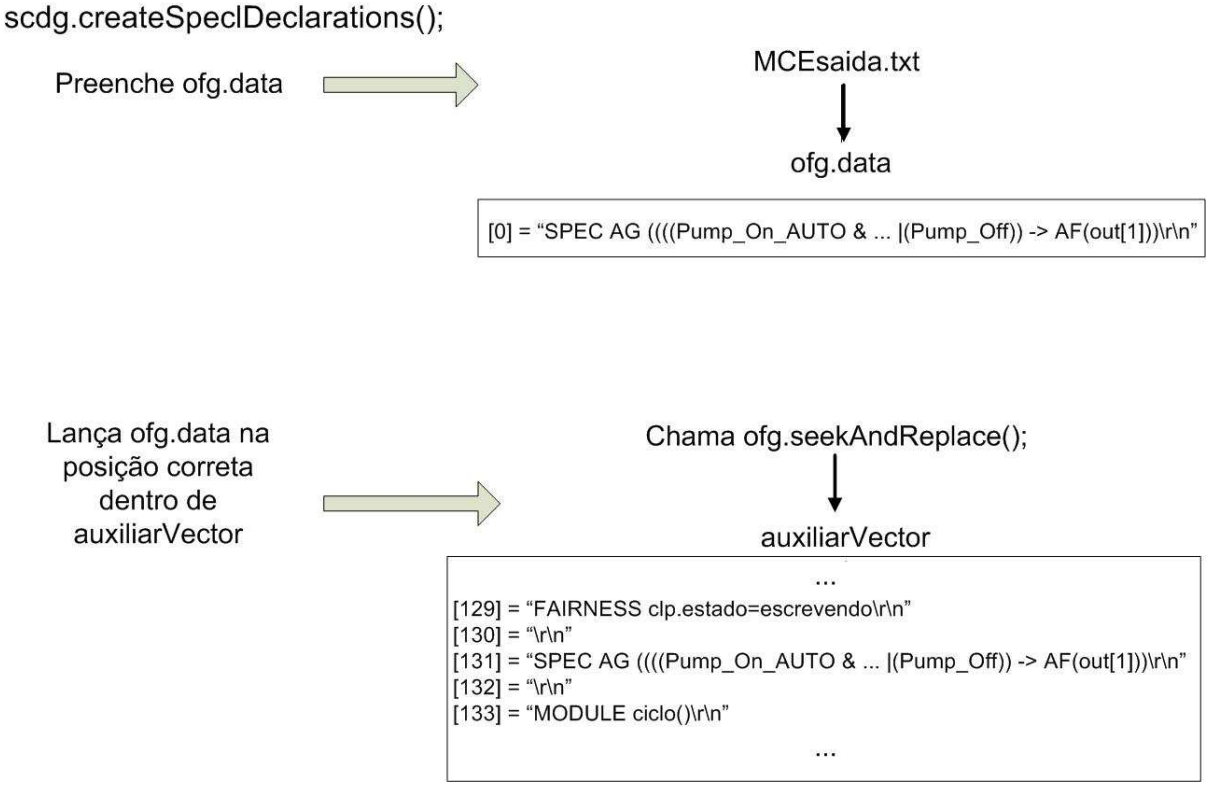


FIG.7.4: 3º passo da ferramenta *XML2SMV*.

7.2 EXECUÇÃO DE EXEMPLO

A partir da MCE da TAB. 6.2 e do arquivo *Instance_Example_ISA.xml* (vide Modelo 1 do ANEXO 2), chegou-se ao modelo com as especificações também contido no Modelo 3 do ANEXO 2.

A verificação obteve os seguintes resultados:

```
Model checking results
======================
(AG (((((((((Pump_On_AUTO&(~Valve_A_Opened))&Valve_B_Closed)|(......true

See file "SMV_XML_out.warn" for warnings.

user time.............................................0.078125 s system
```

```
time.................................................0.015625 s

Resources used
==============
user time...........................................0.078125 s system
time................................................0.015625 s BDD nodes
allocated...........................................25939 data segment
size................................................0
```

Ou seja, a verificação não só foi possível, como foi realizada em cerca de um décimo de segundo. Além disso, foi confirmado que o modelo satisfaz às especificações contidas na MCE.

7.3 RESUMO DO CAPÍTULO

Neste capítulo foi realizada a validação do exemplo ISA, por meio da verificação não temporal utilizando-se a ferramenta SMV, já a mesma não foi possível de ser feita diretamente utilizando-se o verificador UPPAAL. Perdeu-se a verificação das propriedades em tempo real, mas os intertravamentos lógicos do programa, que não levam em conta o tempo, puderam ser analisados.

E mais, esta validação foi feita com grande parte do processo automatizado recorrendo-se às ferramentas *MCE2tctl* e *XML2SMV* também descritas neste capítulo.

8 CONCLUSÃO

8.1 PONTOS ABORDADOS E CONTRIBUIÇÕES

Esta dissertação é uma seqüência do trabalho de OLIVEIRA (2006) e explorou a verificação de modelos aplicadas a sistemas de tempo real. Isto foi feito dentro do contexto do Projeto SIS o qual impôs três aspectos: o sistema de tempo real tratado é um programa de CLP, o sistema possui escala industrial e as etapas propostas para melhoria do processo de engenharia de um SIS devem ser o máximo automatizadas. Do estudo orientado por estes requisitos resultaram duas contribuições:

- Uma proposta de procedimento para escrita de especificações em TCTL tomando como base a modelagem proposta por GORGÔNIO et al. (2006).
- Duas ferramentas para validar um modelo xml do UPPAAL que será utilizado para geração automática de testes. A ferramenta *MCE2ctl* gera um arquivo texto contendo a especificação em CTL no formato aceito pelo verificador SMV e a *XML2SMV* converte um modelo xml extraído do diagrama lógico conforme BARBOSA et al. e base do modelo xml do UPPAAL a ser validado, num modelo aceito pelo SMV, com as especificações que podem ser geradas pela *MCE2ctl*.

8.2 PONTOS NÃO ABORDADOS E LIMITAÇÕES

A verificação de modelos aplicadas a sistemas de tempo real de médio e grande porte, sem se recorrer a intensa intervenção humana por meio de abstrações, é um desafio que ainda resiste. Neste trabalho os limites da ferramenta UPPAAL foram explorados. Não se pesquisou sobre os limites de outras ferramentas similares.

Percebeu-se que um ponto crítico deste obstáculo é a linguagem de entrada da ferramenta em conjunto com a estrutura de dados, autômatos temporizados e DBM, respectivamente. Se este conjunto permitiu a expansão dos limites da verificação de sistemas de tempo real até o padrão atual, para novas expansões dever-se-á sofrer alterações ou mesmo ser substituído.

A escrita de especificações mais complexas, tanto para sistemas de tempo real como para intertravamentos puramente lógicos, ainda requer a intervenção manual.

8.3 PERSPECTIVAS DE TRABALHOS FUTUROS

Para se vencer as limitações com relação à escala do sistema de tempo real a ser verificado, a direção de pesquisa deve passar pela investigação de estruturas de dados e de linguagens de descrição de sistemas de tempo real capazes de suportar sistemas de porte maiores. Entre as linguagens de descrição, pode-se citar as Redes de Petri Temporizadas (WANG, 2004).

9 REFERÊNCIAS BIBLIOGRÁFICAS

ALUR, R., COURCOUBETIS, C. e DILL, D. L. Model-Checking in Dense Real-time. **Information and Computation**, 104(1):2–34, 1993. Disponível: `citeseer.ist.psu.edu/alur93modelchecking.html`.

ALUR, R. e DILL, D. L. A theory of timed automata. **Theoretical Computer Science**, 126(2):183–235, 1994. Disponível: `citeseer.ist.psu.edu/alur94theory.html`.

BARBOSA, L. P. A., GORGÔNIO, K., LIMA, A. M. N., PERKUSICH, A. e DA SILVA, L. D. On the automatic generation of timed automata models from ISA 5.2 diagrams. Em **Proceedings of the 12th IEEE Conference on Emerging Technologies and Factory Automation**.

BEHRMANN, G., DAVID, A. e LARSEN, K. G. A Tutorial on Uppaal, novembro 2004.

BENGTSSON, J., LARSEN, K., PETTERSSON, P. e YI, W. UPPAAL - a tool suite for automatic verification of real-time systems. Em **Proc. of Workshop Hybrid Systems III: Verification and Control**, volume 1066 of **Lecture Notes in Computer Science**, págs. 232–243, New Brunswick, NJ, USA, 1996. Springer – Verlag. Disponível: `www.uppaal.com/`.

BENGTSSON, J. e YI, W. Timed automata: Semantics, Algorithms and Tolls. **Theoretical Computer Science**, 126(2):183–235, 2004. Disponível: `www.uppaal.com/`.

CLARKE, E. M. e EMERSON, E. A. Design and synthesis of synchronization skeletons using branching time temporal logic. Em **Logic of programs, Workshop**, págs. 52–71, London, UK, maio 1981. Springer-Verilag.

CLARKE, E. M., GRUMBERG, O. e PELED, D. A. **Model Checking**. The MIT Press, Cambridge, Massachusetts 02142, 1st edition, 1999.

DE SMET, O. e ROSSI, O. Verification of a controller for a flexible manufacturing line written in Ladder Diagram via model-checking. Em **Proc. of the American Control Conference**, volume 5, págs. 4147–4152, May 2002.

DWYER, M. B., AVRUNIN, G. S. e CORBETT, J. C. Patterns in property specifications for finite-state verification. Em **2nd Workshop on formal methods in software practice**, págs. 7–15, março 1998. Disponível: `citeseer.ist.psu.edu/dwyer99patterns.html`.

FREY, G. e LITZ, L. Formal methods in PLC programming. Em **Proceedings of the IEEE SMC 2000**, págs. 2431–2436, Nashville, Outubro 2000. Disponível: `www.eit.uni-kl.de/frey/papers/PDF/V132.pdf`.

GORGÔNIO, K. C., DA SILVA, L. D., PAULO, L. e PACÍFICO, R. Projeto SIS. Technical report, Universidade Federal de Campina Grande, Campina Grande,PB, dezembro 2006.

GOURCUFF, V., DE SMET, O. e FAURE, J.-M. Efficient representation for formal verification of PLC programs. Em **8o International Workshop on Discrete Event Systems**, págs. 182–187, Ann Arbor, Michigan, USA, Julho 2006.

HENZINGER, T. A., HO, P. H. e WONG-TOI, H. HiTech: a model-checker for hybrid systems. **Int. J. Software Tools Technologies Transf.**, 1:110–122, 1997.

IEC. Specification Languages GRAFCET for Sequential Function Charts. IEC international Standard 60848, Fev 1999.

IEC. Programmable controllers-Part3: Programming languages. IEC international Standard 61131-3, Jan 2003.

ISA. Binary Logic Diagrams for Process Operations ANSI/ISA 5.2. ISA-The Instrumentation, System and Automation Society, Jul 1992.

LAMPÉRIÈRE-COUFFIN, S., ROSSI, O., ROUSSEL, J.-M. e LESAGE, J.-J. Formal Validation of PLC Programs: A Survey. Em **European Control Conference 1999 (ECC'99)**, Karlsruhe, Germany, Ago.-Set. 1999.

LARSEN, K., PETTERSSON, P. e YI, W. Model-Checking for Real-Time Systems. Em **Proc. of the 10th International Conference on Fundamentals of Computation Theory**, volume 965 of **Lecture Notes in Computer Science**, págs. 62–88, Dresden, Germany, Agosto 1995. Horst Reichel (Ed.). Disponível: `www.uppaal.com/`.

LEVESON, N. e TURNER, C. An investigation of the Therac-25 accidents. **Computer pages**, págs. 18–41, 1993.

MADER, A. e WUPPER, H. Timed automaton for simple programmable logic controllers. Em **Proc. of the 11th Euromicro Conference of Real-Time Systems (ECRTS'99)**, págs. 114–122, 1999.

MALER, O. e YOVINE, S. Hardware Timing Verification using KRONOS. Em **Proc. of 7 th Israeli Conference on Computer Systems and Software Engineering**, págs. 12–13, Herzliya, Israel, junho 1996. Disponível: `citeseer.ist.psu.edu/maler96hardware.html`.

MCMILLAN, K. L. **Getting started with SMV**. Cadence Berkeley Labs, 2001 Addison St., Berkeley, USA, Março 1999.

MCMILLAN, K. L. **The SMV system, for SMV version 2.5.4**. Carnegie Mellon University, Novembro 2001. Disponível: `www.cs.cmu.edu/ modelcheck/smv.html`. do original de fevereiro de 1992.

MOKADEM, H. B. **Vérification des propriétés temporisées des automates programmables industriels.** Tese de Doutorado, École Normale Supérieure de Cachan, setembro 2006.

MOON, I. Modelling Programmable Logic Controllers for Logic Verification. **IEEE Control Systems Magazine**, 14(6), 1994.

OLIVEIRA, C. H. C. Verificação de Modelos Aplicada ao Projeto de Sistemas Industriais Automatizados por Controladores Lógicos Programáveis. Dissertação de Mestrado, Instituto Militar de Engenharia, Rio ed Janeiro, BR, 2006.

QUIELLE, J. P. e SIFAKIS, J. Specification and verification of concurrent systems in CESAR. Em **5th International Symposium on Programming**, volume 137, págs. 337–351. Springer-Verilag, 1982.

RAUSCH, M. e KROGH, B.-H. Formal Verification of PLC Programs. Em **Proc. American Control Conference**, págs. 234–238, Philadelphia, PA, USA, Junho 1998. Disponível: `citeseer.ist.psu.edu/rausch98formal.html`.

SPARACO, P. Board Faults Ariane 5 Software. **Aviation Week & Space Technology**, págs. 33–35, Julho 1996.

T. A. HENZINGER, X. NICOLLIN, J. SIFAKIS e S. YOVINE. Symbolic Model Checking for Real-Time Systems. Em **7th. Symposium of Logics in Computer Science**, págs. 394–406, Santa-Cruz, California, 1992. IEEE Computer Scienty Press. Disponível: `citeseer.ist.psu.edu/henzinger92symbolic.html`.

UPPAAL. **Endereço da página onde está disponível o programa**. 2008. Disponível: `http://www.uppaal.com`. [último acesso em 09 jan. 2008].

WANG, F. Formal Verification of Timed Systems: A Survey and Perspectives. Em **Proc. of the IEEE**, volume 92, págs. 1283–1305, Agosto 2004.

WILLEMS, H. X. Compact timed automata for PLC programs. Technical Report CSI-R9925, University of Nijmegem, Nijmegem,NT, novembro 1999.

YOVINE, S. Kronos: a verification tool for rel-time systems. **on Software Tools for Technology Transfer**, (1), outubro 1997.

ZOUBEK, B. **Automatic verification of temporal and timed properties of control programs.** Tese de Doutorado, University of Birmingham, Birmingham, UK, setembro 2004.

10 ANEXOS

10.1 ANEXO 1: ARQUIVOS UTILIZADOS

Este anexo apresenta o modelo do sistema do programa de controle de prensas do exemplo 6.1 da página 85, utilizado no verificador UPPAAL.

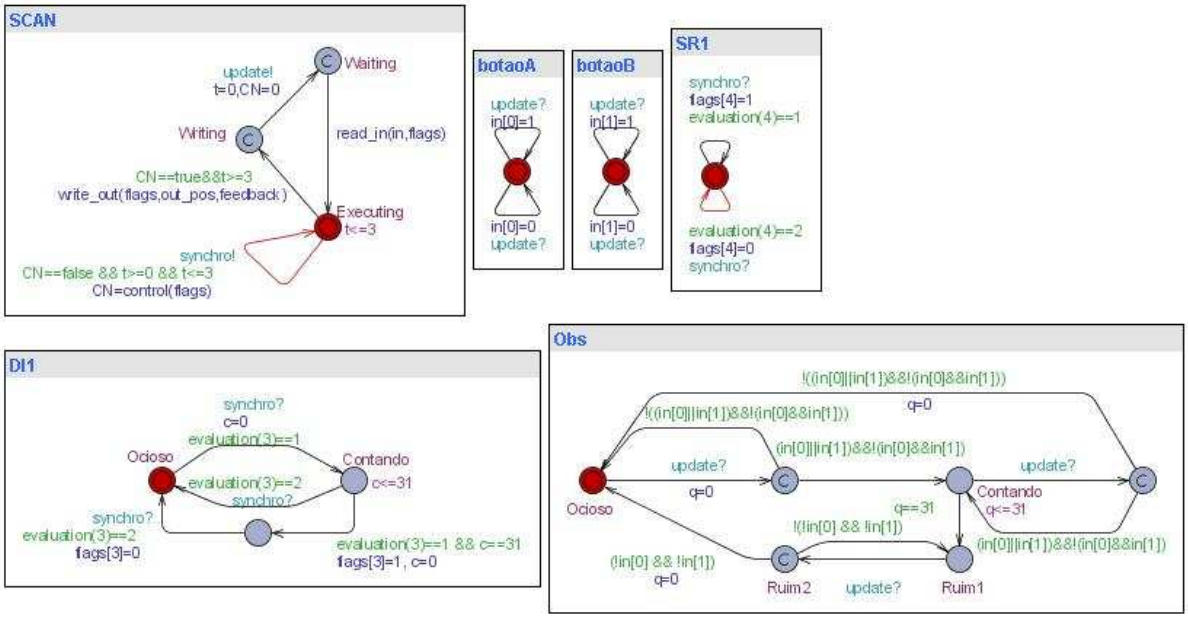


FIG.10.1: Rede de autômatos temporizados do modelo do programa de controle das prensas do exemplo 6.1

Declaração global

```
// Modelo Gerado Automaticamente!

broadcast chan update, synchro;

const int NUM_FEEDBACKS = 1;
const int NUM_ELEM = 4 + NUM_FEEDBACKS;
const int NUM_INPUTS = 2 + NUM_FEEDBACKS;
const int NUM_OUTPUTS = 1;
const int NUM_NEG_OUTPUTS = 0;
const int NUM_INT_SIG = NUM_ELEM - NUM_INPUTS;
const int NUM_TESTE = 2;
```

```
bool CN; bool flags[NUM_ELEM];
bool in[NUM_INPUTS];
bool out[NUM_OUTPUTS];


int out_pos[NUM_OUTPUTS - NUM_NEG_OUTPUTS] = {4};
//int neg_out_pos[NUM_NEG_OUTPUTS] = {};
int feedback[NUM_FEEDBACKS]={4};
teste_pos[NUM_TESTE]={0,1};

clock t;   // t   o tempo dentro de cada ciclo

int id_botaoA = 0; int id_botaoB = 1;

int id_DI1 = 3; int id_SR1 = 4;


int evaluation(int id) {

if (id == id_DI1) {

  if ((flags[id_botaoA] || flags[id_botaoB]) &&
                        !flags[feedback[0]]) return 1;
  if (!((flags[id_botaoA] || flags[id_botaoB]) &&
                        !flags[feedback[0]])) return 2;
  else return 0;

}

if (id == id_SR1) {

  if ((!flags[id_DI1] && flags[id_botaoA] &&
                        flags[id_botaoB]) && !(!flags[id_botaoA]
                        || !flags[id_botaoB])) return 1;
```

```
  if (!flags[id_botaoA] || !flags[id_botaoB]) return 2;
  else return 0;

}

return 0;

}
```

Declarações do sistema

```
SCAN = Scan_Cicle(3); // 3 d cimos de segundos

botaoA = Env_(0);
botaoB = Env_(1);
DI1 = DI_(3,31);
SR1 = SR_(4);
Obs = Observ_();

system SCAN, botaoA, botaoB, SR1, DI1, Obs;
```

10.2 ANEXO 2: MODELOS OBTIDOS

Este anexo exibe alguns dos modelos usados e gerados pelas ferramentas *MCE2tctl* e *XML2SMV* no capítulo 7.

Modelo 1: Modelo xml extraído do diagrama lógico da FIG. 6.2 (página 70), arquivo *Instance_Example_ISA.xml*, primeiras páginas.


```
<?xml version="1.0" encoding="UTF-8"?> <project
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation='Esquema2.xsd'>
    <header>
        <typeOfDocument>EXEMPLO</typeOfDocument>
        <idDocument>AA-0000.00-0000-000-AAA-000</idDocument>
        <client>XXXX</client>
        <page>01 de 03</page>
        <program>SIS</program>
        <area>PESQUISA</area>
        <title>Opera § fo de Enchimento de Tanques</title>
        <revision>
            <idRev>REV.0</idRev>
            <!--Element date is optional-->
            <date>2007-04-25</date>
            <!--Element idProject is optional-->
            <idProject>XXXX</idProject>
            <!--Element execution is optional-->
            <execution>XXXX</execution>
            <!--Element verification is optional-->
            <verification>XXXX</verification>
            <!--Element approval is optional-->
            <approval>XXXX</approval>
        </revision>
    </header>
    <sheet id="01">
        <sheetHeader>
            <page>01 de 02</page>
        </sheetHeader>
        <description>Diagrama L gico - Parte1</description>
        <!--Element symbol, maxOccurs=unbounded-->
        <symbol>
            <!--Element name is optional-->
```

```
            <name>Tank A Level High</name>
            <idSymbol>sym1</idSymbol>
            <symType>fisical</symType>
            <!--Element symSubType, maxOccurs=unbounded-->
            <symSubType>IN</symSubType>
            <output feedback="false" negated="false">
                <!--Attribute negated is optional-->
                <!--Attribute feedback is optional-->
                <idOutput>sym1_out1</idOutput>
                <!--Element outConInfo, maxOccurs=unbounded-->
                <outConInfo>
                   <!--Element conSymName is optional-->
                   <conSymName>Tank A Level High Indicator</conSymName>
                   <conSymId>sym2</conSymId>
                   <conInpId>sym2_in1</conInpId>
                </outConInfo>
                <outConInfo>
                   <!--Element conSymName is optional-->
                   <conSymName>AND1</conSymName>
                   <conSymId>sym3</conSymId>
                   <conInpId>sym3_in1</conInpId>
                </outConInfo>
            </output>
        </symbol>
    <symbol>
```

Modelo 3: Conteúdo do arquivo *MCEsaida.txt*, gerado pela MCE contida na TAB. 6.2, por meio do programa *MCE2ctl*.

```
SPEC AG ((((Pump_On_AUTO & !Valve_A_Opened & Valve_B_Closed ) |
    (Pump_On_AUTO & Valve_A_Opened & !Valve_B_Closed ) )|
    ((Pump_On_AUTO & !Valve_B_Opened & Valve_A_Closed) |
    (Pump_On_AUTO & Valve_B_Opened & !Valve_A_Closed ) )|
    ((Pump_On_AUTO & Valve_A_Opened & Valve_B_Closed &
    Tank_A_Level_High))|((Pump_On_AUTO & Valve_B_Opened &
    Valve_A_Closed & Tank_B_Level_High))|(Pump_Motor_Overloaded)|
    (Pump_Off)) -> AF(!out[5]))
```

Modelo 3: Código em SMV completo para o modelo ISA. Gerado por meio do programa *XML2SMV* com os arquivos *Instance_Example_ISA.xml* e *MCEsaida.txt* de entrada.

```
MODULE main

 VAR

Tank_A_Level_High, fTank_A_Level_High: boolean;
Start_Filling_Tank_A, fStart_Filling_Tank_A: boolean;
Valve_A_Opened, fValve_A_Opened: boolean;
Stop_Filling_Tank_A, fStop_Filling_Tank_A: boolean;
Valve_B_Closed, fValve_B_Closed: boolean;
Start_Filling_Tank_B, fStart_Filling_Tank_B: boolean;
Valve_B_Opened, fValve_B_Opened: boolean;
Stop_Filling_Tank_B, fStop_Filling_Tank_B: boolean;
Valve_A_Closed, fValve_A_Closed: boolean;
Tank_B_Level_High, fTank_B_Level_High: boolean;
Pump_On, fPump_On: boolean;
Pump_On_AUTO, fPump_On_AUTO: boolean;
Pump_Off, fPump_Off: boolean;
Pump_Suct_Press_Low, fPump_Suct_Press_Low: boolean;
Pump_Motor_Overloaded, fPump_Motor_Overloaded: boolean;
Reset_Pump_Motor_Starter, fReset_Pump_Motor_Starter: boolean;
Pump_Seal_Water_Press_Low, fPump_Seal_Water_Press_Low: boolean;
OUT0, out[1], OUT2, OUT3, OUT4, out[5], OUT6, OUT7, OUT8 : boolean;

clp: ciclo();
fSR1: resetset((fStart_Filling_Tank_A), (fStop_Filling_Tank_A),
    clp.estado);

fSR2: resetset((fStart_Filling_Tank_B), (fStop_Filling_Tank_B),
    clp.estado);

fSR3: resetset((((((fTank_B_Level_High & fValve_A_Closed &
    fValve_B_Opened) | (fValve_B_Closed & fValve_A_Opened &
    fTank_A_Level_High)) & fPump_On_AUTO) | fPump_On)),
    ((next(fSR4.saida) | next(fDI1.saida) | fPump_Off |
    ((((fTank_B_Level_High & fValve_A_Closed & fValve_B_Opened) |
    (fValve_B_Closed & fValve_A_Opened & fTank_A_Level_High)) &
    fPump_On_AUTO) | fPump_On))), clp.estado);

fDI1: DI(fPump_Suct_Press_Low, clp.estado);

fSR4: setreset((fPump_Motor_Overloaded),(fReset_Pump_Motor_Starter),
    clp.estado);
```

```
DEFINE

fOUT0 := fTank_A_Level_High;
fout[1] := fSR1;
fOUT2 := fSR2;
fOUT3 := fTank_B_Level_High;
fOUT4 := fSR3;
fout[5] := fSR3;
fOUT6 := fPump_Seal_Water_Press_Low;

ASSIGN

next(fTank_A_Level_High):= case
clp.estado = lendo : Tank_A_Level_High;
1 : fTank_A_Level_High;
 esac;

next(fStart_Filling_Tank_A):= case
clp.estado = lendo : Start_Filling_Tank_A;
1 : fStart_Filling_Tank_A;
 esac;

next(fValve_A_Opened):= case
clp.estado = lendo : Valve_A_Opened;
1 : fValve_A_Opened;
 esac;

next(fStop_Filling_Tank_A):= case
clp.estado = lendo : Stop_Filling_Tank_A;
1 : fStop_Filling_Tank_A;
 esac;

next(fValve_B_Closed):= case
clp.estado = lendo : Valve_B_Closed;
1 : fValve_B_Closed;
 esac;

next(fStart_Filling_Tank_B):= case
clp.estado = lendo : Start_Filling_Tank_B;
1 : fStart_Filling_Tank_B;
 esac;
```

```
next(fValve_B_Opened):= case
clp.estado = lendo : Valve_B_Opened;
1 : fValve_B_Opened;
 esac;

next(fStop_Filling_Tank_B):= case
clp.estado = lendo : Stop_Filling_Tank_B;
1 : fStop_Filling_Tank_B;
 esac;

next(fValve_A_Closed):= case
clp.estado = lendo : Valve_A_Closed;
1 : fValve_A_Closed;
 esac;

next(fTank_B_Level_High):= case
clp.estado = lendo : Tank_B_Level_High;
1 : fTank_B_Level_High;
 esac;

next(fPump_On):= case
clp.estado = lendo : Pump_On;
1 : fPump_On;
 esac;

next(fPump_On_AUTO):= case
clp.estado = lendo : Pump_On_AUTO;
1 : fPump_On_AUTO;
 esac;

next(fPump_Off):= case
clp.estado = lendo : Pump_Off;
1 : fPump_Off;
 esac;

next(fPump_Suct_Press_Low):= case
clp.estado = lendo : Pump_Suct_Press_Low;
1 : fPump_Suct_Press_Low;
 esac;

next(fPump_Motor_Overloaded):= case
```

```
clp.estado = lendo : Pump_Motor_Overloaded;
1 : fPump_Motor_Overloaded;
 esac;

next(fReset_Pump_Motor_Starter):= case
clp.estado = lendo : Reset_Pump_Motor_Starter;
1 : fReset_Pump_Motor_Starter;
 esac;

next(fPump_Seal_Water_Press_Low):= case
clp.estado = lendo : Pump_Seal_Water_Press_Low;
1 : fPump_Seal_Water_Press_Low;
 esac;

next(OUT0):= case
clp.estado = escrevendo : fOUT0;
1 : OUT0;
 esac;

next(out[1]):= case
clp.estado = escrevendo : fout[1];
1 : out[1];
 esac;

next(OUT2):= case
clp.estado = escrevendo : fOUT2;
1 : OUT2;
 esac;

next(OUT3):= case
clp.estado = escrevendo : fOUT3;
1 : OUT3;
 esac;

next(OUT4):= case
clp.estado = escrevendo : fOUT4;
1 : OUT4;
 esac;

next(out[5]):= case
clp.estado = escrevendo : fout[5];
1 : out[5];
```

```
 esac;

next(OUT6):= case
clp.estado = escrevendo : fOUT6;
1 : OUT6;
 esac;


FAIRNESS clp.estado=escrevendo

SPEC AG ((((Pump_On_AUTO & !Valve_A_Opened & Valve_B_Closed ) |
    (Pump_On_AUTO & Valve_A_Opened & !Valve_B_Closed ) )|
    ((Pump_On_AUTO & !Valve_B_Opened & Valve_A_Closed) |
    (Pump_On_AUTO & Valve_B_Opened & !Valve_A_Closed ) )|
    ((Pump_On_AUTO & Valve_A_Opened & Valve_B_Closed &
    Tank_A_Level_High))|((Pump_On_AUTO & Valve_B_Opened &
    Valve_A_Closed & Tank_B_Level_High))|(Pump_Motor_Overloaded)|
    (Pump_Off)) -> AF(!out[5]))

MODULE ciclo()
VAR
  estado : {lendo, executando, escrevendo};
ASSIGN
  init(estado) := lendo;
  next(estado) := case
              estado = lendo : {executando};
              estado = executando : {executando, escrevendo};
              estado = escrevendo : {lendo};
              1 : estado;
              esac;

MODULE setreset(entrada_set,entrada_reset,momento)
VAR
  estado : boolean;
ASSIGN
  init(estado) := 0;
  next(estado) := case
              (momento = executando) & entrada_set &
                    !entrada_reset: {1};
              (momento = executando) & entrada_reset &
                    !entrada_set: {0};
              (momento = executando) & entrada_set &
```

```
                    entrada_reset : {1};
              1 : estado;
              esac;
DEFINE
   saida := estado;

MODULE resetset(entrada_set,entrada_reset, momento)
VAR
  estado : boolean;
ASSIGN
  init(estado) := 0;
  next(estado) := case
              (momento = executando) & entrada_set &
                    !entrada_reset: {1};
              (momento = executando) & entrada_reset &
                    !entrada_set: {0};
              (momento = executando) & entrada_set &
                    entrada_reset : {0};
              1 : estado;
              esac;
DEFINE
   saida := estado;

MODULE DI(entrada, momento)
VAR
  estado : {parado,contando,terminado};
ASSIGN
  init(estado) := parado;
  next(estado) := case
              (momento = executando) & estado=parado &
                    !entrada: {parado};
              (momento = executando) & estado=parado &
                    entrada : {contando};
              (momento = executando) & estado=contando &
                    entrada: {terminado,contando};
              (momento = executando) & estado=contando &
                    !entrada: {parado};
              (momento = executando) & estado=terminado &
                    entrada : {terminado};
              (momento = executando) & estado=terminado &
                    !entrada : {parado};
              1 : estado;
```

```
            esac;
DEFINE
   saida := estado = terminado;

MODULE DT(entrada, momento)
VAR
  estado : {parado,contando,alto};
ASSIGN
  init(estado) := parado;
  next(estado) := case
              (momento = executando) & estado=parado &
                    !entrada: {parado};
              (momento = executando) & estado=parado &
                    entrada : {alto};
              (momento = executando) & estado=alto &
                    entrada: {alto};
              (momento = executando) & estado=alto &
                    !entrada: {contando};
              (momento = executando) & estado=contando &
                    entrada : {alto};
              (momento = executando) & estado=contando &
                    !entrada : {contando,parado};
              1 : estado;
             esac;
DEFINE
   saida := estado = alto;

MODULE PO(entrada, momento)
VAR
  estado : {parado,contando,terminado};
ASSIGN
  init(estado) := parado;
  next(estado) := case
              (momento = executando) & estado=parado &
                    !entrada: {parado};
              (momento = executando) & estado=parado &
                    entrada: {contando};
              (momento = executando) & estado=contando:
                    {contando, terminado};
              (momento = executando) & estado=terminado &
                    entrada: {terminado};
              (momento = executando) & estado=terminado &
```

```
                    !entrada: {parado};
              1 : estado;
             esac;
DEFINE
   saida := estado = contando;
```

11 APÊNDICES

11.1 APÊNDICE 1: TRANSCRIÇÃO DO EXEMPLO DO APÊNDICE A DA NORMA ISA 5.2

A operação pode ser feita manualmente (posição ON) ou de forma automática (posição AUTO) de acordo com o selecionado na chave seletora HS-7 que ainda possui a posição OFF para desligamento manual. Quando a bomba está em operação, a lâmpada piloto vermelha L-8A fica ligada, quando não, é a lâmpada piloto verde L-8B que fica ligada. Uma vez ligada, a bomba continua a operar até a ocorrência de um comando de *parar* (descrito mais a frente) ou até a perda da energização do sistema de controle.

A bomba pode ser operada manualmente a qualquer tempo desde que não exista uma condição impeditiva (que é um subconjunto das condições de habilitação do comando de "parar"), quais sejam:

1. A pressão de sucção ficar baixa por 5 s continuamente (PSL-5 > 5 s).
2. A pressão d'água de lastro não deve estar baixa (PSL-6).
3. O motor da bomba não pode estar sobrecarregado e, ocorrido a sobrecarga, a indicação deste fato (o relé térmico) deve estar *resetada*.

As chaves de contato manuais montadas em painel(éis), HS-1 e HS-2, correspondem ao comando da operação para encher o tanques A ou B respectivamente. Cada chave possui duas posições LIGA e DESLIGA. LIGA HS-1 (HS-2) energiza a solenóide HY-1 (HY-2). A desernegização de uma válvula solenóide leva-a para a posição de falha-segura, isto é, fechada. Isto despressuriza o atuador pneumático da válvula de controle associada, HV-1 ou HV-2. A despressurização de uma válvula de controle leva-a para a posição de falha-segura, isto é, aberta. As válvulas de controle possuem sensores que indicam sua posição aberta, ZSH-1 e ZSH-2, e sua posição fechada ZSL-1 e ZSL-2.

A posição DESLIGADA das chaves HS-1 e HS-2 causam ações opostas, ou seja, as válvulas solenóides ficam energizadas, os atuadores de controle são pressurizados e as válvulas de controle são fechadas.

Se a energia do circuito cai durante a partida, a memória do mesmo se perde e a operação é interrompida. O comando para parar a operação deve prevalecer sobre o

comando de início da operação.

Para operar a bomba automaticamente, todas as seguintes condições precisam estar satisfeitas.

1. Uma das duas válvulas HS-1 ou HS-2 deve estar aberta e a outra válvula de controle deve estar fechada, dependendo do tanque A ou B estar enchendo.

2. A pressão da bomba de sucção deve estar acima de um valor pré-estabelecido, como assinalado pelo pressostato PSL-5.

3. Se a válvula HV-1 estiver aberta para permitir o bombeamento para o tanque A, o nível do tanque deve estar abaixo de um valor determinado, como indicado pelo sensor de nível LSH-3 o qual também acende a lâmpada piloto de nível alto no tanque LLH-3, localizada no painel. De forma similar, o sensor de nível, LSH-4, permite o bombeamento para o tanque B se não atuado (ativo) e acende a lâmpada piloto LLH-4 caso contrário.

4. A pressão d'água de lastro deve estar adequada e pode ser acompanhada no indicador de pressão PI-6. Este é um requisito não intertravado que depende da atenção do operador antes dele dar início à operação. O pressostato, PSL-6, atrás do painel, somente aciona o alarme de pressão baixa, PAL-6, também no painel.

5. O motor da bomba não deve estar sobrecarregado e, tendo a sobrecarga ocorrido, a indicação deste fato deve estar *resetada*.

O comando de *parar* a bomba, por sua vez, é provocado por quaisquer das condições abaixo:

1. Estando a bomba no comando automático e enchendo um tanque, a válvula de controle correspondente deixa sua posição de totalmente aberta, ou a válvula correspondente ao outro tanque deixa sua posição de totalmente fechada.

2. Estando a bomba no comando automático e o tanque selecionado para encher atinge o nível desejado (LLH).

3. A pressão da bomba de sucção está continuamente baixa por 5 segundos (PSL-5 > 5 s).

4. O motor da bomba está sobrecarregado.

5. A seqüência é parada manualmente através de HS-1 ou HS-2. Se o comando de parar e ligar ocorrem simultaneamente, então o comando de parar deve prevalecer sobre o comando de ligar.

6. A bomba é parada manualmente através de HS-7.

7. A pressão da bomba d'água de selagem está baixa (PSL-6). Esta condição não está intertravada, e requer intervenção manual em ordem de parar a bomba.

Livros Grátis

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

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas