UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TESE DE DOUTORADO

GERAÇÃO E EXECUÇÃO AUTOMÁTICA DE TESTES PARA PROGRAMAS DE CONTROLADORES LÓGICOS PROGRAMÁVEIS PARA SISTEMAS INSTRUMENTADOS DE SEGURANÇA

KÉZIA DE VASCONCELOS OLIVEIRA

CAMPINA GRANDE - PB
FEVEREIRO DE 2014

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Geração e Execução Automática de Testes para Programas de Controladores Lógicos Programáveis para Sistemas Instrumentados de Segurança

Kézia de Vasconcelos Oliveira

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Angelo Perkusich e Leandro Dias da Silva
(Orientadores)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

O48g Oliveira, Kézia de Vasconcelos.

Geração e execução automática de testes para programas de controladores lógicos programáveis para sistemas instrumentados de segurança / Kézia de Vasconcelos Oliveira. – Campina Grande, 2014.

125 f. : il.

Tese (Doutorado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

"Orientação: Prof. Angelo Perkusich, Prof. Leandro Dias da Silva".

Referências.

1. Sistemas Instrumentados de Segurança. 2. Rede de Autômatos Temporizados. 3. Controladores Lógicos Programáveis. I. Perkusich, Angelo. II. Silva, Leandro Dias da. III. Título.

CDU 004.72(043)

"GERAÇÃO E EXECUÇÃO AUTOMÁTICA DE TESTES PARA PROGRAMAS DE CONTROLADORES LÓGICOS PROGRAMÁVEIS PARA SISTEMAS INSTRUMENTADOS DE SEGURANÇA"

KÉZIA DE VASCONCELOS OLIVEIRA

TESE APROVADA EM 17/02/2014

ANGELO PERKUSICH, D.Sc, UFCG
Orientador(a)

LEANDRO DIAS DA SILVA, D.Sc, UFAL
Orientador(a)

HYGGO OLIVEIRA DE ALMEIDA, D.Sc, UFCG
Examinador(a)

KYLLER COSTA GORGÔNIO, Dr., UFCG
Examinador(a)

GIOVANNI CORDEIRO BARROSO, D.Sc, UFC
Examinador(a)

ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFCG
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Sistemas Instrumentados de Segurança (SIS) são desenvolvidos para garantir a segurança operacional de sistemas industriais prevenindo a ocorrência de situações indesejadas quando da execução de procedimentos realizados automaticamente ou sob a interferência de operadores humanos. No contexto de SIS é fundamental garantir a confiança e a segurança no funcionamento, pois defeitos no hardware, no software ou ainda erros humanos podem ocasionar danos às instalações, aos seres humanos e ao meio ambiente. O objetivo neste trabalho é apresentar um método que aumente a confiança e a segurança em programas de Controladores Lógicos Programáveis (CLP) para SIS. Para tanto, geração e execução automática de casos de teste, que contemplam os estados das saídas e propriedades temporizadas do sistema, são utilizadas para avaliar se o programa do SIS está em conformidade com sua especificação. Para este propósito, faremos uso de uma arquitetura com suporte à verificação dinâmica de programas de CLP para SIS, arquitetura *Hardware in the loop* (HIL). Além disso, o formalismo de redes de autômatos temporizados em conjunto com diagramas de decisão binária ordenados e reduzidos (ROBDD) são utilizados para gerar casos de teste não redundantes. Três estudos de caso são utilizados para avaliar o método proposto e os resultados confirmam a sua eficiência.

Abstract

Safety Instrumented Systems (SIS) are designed to guarantee the industrial system safety preventing undesirable situations when executing procedures performed automatically or by human being. In the context of SIS, ensuring reliable and safe operation is vital because hardware and software faults or human error can cause damage to the plants, humans and the environment. The goal of this work is to present a method to increase reliability and safety in Programmable Logic Controllers (PLC) programs for SIS. Automatic generation and execution of test cases, which include the the system outputs and timer properties states, are used to evaluate whether the SIS program is in conformance to its specification. The Hardware in the loop (HIL) architecture, which supports the dynamic verification of PLC programs for SIS, is used here. Furthermore, timed automata networks together with reduced ordered binary decision diagrams (ROBDD) are used to generate non-redundant test cases. Three case studies are used to evaluate the proposed method and the results attest its efficiency.

Agradecimentos

Agradeço a Deus, em primeiro lugar, razão de toda a minha história, pelo dom da vida e pela oportunidade de realizar este trabalho, dando-me forças nos momentos difíceis. Obrigada senhor!

Agradeço a meus pais José Florêncio e Josélia, e as minhas irmãs Kéllen e Kivânia, por todo o incentivo, apoio, confiança e exemplo dado por toda vida.

A meu filho Arthur Victor que soube me dar inspiração e apoio nas horas difíceis. E a meu marido Karcius por todo carinho, incentivo, dedicação e paciência.

A minha segunda família Juarez, Socorro e Kelly, por toda confiança e incentivo para a conclusão deste trabalho.

Aos meus parentes que sempre tiveram uma palavra de apoio e ternura.

Aos professores Angelo Perkusich e Leandro Dias da Silva pela orientação necessária ao desenvolvimento desta tese e pela contribuição em minha formação acadêmica e profissional.

Aos professores Kyller Costa Gorgônio, Hyggo Oliveira de Almeida e Péricles Rezende Barros pelas contribuições na realização deste trabalho.

Ao pessoal do Laboratório Embedded pela amizade.

A meu avô José e, in memorian, a meus avós Maria, Irene e João pelo exemplo de vida.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro.

Aos meus amigos que compreenderam minha ausência.

Em fim, a todos que contribuíram de forma direta ou indireta para a realização deste trabalho.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Sistemas Instrumentados de Segurança (SIS) são desenvolvidos para garantir a segurança operacional de sistemas industriais prevenindo a ocorrência de situações indesejadas quando da execução de procedimentos realizados automaticamente ou sob a interferência de operadores humanos (Kapasi *et al.* , 2011; Zhanyuan *et al.* , 2010). No caso de falhas, a atuação do SIS deve garantir que a planta, ou parte dela, manter-se-á em um estado operacional seguro (Fang *et al.* , 2008). As operações de controle de SIS são realizadas por Controladores Lógicos Programáveis (CLP) de segurança (Ljungkrantz *et al.* , 2012), uma classe de CLP (Bryan & Bryan, 1997) especialmente projetada para atuar em áreas de segurança.

No contexto de SIS é fundamental garantir a confiança e a segurança no funcionamento pois, defeitos no hardware, no software ou erros humanos podem ocasionar danos às instalações, aos seres humanos e ao meio ambiente (Goble & Cheddie, 2005; Gruhn & Cheddie, 2006). Desta forma, torna-se necessária a utilização de técnicas de verificação durante o processo de desenvolvimento de um programa de CLP para SIS.

Existem diversas técnicas para verificar programas de CLP para SIS, desde técnicas estáticas a dinâmicas (Patil *et al.* , 2011; Pressman, 2006). Técnicas de verificação estática são aquelas que não requerem a execução ou mesmo a existência de um programa para serem conduzidas (Pressman, 2006). Verificação de modelos é um exemplo de técnica de verificação estática (Katoen, 1999; Frey & Litz, 2000; Tschannen *et al.* , 2011). Técnicas de verificação dinâmica são aquelas que se baseiam na execução de um programa com a finalidade de detectar erros (Sommerville, 2007). Testes e avaliação de asserções no código são exemplos de técnicas de verificação dinâmica (Ge *et al.* , 2011; Sugai *et al.* , 2008).

De acordo com a revisão bibliográfica realizada podemos observar que a técnica verificação de modelos é aplicada no contexto de programas para CLP. Bender et al. (2008) convertem programas Ladder em redes de Petri e realizam a verificação de propriedades na ferramenta Tina. Canet et al. (2000) mostram como transformar um programa escrito em *IL* em uma estrutura de Kripke e, como a verificação de propriedades, segundo a lógica temporal *LTL* pode ser analisada. Gergely et al. (2010) combinam o formalismo de redes de Petri e a verificação de modelos para avaliar programas para CLPs. Mokadem et al. (2010) utilizam o formalismo de redes autômatos temporizados para modelar programas para CLP e a verificação dos modelos é realizada na ferramenta Uppaal. Porém, apesar dos trabalhos analisados utilizarem esta técnica para verificar programas, o uso da mesma é inadequado para sistemas complexos devido ao problema conhecido como explosão do espaço de estados (Kalita & Khargonekar, 2002; Pu & Zhang, 2007; Schneider & Brandt, 2008).

Outra técnica utilizada para verificar programas de CLP para SIS é a realização de testes. Teste é uma técnica utilizada para avaliar a qualidade, a segurança e a confiança das aplicações e é normalmente usado na prática como uma atividade do processo de verificação e validação. Com o uso de testes é possível validar tanto requisitos funcionais quanto não funcionais de um sistema de software (Sommerville, 2007). No contexto de programas para CLPs e sistemas de tempo real (Cooling, 2003) dois tipos de técnicas são propostas para realização de testes: testes baseados em modelos (do inglês, *Model-Based testing* - MBT) (Utting & Legeard, 2006) e testes no código fonte.

Na literatura, o que difere os trabalhos que utilizam MBT como técnica de verificação é a forma como as métricas de cobertura do modelo são estabelecidas. Em (Larsen *et al.* , 2005; Oliveira *et al.* , 2010a; Silva *et al.* , 2008) a métrica utilizada é a escolha das variáveis de entrada de forma aleatória. Em (Li *et al.* , 2008) é utilizada a métrica conjuntos de ações e estados. O problema com o uso destas duas métricas é que nem sempre é possível detectar erros na implementação com o conjunto de casos de teste gerados. Nos trabalhos que abordam o uso das métricas fluxo de eventos (Timo & Rollet, 2010) e cobertura de falhas (En-Nouaary *et al.* , 2002) não são gerados casos de teste para avaliar o comportamento de elementos temporizados. Para trabalhos que apresentam as métricas cobertura de transições (Hessel & Pettersson, 2004; Jee *et al.* , 2006) e heurística do fator determinante (Peixoto *et al.* , 2010), os casos de testes gerados apresentam um considerável grau de redundância entre si e não

adicionam valor ao conjunto de testes, pois não detectam novos erros e não garantem uma melhor cobertura, seja do modelo, seja dos requisitos em análise.

Com a técnica de teste no código fonte a verificação de propriedades específicas é realizada no próprio código fonte do sistema sob teste. Nesta técnica o hardware dedicado é inserido no processo de teste, através do uso da arquitetura HIL (do inglês, *Hardware-In-the-Loop*) (Bacic, 2005; Crăciun *et al.* , 2010; Michalek *et al.* , 2005; Rankin & Jiang, 2011), e o comportamento do sistema pode ser testado em tempo real.

O restante deste capítulo está organizado da seguinte forma. Na Seção 1.1 são apresentados alguns trabalhos na área de verificação de programas baseados na arquitetura HIL. Na Seção 1.2 é discutida a contribuição deste trabalho para a área de Automação Industrial e Engenharia de Software. Na Seção 1.3 é abordado o problema a ser resolvido neste trabalho e, na Seção 1.4 é apresentada a solução para o mesmo. Na Seção 1.5 são apresentados o objetivo geral e os objetivos específicos deste trabalho. Por fim, na seção 1.6 é apresentada a estrutura deste documento.

1.1 Verificação de Programas para CLP no contexto HIL

Como nesta tese desejamos aplicar técnicas de verificação a programas para CLP em sua plataforma alvo, buscamos na literatura a forma adequada de definir uma arquitetura HIL para um sistema de tempo real. Jacobs el al. (2005) propõem uma arquitetura HIL para validar sistemas que são controlados por CLPs. Os autores discutem os passos que devem ser realizados para aplicar a técnica HIL na validação de programas para CLPs porém, não apresentam procedimentos para gerar casos de teste, de forma automática, a partir do modelo que representa a especificação do sistema.

Nos trabalhos de Iacob et al. (2011a; 2011b) é apresentada uma arquitetura HIL para um sistema de controle de uma caldeira. A idéia com o uso de HIL é dividir o sistema em duas partes, simulação da caldeira (planta do sistema) e sistema sob teste (SUT). O modelo de simulação é desenvolvido no *LabView* (ambiente de programação gráfico) em um PC e a implementação dos algoritmos de controle em tempo real é executada em um CLP. A troca de dados entre o PC e o CLP é realizada pelo servidor OPC. Em Iacob *et al.* (2011b) é apresentada a sincronização entre o controlador de tempo real (CLP) e o sistema que não

opera em tempo real (PC), uma das principais contribuições deste trabalho. Com os trabalhos de Iacob et al. foi possível estabelecer uma fronteira entre modelos simulados (planta do sistema) e implementação real do sistema (SUT). Além disso, foi possível entender a comunicação e transmissão de dados entre um sistema de tempo real (CLP) e um sistema que não opera em tempo real (PC). Porém, nenhuma técnica para validar a SUT foi apresentada nestes trabalhos.

Rankin & Jiang (2011) propõem uma plataforma para simulação em HIL para avaliar o desempenho funcional de sistemas eletrônicos programáveis para aplicações de controle de segurança em plantas nucleares. A principal contribuição deste trabalho consiste na sincronização entre a planta simulada e o controlador real (CLP) e não na aplicação de técnicas para avaliar programas para CLP.

Machado et al. (2013) propõem uma técnica para avaliar programas de CLP utilizando a abordagem HIL em conjunto com a verificação de modelos. Os autores modelam a planta segundo o formalismo de redes de autômatos temporizados descrito na ferramenta Uppaal (Larsen *et al.* , 2004). Após realizada a verificação de modelos, este modelo é transformado em um programa de CLP. Ao utilizar a técnica verificação de modelos, os problemas já mencionados anteriormente no início do Capítulo também se aplicam a este trabalho e ao trabalho de Bohn et al. (2002).

Uma vez analisadas as arquiteturas HIL propostas, torna-se necessário definir qual tipo de teste será utilizado para avaliar programas de CLP para SIS. O foco neste trabalho é nas técnicas de teste funcional ou de caixa preta (Beizer, 1995; Young & Pezze, 2005), pois o CLP é tratado como uma caixa preta e temos acesso apenas às entradas e saídas do sistema sob teste. Além disso, a especificação do sistema é dada por uma representação abstrata na forma de diagramas ISA 5.2 (ISA, 1992).

Teste funcional é uma técnica de verificação dinâmica bastante utilizada para avaliar programas de CLP (Oliveira *et al.* , 2010a; Chaaban *et al.* , 2011b). Realizar teste de caixa preta completo num sistema requer a aplicação de todas as possíveis combinações de valores das entradas, o que torna impraticável o teste completo do sistema (Schiffmann & Steinbach, 2012). Desta forma, critérios para reduzir a quantidade de casos de teste gerados são empregados na elaboração dos casos de teste. Alguns exemplos destes critérios são (Schiffmann & Steinbach, 2012; Nidhra & Dondeti, 2012): grafo de causa-efeito; tabela de decisão; análise

de valor limite; particionamento em classes de equivalência e diagrama de decisão binária (BDD).

Grafo de causa-efeito pode ser confuso para especificações de software complexos, o que dificulta a automatização do processo de teste. Tabelas de decisão são formuladas em linguagem natural o que representa uma vantagem, uma vez que casos de teste podem ser entendidos mais facilmente. Porém, é necessário decidir quais condições são relevantes para gerar os casos de teste, pois para sistemas com muitas condições a tabela pode crescer exponencialmente, dificultando sua criação e preenchimento.

Chaaban et al. (2009; 2011a; 2011b) apresentam os procedimentos necessários para gerar casos de teste em ambiente *Matlab/Simulink*, em que a verificação do programa para CLP é realizada quando o mesmo está sendo executado em um CLP. A idéia consiste na atribuição de valores para as variáveis de entrada. Para variáveis de entrada que não são booleanas é utilizada a abordagem de análise de valor limite. Para variáveis booleanas, todas as combinações de entrada são construídas e depois estas são combinadas com os valores das entradas não booleanas, de modo que no final obtenha-se uma matriz com todas as sequências de valores. O problema com o uso deste método é que para variáveis de entrada booleanas é realizado o teste exaustivo, que na prática é inviável. Além disso, na prática, a abordagem análise do valor limite é utilizada em conjunto com classes de equivalência de forma que variáveis booleanas e lógicas possam ser avaliadas (Nidhra & Dondeti, 2012).

Oliveira et al. (2013), num estágio anterior deste trabalho, utilizaram classes de equivalência para gerar casos de teste a partir de diagramas ISA 5.2, especificação do sistema. Os casos de teste gerados são executados em um programa de CLP para SIS em sua plataforma alvo e a conformidade entre especificação e implementação do sistema é avaliada. Os autores apresentaram um algoritmo no qual é possível avaliar os estados das saídas e propriedades temporizadas do sistema de forma que comportamentos redundantes da implementação não sejam avaliados. Contudo, o mecanismo para gerar os casos de teste não é eficiente para aplicações com um grande número de variáveis de entrada pois, no algoritmo, primeiro todas as combinações de valores para as variáveis de entrada são geradas e depois algumas destas combinações são selecionadas com o uso de classes de equivalência, para gerar os casos de teste. Desta forma, torna-se necessária a aplicação de uma técnica para minimizar a quantidade de combinações de variáveis de entrada que são geradas. Neste trabalho, não

é realizada uma análise para verificar se os estados das saídas do sistema e propriedades temporizadas podem realmente ser verificadas para qualquer programa de CLP para SIS. Além disso, particionamento em classes de equivalência não deve ser aplicado a variáveis do tipo booleana, pois o número de classes de equivalência gerado é exponencial ao número de variáveis booleanas utilizadas para definir as classes (Arnicane, 2009).

Angelov et al. (2008) apresentaram um método de projeto composicional para aplicações de sistemas embarcados, utilizando componentes de software reconfiguráveis, tais como máquina de estados e blocos de função. Para tanto, os autores modelaram blocos de funções utilizando a abordagem de diagramas de decisão binária (BDD), porém não avaliaram propriedades temporizadas em conjunto com a lógica completa do sistema.

Miremadi et al. (2011; 2012) propõem um método para minimizar o tamanho do supervisor de aplicações industriais. O objetivo é gerar fórmulas proposicionais, denominadas de guardas, as quais representam o comportamento do supervisor. Para lidar com sistemas complexos de forma eficiente, os modelos são simbolicamente representados por BDDs ordenados e reduzidos (ROBDDs), e todas as computações são executadas sobre estas estruturas de dados. Utilizando os ROBDDs gerados e aplicando algumas heurísticas a guarda é gerada. De forma geral, com a aplicação destas técnicas o tamanho da guarda é reduzido. Finalmente, as guardas geradas podem ser adicionadas aos modelos originais através da criação de *Extended Finite Automata* (EFA) (Skoldstam *et al.* , 2007). Na maioria das vezes, este passo facilita a implementação do supervisor devido o tamanho reduzido do modelo. Contudo, aplicar este método no contexto de SIS em sua plataforma alvo requer que os modelos de relógios sejam contínuos, pois, muitas arquiteturas HIL são baseadas em simulação de tempo real contínuo. Além disso, neste trabalho o objetivo é gerar um modelo para o supervisor e não testar o código do supervisor. Da mesma maneira, no trabalho de Mari et al. (2012) o objetivo é gerar a implementação do sistema a partir de OBDDs, um BDD ordenado, e não testar o código do sistema.

Schiffmann et al. (2012) demonstraram que BDDs e seus variantes (OBDD, ROBDD), podem ser utilizados para gerar casos de teste de caixa preta com alto nível de qualidade pois, com BDDs, é possível ter uma representação compacta e eficiente de estados e conjunto de estados que representam o modelo do sistema, resultando assim num menor espaço ocupado quando comparado com outras técnicas, tais como árvore de decisão e tabela verdade.

1.2 Relevância

No contexto de SIS é fundamental garantir a confiança na execução de processos automatizados e a segurança tanto de equipamentos e instalações como de funcionários (Goble & Cheddie, 2005; Gruhn & Cheddie, 2006). Desta forma, torna-se necessária a utilização de técnicas de verificação durante o processo de desenvolvimento de um programa de CLP para SIS.

Para assegurar o correto funcionamento do SIS é fundamental que propriedades temporizadas sejam avaliadas em conjunto com a lógica completa do sistema. De acordo com a revisão bibliográfica realizada para o desenvolvimento deste trabalho, avaliar propriedades temporizadas e os estados das saídas do sistema de forma a reduzir comportamentos redundantes da implementação são características não consideradas na avaliação de programas de CLP para SIS, o que reforça o caráter de originalidade e a contribuição científica desta tese. Além disso, a arquitetura HIL desenvolvida bem como o processo de geração e execução dos casos de teste, no contexto deste trabalho, tem como objetivo demonstrar que o método proposto é viável e praticável, uma vez que o mesmo pode ser utilizado de forma paralela ao processo de desenvolvimento do programa do CLP para SIS.

1.3 Problema Abordado

O problema de interesse neste trabalho é: *Como aumentar a confiança e a segurança na execução de processos realizados, em tempo real, por programas de controladores lógicos programáveis no contexto de sistemas instrumentados de segurança, de forma que propriedades temporizadas e os estados das saídas do sistema possam ser avaliados e que comportamentos redundantes da implementação do sistema não sejam avaliados?*

1.4 Hipóteses

Para verificar programas de CLP para SIS, considerando o problema abordado neste trabalho, é necessária a aplicação de técnicas para avaliar comportamentos não redundantes da lógica do sistema e de elementos temporizados. Desta forma, para o problema abordado neste trabalho são consideradas as seguintes hipóteses:

- *H1: a arquitetura HIL pode ser adaptada no contexto de SIS de forma que a conformidade entre a especificação e a implementação do sistema possa ser avaliada;*
- *H2: com o uso de ROBDDs é possível testar os estados das saídas e propriedades temporizadas do sistema;*
- *H3: com o uso de ROBDDs é possível reduzir o número de casos de teste redundantes.*

1.5 Objetivos Gerais e Específicos

O objetivo geral neste trabalho é desenvolver um método para verificar se um programa de CLP para SIS reflete o comportamento definido em sua especificação, de forma que propriedades temporizadas e os estados das saídas do sistema possam ser avaliados e que comportamentos redundantes da implementação do sistema não sejam avaliados. No sentido de atingir o objetivo principal, os seguintes objetivos específicos foram definidos:

- propor uma arquitetura HIL para testar programas de CLP para SIS;
- desenvolver um modelo, a partir da especificação do sistema, para gerar os casos de teste;
- desenvolver um método e uma ferramenta para gerar e executar automaticamente casos de testes, não redundantes, que contemplem propriedades temporizadas e os estados das saídas do sistema.

1.6 Estrutura do Documento

Este documento está estruturado da seguinte forma: no Capítulo 2, Fundamentação Teórica, é fornecido o embasamento teórico necessário para o entendimento deste trabalho. Desta forma, os principais conceitos relativos a Controladores Lógicos Programáveis (CLPs), Padrão IEC 61131-3, Protocolo OPC, Diagramas de Lógica Binária ISA 5.2, Rede de Autômatos Temporizados, Diagramas de Decisão Binária e Técnica HIL são apresentados. No Capítulo 3 é apresentado o método introduzido nesta tese para aumentar a confiança e a segurança de programas de CLP para SIS. Neste capítulo são apresentados os modelos, os algoritmos e

a geração de autômatos temporizados a partir de Diagramas de Lógica Binária ISA 5.2 e os fundamentos teóricos utilizados para gerar e executar os casos de teste. Neste capítulo também é realizada uma análise do método proposto. Esta análise tem o objetivo de validar as hipóteses propostas nesta tese. No Capítulo 4, Avaliação do Método Proposto, três estudos de caso são apresentados. Por fim, no Capítulo 5 são apresentadas as considerações finais e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

O objetivo neste capítulo é fornecer um embasamento teórico necessário para o entendimento deste trabalho. São apresentados os principais conceitos relacionados a controladores lógicos programáveis, padrão IEC 61131-3, protocolo OPC, diagramas de lógica binária ISA 5.2, redes de autômatos temporizados, diagramas de decisão binária e a técnica HIL.

2.1 Controladores Lógicos Programáveis

O Controlador Lógico Programável (CLP) surgiu no final da década de 1960, na empresa General Motors, com o intuito de substituir os relés, pois, a cada nova mudança na linha de montagem desta empresa, a lógica de controle dos painéis tinha que ser reprogramada, fazendo com que tal tarefa fosse onerosa (Parr, 2003). CLPs evoluíram passando a ser mais versáteis e de fácil utilização tornando-se assim um dos equipamentos mais utilizados na automação industrial. Segundo a ABNT (Associação Brasileira de Normas Técnicas), um CLP é um equipamento eletrônico digital com hardware e software compatíveis com aplicações industriais.

As principais vantagens da utilização dos CLPs, quando comparados a outros dispositivos de controle industrial, são (John & Tiegelkamp, 2001; Parr, 2003): menor espaço ocupado; menor potência elétrica requerida; fácil programação e manutenção; maior flexibilidade e apresentam interface de comunicação com outros CLPs e computadores de controle.

Na Figura 2.1 um esquema de um CLP é apresentado. Considerando este contexto, um CLP é dividido em três partes: entradas e saídas, que podem ser digitais ou analógicas e

unidade central de processamento (CPU), local onde são realizados o processamento do programa do usuário e a atualização de dados na memória.

Figura 2.1: Esquema básico de um CLP (Fonte: Bryan & Bryan, 1997).

O funcionamento de um CLP é sequencial, no qual ciclos de varredura são realizados em etapas. Cada uma destas etapas é exclusiva, ou seja, quando uma é executada as demais ficam inativas. O tempo total para executar cada ciclo é denominado tempo de varredura.

Na Figura 2.2 são apresentadas as etapas do ciclo de varredura de um CLP (Bryan & Bryan, 1997). Quando o CLP é inicializado uma série de pré-operações são realizadas, tais como: inativação de todas as saídas e verificação do funcionamento do CLP, da memória, da configuração interna e da existência de um programa de usuário. Após isso, o ciclo de varredura é inicializado. Seu funcionamento ocorre da seguinte forma: primeiramente, as entradas do sistema são lidas e os seus valores armazenados na memória. Depois o programa é executado e, na última etapa, os valores das saídas são liberados com base em seus valores armazenados na memória.

Figura 2.2: Etapas do ciclo de varredura de um CLP.

2.2 Padrão IEC 61131-3

Em 1979, a Comissão Eletrotécnica Internacional *(International Electrotechnical Commission - IEC)*, com o objetivo de definir um padrão específico para CLPs, criou o padrão IEC 1131 (Verwer, 1996). O desenvolvimento deste padrão, hoje denominado de IEC 61131, tornou-se necessário pelo fato de, cada vez mais, haver um aumento do nível de complexidade dos sistemas de controle e da incompatibilidade entre controladores, fazendo com que o desenvolvimento e até mesmo a manutenção dos sistemas seja uma tarefa onerosa. Para minimizar a ocorrência destes fatos, a IEC designou cinco frentes de trabalho para desenvolver diferentes partes do padrão para CLPs. A primeira parte do padrão, conceitos e definições de terminologias básicas, foi publicada em 1992. A parte 3, referente ao conjunto de linguagens de programação e foco deste trabalho, foi publicada em 1993 (International Electrotechnical Commission, 1993).

Com o uso do padrão IEC 61131-3 é possível ter (Oliveira *et al.* , 2010b; Verwer, 1996):

- sintaxe e semântica bem definidas de cinco linguagens de programação, mencionadas posteriormente, bem como o funcionamento delas em controladores industriais;
- a criação de diferentes componentes de um programa durante qualquer fase do ciclo de desenvolvimento de um programa para CLP;
- métodos modernos de engenharia de software: bibliotecas de funções podem ser desenvolvidas, incentivando a reutilização de código e a geração de documentação dos programas. Além disso, o programa pode ser decomposto em partes menores (componentes) que podem ser gerenciadas de forma independente;
- a utilização do conceito de multitarefa, ou seja, cada tarefa (atualização de entradas e saídas (E/S), comunicação, funções internas, dentre outras) pode ser composta de múltiplos programas, cada um deles com múltiplas rotinas e funções. Esta característica é muito comum em sistemas de tempo real;
- um diversificado conjunto de tipos de variáveis e estruturas que possibilitam manipular sinais analógicos e digitais, datas, texto e números.

De acordo com o padrão IEC 61131-3, programas para CLPs podem ser escritos em uma das cinco linguagens descritas a seguir (John & Tiegelkamp, 2001; Otto & Hellmann, 2009):

- *Instruction List (IL)* é uma linguagem textual que se assemelha à linguagem *assembler*;
- *Ladder Diagram (Ladder)* é baseada em uma representação gráfica de *Relay Ladder Logic (RLL)*;
- *Function Block Diagram (FBD)* é utilizada para expressar o comportamento de um controlador como um conjunto de blocos gráficos interligados;
- *Structured Text (ST)* é uma linguagem de alto nível que permite programação estruturada e se assemelha à linguagem *Pascal*;
- *Sequential Function Chart (SFC)* é usada para modelar lógicas de controle baseadas na sequência temporal de eventos de processo.

Neste trabalho serão apresentadas as linguagens FBD e Ladder por serem as mais utilizadas no âmbito industrial. Nas Subseções 2.2.1 e 2.2.2 são apresentados os elementos e o princípio de funcionamento destas linguagens.

2.2.1 Diagramas de Blocos Funcionais (FBD)

FBD é uma linguagem gráfica na qual blocos funcionais são utilizados para representar um conjunto de ações a serem realizadas (Bryan & Bryan, 1997). Estes blocos, que podem ser escolhidos dentro de um conjunto pré-definido pelo fabricante do CLP ou podem ser definidos pelo usuário, são conectados em conjunto de forma semelhante a um diagrama de circuito.

Na Figura 2.3 é apresentado o esquema geral de um bloco funcional. Um bloco funcional é composto por um conjunto de variáveis de entrada e saída, a função a ser computada e um conjunto de variáveis internas, necessárias para o processamento da função.

Após definida a estrutura geral de um bloco funcional, alguns blocos funcionais genéricos utilizados pela maioria dos fabricantes de CLPs são definidos. Estas informações estão presentes na Tabela 2.1 (John & Tiegelkamp, 2001).

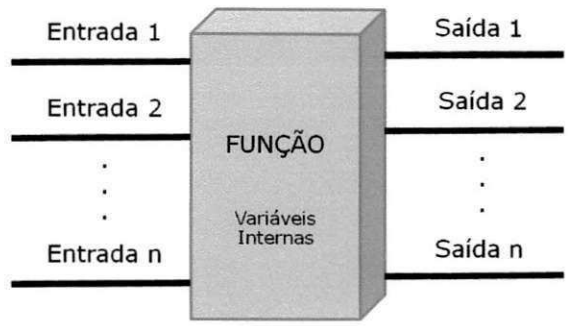


Figura 2.3: Esquema geral de um bloco funcional.

Em FBD, o fluxo dos dados ocorre da esquerda para a direita, das saídas de um bloco funcional para as entradas de outro bloco funcional. Os blocos funcionais são conectados por linhas horizontais, linhas verticais e *jumpers*, que são utilizados para modificar a sequência de execução do fluxo de dados. A negação dos sinais é feita de forma semelhante aos diagramas lógicos, pela presença de um "o" ou pelo uso do bloco NOT.

Na Figura 2.4 é apresentado um exemplo de programa FBD adaptado de (Parr, 2003). Para o exemplo, itens, ao longo de uma esteira, são detectados por uma fotocélula e contados. Quando um lote estiver concluído, total de 10 itens, a esteira é parada e uma luz é acesa informando ao operador que o lote está completo e pode ser removido. Após a remoção do lote, o botão de reiniciar é acionado e a contagem de itens pode ser reiniciada.

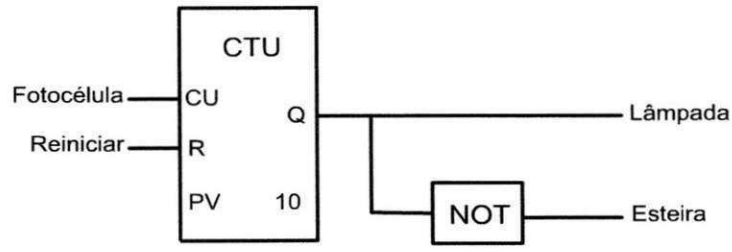
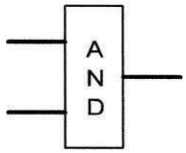
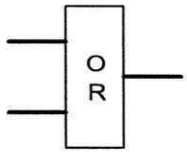
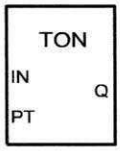
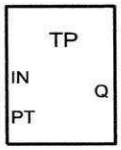
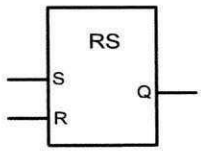
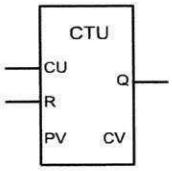


Figura 2.4: Exemplo de um programa FBD com contador (Fonte: Parr, 2003).

Tabela 2.1: Blocos funcionais definidos pela maioria dos fabricantes de CLPs.

Símbolo	Descrição
	Representa uma função booleana AND.
	Representa uma função booleana OR.
	Temporizador TON ou TMR: representa um elemento temporizado. A saída é ativada (Q = 1) depois de um período de tempo (tempo = PT) quando existe lógica 1 na entrada (IN = 1).
	Temporizador TOF: representa um elemento temporizado. A saída é ativada (Q = 1) quando existe lógica 1 na entrada (IN = 1). Quando a lógica na entrada passar a ser 0 (IN = 0) a saída será desativada (Q = 0) após um período de tempo (tempo = PT).
	Temporizador TP: representa um elemento temporizado. A saída é ativada (Q = 1) durante um período de tempo (tempo = PT) quando existe lógica 1 na entrada (IN = 1). Mesmo que exista lógica 0 na entrada (IN = 0) a saída continua ativada (Q = 1) durante o tempo pré-definido (tempo = PT).
	Flip-flop SR. A entrada set (S) é a dominante. Caso R esteja ativado (R = 1) então a saída Q é desativada (Q = 0).
	Contador incremental. Pulsos positivos em CU são contados até que o valor máximo PV seja atingido. Neste instante o valor de contagem (CV) é congelado e a saída Q é ativada (Q = 1). Quando a entrada R (Reset) for ativada (R = 1) o contador e a saída são restaurados. Quando CV = PV a saída será ativada (Q = 1).

2.2.2 Linguagem de Diagramas Ladder

A linguagem de diagramas Ladder, ou simplesmente Ladder, é uma das cinco linguagens definidas pelo padrão internacional IEC 61131-3 as quais são utilizadas para construir aplicações para CLPs. Ladder é uma linguagem gráfica que se assemelha muito aos circuitos de relés.

Ladder possui esse nome pelo fato da sua representação se parecer com uma escada, na qual duas barras verticais paralelas, uma esquerda e uma direita, que representam respectivamente o barramento energizado e barramento terra, são utilizadas para interligar a lógica de controle que forma os degraus (Bender *et al.* , 2008). Cada degrau é formado por uma lógica de controle que por sua vez é constituída de linhas e colunas, nas quais estão localizados os elementos da linguagem, cuja quantidade é definida pelo fabricante do CLP. Estes detalhes podem ser visualizados na Figura 2.5.

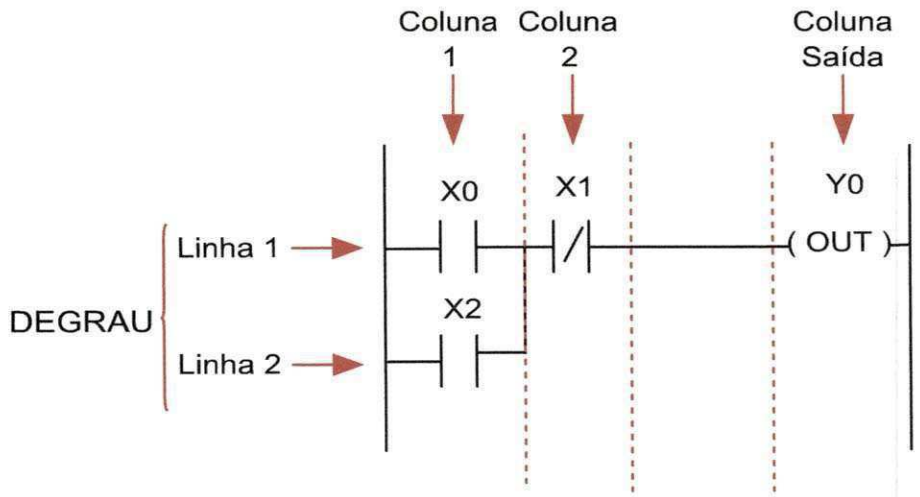


Figura 2.5: Esquema gráfico da linguagem Ladder.

Cada lógica de controle deve ser programada de forma que as instruções sejam energizadas no sentido da corrente elétrica entre as duas barras. A corrente flui da esquerda para a direita em cada linha e energiza sequencialmente cada coluna da linha que está percorrendo. Dizemos que um degrau de um programa Ladder está habilitado (saída energizada), segundo o estado atual de suas variáveis de entrada, quando existe um caminho que gera continuidade lógica entre as duas barras e que este degrau está desabilitado quando não há continuidade lógica entre as mesmas (Bryan & Bryan, 1997). Na Figura 2.6 é apresentado

um exemplo de um degrau de um programa Ladder com os possíveis caminhos que proveem continuidade lógica e energizam a saída do degrau.

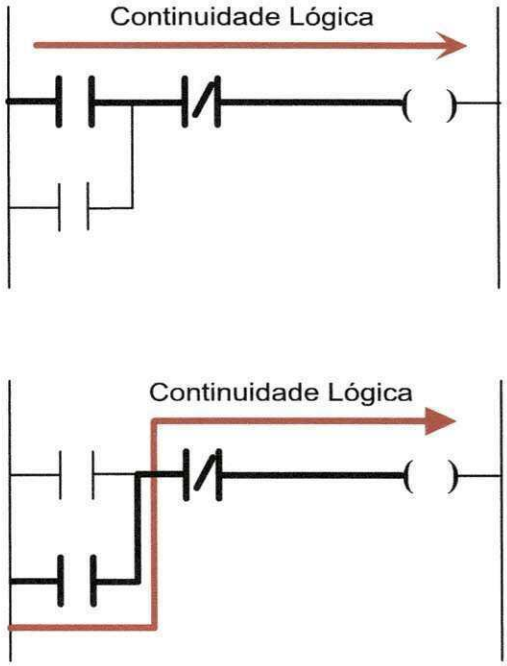
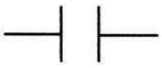
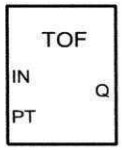
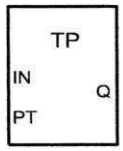


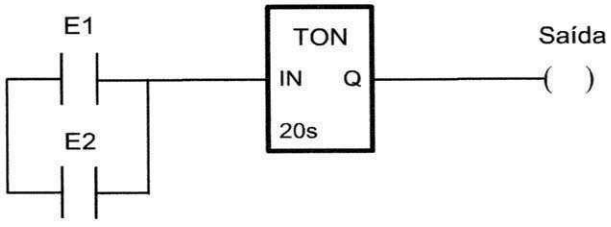
Figura 2.6: Caminhos que proveem continuidade lógica de um programa Ladder.

Nos parágrafos acima vimos como ocorre o princípio de funcionamento da linguagem Ladder, agora veremos, na Tabela 2.2, os principais elementos que compõem esta linguagem. Estes elementos podem ser combinados em série, formando uma operação booleana AND, em paralelo, formando uma operação booleana OR, ou uma composição destas combinações.

Na Figura 2.7(a), um exemplo de um programa Ladder é ilustrado. As variáveis *E1* e *E2* são chamadas de contatos normalmente abertos e a variável *Saída* é chamada de bobina. As variáveis *E1* e *E2* são conectadas em paralelo (operação Booleana OR). Como existe um temporizador no degrau então, o valor lógico de *Saída* depende da saída do temporizador TON. Portanto, se a operação lógica de entrada do temporizador é verdadeira, *E1 or E2 = 1*, então a operação de contagem do temporizador é iniciada; quando o tempo de 20s (valor de PT) for atigindo a variável *Saída* é ativada (valor lógico 1); caso a operação lógica de entrada seja falsa, *E1 or E2 = 0*, então a variável *Saída* é desativada (valor lógico 0). Na Figura 2.7(b) a tabela verdade para a variável *Saída* é ilustrada.

Tabela 2.2: Elementos da linguagem Ladder.

Símbolo	Descrição
Entrada 	Contato normalmente aberto: representa uma entrada da lógica de controle. No CLP é examinado o bit especifico correspondente a este símbolo e é retornado 0 se o bit é 0 e 1 se o bit é 1.
Entrada 	Contato normalmente fechado: representa uma entrada da lógica de controle. No CLP é examinado o bit especifico correspondente a este símbolo e é retornado 1 se o bit é 0 e 0 se o bit é 1.
Saída ( )	Bobina: representa uma saída. É um elemento atuador, o qual é acionado ou desligado de acordo com a lógica de controle.
	Temporizador TON ou TMR: representa um elemento temporizado. A saída é ativada ($Q = 1$) quando a entrada é verdadeira ($IN = 1$). Quando a entrada for falsa a saída será desativada após um período de tempo (PT).
	Temporizador TOF: representa um elemento temporizado. A saída é ativada (Q = 1) quando existe lógica 1 na entrada (IN = 1). Quando a lógica na entrada passar a ser 0 (IN = 0) a saída será desativada (Q = 0) após um período de tempo (tempo = PT).
	Temporizador TP: representa um elemento temporizado. A saída é ativada ($Q = 1$) durante um período de tempo (PT) quando a entrada é verdadeira. Mesmo que a entrada seja falsa ($IN = 0$) a saída continua ativada durante o tempo pré-definido (PT).



(a) Degrau

E1	E2	Saída
0	0	0
0	1	0 - antes 20 s 1 - depois 20s
1	0	0 - antes 20 s 1 - depois 20s
1	1	0 - antes 20 s 1 - depois 20s

(b) Tabela Verdade

Figura 2.7: Exemplo de um degrau de um programa Ladder com temporizador.

2.3 Protocolo OPC

Fazer com que dispositivos de chão de fábrica e programas aplicativos funcionem em conjunto é um dos grandes problemas do processo manufatureiro. Uma das principais razões para este problema é a não padronização de interfaces, fazendo com que a escolha de hardware e software seja bastante restringida devido à limitada conectividade entre tais componentes (Iwanitz & Lange, 2002).

Na ausência de qualquer padrão, fornecedores desenvolvem soluções proprietárias de hardware e de software fazendo com que o custo de integração e manutenção a longo prazo de sistemas seja alto. O desenvolvimento de *drivers* e interfaces padronizadas aumentou rapidamente por causa dos milhares de diferentes tipos de dispositivos de controle e pacotes de software que precisam se comunicar. Com esta proliferação de *drivers* surgem alguns problemas, tais como (OPC Foundation, 1998): inconsistências entre *drivers* de diferentes fornecedores; recursos de hardware que não são universalmente suportados e conflitos de acesso.

A solução para tais problemas consiste em ter um padrão que forneça tecnologia de software *plug-and-play* para controle de processos, de forma que a conexão e a comunicação de todos os dispositivos e *drivers* possam ocorrer livremente. Para atender a esta necessidade, o padrão OPC (*OLE - Object Linking and Embedding* (renomeado para *ActiveX*) - *for Process Control*) foi desenvolvido (Mahnke *et al.* , 2009).

O padrão OPC foi criado com a colaboração de líderes mundiais de automação e fornecedores de hardware e software em cooperação com a Microsoft (Iwanitz & Lange, 2002). A organização que gerencia este padrão é a *OPC Foundation*, criada em 1996. O objetivo desta fundação é desenvolver um padrão aberto, flexível e *plug-and-play* que permita que os usuários finais desfrutem de uma maior escolha de soluções, reduzindo os custos de desenvolvimento e manutenção de hardware e software dos fornecedores (OPC Foundation, 1998). Com o uso do padrão OPC o compartilhamento de informações e interação entre um sistema de automação com outros sistemas, em toda a sua planta ou fábrica, torna-se possível (OPC Foundation, 2006).

Baseado em tecnologias desenvolvidas pela Microsoft - *OLE*, *COM* (*Component Object Model*) e *DCOM* (*Distributed Component Object Model*) (Iwanitz & Lange, 2002) - OPC

é composto por um conjunto padrões de interfaces, propriedades e métodos para o uso em controle de processos e aplicações de automação de fábricas. As tecnologias *ActiveX e COM* são utilizadas para definir como os componentes individuais de software podem interagir e compartilhar dados. Apoiado pela tecnologia NT da Microsoft, com OPC é possível fornecer uma interface de comunicação comum para diversos dispositivos de controle de processo, independentemente do controlador de software ou dispositivos no processo.

OPC é implementado em pares cliente/servidor (Mahnke *et al.* , 2009). Normalmente, os produtos para monitoração de dados (IHMs; sistemas supervisórios, etc.) são clientes OPC. Já os produtos que são utilizados para fazer a comunicação direta com os dispositivos de campo são servidores OPC.

A seguir, algumas especificações OPC desenvolvidas pela fundação OPC são apresentadas. Uma especificação OPC é um documento que contém informações sobre como dois produtos OPC devem comunicar-se entre si. Este documento também inclui os parâmetros necessários e variáveis para cada requisição individual. O objetivo nestas especificações é facilitar o desenvolvimento tanto de servidores OPC como aplicações clientes. Em cada uma dessas especificações são definidos os tipos de dados que devem ser transferidos entre aplicações.

- *OPC Data Access* (OPC Foundation, 2003a): é atualmente a especificação OPC mais utilizada. No documento desta especificação é definido como dados são transferidos entre as aplicações em tempo real;
- *OPC Alarms and Events* (OPC Foundation, 2002): é utilizado para detectar eventos e identificar condições anormais em alguma parte do sistema;
- *OPC Historical Data Access* (OPC Foundation, 2003b): é utilizado para recuperar e analisar dados históricos para fins de análise, controle de estoque, etc. Estes dados são recuperados de um banco de dados ou de uma unidade terminal remota.

Com a publicação destas especificações OPC, um número maior de produtos voltados para a automação industrial podem ser desenvolvidos, pois estes se beneficiam das seguintes vantagens oferecidas por tal padrão (Iwanitz & Lange, 2002; OPC Foundation, 1998):

- padronização das interfaces de comunicação entre servidores e aplicações cliente, facilitando a integração e manutenção dos sistemas;

- redução do desenvolvimento de *drivers* proprietários;
- redução do tempo necessário para integração de sistemas e treinamento. No padrão é apresentada uma interface padronizada para todos os produtos;
- integração com sistemas *MES* (*Manufacturing Execution System*), *ERP* (*Enterprise Resource Planning*) e aplicações windows (Excel, etc);
- aumento da variedade de produtos disponíveis no mercado, o que deve, no futuro, levar a uma redução significativa de custos com o consequente aumento da qualidade dos mesmos;
- melhoria do desempenho e otimização da comunicação entre dispositivos industriais;
- interoperabilidade entre sistemas de diversos fabricantes.

Neste trabalho será apresentada apenas a especificação de aquisição de dados (*OPC Data Access Specification*) pois, esta foi utilizada para desenvolver a aplicação cliente OPC. Nesta aplicação é realizada a comunicação entre a ferramenta de testes desenvolvida e o servidor OPC.

2.3.1 Especificação OPC de aquisição de dados

A especificação de aquisição de dados (OPC DA), inicialmente chamada apenas de especificação OPC, é a mais antiga de todas as especificações OPC. Nela estão definidas as regras para a troca de dados em tempo real entre clientes e servidores OPC (Iwanitz & Lange, 2002). Cada cliente OPC pode conectar-se à diferentes servidores, os quais podem estar processando na mesma máquina ou remotamente em máquinas diferentes. De modo análogo, um servidor OPC pode esta conectado a diferentes clientes. De acordo com a especificação OPC DA, os servidores OPC são utilizados para realizar a comunicação com os dispositivos e disponibilizar essas informações para os clientes.

Na especificação OPC DA a arquitetura OPC é composta por uma hierarquia de três objetos (Iwanitz & Lange, 2002):

- servidor: é uma estrutura de armazenagem para grupos que, por sua vez, tem como função básica o armazenamento de itens. O servidor é utilizado para gerenciar a

conexão com o cliente e retornar dados ao cliente. Outra função do servidor é implementar uma estrutura de endereçamento capaz de associar itens com variáveis reais;

- grupo: são definidos pelos clientes. São utilizados para realizar o agrupamento lógico e gerenciamento de itens. Existem dois tipos de grupos, públicos, utilizados por vários clientes OPC, e privados utilizados por apenas um cliente OPC;

- item: representa o dado de campo propriamente dito. Cada item de dado é formado por três componentes básicos: o valor propriamente dito, do tipo VARIANT (com subtipos Float, Integer, String etc); rótulo de tempo (timestamp) que representa a informação do tempo em que o servidor recebeu o dado de um dispositivo; e dois bytes que representam a qualidade associada ao dado (ex: *good* - Dado válido; *bad* - perda do link de comunicação com o dispositivo de campo; e *uncertain* - no caso de existir o link e o dispositivo de campo estiver fora de comunicação).

O acesso aos dados, conforme definido na especificação OPC DA, pode ser realizado de três maneiras distintas (OPC Foundation, 2003a):

- leitura e escrita síncrona: são executadas imediatamente pelo servidor, a requisição de dados é feita pelo cliente e os recursos de sistema só são liberados quando os valores são retornados pelo servidor. É simples de implementar, mas pouco eficiente, ocupando muitos recursos de rede quando existem muitos dados a trafegar. Há dois tipos de acesso diferentes, ao cache normalmente mantido pelo servidor, ou diretamente ao dispositivo. Neste último o uso das operações síncronas pode comprometer seriamente o desempenho do sistema, pois cliente e servidor ficam bloqueados enquanto o dispositivo físico é acessado;

- leitura e escrita assíncronas: são mais eficientes para grandes quantidades de dados, pois o cliente é imediatamente liberado após fazer a requisição. Com o modo assíncrono é permitido que várias solicitações de dados sejam aceitas pelo servidor OPC, até mesmo de clientes OPC diferentes;

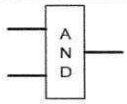
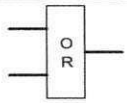
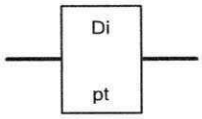
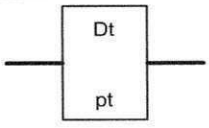
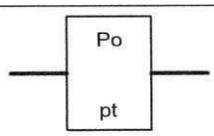
- atualização enviada pelo servidor: permite que mensagens atualizando um determinado conjunto de valores sejam requisitadas, de forma periódica ou por exceções, pelo cliente ao servidor.

2.4 Diagramas de Lógica Binária ISA 5.2

O padrão ISA 5.2 foi criado para prover um método de diagramação de lógica de entreligamento binário e sequenciamento de sistemas. Este padrão foi desenvolvido para facilitar a compreensão do funcionamento dos sistemas binários e melhorar a comunicação entre os técnicos, gerentes, projetistas e o pessoal responsável por operar e manter os sistemas (ISA, 1992). Neste padrão são simbolizadas as funções operacionais binárias de um sistema de maneira que pode ser aplicado a qualquer classe de hardware, seja ela eletrônica, elétrica, hidráulica, mecânica, manual ou óptica.

No padrão ISA 5.2 são fornecidos símbolos para representar operações de processos através de funções operacionais binárias. Alguns destes símbolos são apresentados na Tabela 2.3. A leitura de um diagrama ISA 5.2 é feita da esquerda para a direita e de cima para baixo e alterações no sentido do fluxo convencional devem ser explicitadas através de setas.

Tabela 2.3: Símbolos ISA 5.2.

Símbolo	Descrição
	Representa uma entrada em um diagrama ISA 5.2.
	Representa uma saída em um diagrama ISA 5.2.
	Representa uma operação booleana AND em um diagrama ISA 5.2.
	Representa uma operação booleana OR em um diagrama ISA 5.2.
	Temporizador DI (Delay Initiation of output): a saída é ativada depois de um período de tempo quando existe lógica 1 na entrada. Nas linguagens Ladder e FBD este temporizador é denominado de TON.
	Temporizador DT (Delay Termination of output): a saída é ativada quando existe lógica 1 na entrada. Quando a lógica na entrada passar a ser 0 a saída será desativada após um período de tempo. Nas linguagens Ladder e FBD este temporizador é denominado de TOF.
	Temporizador PO (Pulse Output): a saída é ativada durante um período de tempo quando existe lógica 1 na entrada. Mesmo que exista lógica 0 na entrada a saída continua ativada durante o tempo pré-definido. Nas linguagens Ladder e FBD este temporizador é denominado de TP.

O diagrama ISA 5.2 de um exemplo introduzido em (Parr, 2003) é ilustrado na Figura 2.8. No exemplo é relatado o funcionamento de um motor o qual é acionado e parado com o auxílio dos respectivos botões *Iniciar* e *Parar*. O sistema também é composto por um contato auxiliar, *AuxIniciar*, que é utilizado para energizar *Motor* (motor ativado). Se um erro for cometido, devido à ocorrência de uma sobrecarga, ou devido a uma parada de emergência ser pressionada, ou se houver uma falha no abastecimento, o sinal do contato auxiliar será perdido. Este contato só pode ser verificado depois de 5 segundos após *Motor* ter sido energizado. *Manual* representa o acionamento manual de um alarme caso algum problema aconteça.

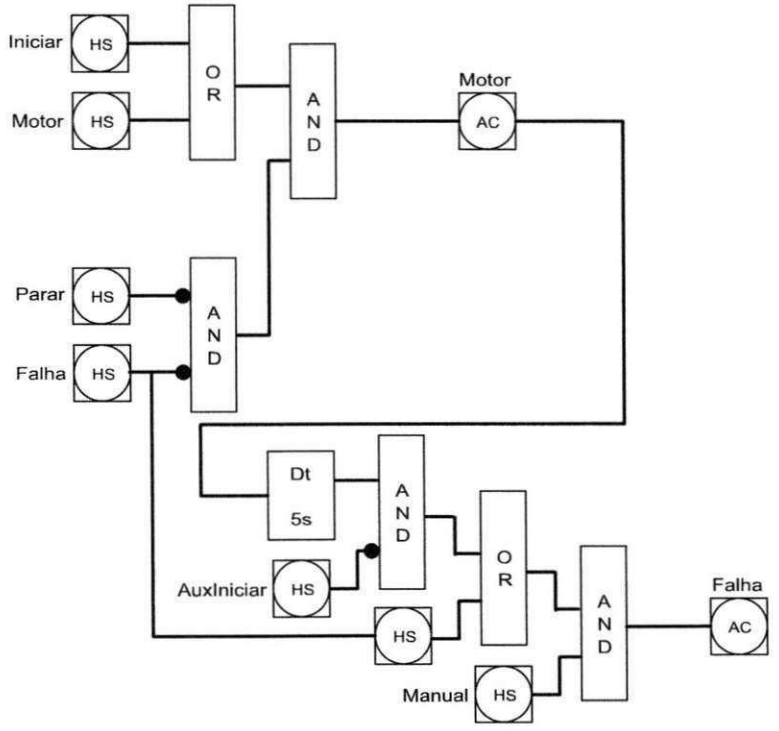


Figura 2.8: Diagrama ISA 5.2 para o funcionamento de um motor.

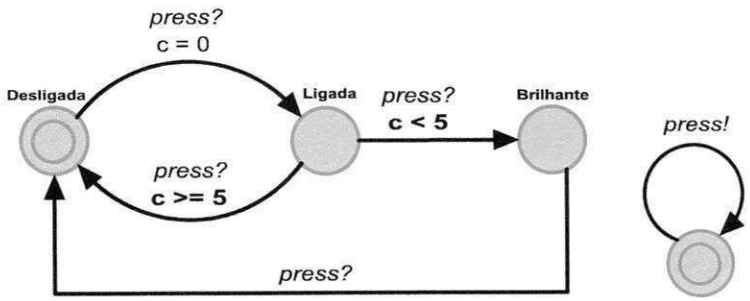
2.5 Rede de Autômatos Temporizados

Autômatos temporizados são autômatos de estados finitos com restrições de temporização associadas as suas arestas e localidades e são utilizados para modelar o comportamento de sistemas de tempo real (Alur & Dill, 1994; Bengtsson & Yi, 2004). As restrições são construídas a partir de variáveis de controle de tempo chamadas de relógios. Estes progridem de

forma síncrona e são declarados como valores reais, pois o tempo considerado é contínuo.

Neste trabalho focaremos apenas nos modelos de autômatos temporizados utilizados pela ferramenta Uppaal (Larsen *et al.* , 2004), pois esta foi utilizada para modelar as redes de autômatos temporizados geradas a partir de diagramas ISA 5.2.

Na Figura 2.9, uma pequena introdução da sintaxe e semântica de uma rede de autômatos temporizados na ferramenta Uppaal é apresentada. Para tanto, o funcionamento de uma lâmpada é modelado. O autômato que representa a lâmpada, ilustrado na Figura 2.9(a), é composto por três localidades: desligada, ligada, e brilhante. As expressões próximas aos arcos, ilustradas em negrito, representam guardas, e expressões do tipo $c = 0$ representam atualização de variáveis. Canais de sincronização são expressos em itálico. O funcionamento desta rede de autômatos ocorre da seguinte forma: se o usuário pressionar o interruptor, isto é, sincronizar com a *press?*, a lâmpada acende. Se o usuário pressionar o interruptor outra vez, a lâmpada apaga-se. No entanto, se o usuário for rápido e pressionar o interruptor duas vezes, a lâmpada acende e torna-se brilhante. O modelo do usuário é mostrado na Figura 2.9(b). O usuário pode pressionar o interruptor aleatoriamente e em qualquer momento ou nunca o pressionar. O relógio c da lâmpada é usado para detectar se o usuário é rápido ($c < 5$) ou lento ($c >= 5$).



(a) Lâmpada (b) Usuário

Figura 2.9: Exemplo simples de uma lâmpada.

A seguir serão apresentadas algumas definições formais da sintaxe e semântica de autômatos temporizados. As seguintes notações serão utilizadas: C é o conjunto de relógios e B(C) um conjunto de conjunções sobre condições simples da forma x $\bowtie$ c ou x-y $\bowtie$ c, de modo que x, y $\in$ C, c $\in$ N e $\bowtie \in \{<, \leq, =, \geq, > \}$.

Definição 1: Um autômato temporizado é um sêxtuplo (L, l_0, C, A, E, I), tal que:

- L é o conjunto de localidades;
- $l_0 \in L$ é a localidade inicial;
- C é o conjunto de relógios;
- A é o conjunto de ações;
- $E \subseteq L \times A \times B(C) \times 2^c \times L$ é o conjunto de arestas entre localidades com uma ação, uma guarda e um conjunto de relógios que serão restaurados;
- $I: L \rightarrow B(C)$ invariantes que são atribuídas às localidades (também pode ser definido o intervalo de tempo que o sistema pode permanecer em um determinado estado).

A valoração do relógio é uma função $u: C \rightarrow R_{\geq 0}$ do conjunto dos relógios para os reais não negativos. Façamos R^c ser o conjunto de todas as valorações dos relógios. Façamos $u_0(x) = 0$ para todo $x \in C$. Notações considerando guardas e invariantes como o conjunto de valoração do relógio são utilizadas como $u \in I(l)$ (ou seja, u satisfaz $I(l)$).

Definição 2: Seja (L, l_0, C, A, E, I) um autômato temporizado. A semântica é definida como um sistema de transição rotulada com $\langle S, s_0, \rightarrow \rangle$, onde $S \subseteq L \times R^c$ é o conjunto de estados, $s_0 = (l_0, u_0)$ é o estado inicial, e $\rightarrow \subseteq S \times \{ R_{\geq 0} \cup A \} \times S$ é a relação de transição tal que:

- $(l, u) \rightarrow^d (l, u + d)$ se $\forall\, d': 0 \leq d' \leq d \Rightarrow u + d' \in I(l)$, e
- $(l, u) \rightarrow^a (l', u')$ se existe $e = (l, a, g, r, l') \in E$ tal que $u \in g$, $u' = [r \mapsto 0]u$, e $u' \in I(l)$, onde para $d \in R_{\geq 0}$, $u + d$ mapeia cada relógio x em C para o valor $u(x) + d$, e $[r \mapsto 0]u$ denota a valoração do relógio que leva cada relógio de r até 0, ou seja, todos os relógios são restaurados.

Os autômatos temporizados são compostos frequentemente por uma rede de autômatos temporizados sobre um conjunto comum de relógios e ações, consistindo em n autômatos temporizados, $A_i = (L_i, l_i^0, C, A, E_i, I_i)$, em que $1 \leq i \leq n$. Um vetor de localidade é um $\bar{l} = (l_1, ..., l_n)$. As funções invariantes de cada autômato da rede são compostas numa função comum sobre os vetores de posição $I(\bar{l}) = \bigwedge_i I_i(l_i)$. Denota-se por $\bar{l}[l_i' / l_i]$ o vetor onde o

i-ésimo elemento l_i *de* $\bar{l}$ é substituído por l_i'. A seguir a semântica de uma rede de autômatos temporizados é definida.

Definição 3: Seja $A_i = (L_i, l_i^0, C, A, E_i, I_i)$ uma rede de n autômatos temporizados. E seja $\bar{l_0} = (l_1^0, ..., l_n^0)$ o vetor de localidade inicial. A semântica de uma rede de autômatos temporizados é definida como um sistema de transição $\langle S, s_0, \rightarrow \rangle$, tal que $S = (L_1, ..., L_n)$ x R^c é o conjunto de estados, $s_0 = (\bar{l_0}, u_0)$ é o estado inicial, e $\rightarrow \subseteq S \; x \; S$ é a relação de transição definida por:

- $(\bar{l}, u) \rightarrow (\bar{l}, u + d)$ se $\forall \, d' \colon 0 \leq d' \leq d \Rightarrow u + d' \in I(\bar{l})$, e
- $(\bar{l}, u) \rightarrow (\bar{l}[l_i' / l_i], u')$ se existe $l_i \rightarrow^{rgr} l_i'$ tal que $u \in g$, $u' = [r \mapsto 0]u$ e $u' \in I(\bar{l})$
- $(\bar{l}, u) \rightarrow (\bar{l}[l_j' / l_j \, , l_i' / l_i], u')$ se existe $l_i \rightarrow^{c?g_i r_i} l_i'$ *e* $l_j \rightarrow^{c!g_j r_j} l_j'$ tal que $u \in (g_i \bigwedge g_j)$, $u' = [r_i \bigcup r_j \mapsto 0]u$ e $u' \in I(\bar{l})$

Os autômatos temporizados em Uppaal são extensões dos autômatos temporizados com as seguintes propriedades adicionais:

- templates: são definidos com um conjunto de parâmetros que podem ser de diversos tipos (por exemplo, int, chan). Estes parâmetros são substituídos por um dado argumento na declaração de processos;
- constantes: são declaradas como *const nome valor*. As constantes por definição não podem ser modificadas e devem ser inteiras;
- arrays: são permitidos para os relógios, canais, constantes e variáveis inteiras. São definidos adicionando um tamanho ao nome da variável, por exemplo, *chan c[4]*;
- inicializadores: são usados para inicializar variáveis inteiras e arrays de variáveis inteiras. Por exemplo, *int i: = 2;* ou *int i[3]: = 1, 2, 3*;
- variáveis inteiras limitadas: são declaradas como *int[min,max] nome*, onde *min* e *max* são os limites inferior e superior, respectivamente. Guardas, invariantes e atribuições podem ser compostas por expressões que variam sobre variáveis inteiras limitadas. Os limites são avaliados na verificação, e a violação de um limite conduz a um estado inválido que é rejeitado. Se os limites forem omitidos, a escala usada é de *-32768* a *32768*;

- sincronização binária: são declarados como *chan c*. Uma aresta rotulada com *c!* é sincronizada com outra rotulada com *c?*. Um par de sincronizações é escolhido de forma não determinística se forem permitidas diversas combinações;

- canais de broadcast: são declarados como *broadcast chan c*. Numa sincronização um remetente *c!* pode ser sincronizado com um número arbitrário de receptores *c?*. Se não houver receptor, então a ação *c!* pode continuar a ser executada, isto é, a emissão da transmissão nunca será bloqueada;

- canais urgentes de sincronização: são declarados quando na declaração do canal a palavra chave *urgent* é utilizada. Os atrasos não devem ocorrer se a sincronização da transmissão num canal urgente é habilitada. Arestas onde canais urgentes são utilizados para a sincronização não podem ter restrições de tempo, isto é, não devem possuir relógios nas guardas;

- localidades urgentes: são semanticamente equivalentes a adicionar um relógio extra x, que é restaurado em todas as arestas de chegada, e tem uma invariante $x <= 0$ na posição. Em localidades urgentes não deve haver progressão de tempo;

- localidades committed: são ainda mais restritivas na execução do que as localidades urgentes. Um estado é *committed* se algumas das localidades no estado forem *committed*. Num estado *committed* não pode ocorrer atrasos e na próxima transição deve conter uma aresta com origem em uma das localidades *committed*.

As expressões em UPPAAL são construídas com o uso de variáveis de relógios e variáveis inteiras. As expressões são usadas com os seguintes rótulos:

- guarda: é uma expressão particular em que as seguintes condições devem ser satisfeitas: ser livre de efeitos colaterais e ser avaliada por uma expressão booleana. Somente os relógios, as variáveis inteiras, e as constantes são referenciados (ou arrays destes tipos). Os relógios e as diferenças de tempo são comparados somente com expressões inteiras. Guardas com relógios são essencialmente conjunções (as disjunções são permitidas sobre condições inteiras);

- sincronização: é qualquer expressão da forma *Expression!* ou *Expression?* ou é um rótulo vazio. A expressão deve ser livre de efeitos colaterais, avaliada por um canal e deve somente ser utilizada com variáveis inteiras, constantes e canais;

- assignment: é uma lista de expressões separadas por vírgulas com efeito colateral; as expressões devem somente se referir a relógios, variáveis inteiras e constantes. Apenas valores inteiros devem ser atribuídos aos relógios;

- invariante: é uma expressão livre de efeitos colaterais e apenas relógios, variáveis inteiras e constantes são referenciados. Uma invariante é uma conjunção de condições da forma $x < e$ ou $x \leq e$, tal que x é o relógio referido e e é avaliado como um inteiro.

2.6 Diagramas de Decisão Binária

Dado um conjunto de x variáveis booleanas $\mathcal{B}$, uma função booleana $f: \mathbb{B}^x \rightarrow \mathbb{B}$ ($\mathbb{B}$ é o conjunto de valores booleanos, isto é, 0 e 1) pode ser expressa por um grafo direcionado acíclico denominado de diagrama de decisão binária (BDD), no qual os nós intermediários representam as variáveis e os nós folha representam os valores da função, valores booleanos *0* ou *1* (Bryant, 1986; Minato, 1996). Cada nó intermediário x_i tem um filho esquerdo (assumindo $x_i = 1$) e um filho direito (assumindo $x_i = 0$). O número de nós intermediários representa o tamanho de um BDD.

Utilizando a decomposição de Shannon (Shannon, 1948), um BDD f pode ser expresso por:

$$f = (\neg\, b \wedge f[0/b]) \vee (b \wedge f[1/b]) \text{ para } b \in \mathcal{B},$$

onde *f[b'/b]* (denominada de cofator de f) é a função booleana obtida a partir de f, substituindo-se toda ocorrência da variável b pela constante b'. Desta forma, as expressões booleanas *f[0/b]* (cofator negativo) e *f[1/b]* (cofator positivo) referem-se, respectivamente, à atribuição de *0* e *1* para todas as ocorrências da variável booleana b.

Considere $X = \{x_1, x_2, ..., x_n\}$ como sendo um conjunto de variáveis booleanas e π: $X \rightarrow \{1, 2, ..., |X| = n\}$ como sendo o mapeamento bijetivo dos índices das variáveis. Dizemos que uma variável x_i tem uma ordem inferior a uma variável x_j, se x_i está mais perto da raiz do

grafo. Esta ordem é indicada por $\pi(x_i) < \pi(x_j)$. Se as variáveis do BDD ocorrem na mesma ordem em todos os caminhos, o BDD é chamado de BDD ordenado (OBDD) (Bryant, 1992). A forma como as variáveis estão ordenadas terá impacto direto no tamanho do BDD gerado, no entanto, encontrar uma ordenação ótima das variáveis é um problema NP-Completo (Bollig & Wegener, 1996). Pesquisas sobre como encontrar a melhor ordem das variáveis BDDs têm sido feitas nos últimos anos e vários métodos de ordenação têm sido apresentados (Fujita *et al.* , 1993; Rudell, 1993; Rice & Kulhari, 2008; Kumar *et al.* , 2010; Stergiou, 2011; Sensarma *et al.* , 2012). Encontrar a melhor ordem das variáveis não está no escopo desta tese.

Um BDD que satisfaz as seguintes condições é denominado de BDD reduzido (RBDD):

- não possui subgrafos isomorfos;
- não contém nenhum nó redundante, isto é, nós onde ambas as arestas estão apontando para o mesmo nó sucessor.

Os modelos de BDDs utilizados neste trabalho são considerados ordenados e reduzidos, denominados de ROBDDs (Minato, 1996). Com ROBDDs é possível ter uma representação compacta e canônica (única) para uma determinada função e ordenação de variáveis (Bryant, 1986). A seguir, na Subseção 2.6.1, é apresentado o algoritmo utilizado nesta tese para construir ROBDDs a partir de uma dada função Booleana.

2.6.1 Implementação de ROBDDs

Uma maneira usual de gerar ROBDDs é combinar ROBDDs existentes com operadores, como conjunção (AND) ou disjunção (OR). Neste trabalho estamos interessados em um algoritmo que, dado um ROBDD para uma função f e outro para uma função g, então um ROBDD para f <*op*> g pode ser gerado. Considere <*op*> como sendo um operador entre dois ROBDDs f e g que pode ser calculado conforme:

$$f < op > g = (\neg\, b \wedge (f[0/b] < op > g[0/b])) \vee (b \wedge (f[1/b] < op > g[1/b])).$$

Por razões de eficiência, na simulação simbólica com ROBDDs, todas as funções booleanas de duas variáveis são mapeadas para uma única operação geral, a qual é capaz de expressar

todas as operações booleanas, a chamada operação *if-then-else* (*ITE*), veja Tabela 2.4 (Brace *et al.* , 1990). ITE é uma função com três parâmetros, na qual:

$$ITE(F,G,H) = F.G + F'.H \text{ (se } F \text{ então } G, \text{ senão } H).$$

Tabela 2.4: Funções de duas variáveis descritas pela operação *ITE*

Função	**Fórmula Equivalente**
$F \,.\, G$	*ITE(F,G,0)*
$F + G$	*ITE(F,1,G)*
F'	*ITE(F,0,1)*
$F \oplus G$	*ITE(F,G',G)*

Na decomposição de Shannon temos que:

$$F = (\neg\, v \wedge F[0/v]) \vee (v \wedge F[1/v])$$

tal que *F[0/v]* e *F[1/v]* referem-se, respectivamente, à atribuição de $v = 0$ e $v = 1$. Seja $F = (w,T,E)$ e considerando $\pi(v) < \pi(w)$. Encontar o cofator positivo e o negativo de F em relação a v pode ser obtido da seguinte forma (Brace *et al.* , 1990):

- *F[1/v]* = F, se $\pi(v) < \pi(w)$, e *F[1/v]* = T, se $\pi(v) = \pi(w)$;
- *F[0/v]* = F, se $\pi(v) < \pi(w)$, e *F[0/v]* = E, se $\pi(v) = \pi(w)$.

Sejam F, G e H os vértices de um ROBDD e v a variável topo destes vértices, então *ITE(F,G,H)* pode ser calculado através da seguinte formulação recursiva:

$$ITE(F, G, H) = ITE(v, ITE(F[1/v], G[1/v], H[1/v]), ITE(F[0/v], G[0/v], H[0/v])).$$

Os casos terminais para a recursão são: $ITE(1, F, G) = ITE(0, G, F) = ITE(F, 1, O) = F$.

A seguir, em Algoritmo 1, é apresentado o algoritmo utilizado neste trabalho para gerar ROBDDs. Neste algoritmo, desenvolvido por Brace et. al (1990), é executada a operação *ITE*. As entradas para o algoritmo são três vértices de um ROBDD e a saída é um vértice que representa a operação. Note que foi utilizado uma tabela hash, denominada de *tabela única*, para armazenar os resultados já computados. Esta tabela é utilizada para manter uma

forma canônica sobre os nós no ROBDD de modo que cada nó represente uma função lógica exclusiva.

Algoritmo 1 operação ITE

Input: f, g, h

Output: resultado = ITE(f,g,h)

```
1: if caso terminal da recursão then
2:     return resultado
3: else if tabela calculada tem a entrada (f,g,h) then
4:     return resultado
5: else
6:     x = variavel_topo(f,g,h)
7:     T = ITE(f[1/x],g[1/x],h[1/x])
8:     E = ITE(f[0/x],g[0/x],h[0/x])
9:     if T == E then
10:        return T
11:    end if
12:    r = ache_ou_adicione_à_tabela_única(x,T,E)
13:    insira_na_tabela_calculada({f,g,h},r))
14:    return r
15: end if
```

Na tabela única, cada tripla (v, G, H) é mapeada a um vértice $F = (v, G, H)$ do ROBDD. Cada vértice do ROBDD tem uma entrada na tabela única. Antes de um novo vértice ser adicionado ao ROBDD é realizada uma pesquisa na tabela única para determinar se já existe um vértice para essa função. Se existir, o vértice existente será utilizado, caso contrário o novo vértice será adicionado ao ROBDD e uma nova entrada para ele será criada na tabela única. Assume-se que quando for criado um novo vértice F, os vértices G e H já tenham sido criados. Portanto, a função F só será representada no ROBDD se, e somente se, a tripla (v, G, H) pertencer à tabela única.

A tabela calculada é uma função de memória para acelerar a operação ITE. A manutenção desta tabela é muito importante, pois com o uso desta é possível reduzir o tempo de execução do algoritmo de exponencial para polinomial (Meinel *et al.* , 2002).

2.7 Técnica *Hardware-in-the-loop*

Hardware-In-the-Loop (HIL) é uma técnica utilizada em projeto e avaliação de sistemas de tempo real, principalmente quando estes não podem ser testados facilmente, completamente, e repetidamente em seus ambientes operacionais (Bacic, 2005; Crăciun *et al.* , 2010). A ideia básica de HIL consiste em incluir um hardware real no processo de teste durante o desenvolvimento do sistema (Bacic, 2005; Karimi *et al.* , 2010). Desta forma, com a utilização desta técnica é permitida a simulação em tempo real na qual a planta simulada é acoplada com o sistema real sob teste (SUT) (Michalek *et al.* , 2005; Crăciun *et al.* , 2010; Rankin & Jiang, 2011). HIL é uma forma de simulação em tempo real, que difere dessa pelo simples fato de possuir um componente real no processo.

Hardware-In-the-Loop é uma técnica que vem sendo estudada há mais de 40 anos. Um dos primeiros usos de HIL foi para simulação de vôos (Isermann *et al.* , 1999). Esta técnica também tem sido utilizada em sistemas de teste de orientação de mísseis (Sisle & McCarthy, 1982; Shi-peng *et al.* , 2011). Nos últimos 20 anos, HIL vem sendo utilizada na indústria automotiva para projetar sistemas de freios ABS, sistemas de suspensão, dentre outros (Schaffnit *et al.* , 1998; Short & Pont, 2005). HIL pode ser utilizada em quase todas as áreas em que são necessários testes mais realísticos com componentes de um sistema antes de sua construção final, como por exemplo, softwares de simulação de tráfego (Li *et al.* , 2004) e aplicações médicas, como sistemas usados para realizar cirurgias (Kim *et al.* , 2002). Outras aplicações desta técnica são, principalmente, nos campos da engenharia e da robótica (Williams *et al.* , 2003; Martin & Emami, 2006; Hamelin *et al.* , 2008; Zhou *et al.* , 2011).

A técnica HIL também vem sendo utilizada na área de automação industrial como instrumento de apoio à elaboração de sistemas de controle. A consequência do uso desta técnica tem sido a redução do tempo de desenvolvimento bem como o aumento na qualidade, confiabilidade e segurança do produto final (Albuquerque, 2007; Crăciun *et al.* , 2010).

Na Figura 2.10 é apresentada a arquitetura de um sistema HIL. Para este sistema o modelo da planta é simulado e a unidade de controle é um dispositivo real (Rudas *et al.* , 2010). Este sistema é composto por sensores e atuadores que são utilizados para fornecer uma interface física com a unidade de controle; um controlador para processar dados e uma interface

homem-máquina (IHM), utilizada para controlar processos de tempo real (tais como, observação de variáveis e coleta de dados) e prover a automação de testes. Na arquitetura, os sinais dos sensores são gerados no modelo da planta e processados na unidade de controle que, por sua vez emite sinais de controle ao atuador. Alterações nos sinais dos atuadores resultam na mudança dos valores das variáveis no modelo da planta.

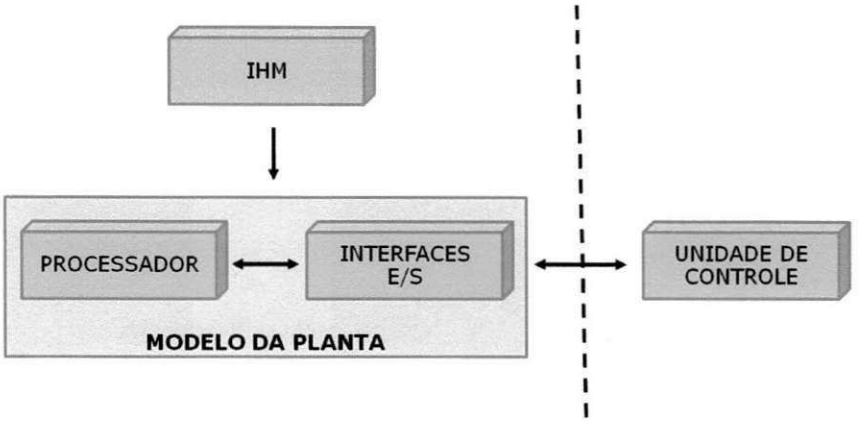


Figura 2.10: Arquitetura de um sistema HIL

HIL é uma das técnicas mais conhecidas para teste de CLPs, pois, com o uso desta técnica é possível emular o comportamento de uma planta industrial antes mesmo da planta real está disponível. Além disso, com HIL também é possível validar um software sem que danos ocorram devido o mau funcionamento do controlador em combinação com o sistema mecânico real (Kain *et al.* , 2011).

A vantagem mais evidente do uso de HIL é que condições do mundo real podem ser testadas sem os riscos reais envolvidos. Por exemplo, um piloto automático pode ser testado sem colocar um avião em risco. Com HIL é possível testar unidades de controle com condições extremas, o que pode não ser viável no mundo real (por exemplo, alta/baixa temperatura). Ainda, com HIL é possível (Machado & Seabra, 2013; Shah & Irfan, 2005): testar um subsistema sem que todo o sistema esteja pronto, tornando o teste uma parte efetiva do processo de desenvolvimento, desde a concepção a implantação; realizar testes no software de controle sem acionar um processo real; testar os efeitos de falhas em atuadores, sensores e computadores em todo o sistema; reproduzir experimentos de forma frequente sem que haja danos reais ao sistema e reduzir custos e o tempo de desenvolvimento do sistema.

2.8 Considerações Finais do Capítulo

Nesta seção são recapitulados os principais conceitos apresentados nesse capítulo. Na Seção 2.1 foi mostrado o funcionamento geral de um CLP. Na Seção 2.2 foi introduzido o padrão IEC 61131-3, dando destaque as linguagens Ladder e FBD pelo fato de serem as mais utilizadas no âmbito industrial. Na Seção 2.3, o protocolo OPC com foco na aquisição e envio de dados para um CLP foi exposto. Na seção 2.4 o padrão ISA 5.2 foi apresentado. Este padrão é utilizado para representar a especificação do programa do CLP. Na seção 2.5 foi introduzida a sintaxe e a semântica das redes de autômatos temporizados. Este formalismo foi utilizado para modelar a especificação do sistema para que seja possível gerar casos de teste a partir de um modelo. Na Seção 2.6, conceitos relacionados à diagramas BDDs foram exibidos. Por fim, na Seção 2.7, a técnica HIL na qual o hardware real (CLP) é inserido no processo de teste é apresentada.

Capítulo 3

Método para Verificar Programas de CLP para SIS

Neste capítulo é introduzido um método para verificar se um programa de CLP para SIS reflete o comportamento definido em sua especificação, de forma que propriedades temporizadas e os estados das saídas do sistema possam ser avaliados e que comportamentos redundantes da implementação do sistema não sejam avaliados. Para o método, os seguintes atores estão envolvidos:

- projetista do sistema: responsável por identificar as funcionalidades do sistema que são necessárias para resolver o problema do domínio. O projetista do sistema também é responsável por gerar um documento, especificação do sistema, a partir das funcionalidades estabelecidas. A especificação do sistema deve ser na forma de diagramas ISA 5.2, conforme descrito na Seção 2.4;
- programador do sistema: responsável por desenvolver o programa do CLP para SIS. Este programa deve ser escrito em uma das cinco linguagens definida pelo padrão IEC 61131-3, conforme descrito na Seção 2.2;
- testador: responsável por realizar os testes no sistema desenvolvido. Este ator é o usuário do método proposto.

O desenvolvimento de programas de CLPs para SIS, conforme apresentado na Figura 3.1, é realizado em uma sequência de etapas as quais definem o processo de desenvolvimento de um programa para CLP. Neste trabalho é considerado um processo de desenvolvimento constituído de quatro etapas:

- definição dos requisitos: nesta etapa é definido o que o sistema deve fazer (suas funções) e suas propriedades essenciais e desejáveis (Sommerville, 2007);
- especificação: com base nos requisitos coletados é produzida a especificação do sistema na forma de diagramas de lógica binária seguindo o padrão ISA 5.2;
- implementação e teste: nesta etapa o código do programa do CLP é desenvolvido e testado. Este programa é escrito segundo uma das cinco linguagens definidas no padrão IEC 61131-3;
- implantação do código no CLP: nesta etapa o programa desenvolvido é carregado em um CLP e colocado em produção.

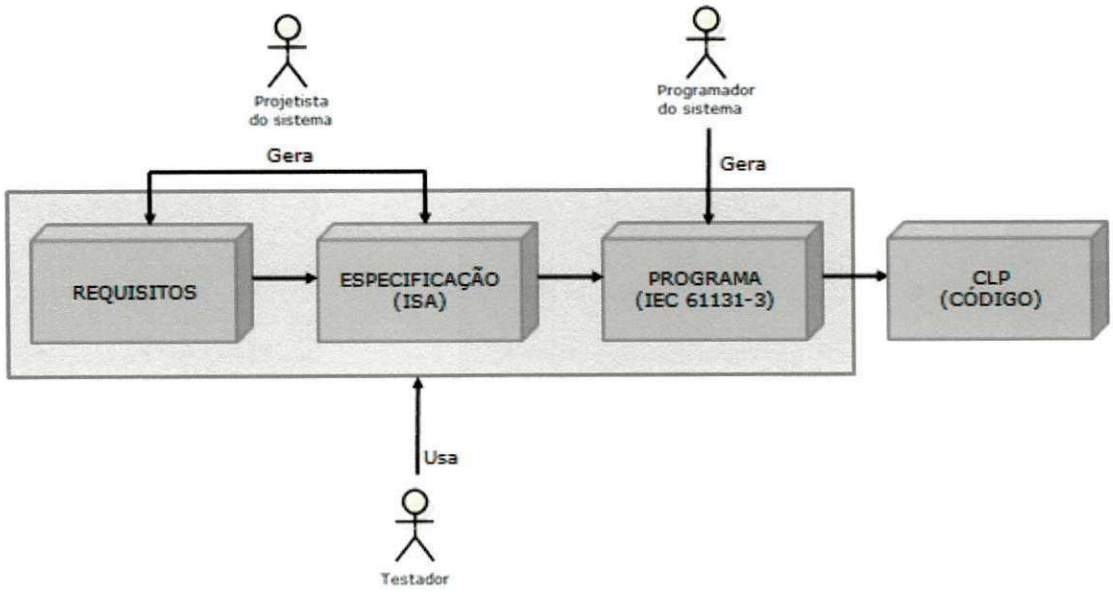


Figura 3.1: Processo de desenvolvimento de um programa de CLP para SIS.

A seguir, na Figura 3.2, é apresentado o método proposto nesta tese. O método, baseado em MBT e HIL, é descrito como segue: primeiro a especificação do sistema, fornecida pelo projetista do sistema, é convertida, de forma automática, em uma rede de autômatos temporizados. Após a geração desta rede de autômatos temporizados, casos de teste são gerados, de forma automática, a partir desta rede de autômatos. Para esta tarefa, uma ferramenta de teste foi desenvolvida com o objetivo de gerar um conjunto de casos de testes em que os estados das saídas e propriedades temporizadas do sistema são avaliados. Depois, os valores das entradas dos casos de teste gerados são enviados para o CLP, o qual está presente a implementação do sistema a ser avaliada. Este programa é desenvolvido pelo programador do sistema. Por fim, um veredito sobre a conformidade entre especificação e implementação do sistema é fornecido, ou seja, as saídas geradas a partir da rede de autômatos temporizados são comparadas com as saídas geradas pelo CLP. Detalhes sobre a geração da rede de autômatos temporizados a partir de diagramas ISA 5.2 e do funcionamento da ferramenta de teste serão apresentados nas seções a seguir.

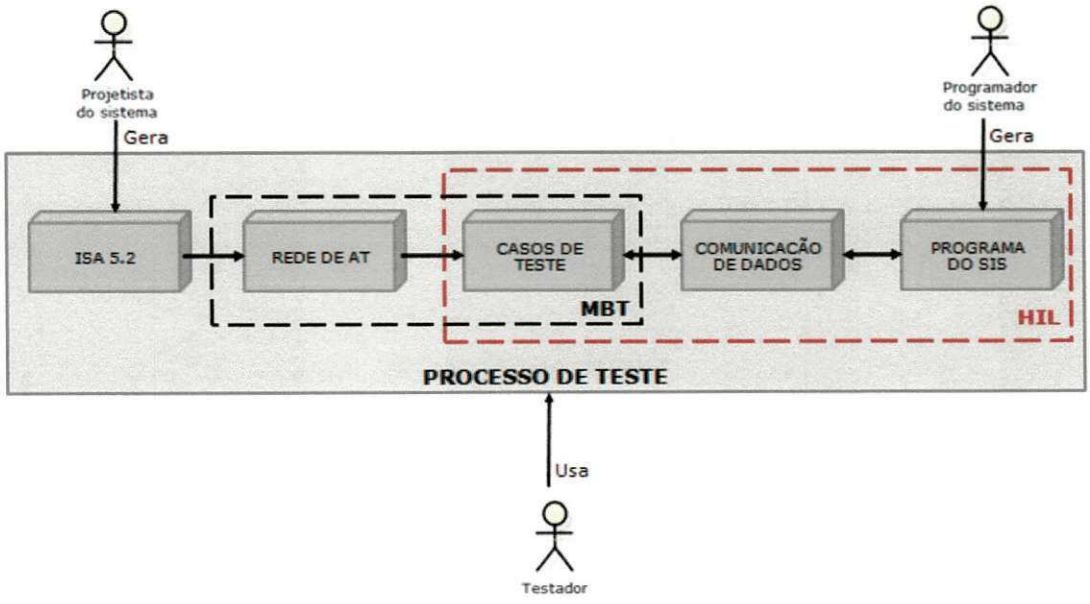


Figura 3.2: Método para verificar programas de CLP para SIS.

O testador em posse da especificação e da implementação do SIS pode utilizar o método proposto de forma paralela ao processo de desenvolvimento minimizando o custo adicional para produção de artefatos específicos para teste. Além disso, a complexidade de construção

dos modelos de autômatos temporizados bem como dos casos de teste gerados é transparente para o testador, uma vez que o mesmo necessita apenas fornecer a especificação e a implementação do SIS para obter o veredito dos testes. Para utilizar o método é necessário que o testador tenha conhecimento acerca de diagramas ISA e linguagens do padrão IEC 61131-3, uma vez que os possíveis erros apresentados no veredito de teste não são reportados para os diagramas ISA ou para a implementação do sistema. Assim, o testador terá que identificar qual tipo de erro ocorreu no programa.

Nas seções seguintes serão apresentados a arquitetura HIL proposta e os três módulos desenvolvidos para verificar programas de CLP para SIS: módulo para geração dos modelos de autômatos temporizados, descrito na Seção 3.2, módulo para geração dos casos de tese, exposto na Seção 3.3 e módulo para comunicação de dados, descrito na Seção 3.4. A validação destes módulos será apresentada no Capítulo 4.

3.1 Arquitetura HIL para Verificação de Programas de CLP para SIS

Nesta tese propomos uma arquitetura HIL, conforme Figura 3.3, de forma que a conformidade entre a especificação e a implementação do sistema executado na unidade de controle, sistema sob teste (SUT), possa ser verificada. Neste trabalho, a unidade de controle é representada por um controlador lógico programável (CLP) e o programa executado nesta unidade é denominado de programa de CLP para SIS. Na arquitetura proposta o modelo da planta não é utilizado, uma vez que estamos interessados apenas em verificar a conformidade do programa do SIS e não o funcionamento da planta industrial.

A arquitetura HIL proposta é constituída de dois módulos: unidade de controle e uma interface homem-máquina (IHM). No módulo IHM utilizamos aplicações para modelar a especificação do sistema, gerar e executar os casos de teste de conformidade e controlar processos de tempo real. Os componentes do módulo IHM (aplicações e servidor OPC) devem ser utilizados no mesmo computador pessoal (PC). O fluxo de dados no sistema é como segue: as aplicações do módulo IHM são utilizadas para fornecer valores de simulação para o servidor OPC, que por sua vez os envia para o CLP, no qual está presente o programa do CLP para SIS. Estes valores são processados e ao final do processamento a tabela de variáveis do

CLP é atualizada e enviada para o servidor OPC, que por sua vez a envia para as aplicações IHM, fechando o ciclo. A comunicação de dados entre os componentes (aplicações, servidor OPC e CLP) é bidirecional.

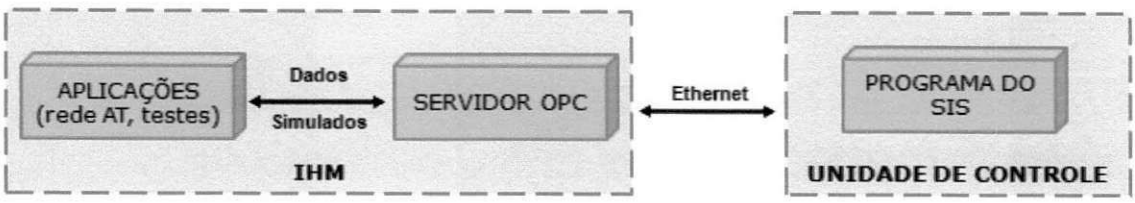


Figura 3.3: Arquitetura HIL proposta.

Na Figura 3.4 é apresentado o fluxograma para o processo de envio e recebimento de dados entre os módulos IHM e a unidade de controle. Este processo é composto pelos seguintes passos: (1) Leitura da especificação do sistema, dada na forma de diagramas ISA 5.2. Neste trabalho consideramos apenas diagramas ISA com variáveis do tipo Booleana, além disso, a lógica do sistema deve ser composta apenas por temporizadores e operações Booleanas AND, OR ou uma combinação destas. (2) Geração automática da rede de autômatos temporizados a partir da especificação do sistema. A rede é gerada segundo a sintaxe e semântica da ferramenta Uppaal. (3) Geração automática dos casos de teste a partir da rede de autômatos temporizados. Com os casos de teste gerados é possível verificar os estados das saídas e propriedades temporizadas do sistema. Cada caso de teste é composto por um conjunto de valores Booleanos definidos para as entradas, um conjunto de valores Booleanos definidos para as saídas e tempo de execução do caso de teste. (4) Envio dos valores das entradas dos casos de teste gerados para o servidor OPC. (5) Envio dos valores das entradas dos casos de teste gerados para o CLP. (6) Recebimento dos valores das saídas processadas pelo programa do CLP para SIS a partir dos valores das entradas dos casos de teste. Estes dados são enviados do CLP para o servidor OPC. T_{test} é tempo de execução de cada caso de teste. Este valor é definido quando o caso de teste é gerado. (7) Comparação entre os valores das saídas geradas a partir da rede de autômatos temporizados (saídas dos casos de teste) e as saídas geradas pelo programa do CLP. (8) Impressão do veredito de teste. Existem dois possíveis vereditos: *conforme*, as saídas esperadas foram realmente as geradas e *não conforme* caso contrário.

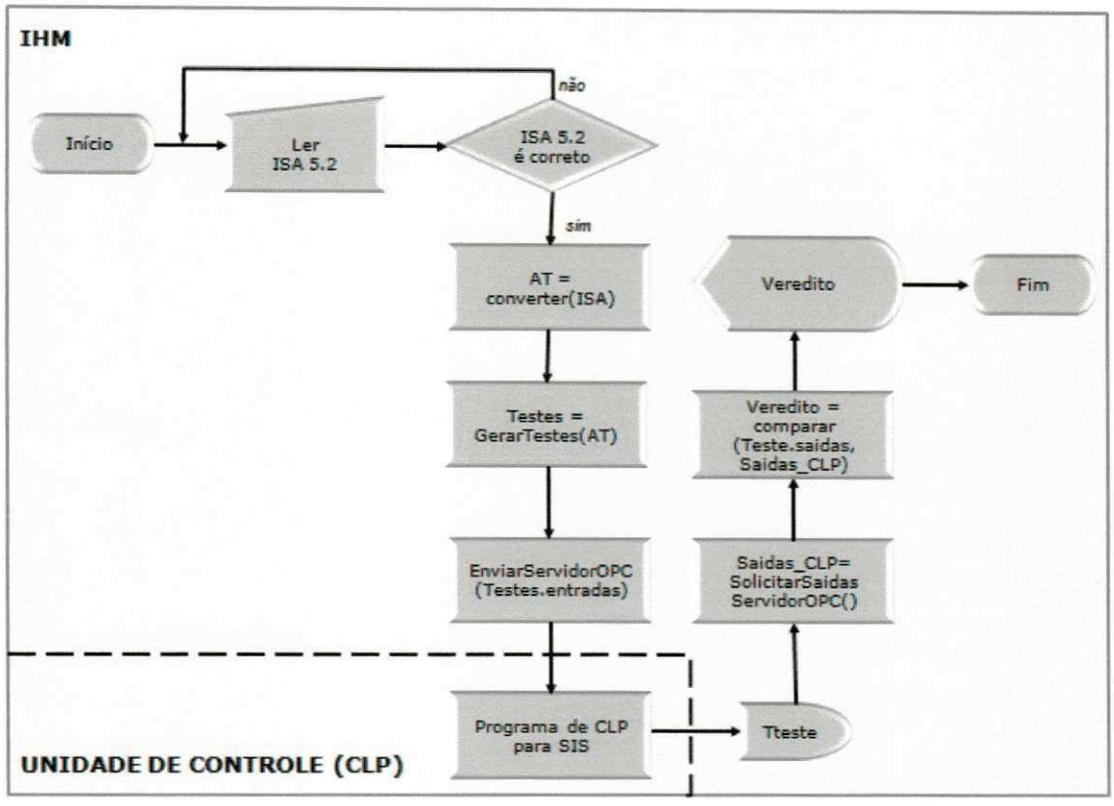


Figura 3.4: Fluxograma para processo de envio e recebimento de dados na arquitetura HIL proposta.

3.2 Autômatos Temporizados para Diagramas ISA 5.2

Nesta seção são apresentados os modelos extraídos a partir da especificação do sistema dada na forma de diagramas ISA 5.2. A partir dos trabalhos relacionados, apresentados na Seção 3.2.1, o formalismo de autômatos temporizados foi definido para ser utilizado no processo de geração dos casos de teste, descrito na Seção 3.2.2.

3.2.1 Métodos para Modelagem de Diagramas ISA 5.2

Nos trabalhos de Barbosa et al. (2007; 2008) é introduzido um método para converter diagramas de lógica binária ISA 5.2 em uma rede de autômatos temporizados. A rede gerada é composta por sete tipos de templates de autômatos temporizados, dentre os quais temos: templates para temporizadores, templates para variáveis de entrada, templates para células

de memória (flip-flop SR), template para o ciclo de varredura, template para atualizar os valores das variáveis de entrada e dois templates para controlar o envio e recebimento de sinais de entrada e saída.

Na Figura 3.5 é ilustrado o modelo de autômato temporizado para um temporizador do tipo DI. Observe que no modelo não há variáveis de relógio e a temporização é realizada por meio de ciclos (*cycles* = *PT/tScan*, em que *PT* é o valor presente do temporizador e *tScan* é o tempo de scan. A variável *cycles* é do tipo inteiro). Com o uso deste artifício não podemos garantir que a saída do temporizador será liberada no tempo indicado (*PT*) pois, considerando, por exemplo, um temporizador com $PT = 11s$ e um $tScan = 10s$, temos que a saída deste temporizador será liberada após 1 ciclo de scan ($10s$) e não após $11s$, durante o segundo ciclo, como deveria ser. Além de problemas com a temporização, nos modelos desenvolvidos cada flip-flop é modelado como um autômato temporizado e estes modelos são sincronizados com o modelo de autômato que representa o ciclo de varredura, aumentando assim a quantidade de localidades e, consequentemente o número de estados gerados durante a geração dos casos de teste. Para solucionar estes problemas, nesta tese variáveis de relógio com tempo contínuo foram inseridas nos modelos de autômato para temporizadores e as células de memória são implementadas como funções.

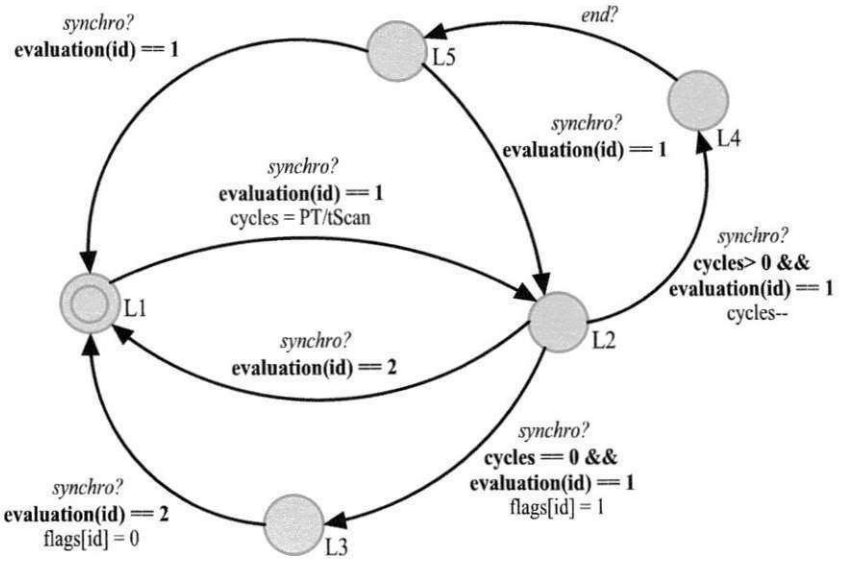


Figura 3.5: Autômato temporizado proposto por Barbosa et al. para um temporizador DI.

3.2.2 Geração dos Modelos de Autômatos Temporizados

Nesta seção são apresentados os modelos de autômatos temporizados extraídos a partir da especificação de um programa para CLP, descrita na forma de diagramas de lógica binária ISA 5.2. Estes modelos propostos são gerados conforme a sintaxe e a semântica da ferramenta Uppaal. A rede gerada é composta por dois tipos de templates de autômatos temporizados: autômato que representa o comportamento de elementos temporizados e autômato que representa o ciclo de varredura do CLP. Nas subseções a seguir estes modelos serão apresentados.

A principal razão para a escolha do formalismo de autômatos temporizados deve-se ao fato de que a semântica de autômatos adapta-se bem a modelagem de sistemas de tempo real. Neste trabalho utilizamos uma variante de autômatos temporizados, redes de autômatos temporizados, em que canais de sincronização, variáveis de relógios com tempo contínuo e ações são introduzidas nos modelos. Nesta tese, apenas sistemas determinísticos são considerados.

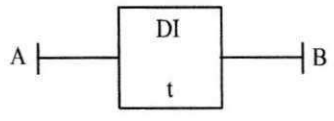
Autômatos para Temporizadores

Nesta seção são apresentados os modelos de autômatos temporizados para temporizadores. Para tanto, três templates de autômatos são considerados: autômato que representa temporizadores do tipo DI (*Delay Initiation of output*), autômato que representa temporizadores do tipo DT (*Delay Termination of output*) e autômato que representa temporizadores do tipo PO (*Pulse Output*). Cada temporizador da especificação do sistema é modelado como um autômato temporizado.

Na Figura 3.6(a) é apresentada uma representação de um temporizador DI. Em um temporizador DI a saída (B) é energizada depois de um período de tempo (t) quando a entrada é verdadeira (A = *verdadeiro*). Quando a entrada for falsa (A = *falso*) a saída será desativada (B = *falso*). Na Figura 3.6(b) é apresentada a tabela verdade para a variável B. Considere a variável *Tempo* como sendo o tempo decorrido.

Para definirmos o autômato temporizado que representa o comportamento de um temporizador DI vamos fazer as seguintes considerações. O temporizador DI será composto por três localidades: *timer enable* (L_1), *timer timing* (L_2) e *timer done* (L_3). Estas localidades representam as condições correspondentes do temporizador. A tabela apresentada na Figura 3.6(b) será utilizada para determinar a mudança entre as localidades do autômato.

Para cada mudança de localidade será gerado um arco, no qual as entradas representarão guardas e as saídas representarão atribuições. A primeira linha da tabela será utilizada para o temporizador mover da localidade *timer enable* para a localidade *timer timing*. A segunda linha será utilizada para o temporizador mover da localidade *timer timing* para a localidade *timer done*. A terceira linha será utilizada para o temporizador mover da localidade *timer timing* para a localidade *timer enable* e, para mover da localidade *timer done* para a localidade *timer enable*. As variáveis *control_nTimer* e *acc_nTimer* são utilizadas para indicar qual temporizador está apto a executar em cada ciclo de varredura. O sufixo *nTimer* representa a posição do temporizador nos diagramas ISA. A seguir, apresenta-se a definição formal do autômato que representa o comportamento de um temporizador DI.



(a) Símbolo.

A	Tempo	B
verdadeiro	< t	falso
verdadeiro	⩾t	verdadeiro
falso	-	falso

(b) Tabela Verdade para B.

Figura 3.6: Temporizador DI.

Definição 1: Um autômato temporizado para um temporizador DI é a sêxtupla (L, l_0, C, Ac, E, I), tal que:

- L = $\{L_1, L_2, L_3\}$;
- $l_0 = L_1$;
- C = {time};
- Ac = {synchro?};
- E = $\{E_1, E_2, E_3, E_4\}$, tal que:
  - E_1 = (L_1, synchro?, A, {B = false, time = 0, acc_nTimer = t, control_nTimer = 1}, L_2);

- E_2 = (L_2, synchro?, !A, {B = false, acc_nTimer = 0, control_nTimer = 0}, L_1);
- E_3 = (L_2, synchro?, A && time $\geq$ t, {B = true, acc_nTimer = 0}, L_3);
- E_4 = (L_3, synchro?, !A, {B = false, acc_nTimer = 0}, L_1);

- I = {$\emptyset$}, *não possui invariantes.*

Na Figura 3.7 é ilustrada a representação do autômato temporizado para um temporizador DI. Este autômato é executado da seguinte forma: quando o autômato que representa o ciclo de varredura do CLP for sincronizado com *synchro?* e a entrada do temporizador DI for verdadeira (*A*), então, o arco com origem na localidade *L1* e destino na localidade *L2* é ativado, o relógio *time* é inicializado e as variáveis específicas para cada temporizador (*acc_nTimer* e *control_nTimer*) são inicializadas. Estas variáveis são utilizadas para identificar qual temporizador está sendo executado num determinado ciclo de varredura. A partir deste ponto pode ocorrer duas situações. A primeira ocorre quando a entrada do temporizador for falsa (*!A*), assim, o arco com origem na localidade *L2* e destino na localidade *L1* é ativado e o temporizador é restaurado (*acc_nTimer = 0, control_nTimer = 0*). A segunda situação ocorre quando o tempo de simulação decorrido for igual ou superior ao valor de *t* e a entrada do temporizador for verdadeira (*A && time* $\geq$ *t*), desta forma, a saída do temporizador será verdadeira. Na localidade *L3*, quando a entrada do temporizador for falsa o arco com origem na localidade *L3* e destino na localidade *L1* é ativado e o temporizador é desativado (*B = false*).

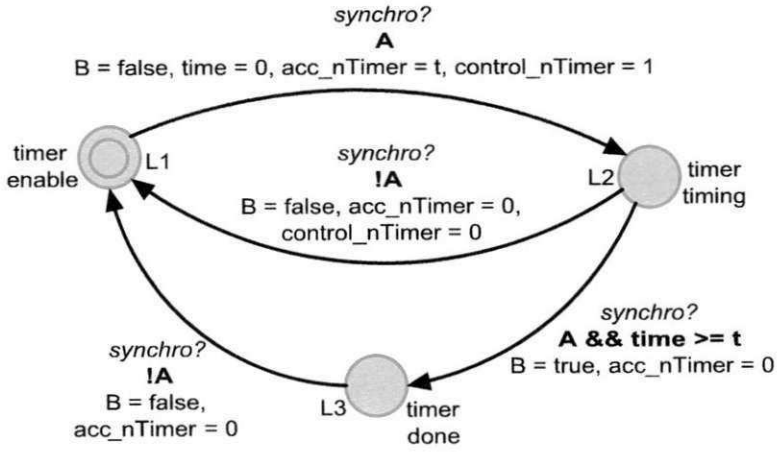
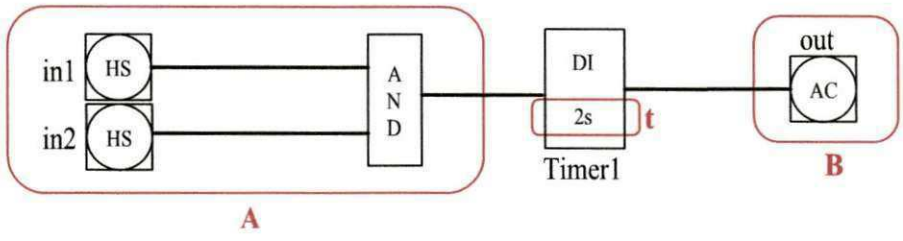
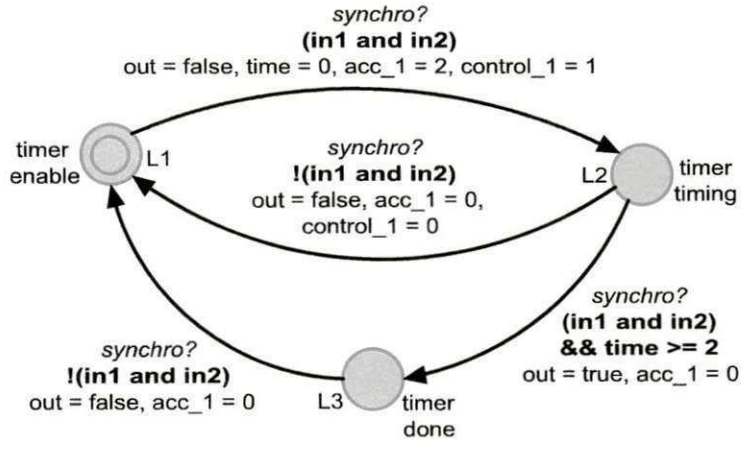


Figura 3.7: Autômato temporizado para um temporizador DI.

Na Figura 3.8(a) é apresentado um exemplo de um diagrama ISA 5.2 com um temporizador do tipo DI. De acordo com a **Definição 1**, o modelo de autômato temporizado para o temporizador *Timer1* é apresentado na Figura 3.8(b). Para este autômato temos que a entrada do temporizador (A) será representada pela função Booleana $in1 \wedge in2$, a variável t (valor presente do temporizador) será inicializada com a constante *2* e a saída do temporizador (B) será representada pela variável *out*. Como temos apenas um temporizador, então o sufixo *nTimer* das variáveis *acc_nTimer* e *control_nTimer* será representado pelo valor 1.



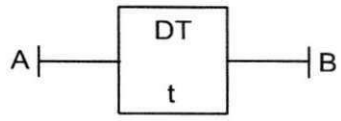
(a) Diagrama ISA.



(b) Autômato para o temporizador DI.

Figura 3.8: Exemplo de um diagrama ISA 5.2 com um temporizador DI.

Na Figura 3.9(a) é apresentada uma representação de um temporizador do tipo DT. Em um temporizador DT a saída é ativada (B = *true*) quando a entrada é verdadeira (A = *true*). Quando a entrada for falsa a saída será desativada após um período de tempo (t). Na Figura 3.9(b) é apresentada a tabela verdade para a variável B. Considere a variável *Tempo* como sendo o tempo decorrido.



(a) Símbolo.

A	Tempo	B
verdadeiro	-	verdadeiro
falso	⩾t	falso
falso	< t	verdadeiro

(b) Tabela Verdade para B.

Figura 3.9: Temporizador DT.

Para definirmos o autômato temporizado que representa o comportamento de um temporizador DT vamos fazer as seguintes considerações. O temporizador DT será composto por três localidades: *timer enable* (L_1), *timer timing* (L_2) e *timer done* (L_3). Estas localidades representam as condições correspondentes do temporizador. A tabela apresentada na Figura 3.9(b) será utilizada para determinar a mudança entre as localidades do autômato. Para cada mudança de localidade será gerado um arco, no qual as entradas representarão guardas e as saídas representarão atribuições. A primeira linha da tabela será utilizada para o temporizador mover da localidade *timer enable* para a localidade *timer timing* e, para mover da localidade *timer done* para a localidade *timer timing*. A segunda linha será utilizada para o temporizador mover da localidade *timer done* para a localidade *timer enable*. A terceira linha será utilizada para o temporizador mover da localidade *timer timing* para a localidade *timer done*. As variáveis *control_nTimer* e *acc_nTimer* são utilizadas para indicar qual temporizador está apto a executar em cada ciclo de varredura. A seguir, é apresentada a definição formal do autômato utilizado para modelar o comportamento de um temporizador DT.

Definição 2: Um autômato temporizado para um temporizador DT é a sêxtupla (L, l_0, C, Ac, E, I), tal que:

- L = $\{L_1, L_2, L_3\}$;
- $l_0 = L_1$;
- C = {time};
- Ac = {synchro?};
- E = $\{E_1, E_2, E_3, E_4\}$, tal que:

- E_1 = (L_1, synchro?, A, {B = true, control_nTimer = 1}, L_2);
- E_2 = (L_2, synchro?, !A, {acc_nTimer = t, time = 0}, L_3);
- E_3 = (L_3, synchro?, A, {acc_nTimer = 0}, L_2);
- E_4 = (L_3, synchro?, !A && acc_nTimer == 0 && time $\geq$ t, {control_nTimer = 0, B = false}, L_1);

- I = {$\emptyset$}, *não possui invariantes.*

Na Figura 3.10 é ilustrada a representação do autômato temporizado para um temporizador DT. Este autômato é executado da seguinte forma: quando o autômato que representa o ciclo de varredura do CLP for sincronizado com *synchro?* e a entrada do temporizador DT for verdadeira (*A*), então, o arco com origem na localidade *L1* e destino na localidade *L2* é ativado e a saída do temporizador com valor lógico *true* é liberada. Quando a entrada do temporizador for falsa (*!A*), então, o arco com origem na localidade *L2* e destino na localidade *L3* é ativado e o temporizador é inicializado (*acc_nTimer = t, time = 0*). A partir deste ponto pode ocorrer duas situações. A primeira ocorre quando a entrada do temporizador for verdadeira (*A*), assim, o arco com origem na localidade *L3* e destino na localidade *L1* é ativado e o temporizador é restaurado (*acc_nTimer = 0*). A segunda situação ocorre quando o tempo de simulação decorrido for igual ou superior ao valor de *t* e a entrada do temporizador for falsa (*A && time $\geq$ t && acc_nTimer == 0*), desta forma, a saída do temporizador será falsa.

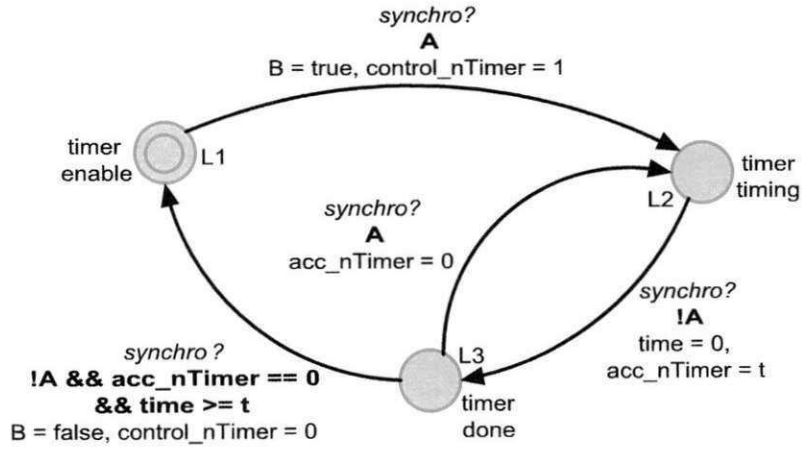
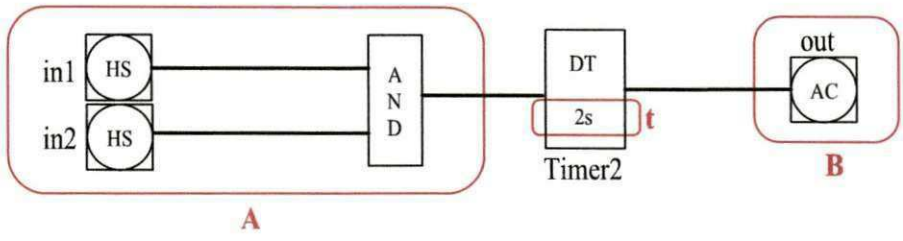
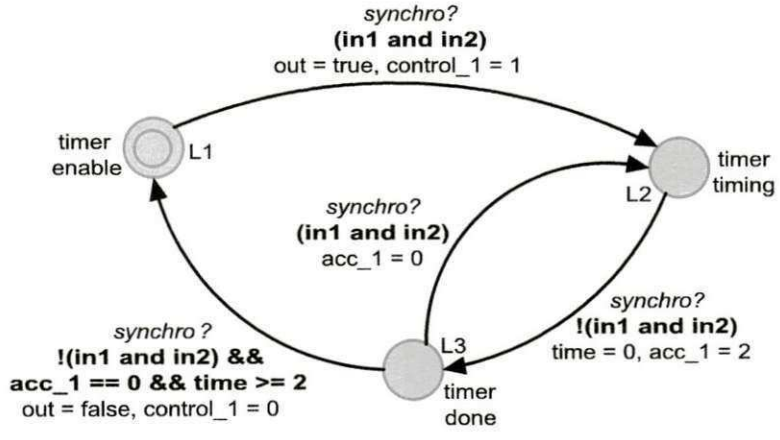


Figura 3.10: Autômato temporizado para um temporizador DT.

Na Figura 3.11(a) é apresentado um exemplo de um diagrama ISA 5.2 com um temporizador do tipo DT. De acordo com a **Definição 2**, o modelo de autômato temporizado para o temporizador *Timer2* é apresentado na Figura 3.11(b). Para este autômato temos que a entrada do temporizador (A) será representada pela função Booleana $in1 \wedge in2$, a variável t (valor presente do temporizador) será inicializada com a constante 2 e a saída do temporizador(B) será representada pela variável *out*. Como temos apenas um temporizador, então o sufixo *nTimer* das variáveis *acc_nTimer* e *control_nTimer* será representado pelo valor 1.



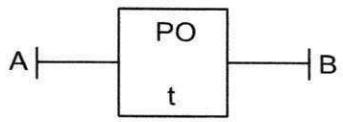
(a) Diagrama ISA.



(b) Autômato para o temporizador DT.

Figura 3.11: Exemplo de um diagrama ISA 5.2 com um temporizador DT.

Na Figura 3.12(a) é apresentada uma representação de um temporizador do tipo PO. Em um temporizador PO a saída é ativada (B = *true*) durante um período de tempo (t) quando a entrada é verdadeira. Mesmo que a entrada seja falsa (A = *false*) a saída continua ativada durante o tempo pré-definido (*Time* = t). Na Figura 3.12(b) é apresentada a tabela verdade para a variável B. Considere a variável *Tempo* como sendo o tempo decorrido.



(a) Símbolo.

A	Tempo	B
verdadeiro	< t	verdadeiro
-	≥t	falso

(b) Tabela Verdade para B.

Figura 3.12: Temporizador PO.

Para definirmos o autômato temporizado que representa o comportamento de um temporizador PO vamos fazer as seguintes considerações. O temporizador PO será composto por duas localidades: *timer enable* (L_1) e *timer timing* (L_2). Estas localidades representam as condições correspondentes do temporizador. A tabela apresentada na Figura 3.12(b) será utilizada para determinar a mudança entre as localidades do autômato. Para cada mudança de localidade será gerado um arco, no qual as entradas representarão guardas e as saídas representarão atribuições. A primeira linha da tabela será utilizada para o temporizador mover da localidade *timer enable* para a localidade *timer timing*. A segunda linha será utilizada para o temporizador mover da localidade *timer timing* para a localidade *timer enable*. As variáveis *control_nTimer* e *acc_nTimer* são utilizadas para indicar qual temporizador está apto a executar em cada ciclo de varredura. A seguir, é apresentada a definição formal do autômato utilizado para modelar o comportamento de um temporizador PO.

Definição 3: Um autômato temporizado para um temporizador PO é a sêxtupla (L, l_0, C, Ac, E, I), tal que:

- L = $\{L_1, L_2\}$;
- $l_0 = L_1$;
- C = {time};
- Ac = {synchro?};
- E = $\{E_1, E_2\}$, tal que:
  - E_1 = (L_1, synchro?, A && control_nTimer == 0, {B = true, acc_nTimer = t, time = 0}, L_2);

- E_2 = (L_2, synchro?, time $\geq$ t && acc_nTimer == 0, {B = false, control_nTimer = 1}, L_1);

- I = {$\emptyset$}, *não possui invariantes.*

Na Figura 3.13 é ilustrada a representação do autômato temporizado para um temporizador PO. Este autômato é executado da seguinte forma: quando o autômato que representa o ciclo de varredura do CLP for sincronizado com *synchro?* e a entrada do temporizador PO for verdadeira (*A*), então, o arco com origem na localidade *L1* e destino na localidade *L2* é ativado e a saída do temporizador com valor lógico *true* é liberada. Quando o tempo de simulação decorrido for igual ou superior ao valor de *t*, então o arco com origem na localidade *L2* e destino na localidade *L1* é ativado, o temporizador é reiniciado e sua saída com valor lógico *false* é liberada.

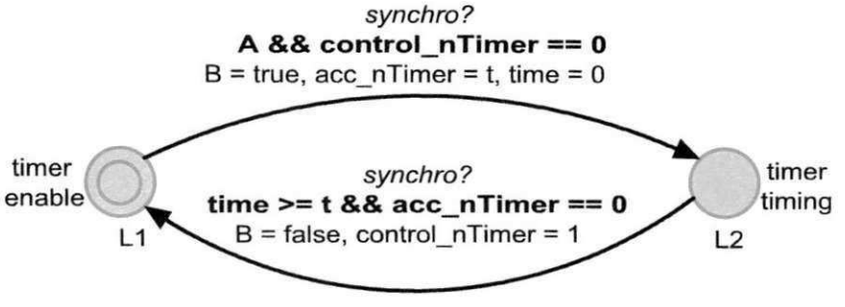
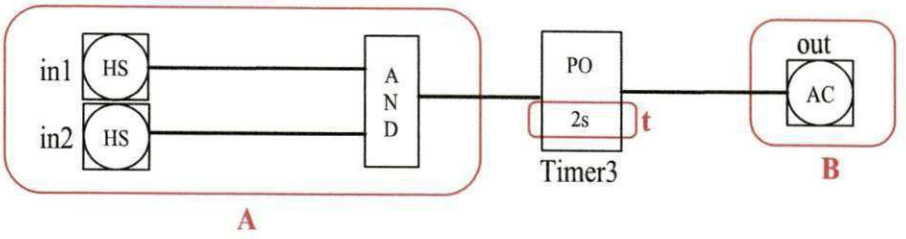
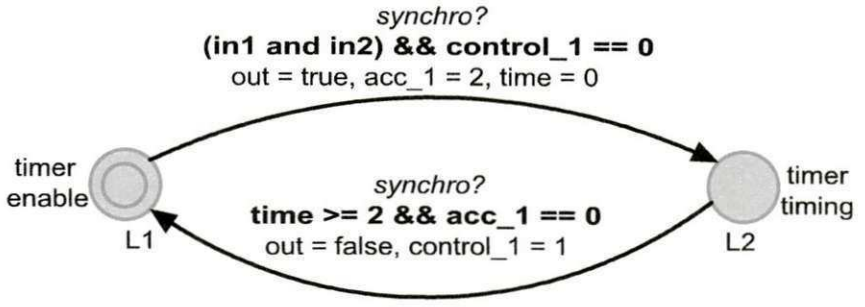


Figura 3.13: Autômato temporizado para um temporizador PO.

Na Figura 3.14(a) é apresentado um exemplo de um diagrama ISA 5.2 com um temporizador do tipo PO. De acordo com a **Definição 3**, o modelo de autômato temporizado para o temporizador *Timer3* é apresentado na Figura 3.14(b). Para este autômato temos que a entrada do temporizador (*A*) será representada pela função Booleana $in1 \wedge in2$, a variável *t* (valor presente do temporizador) será inicializada com a constante *2* e a saída do temporizador (*B*) será representada pela variável *out*. Como temos apenas um temporizador, então o sufixo *nTimer* das variáveis *acc_nTimer* e *control_nTimer* será representado pelo valor 1.



(a) Diagrama ISA.



(b) Autômato para o temporizador PO.

Figura 3.14: Exemplo de um diagrama ISA 5.2 com um temporizador PO.

Autômato para o Ciclo de Varredura do CLP

Nesta seção é apresentado o autômato temporizado que representa o ciclo de varredura de um CLP. No modelo do autômato são consideradas as três etapas que compõem o ciclo de varredura: leitura das variáveis de entrada, execução do programa e atualização das saídas. Desta forma, para o autômato, são necessários três arcos, um para processar cada etapa do ciclo. O autômato é composto por duas localidades: *initialization mode* (L_1) e *run mode* (L_2). Estas localidades representam as condições correspondentes do ciclo de varredura do CLP. A definição dos valores para as variáveis de entrada e a execução da lógica do sistema para geração das saídas envolvem operações de atribuição. Portanto, na geração do modelo do autômato estas operações serão representadas como atribuições através, respectivamente, das funções *readInputs()* e *updateOutputs()*.

Na Figura 3.15 é ilustrado o autômato temporizado que representa o ciclo de varredura de um CLP. Este autômato é executado da seguinte forma: após serem definidos os valores das variáveis de entrada na função *readInputs()* a execução da lógica do sistema é inicializada, esta etapa de processamento não deve ultrapassar o tempo de varredura ($c \leq tScan$). A função

checkExecutionTimers() é utilizada para verificar se algum temporizador deve ser executado. Após a execução da lógica do sistema, as saídas são liberadas, função *updateOutputs()* e o valor de *c* é restaurado. O estado inicial deste autômato é *committed* porque a evolução dos relógios não deve ser considerada durante a atualização das variáveis de entrada. A função *evaluate()* é utilizada para verificar se os temporizadores foram executados. A seguir é apresentada a definição formal deste autômato.

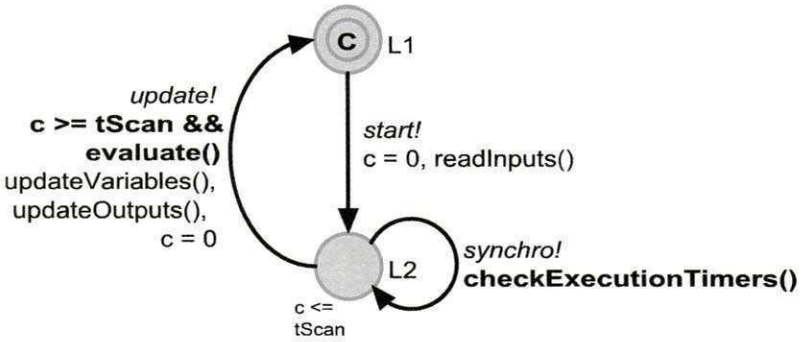


Figura 3.15: Autômato temporizado para o ciclo de varredura de um CLP

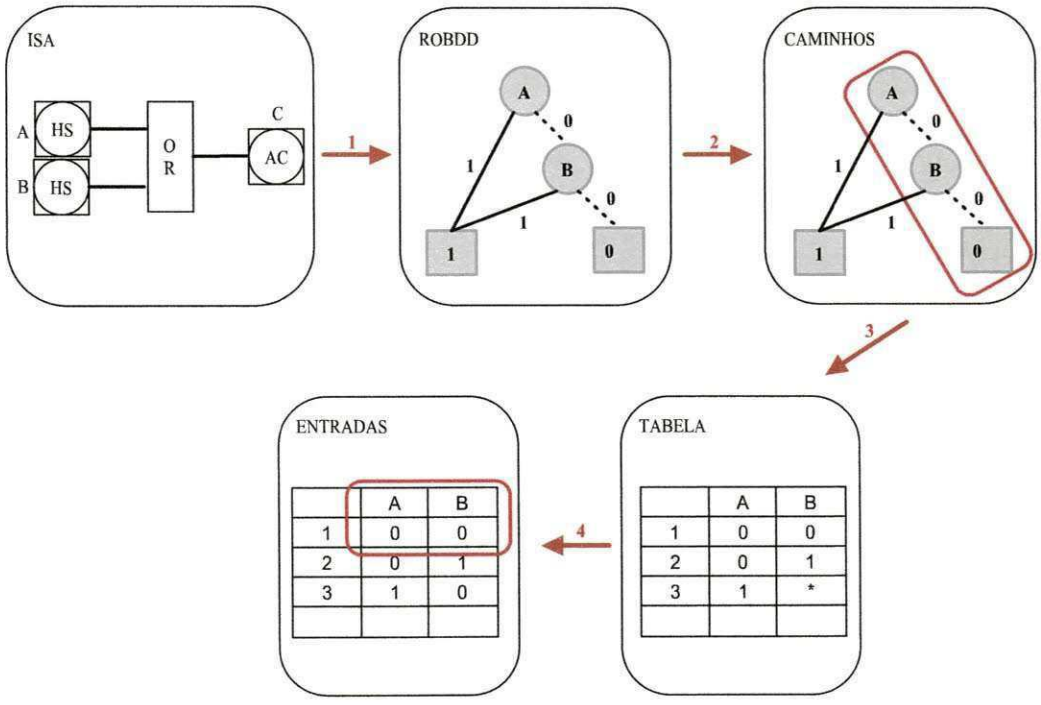
Definição 4: Um autômato temporizado para o ciclo de varredura de um CLP é a sêxtupla (L, l_0, C, Ac, E, I), tal que:

- L = {L_1, L_2}, L_1 *é committed*;
- $l_0 = L_1$;
- C = {c};
- Ac = {start!, sync!, update!};
- E = {E_1, E_2, E_3}, where:
  - E_1 = (L_1, start!, ,× {c = 0, readInputs()}, L_2);
  - E_2 = (L_2, synchro!, checkExecutionTimers(), {}, L_2);
  - E_3 = (L_2, update!, c ≥ tScan && evaluate(), {c = 0, updateOutputs(), updateVariables()}, L_1);
- I = { I(L_2) → time ≤ tScan}.

Funções para o autômato que representa o ciclo de varredura de um CLP

Nesta seção definem-se as funções utilizadas para construir o autômato temporizado que representa o ciclo de varredura de um CLP. São apresentados os algoritmos utilizados para definir as funções *readInputs* e *updateOutputs*.

Na Figura 3.16 o esquema utilizado para definir os valores para as variáveis de entrada do sistema é apresentado. O processo é composto por quatro passos: (1) Geração do ROBDD para cada bloco do diagrama ISA que determina uma saída do sistema. (2) Extração de todos os caminhos do ROBDD. Em cada caminho são definidos os valores para as variáveis de entrada (arestas de cada nó intermediário), e o valor da saída, valor do nó folha. (3) Geração de uma tabela na qual cada variável de entrada compõe uma coluna da tabela e os valores definidos para cada variável correspondem a uma linha da tabela. (4) Cada linha da tabela é utilizada para testar o sistema. A forma como cada linha da tabela é utilizada depende do tipo de função que a gerou. Neste trabalho dois tipos de função são consideradas: as com temporização, função composta por temporizadores, e funções sem temporização.



	A	B
1	0	0
2	0	1
3	1	0

	A	B
1	0	0
2	0	1
3	1	*

Figura 3.16: Esquema para definir valores para as entradas do sistema.

A seguir, o algoritmo utilizado para definir os valores para as variáveis de entradas é apresentado. Para o algoritmo, considere um diagrama ISA como sendo uma tupla *(E, S)*, no qual $E = \{e_1,...,e_n\}$ é o conjunto de entradas e $S = \{s_1,...,s_j\}$ é o conjunto de saídas do diagrama ISA. Consideramos que a saída de cada temporizador também compõe o conjunto S. Cada saída $s \in S$ é composta por um conjunto de variáveis de entrada Y e uma função Booleana f (lógica Booleana do bloco). A entrada para o algoritmo é um diagrama ISA e a saída é um conjunto de valores $\{B_1,...,B_m\}$, em que $B_i = (b_1,...,b_n)$ é uma atribuição de variáveis e $1 \leq i \leq m$. Este algoritmo é utilizado para gerar a função *readInputs()* na qual cada conjunto B_i é executado para cada ciclo de varredura do CLP.

Algoritmo 2 generateInputs()

Input: Um diagrama ISA 5.2, no qual ISA = (E, S)

Output: Um conjunto de atribuições B = $\{B_1, ..., B_m\}$, em que $B_i = (b_1, ..., b_n)$ e $1 \leq i \leq m$

```
1: for all s ∈ S do
2:    robdd ← generateROBDD(f(y1, ..., yn))
3:    listPaths ← extractPaths(robdd, f(y1, ..., yn))
4:    for all l ∈ listPaths do
5:       table ← table ∪ createTable(l, E)
6:    end for
7:    B ← B ∪ defineInputs(f(y1, ..., yn), table)
8: end for
9: return B
```

Na linha 2 do algoritmo é construído um ROBDD para cada função Booleana de S, conforme o passo 1 na Figura 3.16. O procedimento ITE, apresentado na seção 2.6 e desenvolvido por Brace et. al Brace *et al.* (1990), foi utilizado para gerar os ROBDDs. Desta forma, para um conjunto de variáveis Booleanas Y = $\{y_1,...,y_n\}$ e uma função Booleana f(y_1,...,y_n) sobre Y com uma ordem fixa de seus argumentos é gerado um ROBDD. Neste trabalho, a ordem das variáveis é estabelecida de forma automática, assim, consideramos a seguinte ordem das variáveis $\pi(y_1) > \pi(y_2) > ... > \pi(y_n)$. Considere π: $Y \rightarrow \{1, 2,..., |Y| = n\}$ como sendo o mapeamento bijetivo dos índices das variáveis. O tamanho dos ROBDDs gerados pode variar de linear O(n) a exponencial $O(2^n)$ dependendo da ordem das variáveis. Encontrar a melhor ordem das variáveis não está no escopo deste trabalho.

Na linha 3 do algoritmo, todos os caminhos cujo valor da função f é $f(y_1,...,y_n) \in \{0,1\}$ são extraídos, conforme o passo 2 na Figura 3.16. Para calcular o valor de uma função f dada em termos de um ROBDD para uma dada atribuição de variável $(b_1,b_2,...,b_n)$, com $b_i \in \{0,1\}$ e $1 \leq i \leq n$, basta seguir um caminho começando na raiz, alternando em cada nó para a extremidade dada pela atribuição de acordo com a variável $y_i = b_i$. Quando um nó folha for atingido (nó *0* ou nó *1*), então o valor da função foi determinado em função dos valores das entradas especificadas.

Nas linhas 4-6 do *Algoritmo 2* é criada uma tabela, conforme o passo 3 na Figura 3.16, na qual cada variável de entrada $e_i \in$ E, com $1 \leq i \leq n$, compõe uma coluna da tabela e $(b_1,...,b_n)$ é uma atribuição de variáveis. Cada atribuição $(b_1,...,b_n)$ corresponde a uma linha da tabela. Uma atribuição $(b_1,...,b_n)$ é gerada a partir dos valores das variáveis definidos em l (caminho de cada ROBDD gerado). Caso alguma variável de entrada e_i não seja definida em l, então consideramos $b_i = 0$.

Na linha 7 do *Algoritmo 2* é gerado um conjunto com todas as atribuições para as variáveis de entrada do sistema, conforme o passo 4 na Figura 3.16. Estas atribuições são utilizadas para testar o sistema. O conjunto gerado é da forma: $B = \{B_1,...,B_m\}$, em que $B_i = (b_1,...,b_n)$, $1 \leq i \leq m$ e $(b_1,...,b_n)$ é uma atribuição para as variáveis de entrada. B é gerado de acordo com cada função f, conforme Algoritmo 3. Caso a função f não possua temporizadores, então todas as atribuições $(b_1,...,b_n)$ obtidas a partir da função f serão utilizadas para testar o sistema. Caso f possua temporizadores, então de acordo com cada tipo de temporizador serão definidas atribuições da forma $(b_1,...,b_n)$ para testar o comportamento de cada temporizador. Estas atribuições serão utilizadas para ativar os arcos dos autômatos que representam cada temporizador. Com este passo do algoritmo é possível testar os estados das saídas e propriedades temporizadas do sistema.

Ainda, no Algoritmo 2, caso alguma atribuição $(b_1,...,b_n)$ da tabela l não tenha sido utilizada para testar o sistema, esta deve ser incluída nos testes. Neste passo do algoritmo cada conjunto $(b_1,...,b_n)$ é único. Com este artifício é possível reduzir o número de casos de teste redundantes.

De acordo com o *Algoritmo 3*, se o temporizador for do tipo DI ou DT serão necessárias quatro atribuições para testar o comportamento do temporizador. Desta forma, as seguintes atribuições serão utilizadas: uma atribuição em que f$(b_1,...,b_n) = 1$ (a entrada do temporizador

será verdadeira e a aresta 1 será ativada); uma atribuição em que f(b_1,...,b_n) = 0 (a entrada do temporizador será falsa e a aresta 2 será ativada); uma atribuição em que f(b_1,...,b_n) = 1 (a entrada do temporizador será verdadeira e as arestas 1 e 3 serão ativadas) e uma atribuição em que f(b_1,...,b_n) = 0 (a entrada do temporizador será falsa e a aresta 4 será ativada). Se o

Algoritmo 3 defineInputs(f(y_1, ..., y_n), table)

Input: Uma função e sua tabela de atribuições

Output: Um conjunto de atribuições B = {B_1, ..., B_m}, onde B_i = (b_1, ..., b_n) e $1 \leq i \leq m$

```
1:  if f.type != 'timer' then
2:      B = table
3:  else
4:      if f.timer == 'DI' || f.timer == 'DT' then
5:          B ← selectAssignmentTrue(table)
6:          B ← B ∪ verifyAssignmentsDependents(table)
7:          B ← B ∪ selectAssignmentFalse(table)
8:          B ← B ∪ selectAssignmentTrue(table)
9:          B ← B ∪ verifyAssignmentsDependents(table)
10:         B ← B ∪ selectAssignmentFalse(table)
11:     else
12:         B ← selectAssignmentTrue(table)
13:         B ← B ∪ verifyAssignmentsDependents(table)
14:         B ← B ∪ radom(selectAssignment(table))
15:     end if
16:     for all t ∈ table do
17:         if t ∉ B then
18:             B ← B ∪ t
19:         end if
20:     end for
21: end if
22: return B
```

temporizador for do tipo PO serão necessárias duas atribuições para testar o comportamento do temporizador: uma atribuição em que f(b_1,...,b_n) = 1 (a entrada do temporizador será verdadeira e a aresta 1 será ativada) e uma atribuição em que f(b_1,...,b_n) = 0 ou uma atribuição

em que f(b_1,...,b_n) = 1 (a aresta 2 será ativada). A escolha desta última atribuição é feita de forma randômica visto que a aresta 2 do autômato é ativada independente do valor da entrada do temporizador. Durante a seleção das atribuições é avaliado se a saída ativada de um temporizador irá afetar a execução dos testes. Desta forma, no algoritmo é avaliado se alguma atribuição deve ser executada após a saída do temporizador ser ativada. Para lógicas com dependência de temporizadores, a seleção das atribuições dependentes é realizada sempre com a saída do temporizador não subordinado ativada. Desta forma, a entrada do temporizador não dependente deve ser mantida ativada até todas as atribuições subordinadas serem executadas.

Na Figura 3.17 apresenta-se um diagrama ISA com dependência de temporizadores. Observe que o comportamento do temporizador *Timer2* é dependente do comportamento do temporizador *Timer1* pois, a entrada do temporizador *Timer2* é definida em função da saída do temporizador *Timer1*. Desta forma, de acordo com o Algoritmo 3, a entrada do temporizador *Timer1* (temporizador não dependente) deve se manter ativada até que todas as atribuições para o temporizador *Timer2* (atribuições subordinadas) que dependam da saída ativada de *Timer1* sejam executadas. Na Figura 3.18 são apresentadas as atribuições definidas para avaliar o comportamento dos temporizadores *Timer1* e *Timer2*. A variável *O1* representa a saída do temporizador *Timer1*. Para o exemplo, primeiro são definidas as atribuições para o temporizador *Timer1* e, logo após a saída do *Timer1* ser ativada as atribuições dependentes são executadas, no caso a terceira atribuição para avaliar o comportamento do temporizador *Timer2*.

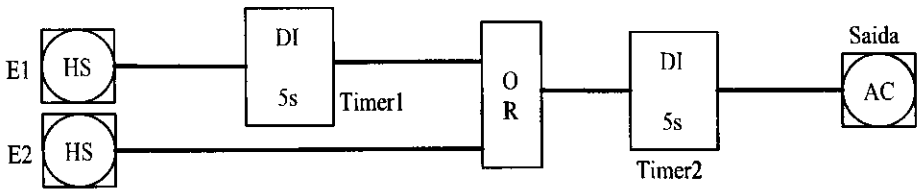


Figura 3.17: Exemplo de um diagrama ISA com dependência de temporizadores.

Na função updateOutputs(), apresentada em Algoritmo 4, são processadas as saídas do diagrama ISA. Para cada variável $s \in S$ é executada a função f com uma atribuição de variáveis (b_1,...,b_n), com $n = |E|$. Cada atribuição é definida na função readInputs(). A função updateOutputs() é processada para cada ciclo de varredura, então da mesma forma que a

função readInputs(), será executada m vezes. A saída da função é uma atribuição de variáveis $\{a_1,...,a_j\}$, onde j = $|S|$. Esta atribuição representa os valores gerados para as saídas do diagrama ISA a partir de uma dada atribuição para as variáveis de entrada, atruibuição $(b_1,...,b_n)$.

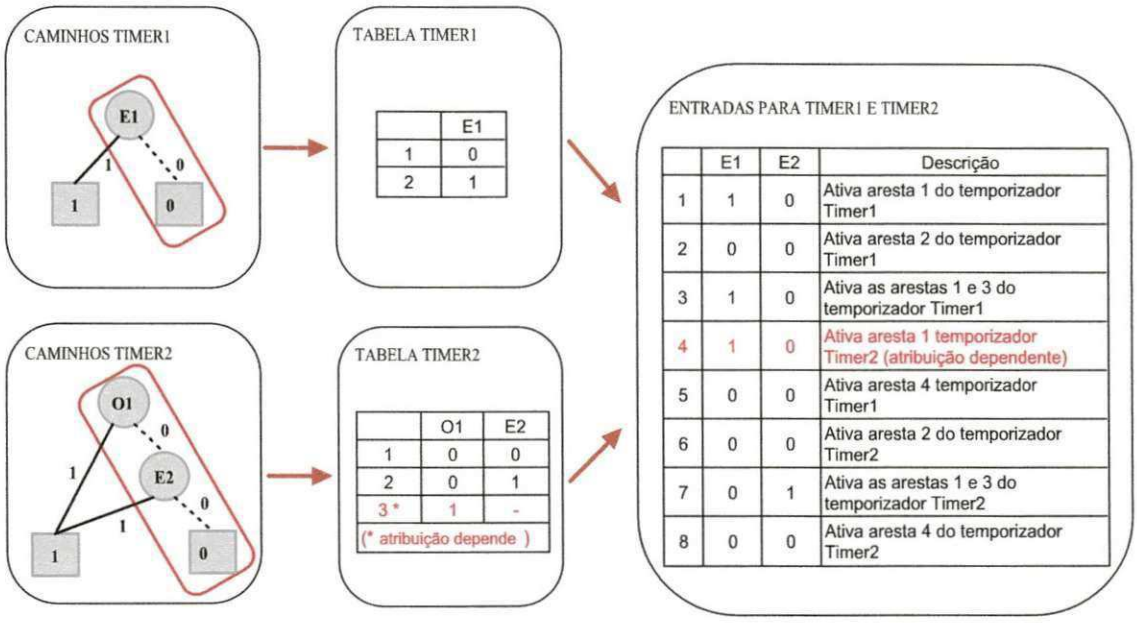


Figura 3.18: Atribuições utilizadas para avaliar o comportamento dos temporizadores Timer1 e Timer2.

Algoritmo 4 updateOutputs(ISA)

Input: Uma atribuição $(b_1,...,b_n)$, em que $n = |E|$

Output: Uma atribuição A = $(a_1,...,a_j)$, em que $j = |S|$

1: i = 1
2: **for all** s ∈ S **do**
3: a_i = execute(f($b_1,...,b_n$)
4: Adicionar a_i a tupla A
5: i = i + 1
6: **end for**
7: **return** A

3.3 Definição dos Casos de Teste

A idéia utilizada para gerar os casos de teste consiste em construir conjuntos de teste para cada ciclo de varredura do CLP. Desta forma, os casos de teste gerados são compostos por tuplas da forma $C = (E, S, T, t)$, tal que:

- E é o conjunto de valores Booleanos gerados para as entradas do modelo da especificação do sistema;
- S é conjunto de valores Booleanos correspondentes às saídas processadas do modelo da especificação a partir do conjunto E;
- T é o traço de execução, ou seja, as sequêcias de ações (eventos de sincronização) utilizadas para gerar as saídas do modelo da especificação;
- t é o tempo de execução do caso de teste.

O conjunto E é obtido a partir da função *readInputs()*, definida na seção anterior. Desta forma, o conjunto será composto por todas as atribuições utilizadas para testar o comportamento de elementos temporizados e saídas do sistema. O conjunto S é obtido a partir dos valores gerados para as saídas do sistema, execução da função *updateOutputs()*. O conjunto T é obtido a partir da sequência de ações da execução da rede de autômatos utilizadas para gerar as saídas. O elemento t da tupla é definido de acordo com a tempo presente do temporizador, no caso de funções com temporização, ou pelo tempo de varredura, no caso de funções sem temporização.

Na Figura 3.19 é apresentado um exemplo de como os casos de teste são definidos a partir de um modelo de autômato temporizado. Para cada caso de teste temos:

- conjunto E, definido a partir da função *readInput()*. O conjunto é composto por dois elementos, representados pelos valores Booleanos determinados para as variáveis de entrada A e B;
- conjunto S, definido a partir da função *updateOutputs()*. O conjunto é composto por um elemento, representado pelo valor Booleano determinado para a variável C a partir do conjunto E;

- conjunto T, obtido a partir da sequência de ações da execução da rede de autômatos utilizadas para gerar as saídas. O conjunto é composto pelos eventos de sincronização *start!* e *update!*;

- como a função não é temporizada, então $t = tScan$. A variável *tScan* representa o tempo de varredura.

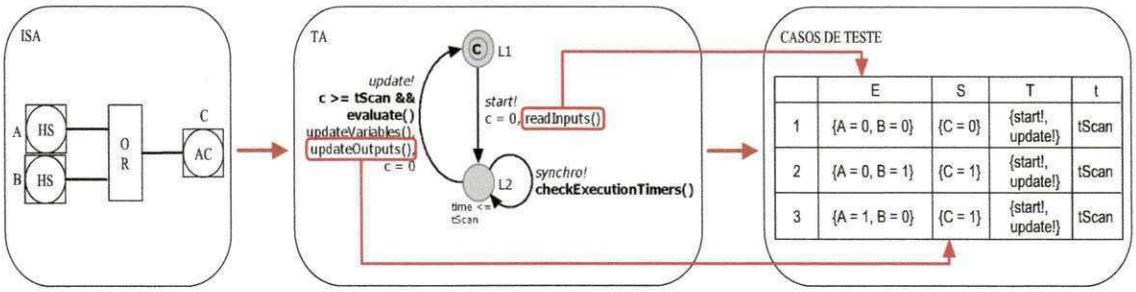


Figura 3.19: Exemplo para definir os casos de teste.

3.4 Módulo para Comunicação de Dados

Nesta seção é apresentado como é realizada a troca de dados entre a unidade de controle e as aplicações do módulo IHM. A comunicação dos dados é feita através de um servidor OPC, via protocolo OPC.

Uma vez gerados os casos de teste, os valores definidos para as entradas (conjunto E) são enviados para o CLP, via protocolo OPC, no qual está presente a implementação do sistema a ser verificada. Para o envio e recebimento de dados do servidor foi utilizada uma biblioteca que dá suporte à especificação OPC de acesso a dados (OPC DA), descrita na Seção 2.3.1. Com o uso desta especificação é possível trocar dados entre aplicações clientes e servidores OPC em tempo real. Na implementação foi considerada a hierarquia dos objetos estabelecida na especificação OPC DA. Desta forma, foram criados:

- um servidor: estrutura utilizada para armazenar grupos e itens, gerenciar a conexão com a aplicação cliente e implementar uma estrutura de endereçamento capaz de associar itens com variáveis reais;

- dois grupos: um para armazenar itens que representam as variáveis de entrada e outro para armazenar itens que representam as variáveis de saída. Estes grupos são utilizados para realizar o agrupamento lógico e gerenciamento dos itens;

- itens: cada variável é representada por um item do tipo Booleano. Caso a variável seja de entrada então, o valor do item que a representa é o estabelecido no caso de teste. Caso a variável seja de saída então, o valor do item será nulo pois a este será atribuído um valor válido no final do ciclo de varredura quando as saídas do sistema são liberadas.

A seguir, em Algoritmo 5, é apresentado o procedimento utilizado para enviar e recuperar dados do servidor OPC. No algoritmo, itens referentes às variáveis de entrada são enviados

Algoritmo 5 dataExchange(C,p)

Input: Uma conjunto de casos de teste C e um programa para CLP p

Output: Um veredito de conformidade entre C e p

```
Criar e inicializar um servidor
Criar dois grupos: g_entradas e g_saídas
for all c = (E,S,T,t) ∈ C do
  for all e in E do
    Criar um item para e com o valor binário atribuído a mesma no caso de teste
    Adicionar o item ao grupo g_entradas
  end for
  for all variável de saída s de c do
    Criar um item para a variável de saída s
    Adicionar o item ao grupo g_saídas
  end for
  Adicionar g_entradas e g_saidas ao servidor
  Enviar os dados dos itens do grupo g_entradas para o servidor de forma assíncrona
  Recuperar os dados dos itens do grupo g_saídas de forma assíncrona
  Comparar S e g_saídas
  Comparar t e o timestamp de g_saídas
  Fornecer um veredito de conformidade entre S e g_saídas
end for
```

para o servidor OPC e, ao final do tempo estabelecido (elemento t da tupla do caso de teste), itens referentes às variáveis de saída são recuperados do servidor OPC. Finalmente, os valores das saídas geradas pelo CLP são comparados com os valores das saídas obtidas a partir da rede de autômatos temporizados (conjunto S) e um veredito de teste é fornecido. A leitura e a escrita dos dados no servidor OPC é feita de forma assíncrona pois, são mais eficientes para grandes quantidades de dados pelo fato de não utilizar muitos recursos de rede, além disso, os recursos do sistema são imediatamente liberados após a requisição dos dados.

3.5 Análise do Método Proposto

Nesta seção, realiza-se uma análise do método proposto nesta tese. Para tanto, três aspectos serão analisados: geração dos casos de teste para avaliar os estados das saídas do sistema; geração dos casos de teste para avaliar o comportamento de elementos temporizados e redução do número de casos de teste redundantes.

Neste trabalho, as entradas para os casos de teste são definidas a partir da abordagem de diagramas de decisão binária ordenados e reduzidos. Para tanto, cada função booleana que determina uma saída é representada por um ROBDD. Uma vez gerado o ROBDD para uma determinada função f, é possível encontrar uma atribuição para as variáveis de entrada que tornem $f = 1$. Caso exista um caminho a partir do nó raiz para o nó folha com valoração 1, denominado de caminho positivo, a atribuição das variáveis para ativar o caminho positivo é uma solução para $f = 1$. De forma análoga, podemos obter os caminhos negativos. Desta forma, se para cada função que determina uma saída for possível obter caminhos positivos e negativos, então, conforme descrito no Algoritmo 2, é possível testar os estados das saídas do sistema. Porém, para garantirmos que os estados das saídas do sistema serão avaliados, cada função Booleana para cada saída do sistema deve possuir as seguintes propriedades:

- ser representada por fórmulas bem formadas (*fbf*);
- a *fbf* não ser uma tautologia e,
- a *fbf* não ser uma contradição.

Uma fórmula bem formada é definida indutivamente de acordo com as seguintes regras de formação:

- qualquer átomo é uma *fbf*;
- se ϕ é uma *fbf*, então $\sim\phi$ também o é;
- se ϕ e ψ são *fbfs*, então $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$ e $(\phi \leftrightarrow \psi)$ também o são.

Qualquer sentença não estabelecida como uma *fbf* por estas três regras não é uma *fbf*. As *fbfs* complexas são construídas a partir das simples, por aplicações repetidas das regras de formação. Dizemos que uma fbf é uma tautologia se todas as interpretações para a *fbf* são verdadeiras e, que uma fbf é uma contradição se todas as interpretações para a *fbf* são falsas.

Caso uma função booleana de uma determinada saída seja representada apenas por uma *fbf* que é tautologia ou uma *fbf* que é contradição, então podemos garantir que ao menos um estado será avaliado para esta saída do sistema, pois ROBDDs têm uma excelente propriedade de que cada nó intermediário é incluído em pelo menos um caminho positivo (caso não houver um caminho positivo, o nó deve ser reduzido para o nó folha com valoração *0*) (Minato, 1996).

Para testar o comportamento dos temporizadores também é necessário que as funções que definem as entradas dos temporizadores sejam representadas por *fbfs* que não são uma tautologia e *fbfs* que não são uma contradição. Pois, para habilitar as transições dos temporizadores, conforme descrito em Algoritmo 3, é necessário, para cada função f de um temporizador, ter ao menos uma atribuição para as variáveis de entrada para satisfazer a função $f = 1$ e outra para atribuição para satisfazer a função $f = 0$. Caso alguma fórmula seja uma *fbf* que é uma tautologia ou uma *fbf* que é uma contradição então o comportamento do temporizador não será avaliado.

Ao utilizamos ROBDD para gerar os casos de teste asseguramos que para uma determinada função e ordenação de variáveis é possível ter uma representação compacta e canônica. Esta propriedade garante que nem nós redundantes nem subgrafos isomorfos serão gerados para cada ROBDD que representa uma determinada função. Desta forma, podemos afirmar que testes redundantes não serão gerados para uma mesma função sem temporização. Porém, para um mesmo sistema, ROBDDs que representam funções diferentes podem gerar o mesmo conjunto de atribuições. Para solucionar este problema, em Algoritmo 2, cada atribuição selecionada é única. Com este artifício é possível reduzir o número de casos de teste redundantes. No caso de funções com temporização, para avaliar o comportamento dos

temporizadores pode ser necessário que uma determinada atribuição seja selecionada mais de uma vez. Isto ocorre quando temos uma única atribuição definida para ativar ou desativar o temporizador.

3.6 Considerações Finais do Capítulo

Neste capítulo foi apresentado um novo método para verificar a implementação de um programa de CLP para SIS a partir de sua especificação, dada na forma de diagramas ISA 5.2, de forma que propriedades temporizadas e os estados das saídas do sistema possam ser verificados e que comportamentos redundantes do sistema não sejam verificados. Na Seção 3.1 foi proposta uma nova arquitetura para verificação de programas de CLP para SIS. Na arquitetura dois módulos são considerados: unidade de controle, representada pelo CLP, e IHM, módulo no qual as aplicações são desenvolvidas. Diferentemente de outros trabalhos, apresentados na Seção 2.7, na arquitetura proposta o modelo da planta não é considerado pois nesta tese estamos interessados em verificar o programa do CLP para SIS e não o funcionamento da planta industrial.

Na Seção 3.2 são apresentados os algoritmos necessários para gerar uma rede de autômatos temporizados a partir da especificação do SIS. Para os modelos gerados temos: os elementos temporizados são modelados e analisados em conjunto com a lógica completa do sistema e, o tempo é modelado com variáveis de relógio com tempo contínuo. Na Seção 3.3 os casos de teste são descritos. Na geração dos casos de teste utilizou-se diagramas ROBDD com dois objetivos: avaliar os estados das saídas e propriedades temporizadas do sistema e reduzir o número de casos de teste redundantes. Na Seção 3.4 é apresentado o módulo de comunicação de dados. Neste módulo, a troca de dados entre o CLP e as aplicações do módulo IHM são realizadas via protocolo OPC.

Na seção 3.5 foi mostrado que para verificar a conformidade entre a especificação e a implementação de um programa de CLP para SIS, de forma que propriedades temporizadas e os estados das saídas do sistema possam ser verificadas, é necessário que todas as funções booleanas utilizadas para determinar as saídas do sistema sejam representadas por *fbfs* que não são uma tautologia e por *fbfs* que não são uma contradição. Caso contrário, não há garantias de que estas propriedades serão avaliadas.

Capítulo 4

Avaliação do Método Proposto

Neste capítulo o método proposto nesta tese é avaliado. Para tanto, são utilizados três estudos de caso cujo cenário de execução é apresentado na Seção 4.1. O primeiro estudo de caso, apresentado na Seção 4.2, representa um sistema para prevenção de incêndio. Nas Seções 4.3 e 4.4 são exibidos, respectivamente, dois estudos de caso, um SIS para detectar a presença de fogo e gás em uma zona e um SIS para controlar o nível de um tanque. Na Seção 4.5, os resultados obtidos nos três estudos de caso são avaliados.

4.1 Cenário para Execução dos Estudos de Caso

Nesta seção é apresentado o cenário utilizado para executar os estudos de caso. Para realizar os experimentos foram utilizados um CLP *Allen-Bradley* 1769-L32E *CompactLogix 5332E* da *Rockwell Automation* e um PC (*Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz 1GB de RAM*) com sistema operacional *Windows XP*. No PC são executadas as seguintes aplicações:

- aplicação na qual diagramas ISA são convertidos em uma rede de autômatos temporizados, denominado de conversor;
- aplicação na qual os casos de teste são gerados, denominada de ferramenta de teste;
- programa RSLogix 5000, utilizado para criar e gerenciar o programa do CLP para SIS;
- programa RSLinx, o servidor OPC;

- aplicação na qual é realizada a troca de dados entre o servidor OPC e a ferramenta de teste. Nesta aplicação, denominada de ClienteOPC, também é fornecido os vereditos de teste.

Para reproduzir os experimentos é necessário aplicar os seguintes passos:

1. estabelecer comunicação entre o PC e o CLP usando o RSLinx. Detalhes de como configurar o servidor OPC com os dispositivos podem ser encontrados na página da *Rockwell Automation* [1];

2. desenvolver a implementação do sistema: criar um projeto no RSLogix 5000, configurando-o com o CLP. Detalhes de como criar um projeto no RSLogix 5000 podem ser encontrados na página da *Rockwell Automation*. Em seguida, desenvolver a implementação do sistema e carregá-la no CLP. Não esquecer de definir os endereços para as variáveis utilizadas na lógica implementada e inserir descrições para as mesmas de acordo com a Tabela 4.1;

Tabela 4.1: Endereçamento de elementos no RSLogix 5000.

Variáveis de Entrada			**Variáveis de Saída**		
Endereço	*Descrição*	*ID do MICA*	*Endereço*	*Descrição*	*ID do MICA*
Local:1:I.Data.0	Botão (A)	RB01/00	Local:3:O.Data.0	Led (C)	RB03/00
Local:1:I.Data.0	Botão (B)	RB01/01			

3. fornecer os diagramas ISA para a aplicação conversor: nesta ferramenta é gerado um arquivo XML. Este arquivo representa uma rede de autômatos temporizados segundo a sintaxe e semântica da ferramenta Uppaal;

4. fornecer a rede de autômatos para a ferramenta de teste: nesta ferramenta os casos de teste são gerados a partir da rede de autômatos gerada no passo anterior;

5. fornecer os casos de teste para a aplicação clienteOPC e aguardar o veredito de teste: nesta aplicação as entradas dos casos de teste são enviadas para o servidor OPC e as saídas geradas a partir da rede de autômatos temporizados (saídas esperadas) são

[1] www.rockwellautomation.com/

comparadas com as saídas geradas pelo CLP (saídas geradas). Se as saídas esperadas foram realmente as geradas então um veredito que indica conformidade é fornecido, caso contrário um veredito que indica não conformidade é informado.

4.2 Sistema de Prevenção de Incêndio

O SIS utilizado neste estudo de caso é um sistema para prevenção de incêndio. O objetivo com este estudo de caso é avaliar o método para especificações sem elementos temporizados. O sistema é composto por: três sensores que são utilizados para detectar fogo (F1, F2 e F3); um alarme; um led; um botão; uma chave manual e um atuador, utilizado para ativar dois extintores de incêndio de forma automática. Nas Figuras 4.1 e 4.2 são ilustradas, respectivamente, a especificação deste sistema na forma de diagramas ISA 5.2 e a sua implementação em FBD. Para o sistema as seguintes operações são realizadas:

- o alarme só é disparado se pelo menos dois sensores acusarem fogo;
- o botão pode ser utilizado para ativar o alarme de incêndio;
- a chave manual pode ser usada para desligar o alarme depois que os sensores forem desligados;
- se for feita uma tentativa de desligar o sistema de alarme com os sensores ativados o alarme permanece ligado;
- o led acionado indica que pelo menos um dos sensores está ativado;
- caso o alarme ou o led sejam acionados então, o atuador é programado para acionar os dois extintores de incêndio.

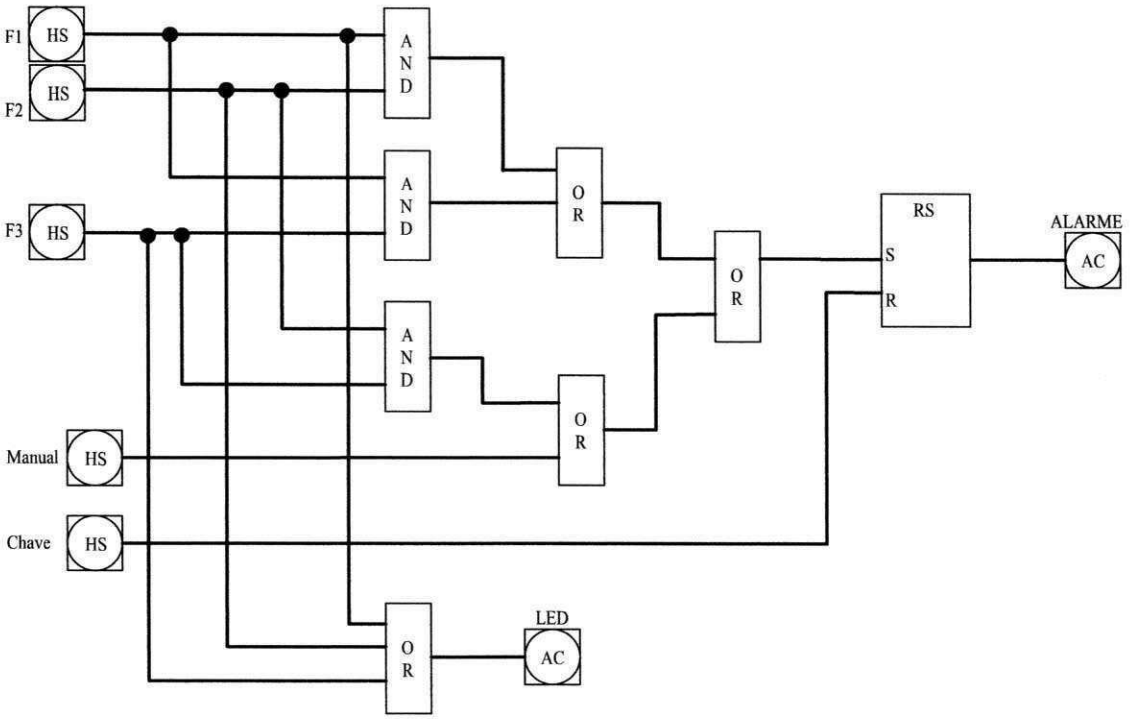


Figura 4.1: Diagrama ISA 5.2 para o sistema de prevenção de incêndio.

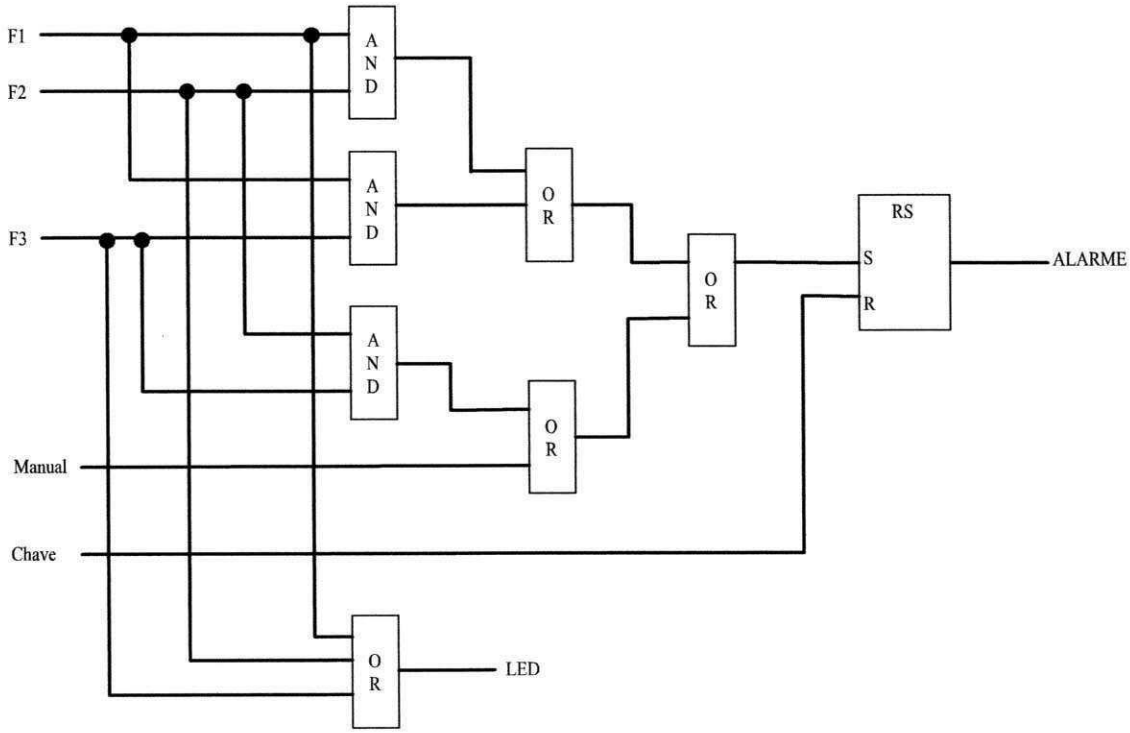


Figura 4.2: Programa FBD para o sistema de prevenção de incêndio.

4.2.1 Geração dos Modelos de Autômatos Temporizados

Como a especificação do sistema não é composta por elementos temporizados então, a rede de autômatos gerada será formada apenas pelo autômato que representa o ciclo de varredura do CLP (ver Figura 4.3). Para o diagrama ISA apresentado na Figura 4.1 temos:

- conjunto das entradas: $E = \{F1, F2, F3, Manual, Chave\}$;
- conjunto das saídas: $S = \{s_1, s_2, s_3, s_4\}$. Cada saída é uma tupla da forma $\langle Y, f \rangle$, em que Y é o conjunto de variáveis de entrada e f é a função Booleana que define a saída.
  - $s_1 = S = \langle \{F1, F2, F3\}, ((F1 \wedge F2) \vee (F1 \wedge F3) \vee (F2 \wedge F3) \vee Manual) \rangle$;
  - $s_2 = R = \langle \{Chave\}, Chave \rangle$;
  - $s_3 = Alarme = \langle \{S, R\}, ((R \wedge Alarme') \vee (R' \wedge S)) \rangle$;
  - $s_4 = Led = \langle \{F1, F2, F3\}, (F1 \vee F2 \vee F3) \rangle$.

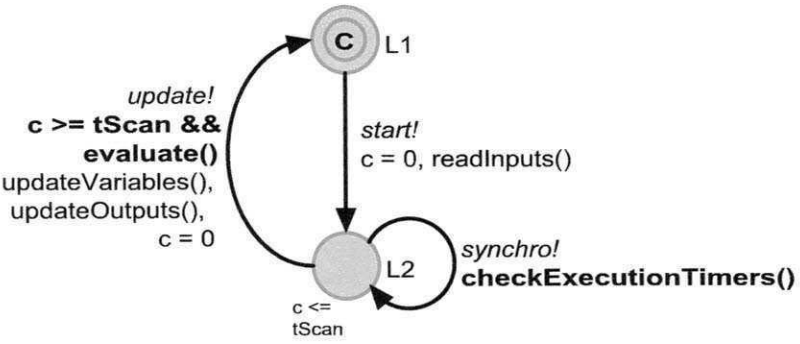
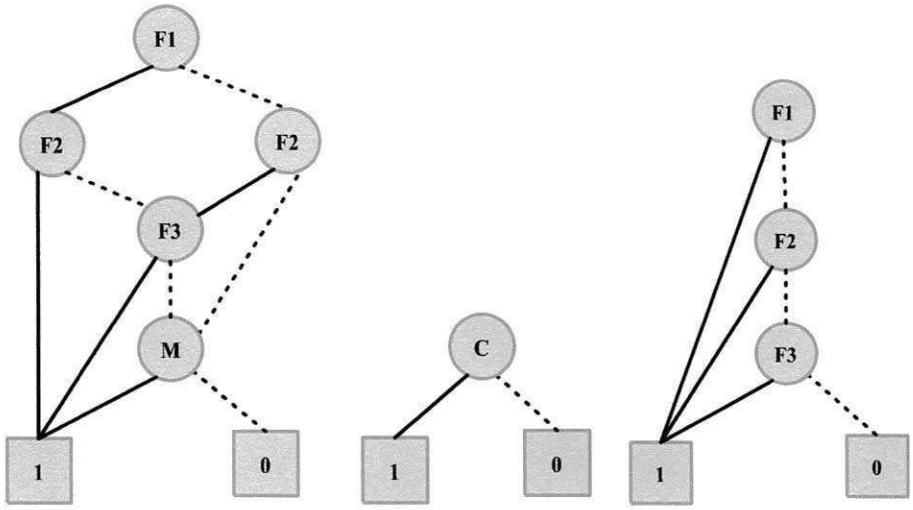


Figura 4.3: Autômato temporizado para o ciclo de varredura do sistema de prevenção de incêndio.

Para construir o autômato temporizado representado na Figura 4.3 as funções *readInputs()* e *updateOutputs()* devem ser definidas. A função *readInputs()* é gerada em quatro passos. No primeiro passo são gerados ROBDDs para cada função Booleana que determina uma saída do diagrama ISA. Na Figura 4.4 estes ROBDDs são apresentados. Observe que apenas 3 ROBDDs foram apresentados para 4 saídas. Isto acontece porque a função Booleana para a saída *Alarme* é representada por uma célula de memória. Desta forma, o resultado para a função é dado pelos valores das variáveis de saída S e R.



(a) S. (b) R. (c) Led.

Figura 4.4: ROBDDs para as saídas do sistema de prevenção de incêndio.

No segundo passo para gerar a função *readInputs()* são extraídos todos os caminhos para cada função *f*. Na Figura 4.5 são apresentados os caminhos para cada função *f*. Na Figura 4.5(c) observe que para a função Booleana $f(F1,F2,F3) = (F1 \vee F2 \vee F3)$ foram extraídos 4 caminhos: f(1,-,-) = 1, f(0,1,-) = 1, f(0,0,1) = 1 e f(0,0,0) = 0. No primeiro caminho os valores para as variáveis *F2* e *F3* não foram definidos porque independente dos seus valores (*0* ou *1*) a saída para a função Booleana seria a mesma (valor *1*).

f(F1,F2,F3,Manual)	S
f(1,1,-,-)	1
f(1,0,1,-)	1
f(1,0,0,0)	0
f(1,0,0,1)	1
f(0,1,1,-)	1
f(0,1,0,0)	0
f(0,1,0,1)	1
f(0,0,-,0)	0
f(0,0,-,1)	1

(a) S.

f(Chave)	R
f(1)	1
f(0)	0

(b) R.

f(F1,F2,F3)	Led
f(1,-,-)	1
f(0,1,-)	1
f(0,0,1)	1
f(0,0,0)	0

(c) Led.

Figura 4.5: Caminhos dos ROBDDs do sistema de prevenção de incêndio.

No terceiro passo para gerar a função *readInputs()* é criada uma tabela, conforme Tabela 4.2, em que cada coluna representa uma variável de entrada e cada linha corresponde a uma atribuição de variáveis, obtida a partir dos caminhos extraídos dos ROBDDs. Por exemplo, considere a primeira linha da tabela. Esta foi obtida a partir do primeiro caminho extraído para S, $f(1,1,-,-) = 1$. Como apenas as variáveis $F1$ e $F2$ foram atribuídas, $F1 = 1$ e $F2 = 1$, então, de acordo com o Algoritmo 2, temos que os valores para as demais variáveis serão 0. Desta forma, uma atribuição da forma $(1,1,0,0,0)$ é gerada e esta é considerada como a primeira linha da tabela.

Tabela 4.2: Tabela para as variáveis de entrada do sistema de prevenção de incêndio.

	F1	F2	F3	Manual	Chave
1	1	1	0	0	0
2	1	0	1	0	0
3	1	0	0	0	0
4	1	0	0	1	0
5	0	1	1	0	0
6	0	1	0	0	0
7	0	1	0	1	0
8	0	0	0	0	0
9	0	0	0	1	0
10	0	0	0	0	1
11	0	0	0	0	0
12	1	0	0	0	0
13	0	1	0	0	0
14	0	0	1	0	0
15	0	0	0	0	0

No quarto passo, para gerar a função *readInputs()* são definidas as atribuições para as variáveis de entrada que serão utilizadas para testar o sistema. Estas atribuições são definidas de acordo com a função f de cada saída do diagrama ISA. Observe que temos apenas funções sem temporização. Neste caso, todas as atribuições não redundantes geradas devem ser consideradas. Fazendo uso da Tabela 4.2, as seguintes atribuições para as variáveis de entrada são definidas:

- atribuições para S: linhas 1-9 da tabela;

- atribuições para R: apenas a atribuição da linha 10 da tabela deve ser selecionada, pois a atribuição definida na linha 11 já foi selecionada;
- atribuições para Led: apenas a atribuição da linha 14 da tabela deve ser selecionada, pois as demais atribuições, linhas 12, 13 e 15 já foram selecionadas.

Na Tabela 4.3 são apresentas as atribuições para as variáveis de entrada utilizadas para testar o sistema. Note que, de acordo com a Tabela 4.2, 15 atribuições de variáveis deveriam ser definidas, mas apenas 11 foram utilizadas. Isto acontece porque atribuições iguais, testes redundantes, não são consideradas. Após gerado o conjunto com todas as atribuições para as variáveis de entrada é iniciada a execução da lógica do sistema. Desta forma, na função *updateOutputs()*, descrita a seguir, cada atribuição é executada e as saídas do sistema são geradas. Cada atribuição é executada em um ciclo de varredura.

Tabela 4.3: Atribuições para as variáveis de entrada do sistema de prevenção de incêndio.

	F1	F2	F3	Manual	Chave
1	1	1	0	0	0
2	1	0	1	0	0
3	1	0	0	0	0
4	1	0	0	1	0
5	0	1	1	0	0
6	0	1	0	0	0
7	0	1	0	1	0
8	0	0	0	0	0
9	0	0	0	1	0
10	0	0	0	0	1
11	0	0	1	0	0

```
void updateOutputs(){
  S = (F1 ∧ F2) ∨ (F1 ∧ F3) ∨ (F2 ∧ F3) ∨ Manual;
  R = Chave;
  if (R == true)
      Alarme = false;
  else Alarme = S;
  Led = F1 ∨ F2 ∨ F3;
}
```

4.2.2 Definição dos Casos de Teste

Para o método proposto nesta tese, os casos de teste gerados são representados por tuplas da forma $C = (E, S, T, t)$. O conjunto E é obtido a partir da função *readInputs()*, definida na Seção 4.2.1. Desta forma, o conjunto será composto por todas as atribuições utilizadas para testar o comportamento de elementos temporizados e saídas do sistema. O conjunto S é obtido a partir dos valores gerados para as saídas do sistema, execução da função *updateOutputs()*. O conjunto T é obtido a partir da sequência de ações da execução da rede de autômatos utilizadas para gerar as saídas. O elemento t da tupla é definido de acordo com a tempo presente do temporizador, no caso de funções com temporização, ou pelo tempo de varredura, no caso de funções sem temporização.

Na Tabela 4.4 são apresentados os casos de teste para o sistema de detecção de incêndio. Estes testes foram gerados a partir dos modelos de autômatos definidos na seção anterior. O valor do tempo de varredura, representado pela variável *tScan*, é de 25 ms.

Tabela 4.4: Casos de teste para o sistema de prevenção de incêndio.

	E	S	T	t
1	{F1 = 1, F2 = 1, F3= 0, Manual = 0, Chave = 0}	S = 1, R = 0, Alarme = 1, Led = 1	{start!, update!}	tScan
2	{F1 = 1, F2 = 0, F3= 1, Manual = 0, Chave = 0}	S = 1, R = 0, Alarme = 1, Led = 1	{start!, update!}	tScan
3	{F1 = 1, F2 = 0, F3= 0, Manual = 0, Chave = 0}	S = 0, R = 0, Alarme = 0, Led = 1	{start!, update!}	tScan
4	{F1 = 1, F2 = 0, F3= 0, Manual = 1, Chave = 0}	S = 1, R = 0, Alarme = 1, Led = 1	{start!, update!}	tScan
5	{F1 = 1, F2 = 1, F3= 1, Manual = 0, Chave = 0}	S = 1, R = 0, Alarme = 1, Led = 1	{start!, update!}	tScan
6	{F1 = 0, F2 = 1, F3= 0, Manual = 0, Chave = 0}	S = 0, R = 0, Alarme = 0, Led = 1	{start!, update!}	tScan
7	{F1 = 0, F2 = 1, F3= 0, Manual = 1, Chave = 0}	S = 1, R = 0, Alarme = 1, Led = 1	{start!, update!}	tScan
8	{F1 =01, F2 = 0, F3= 0, Manual = 0, Chave = 0}	S = 1, R = 0, Alarme = 0, Led = 0	{start!, update!}	tScan
9	{F1 = 1, F2 = 0, F3= 0, Manual = 1, Chave = 0}	S = 1, R = 0, Alarme = 1, Led = 0	{start!, update!}	tScan
10	{F1 = 1, F2 = 0, F3= 0, Manual = 0, Chave = 1}	S = 0, R = 1, Alarme = 0, Led = 0	{start!, update!}	tScan
11	{F1 = 1, F2 = 0, F3= 1, Manual = 0, Chave = 0}	S = 0, R = 0, Alarme = 0, Led = 1	{start!, update!}	tScan

4.2.3 Execução dos Casos de Teste

Nesta seção são apresentados os resultados das execuções dos casos de teste no programa do sistema de prevenção de incêndio. Para tanto, os seguintes testes foram realizados:

- teste na implementação correta: a implementação do sistema executada no CLP corresponde ao programa FBD apresentado na Figura 4.2;
- teste na implementação incorreta, programa FBD com os seguintes erros (ver Figura 4.6):
  - erro 1: acrescentar uma operação *not* no programa, neste caso, a operação *not* será colocada na variável *F2*;
  - erro 2: trocar uma operação lógica *AND* por uma operação lógica *OR*.

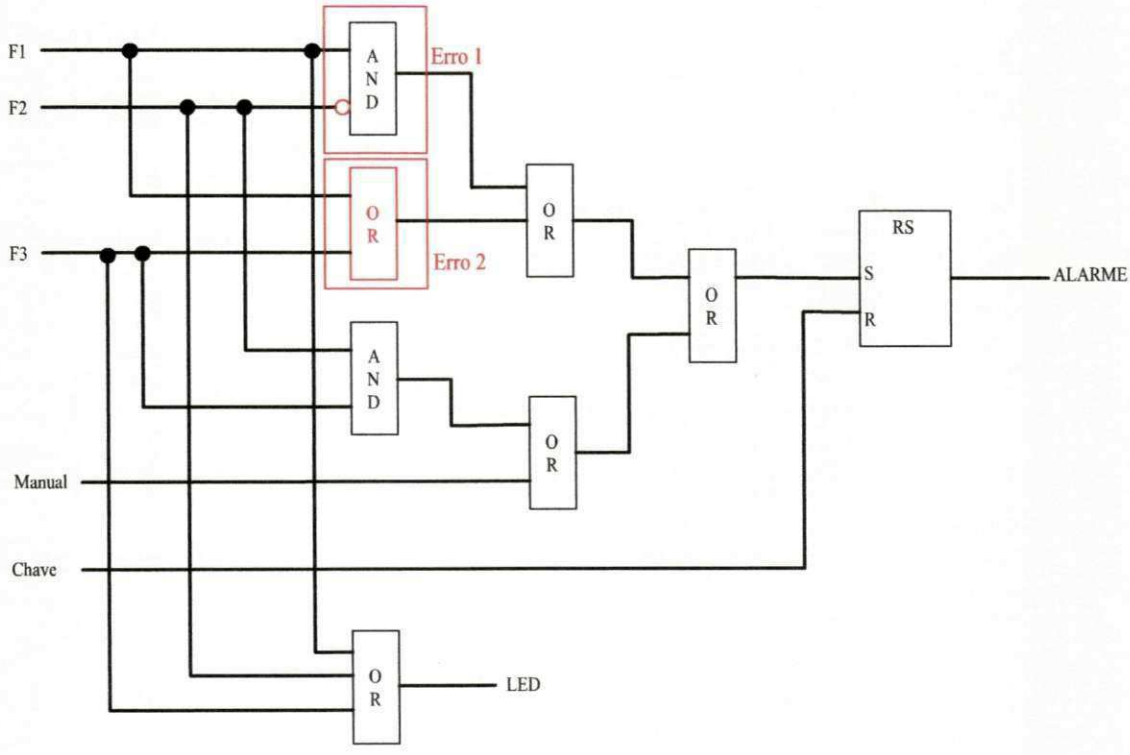


Figura 4.6: Erros no programa FBD para o sistema de prevenção de incêndio.

Para o primeiro teste, implementação correta, o veredito de teste foi: "*A implementação está em conformidade com a especificação*". Desta forma, nenhuma anormalidade relacionada ao comportamento da implementação do sistema foi detectada.

O veredito de teste para o programa FBD com o erro 1, apresentado a seguir, foi: "*Erro no caso de teste 1 - Não há conformidade entre especificação e implementação. Para a combinação de entrada F1 = 1, SF2 = 1, F3 = 0, Manual = 0, Chave = 0 era esperado que a saída Alarme fosse true (saída obtida a partir da rede de AT), mas ela foi false (saída gerada pelo CLP). Também era esperado que a a saída S fosse true, mas ela foi false*".

Caso de teste 1

E: F1 = 1, F2 = 1, F3 = 0, Manual = 0, Chave = 0

S: S = 1, R = 0, Alarme = 1, Led = 1

T: start!, update!

t = 25ms

Saídas geradas no programa do CLP

S: S = 0, R = 0, Alarme = 0, Led = 1

Timestamp = 25ms

O veredito de teste para o programa FBD com o erro 2, apresentado a seguir, foi: "*Erro no caso de teste 3 - Não há conformidade entre especificação e implementação. Para a combinação de entrada F1 = 1, SF2 = 0, F3 = 0, Manual = 0, Chave = 0 era esperado que a saída Alarme fosse false (saída obtida a partir da rede de AT), mas ela foi true (saída gerada pelo CLP). Também era esperado que a a saída S fosse false, mas ela foi true*".

Caso de teste 3

E: F1 = 1, F2 = 0, F3 = 0, Manual = 0, Chave = 0

S: S = 0, R = 0, Alarme = 0, Led = 1

T: start!, update!

t = 25ms

Saídas geradas no programa do CLP

S: S = 1, R = 0, Alarme = 1, Led = 1

Timestamp = 25ms

4.3 Sistema de Segurança para Detecção de Fogo ou Gás

O SIS utilizado neste estudo de caso é um sistema para detectar fogo ou gás em uma determinada zona. O objetivo neste estudo de caso é avaliar o método para especificações com elementos temporizados. Nas Figuras 4.7 e 4.8 são ilustradas, respectivamente, a especificação deste sistema na forma de diagramas ISA 5.2 e sua implementação na linguagem Ladder.

O SIS é composto por cinco sensores, dois de fumaça (SF1 e SF2) e três de gás (SG1, SG2 e SG3), um tanque, um dispositivo que libera gás CO_2 na confirmação de fogo na zona (DispCO2), um alarme sonoro que indica presença de fogo (AlaFDZ), um alarme sonoro que indica presença de gás (AlaGDZ) e duas válvulas, utilizadas para controlar o nível de gás no tanque. Uma destas válvulas é chamada de auxiliar, acionada quando ocorre vazamento de gás na válvula principal e, é utilizada para manter o sistema estável evitando eventuais danos às instalações da zona. As seguintes operações são realizadas no sistema:

- se algum dos sensores de fumaça for acionado então, foi detectado fogo na zona;
- se o sensor de gás 1 e o sensor de gás 2 ou 3 forem acionados então, foi detectado presença de gás na zona;
- se o sensor de gás 2 e o sensor de gás 3 forem acionados então, foi detectado presença de gás na zona;
- se for detectado fogo na zona então, um alarme sonoro que indica presença de fogo é acionado e após 2 segundos o dispositivo que libera gás CO_2 é acionado e a válvula principal é desligada. O gás CO_2 liberado é utilizado para sanar o fogo na zona. O alarme sonoro será desligado quando não houver mais presença de fogo na zona;
- se for detectado gás na zona então, o alarme sonoro que indica presença de gás é acionado e após 4 segundos a válvula que controla o nível de gás no tanque é desligada e a válvula que mantém o sistema estável é aberta. Esta válvula é utilizada para controlar a quantidade de gás no tanque de forma que não haja danificações às instalações e nem desperdício de material. O alarme sonoro será desligado quando não houver mais presença de gás na zona.

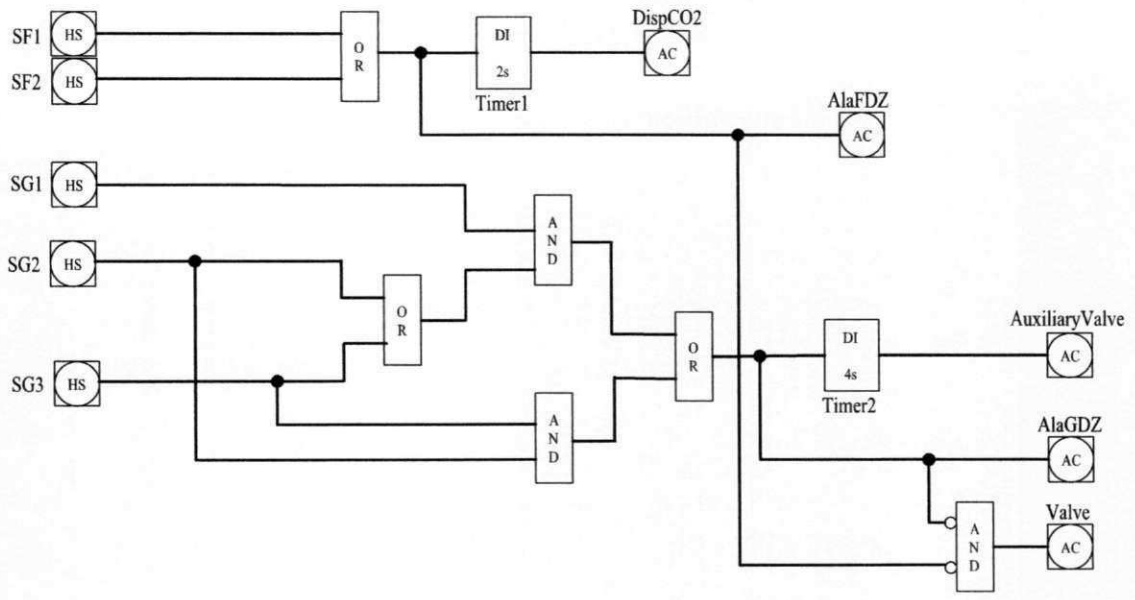


Figura 4.7: Diagrama ISA 5.2 para o sistema de detecção de fogo ou gás.

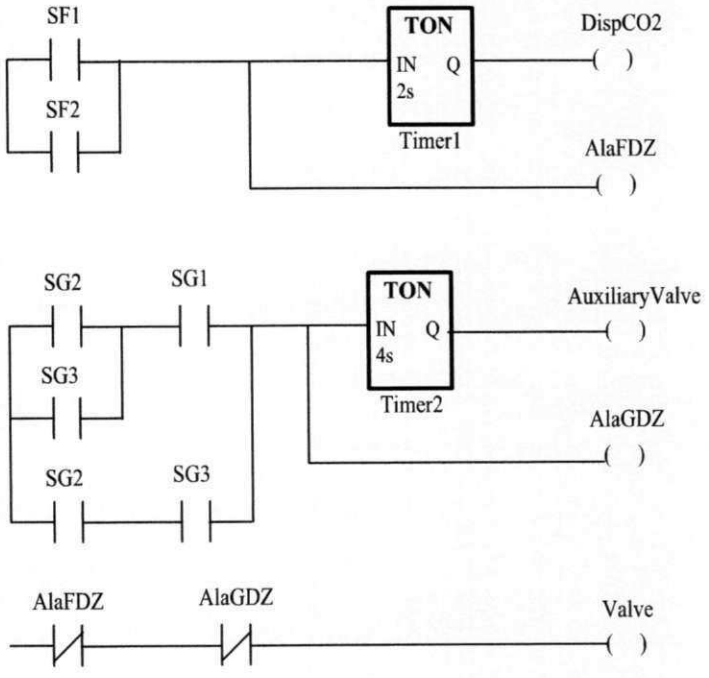


Figura 4.8: Programa Ladder para o sistema de detecção de fogo ou gás.

4.3.1 Geração dos Modelos de Autômatos Temporizados

Nas Figuras 4.9 e 4.10 são apresentados os modelos de autômatos temporizados extraídos a partir da especificação do sistema de detecção de fogo ou gás. De acordo com o método proposto, primeiro são gerados os modelos de autômatos temporizados para temporizadores. Desta forma, de acordo com a **Definição 1**, apresentada na Seção 3.2.2, o modelo de autômato temporizado para o temporizador *Timer1* é apresentado na Figura 4.9. Para este autômato temos que a entrada do temporizador (A) será representada pela função Booleana $SF1 \vee SF2$, a variável t (valor presente do temporizador) será inicializada com a constante 2 e a saída do temporizador (B) será representada pela variável *DispCO2*. Como este temporizador é o primeiro a ser modelado, então o sufixo *nTimer* das variáveis *acc_nTimer* e *control_nTimer* será representado pelo valor 1. De forma análoga, o modelo de autômato temporizado para *Timer2* pode ser gerado.

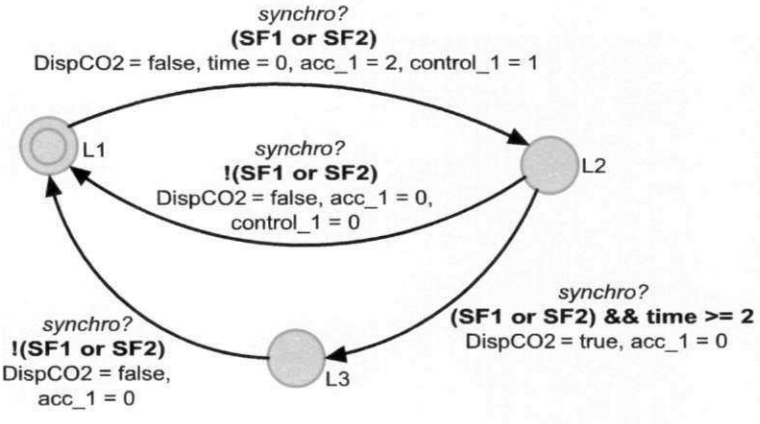


Figura 4.9: Autômato temporizado para o temporizador Timer1 do sistema de detecção de fogo ou gás.

Após definidos os autômatos para os temporizadores, o modelo de autômato que representa o ciclo de varredura do CLP é gerado. Para tanto, as funções *readInputs()* e *updateOutputs()* devem ser definidas. Para o diagrama ISA apresentado na Figura 4.7 temos:

- conjunto das entradas: $E = \{SF1, SF2, SG1, SG2, SG3\}$;
- conjunto das saídas: $S = \{s_1, s_2, s_3, s_4, s_5\}$. Cada saída é uma tupla da forma $\langle Y, f \rangle$, em que Y é o conjunto de variáveis de entrada e f é a função Booleana que define a saída;

- $s_1 = DispCO2 = \langle\{SF1, SF2\}, (SF1 \vee SF2)\rangle$;
- $s_2 = AlaFDZ = \langle\{SF1, SF2\}, (SF1 \vee SF2)\rangle$;
- $s_3 = AuxiliaryValve = \langle\{SG1, SG2, SG3\}, ((SG1 \wedge (SG2 \vee SG3)) \vee (SG2 \wedge SG3))\rangle$;
- $s_4 = AlaGDZ = \langle\{SG1, SG2, SG3\}, ((SG1 \wedge (SG2 \vee SG3)) \vee (SG2 \wedge SG3))\rangle$;
- $s_5 = Valve = \langle\{SF1, SF2, SG1, SG2, SG3\}, (\sim(SF1 \vee SF2) \wedge \sim((SG1 \wedge (SG2 \vee SG3)) \vee (SG2 \wedge SG3)))\rangle$.

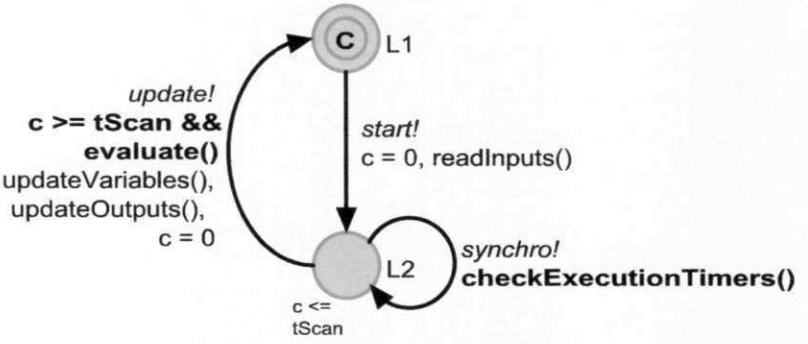


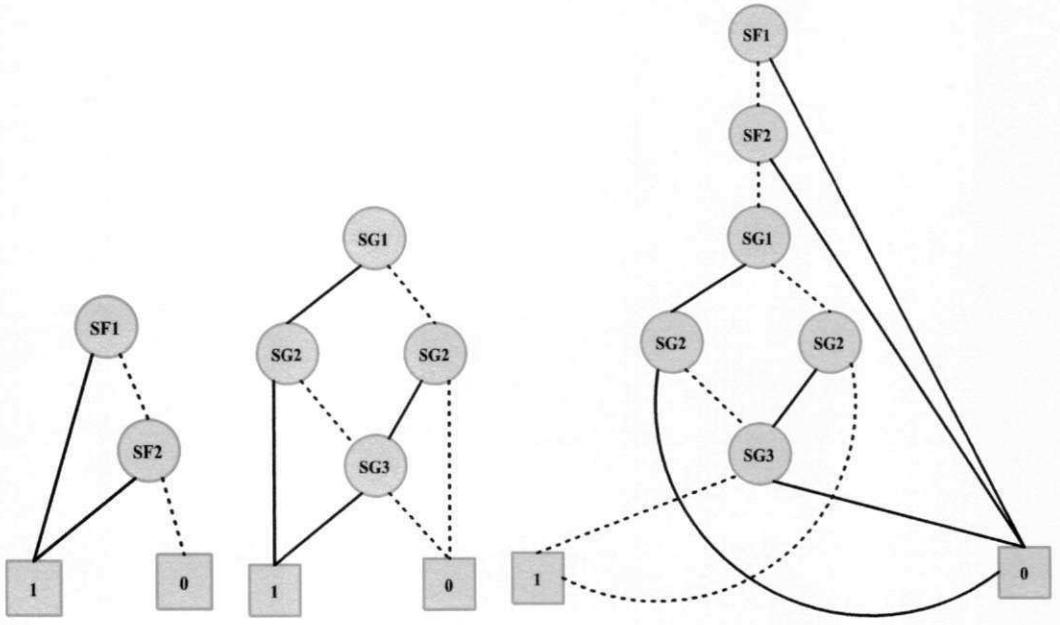
Figura 4.10: Autômato temporizado para o ciclo de varredura do sistema de detecção de fogo ou gás.

A função *readInputs()* é gerada em quatro passos. No primeiro passo são gerados ROBDDs para cada função Booleana que determina uma saída do diagrama ISA. Na Figura 4.11 estes ROBDDs são apresentados. Observe que apenas 3 ROBDDs foram gerados para 5 saídas. Isto acontece porque a função Booleana para as saídas *DispCO2* e *AlaFDZ* é a mesma. O mesmo ocorre com a função para as saídas *AuxiliaryValve* e *AlaGDZ*.

No segundo passo para gerar a função *readInputs()* são extraídos todos os caminhos para cada função *f*. Na Figura 4.12 são apresentados os caminhos para cada função *f*. Na Figura 4.12(a) observe que para a função Booleana *f(SF1,SF2) = SF1* $\vee$ *SF2* foram extraídos 3 caminhos: f(1,-) = 1, f(0,1) = 1 e f(0,0) = 0. No primeiro caminho o valor para a variável *SF2* não foi definido porque independente do seu valor (*0* ou *1*) a saída para a função Booleana seria a mesma (valor *1*).

No terceiro passo para gerar a função *readInputs()* é criada uma tabela, conforme Tabela 4.5, em que cada coluna representa uma variável de entrada e cada linha corresponde a uma

atribuição de variáveis, obtida a partir dos caminhos extraídos dos ROBDDs. Por exemplo, considere a primeira linha da tabela. Esta foi obtida a partir do primeiro caminho extraído para a saída DispCO2, *f(1,-) = 1*. Como apenas a variável *SF1* foi atribuída (*SF1 = 1*) então, de acordo com Algoritmo 2, temos que os valores para as demais variáveis serão *0*. Desta forma, uma atribuição da forma *(1,0,0,0,0)* é gerada e esta é considerada como a primeira linha da tabela.



(a) DispCO2 e AlaFDZ (b) AuxiliaryValve e AlaGDZ (c) Valve

Figura 4.11: ROBDDs para as saídas do sistema de detecção de fogo ou gás.

f(SF1,SF2)	DispCO2 / AlaFDZ
f(1,-)	1
f(0,1)	1
f(0,0)	0

f(SG1,SG2,SG3)	AuxiliaryValve / AlaGDZ
f(1,1,-)	1
f(1,0,1)	1
f(1,0,0)	0
f(0,0,-)	0
f(0,1,0)	0
f(0,1,1)	1

f(SF1, SF2, SG1,SG2,SG3)	Valve
f(1,-,-,-,-)	0
f(0,1,-,-,-)	0
f(0,0,1,1,-)	0
f(0,0,1,0,1)	0
f(0,0,1,0,0)	1
f(0,0,0,0,-)	1
f(0,0,0,1,0)	1
f(0,0,0,1,1)	0

(a) DispCO2 e AlaFDZ. (b) AuxiliaryValve e AlaGDZ. (c) Valve.

Figura 4.12: Caminhos para os ROBDDs do sistema de detecção de fogo ou gás.

Tabela 4.5: Tabela para as variáveis de entrada do sistema de detecção de fogo ou gás.

	SF1	SF2	SG1	SG2	SG3
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	0	0	0
4	0	0	1	1	0
5	0	0	1	0	1
6	0	0	1	0	0
7	0	0	0	0	0
8	0	0	0	1	0
9	0	0	0	1	1
10	1	0	0	0	0
11	0	1	0	0	0
12	0	0	1	1	0
13	0	0	1	0	1
14	0	0	1	0	0
15	0	0	0	0	0
16	0	0	0	1	0
17	0	0	0	1	1

No quarto passo, para gerar a função *readInputs()* são definidas as atribuições para as variáveis de entrada que serão utilizadas para testar o sistema. Estas atribuições são definidas de acordo com a função f de cada saída do diagrama ISA. Observe que temos dois tipos de funções: as que não possuem elementos temporizados, como é o caso da função para a saída *AlaFDZ*, e as que possuem elementos temporizados, como é o caso da função para a saída *DispCO2*. Para funções que não possuem elementos temporizados todas as atribuições devem ser consideradas. No caso de funções que possuem elementos temporizados 4 combinações, uma para ativar cada arco do autômato que representa o temporizador, são consideradas. Fazendo uso da Tabela 4.5, as seguintes atribuições para as variáveis de entrada são definidas:

- atribuições para *DispCO2*:
  - linha 1 da tabela - Ativa a aresta 1 do temporizador Timer1;
  - linha 3 da tabela - Ativa a aresta 2 do temporizador Timer1;
  - linha 2 da tabela - Ativa as aresta 1 e 3 do temporizador Timer1;

  - linha 3 da tabela - Ativa a aresta 4 do temporizador Timer1;

- atribuições para *AlaFDZ*: todas as atribuições definidas para esta saída, linhas 1-3 já foram selecionadas. Desta forma, nenhuma nova atribuição é selecionada;

- atribuições para *AuxiliaryValve*:

  - linha 4 da tabela - Ativa a aresta 1 do temporizador Timer2;

  - linha 6 da tabela - Ativa a aresta 2 do temporizador Timer2;

  - linha 5 da tabela - Ativa as aresta 1 e 3 do temporizador Timer2;

  - linha 8 da tabela - Ativa a aresta 4 do temporizador Timer2;

- atribuições para *AlaGDZ*: apenas a atribuição da linha 9 da tabela deve ser selecionada, pois as demais atribuições, linhas 4-8 já foram selecionadas;

- atribuições para *Valve*: todas as atribuições definidas para esta saída, linhas 10-17 já foram selecionadas. Desta forma, nenhuma nova atribuição é selecionada.

Na Tabela 4.6 são apresentadas as atribuições para as variáveis de entrada utilizadas para testar o sistema. Note que, de acordo com a Tabela 4.5, 17 atribuições de variáveis deveriam ser definidas, mas apenas 9 foram utilizadas. Isto acontece porque atribuições iguais, testes redundantes, não são consideradas. Após gerado o conjunto com todas as atribuições para as variáveis de entrada é iniciada a execução da lógica do sistema. Desta forma, na função *updateOutputs()*, descrita a seguir, cada atribuição é executada e as saídas do sistema são geradas. Cada atribuição é executada em um ciclo de varredura. Os valores para as variáveis *DispCO2* e *AuxiliaryValve* foram definidos nos modelos de autômatos para temporizadores pois estas variáveis representam as saídas dos temporizadores.

```
void updateOutputs(){
  AlaFDZ = SF1 ∨ SF2;
  AlaGDZ = (SG1 ∧ (SG2 ∨ SG3)) ∨ (SG2 ∧ SG3);
  Valve = (SF1 ∨ SF2) ∧ ~((SG1 ∧ (SG2 ∨ SG3)) ∨ (SG2 ∧ SG3));
}
```

Tabela 4.6: Atribuições definidas para as variáveis de entrada do sistema de detecção de fogo ou gás.

	SF1	SF2	SG1	SG2	SG3
1	1	0	0	0	0
2	0	0	0	0	0
3	0	1	0	0	0
4	0	0	0	0	0
5	0	0	1	1	0
6	0	0	1	0	0
7	0	0	1	0	1
8	0	0	0	1	0
9	0	0	0	1	1

4.3.2 Definição dos Casos de Teste

Na Tabela 4.7 são apresentados os casos de teste para o sistema de detecção de fogo e gás. Estes testes foram gerados a partir dos modelos de autômatos definidos na seção anterior. O valor do tempo de varredura, representado pela variável *tScan*, é de 25 ms.

Tabela 4.7: Casos de teste para o sistema de detecção de fogo ou gás.

	E	S	T	t
1	{SF1 = 1, SF2 = 0, SG1 = 0, SG2 = 0, SG3 = 0}	{DispCO2 = 0, AlaFDZ = 1, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 0}	{start!,synchro!, update!}	tScan
2	{SF1 = 0, SF2 = 0, SG1 = 0, SG2 = 0, SG3 = 0}	{DispCO2 = 0, AlaFDZ = 0, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 1}	{start!,synchro!, update!}	tScan
3	{SF1 = 0, SF2 = 1, SG1 = 0, SG2 = 0, SG3 = 0}	{DispCO2 = 1, AlaFDZ = 1, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 0}	start!,synchro!, update!, start!, ..., update!}	2s
4	{SF1 = 0, SF2 = 0, SG1 = 0, SG2 = 0, SG3 = 0}	{DispCO2 = 0, AlaFDZ = 0, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 1}	{start!,synchro!, update!}	tScan
5	{SF1 = 0, SF2 = 0, SG1 = 1, SG2 = 1, SG3 = 0}	{DispCO2 = 0, AlaFDZ = 0, AuxiliaryValve = 0, AlaGDZ = 1, Valve = 1}	{start!,synchro!, update!}	tScan
6	{SF1 = 0, SF2 = 0, SG1 = 1, SG2 = 0, SG3 = 0}	{DispCO2 = 0, AlaFDZ = 0, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 1}	{start!,synchro!, update!}	tScan
7	{SF1 = 0, SF2 = 0, SG1 = 1, SG2 = 0, SG3 = 1}	{DispCO2 = 0, AlaFDZ = 0, AuxiliaryValve = 1, AlaGDZ = 1, Valve = 0}	{start!,synchro!, update!, start!, ..., update!}	4s
8	{SF1 = 0, SF2 = 0, SG1 = 0, SG2 = 1, SG3 = 0}	{DispCO2 = 0, AlaFDZ = 0, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 1}	{start!,synchro!, update!}	tScan
9	{SF1 = 0, SF2 = 0, SG1 = 0, SG2 = 1, SG3 = 1}	{DispCO2 = 0, AlaFDZ = 0, AuxiliaryValve = 0, AlaGDZ = 1, Valve = 1}	{start!,synchro!, update!}	tScan

4.3.3 Execução dos Casos de Teste

Nesta seção são apresentados os resultados das execuções dos casos de teste no programa do sistema de detecção de fogo ou gás. Para tanto, os seguintes testes foram realizados:

- teste na implementação correta: a implementação do sistema executada no CLP corresponde ao programa Ladder apresentado na Figura 4.8;
- teste na implementação incorreta, programa Ladder com os seguintes erros (ver Figura 4.13):
  - erro 1: trocar um contato normalmente aberto por um contato normalmente fechado, neste caso, a variável *SF2* será representada por um contato normalmente fechado;
  - erro 2: trocar uma operação lógica *AND* por uma operação lógica *OR*, no caso, para o terceiro degrau, a variável *AlaGDZ* representada por um contato normalmente fechado será colocada em paralelo com a variável representada pelo contato normalmente fechado *AlaFDZ*;
  - erro 3: alterar o valor de *PT* de *Timer1* para 1 segundo.

Para o primeiro teste, implementação correta, o veredito de teste foi: "*A Implementação está em conformidade com a especificação*". Desta forma, nenhuma anormalidade relacionada ao comportamento da implementação do sistema foi detectada.

O veredito de teste para o programa Ladder com o erro 1 foi: "*Erro no caso de teste 2 - Não há conformidade entre especificação e implementação. Para a combinação de entrada SF1 = 0, SF2 = 0, SG1 = 0, SG2 = 0, SG3 = 0 era esperado que a saída AlaFDZ fosse false (saída obtida a partir da rede de AT), mas ela foi true (saída gerada pelo CLP). Também era esperado que a a saída Valve fosse true, mas ela foi false*". A seguir é apresentado como este veredito foi obtido.

Caso de teste 2

E: SF1 = 0, SF2 = 0, SG1 = 0, SG2 = 0, SG3 = 0

S: DispCO2 = 0, AlaFDZ = 0, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 1

T: start!, synchro!, update!

t = 25ms

Saídas geradas no programa do CLP

DispCO2 = 0, AlaFDZ = 1, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 0

Timestamp = 25ms

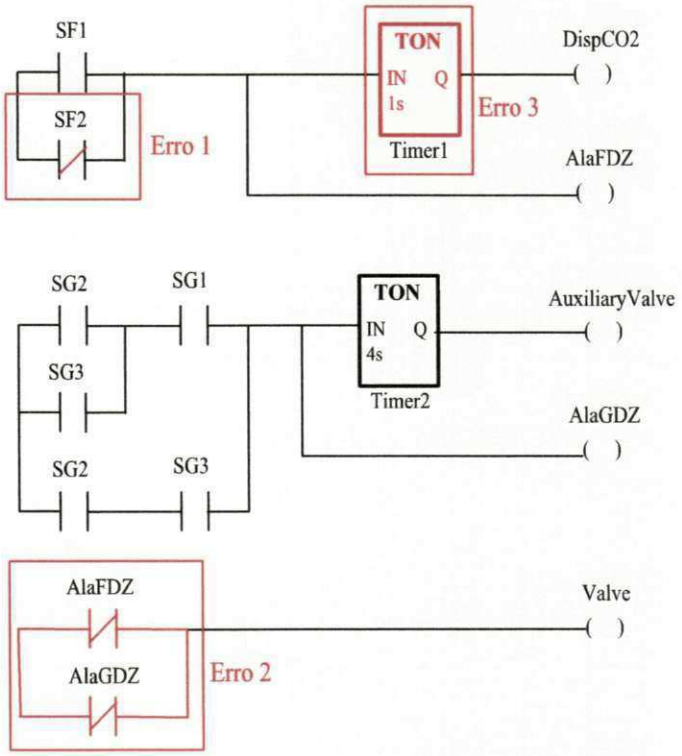


Figura 4.13: Erros no programa Ladder para o sistema de detecção de fogo e gás.

O veredito de teste para o programa Ladder com o erro 2 foi: "*Erro no caso de teste 1 - Não há conformidade entre especificação e implementação. Para a combinação de entrada SF1 = 1, SF2 = 0, SG1 = 0, SG2 = 0, SG3 = 0 era esperado que a a saída Valve fosse false (saída obtida a partir da rede de AT), mas ela foi true (saída gerada pelo CLP)*". A seguir é apresentado como este veredito foi obtido.

Caso de teste 1

E: SF1 = 1, SF2 = 0, SG1 = 0, SG2 = 0, SG3 = 0

S: DispCO2 = 0, AlaFDZ = 1, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 0

T: start!, synchro!, update!

t = 25ms

Saídas geradas no programa do CLP

DispCO2 = 0, AlaFDZ = 1, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 1

Timestamp = 25ms

O veredito de teste para o programa Ladder com o erro 3 foi: "*Erro no caso de teste 3 - Não há conformidade entre especificação e implementação. Para a combinação de entrada SF1 = 0, SF2 = 1, SG1 = 0, SG2 = 0, SG3 = 0 era esperado que o timestamp fosse 2 segundos, mas ele foi 1 segundo*". Este veredito informa que as saídas do programa foram geradas após 1 segundo e não após 2 segundos como descrito na especificação do sistema.

Caso de teste 3

E: SF1 = 0, SF2 = 1, SG1 = 0, SG2 = 0, SG3 = 0

S: DispCO2 = 1, AlaFDZ = 1, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 0

T: start!, synchro!, update!, start!, synchro!, update!

t = 2 segundos

Saídas geradas pelo CLP

DispCO2 = 1, AlaFDZ = 1, AuxiliaryValve = 0, AlaGDZ = 0, Valve = 0

Timestamp = 1 segundo

4.4 Sistema de Controle de Nível de um Tanque

Neste estudo de caso, o controle do nível de um tanque é realizado através do acionamento de um sistema de drenagem composto por uma válvula de escape e uma bomba de sucção. O objetivo com este estudo de caso é avaliar o método para especificações com dependência

de elementos temporizados. Nas Figuras 4.14 e 4.15, são apresentados, respectivamente, o diagrama ISA 5.2 e o programa em FBD para este sistema.

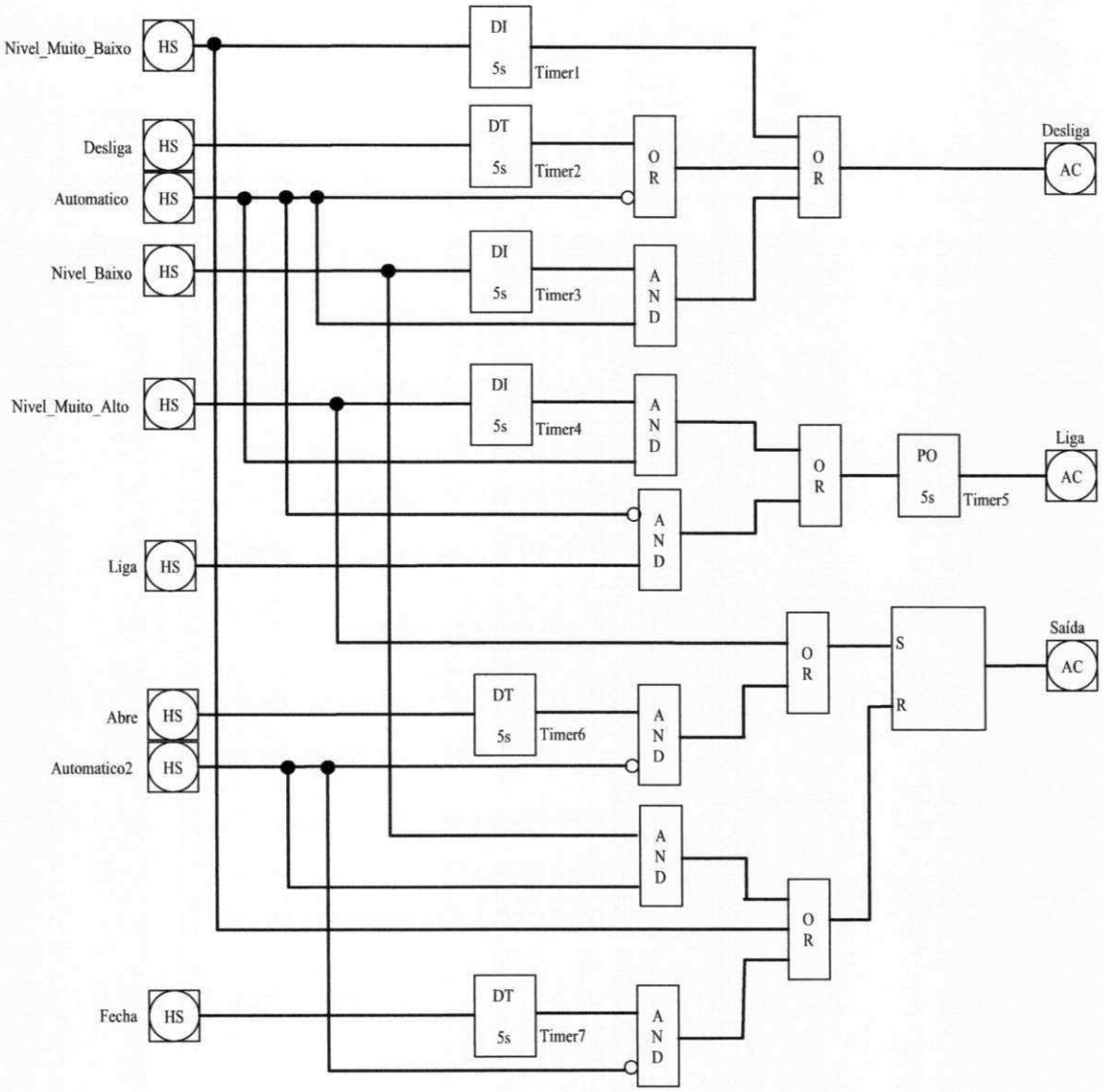


Figura 4.14: Diagrama ISA 5.2 para o sistema de controle de nível de um tanque.

A lógica utilizada para determinar o acionamento do sistema de drenagem é apresentada a seguir, o sistema pode funcionar de quatro maneiras:

- modo 1: completamente manual;
- modo 2: completamente automático;
- modo 3: bomba manual e válvula automática;

- modo 4: bomba automática e válvula manual.

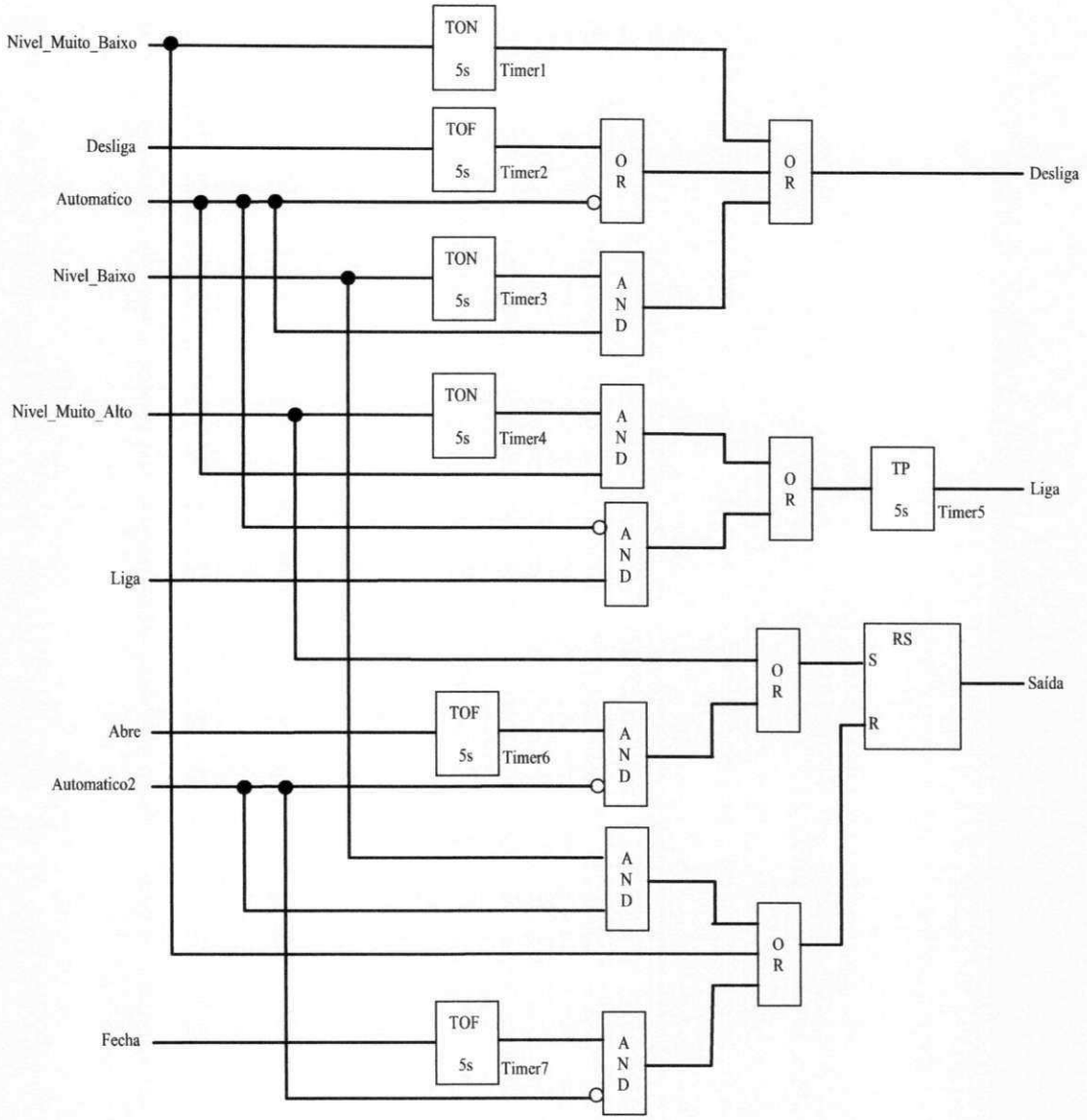


Figura 4.15: Programa FBD para o sistema de controle de nível de um tanque.

O modo de funcionamento desejado é escolhido pela combinação dos estados de duas chaves, *Automatico* e *Automatico2*, que são utilizadas para selecionar, respectivamente, o modo automático ou manual para o acionamento da bomba e da válvula. Com a planta funcionando no modo 1, completamente manual, tanto a válvula quanto a bomba são comandadas independentemente pelo operador. Isto é feito através das chaves *Abre* e *Fecha*, para a válvula, e *Liga* e *Desliga* para a bomba. No modo 2, completamente automático, o estado dos sensores de nível determinam as ações que serão realizadas. Os sinais dos sensores

são *Nivel_Muito_Baixo*, *Nivel_Baixo* e *Nivel_Muito_Alto*. Nos modos em que a bomba ou a válvula estão no modo manual as ações realizadas são função da combinação do estado dos sensores e dos comandos do operador. Existem também ações de emergência que são tomadas independentemente do modo selecionado e que são combinadas com as ações de qualquer modo escolhido para determinar o estado final do sistema. A seguir, uma descrição das possíveis combinações para os sinais de entrada e as correspondentes ações que devem ser efetuadas pelo sistema são apresentadas.

Independente do modo de funcionamento selecionado, se *Nivel_Muito_Baixo* apresentar o valor verdadeiro por 5s consecutivos a ação tomada pelo SIS deve ser, desligar a bomba e fechar a válvula. Se *Nivel_Muito_Alto* apresentar o valor verdadeiro por 5s consecutivos, abrir a válvula. Na descrição dos modos seguintes, os estados apresentados serão combinações entre as ações dos respectivos modos e das ações de emergência. Para diferenciar as ações de emergência elas serão apresentadas em negrito.

No Modo 1, completamente manual, o comportamento dos dois dispositivos é determinado pelo operador. Porém, se verificado alguma das combinações de sinais consideradas críticas, o sistema executará as ações de emergência programadas.

No Modo 2, completamente automático, se *Nivel_Baixo* apresentar o valor verdadeiro por 5s consecutivos, desligar a bomba e fechar a válvula. Se *Nivel_Muito_Alto* apresentar o valor verdadeiro por 5s consecutivos, ligar a bomba e **abrir a válvula**.

No Modo 3, bomba manual e válvula automática, se *Nivel_Muito_Baixo* for verdadeiro por 5s consecutivos, **desligar a bomba e fechar a válvula**. Se *Nivel_Baixo* for verdadeiro por 5s consecutivos, a ação do sistema será fechar a válvula e o estado da bomba será determinado pelo operador. Se *Nivel_Muito_Alto* for verdadeiro por 5s consecutivos, o estado da bomba será determinado pelo operador e a ação do sistema será **abrir a válvula**.

No Modo 4, bomba automática e válvula manual, se *Nivel_Muito_Baixo* for verdadeiro por 5s consecutivos, **desligar a bomba e fechar a válvula**. Se *Nivel_Baixo* for verdadeiro por 5s consecutivos, a ação do sistema será desligar a bomba e o estado da válvula será determinado pelo operador. Se *Nivel_Muito_Baixo* apresentar o valor verdadeiro por 5s consecutivos, ligar a bomba e abrir a válvula.

4.4.1 Geração dos Modelos de Autômatos Temporizados

Nas Figuras 4.16 a 4.19 são apresentados os modelos de autômatos temporizados extraídos a partir da especificação do sistema de controle de nível de um tanque. De acordo com o método proposto, primeiro são gerados os modelos de autômatos temporizados para temporizadores. Desta forma, de acordo com a **Definição 1**, apresentada na Seção 3.2.2, o modelo de autômato temporizado para o temporizador *Timer1* é apresentado na Figura 4.16. Para este autômato temos que a entrada do temporizador (*A*) será representada pela variável *Nivel_Muito_Baixo*, a variável *t* (valor presente do temporizador) será inicializada com a constante *5* e a saída do temporizador (*B*) será representada pela variável *out_Timer1*, uma vez que a mesma será utilizada para gerar a saída *Desliga*. Como este temporizador é o primeiro a ser modelado, então o sufixo *nTimer* das variáveis *acc_nTimer* e *control_nTimer* será representado pelo valor 1. De forma análoga, os modelos de autômatos temporizados para os temporizadores *Timer3* e *Timer4* podem ser gerados.

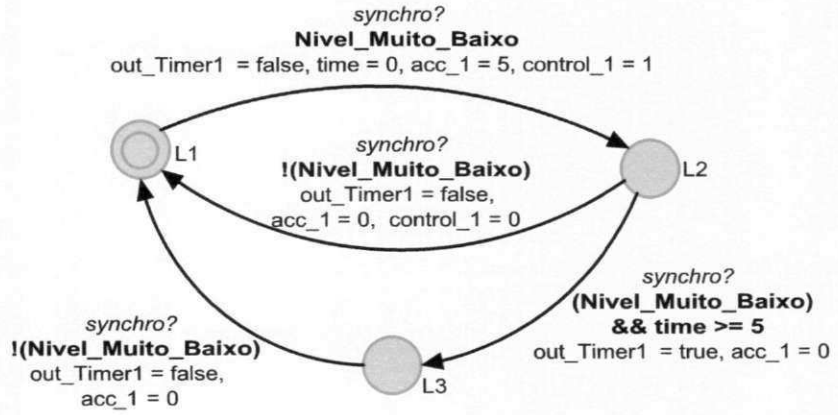


Figura 4.16: Autômato temporizado para o temporizador Timer1 do sistema de controle de nível de um tanque.

De acordo com a **Definição 2**, apresentada na Seção 3.2.2, o modelo de autômato temporizado para o temporizador *Timer2* é apresentado na Figura 4.17. Para este autômato temos que a entrada do temporizador (*A*) será representada pela variável *Desliga*, a variável *t* (valor presente do temporizador) será inicializada com a constante *5* e a saída do temporizador (*B*) será representada pela variável *out_Timer2*. Como este temporizador é o segundo a ser modelado, então o sufixo *nTimer* das variáveis *acc_nTimer* e *control_nTimer* será representado

pelo valor 2. De forma análoga, os modelos de autômatos temporizados para os temporizadores *Timer3* e *Timer4* podem ser gerados.

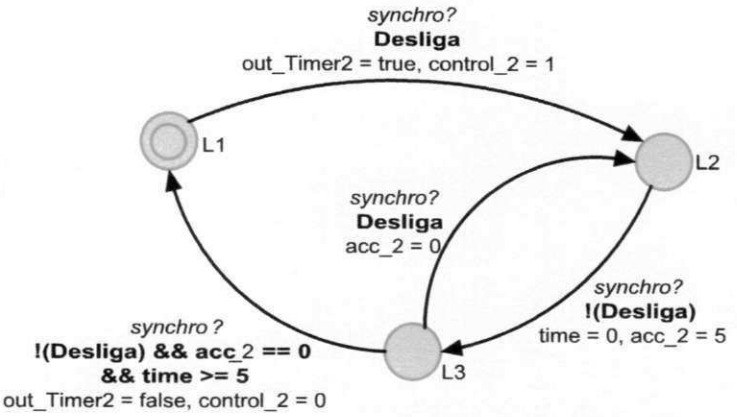


Figura 4.17: Autômato temporizado para o temporizador Timer2 do sistema de controle de nível de um tanque.

De acordo com a **Definição 3**, apresentada na Seção 3.2.2, o modelo de autômato temporizado para o temporizador *Timer5* é apresentado na Figura 4.18. Para este autômato temos que a entrada do temporizador (A) será representada pela função Booleana $((out\_Timer4 \wedge Automatic) \vee (\sim Automatic \wedge Liga))$, a variável t (valor presente do temporizador) será inicializada com a constante 5 e a saída do temporizador (B) será representada pela variável *Liga*. Como este temporizador é o quinto a ser modelado, então o sufixo *nTimer* das variáveis *acc_nTimer* e *control_nTimer* será representado pelo valor 5.

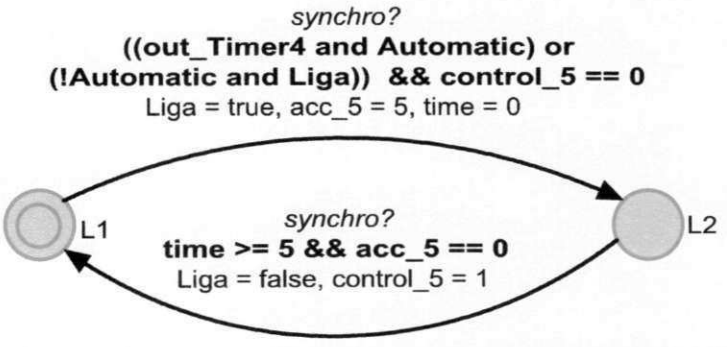


Figura 4.18: Autômato temporizado para o temporizador Timer5 do sistema de controle de nível de um tanque.

Após definidos os autômatos para os temporizadores, o modelo de autômato que representa o ciclo de varredura do CLP é gerado. Para tanto, as funções *readInputs()* e

updateOutputs() devem ser definidas. Para o diagrama ISA apresentado na Figura 4.14 temos:

- conjunto das entradas: E = *{Nivel_Muito_Baixo, Desliga, Automatico, Nivel_Baixo, Nivel_Muito_Alto, Liga, Abre, Automatico2, Fecha}*;
- conjunto das saídas: $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$. Cada saída é uma tupla da forma $\langle Y, f \rangle$, em que Y é o conjunto de variáveis de entrada e f é a função Booleana que define a saída.
  - s_1 = *In_Timer1* = $\langle$*{Nivel_Muito_Baixo}*, *(Nivel_Muito_Baixo)*$\rangle$;
  - s_2 = *In_Timer2* = $\langle$*{Desliga}*, *(Desliga)*$\rangle$;
  - s_3 = *In_Timer3* = $\langle$*{Nivel_Baixo}*, *(Nivel_Baixo)*$\rangle$;
  - s_4 = *Desliga* = $\langle$*{Nivel_Muito_Baixo, Desliga, Automatico, Nivel_Baixo}*, *((out_Timer1) ∨ (out_Timer2 ∨ ∼Automatico) ∨ (out_Timer3 ∧ Automatico))*$\rangle$;
  - s_5 = *In_Timer4* = $\langle$*{Nivel_Muito_Alto}*, *(Nivel_Muito_Alto)*$\rangle$;
  - s_6 = *In_Timer5* = $\langle$*{Nivel_Muito_Alto, Automatico, Liga}*, *(out_Timer5)*$\rangle$;
  - s_7 = *In_Timer6* = $\langle$*{Abre}*, *(Abre)*$\rangle$;
  - s_8 = *In_Timer7* = $\langle$*{Fecha}*, *(Fecha)*$\rangle$;
  - s_9 = S = $\langle$*{Nivel_Muito_Alto, Abre, Automatico2}*, *((Nivel_Muito_Alto) or (out_Timer6 and !Automatico2))*$\rangle$;
  - s_{10} = R = $\langle$*{Nivel_Baixo, Automatico2, Nivel_Muito_Baixo, Fecha}*, *((Nivel_Baixo ∧ Automatico2) ∨ (out_Timer7 ∧ ∼ Automatico2) ∨ (Nivel_Muito_Baixo))*$\rangle$.

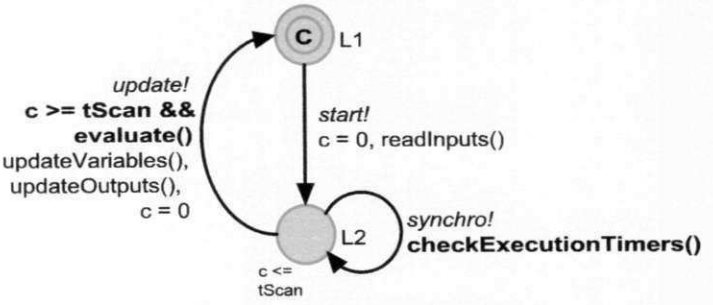
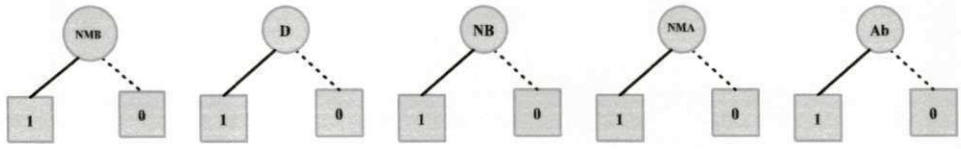
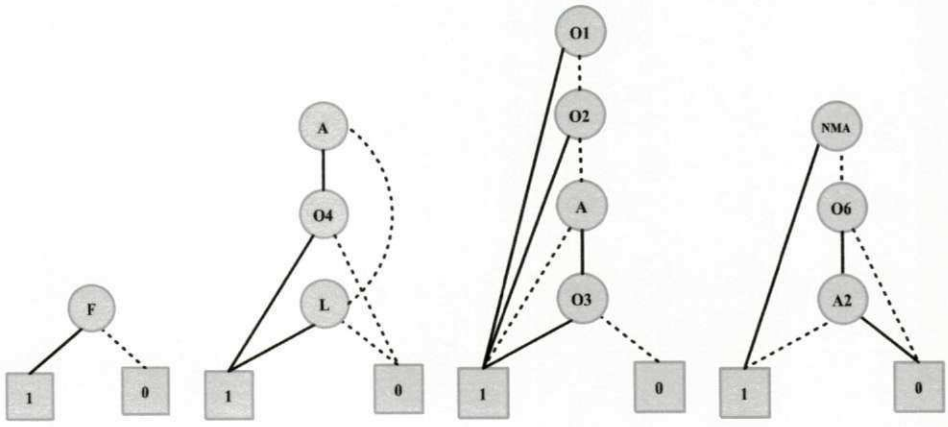


Figura 4.19: Autômato temporizado para o ciclo de varredura do sistema de controle de nível de um tanque.

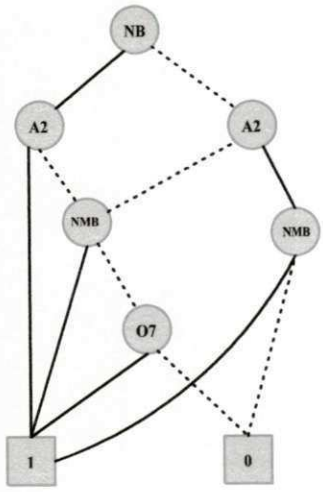
A função *readInputs()* é gerada em quatro passos. No primeiro passo, ROBDDs são gerados para cada função Booleana que determina uma saída do diagrama ISA. Na Figura 4.20 estes ROBDDs são apresentados. As variáveis *In_NomeTimer* e *out_NomeTimer* representam, respectivamente, a entrada e a saída de um temporizador.



(a) In_Timer1. (b) In_Timer2. (c) In_Timer3. (d) In_Timer4. (e) In_Timer6.



(f) In_Timer7. (g) In_Timer5. (h) Desliga. (i) S.



NMB = Nivel_Muito_Baixo, D = Desliga, NB = Nivel_Baixo,
O1 = out_Timer1, O2 = out_Timer2, O3 = out_Timer3,
O4 = out_Timer4, O6 = out_Timer6, O7 = out_Timer7,
A = Automatico, NMA = Nivel_Muito_Alto, Ab = Abre,
F = Fecha, L = Liga, A2 = Automatico2

(j) R.

Figura 4.20: ROBDDs para as saídas do sistema de controle de nível de um tanque.

No segundo passo para gerar a função *readInputs()* são extraídos todos os caminhos para cada função *f*. Na Figura 4.21 são apresentados os caminhos para cada função *f*. Na Figura 4.21(g) observe que para a função Booleana $f(A,O4,L) = (A \wedge O4) \vee (\sim A \wedge L)$ foram extraídos 4 caminhos: f(1,1,-) = 1, f(1,0,-) = 0, f(0,-,0) = 1 e f(0,-,1) = 0. No primeiro caminho os valores para as variáveis *O4* e *L* não foram definidos porque independente dos seus valores (*0* ou *1*) a saída para a função Booleana seria a mesma (valor 1).

f(NMB)	In_Timer1
f(1)	1
f(0)	0

(a) In_Timer1.

f(D)	In_Timer2
f(1)	1
f(0)	0

(b) In_Timer2.

f(NB)	In_Timer3
f(1)	1
f(0)	0

(c) In_Timer3.

f(NMA)	In_Timer4
f(1)	1
f(0)	0

(d) In_Timer4.

f(Ab)	In_Timer6
f(1)	1
f(0)	0

(e) In_Timer6.

f(F)	In_Timer7
f(1)	1
f(0)	0

(f) In_Timer7.

f(A,O4,L)	In_Timer5
f(1,1,-)	1
f(1,0,-)	0
f(0,-,0)	0
f(0,-,1)	1

(g) In_Timer5.

f(O1,O2,A,O3)	Desliga
f(1,-,-,-)	1
f(0,1,-,-)	1
f(0,0,0,-)	1
f(0,0,1,1)	1
f(0,0,1,0)	0

(h) Desliga.

f(NMA,O6,A2)	S
f(1,-,-)	1
f(0,0,-)	0
f(0,1,0)	1
f(0,1,1)	0

(i) S.

f(NB,A2,NMB,O7)	R
f(1,1,-,-)	1
f(1,0,1,-)	1
f(1,0,0,1)	1
f(1,0,0,0)	0
f(0,0,1,-)	1
f(0,0,0,0)	0
f(0,0,0,1)	1
f(0,1,0,-)	0
f(0,1,1,-)	1

(j) R.

Figura 4.21: Caminhos para os ROBDDs do sistema de controle de nível de um tanque.

No terceiro passo para gerar a função *readInputs()* é criada uma tabela, conforme Tabela 4.8, em que cada coluna representa uma variável de entrada e cada linha corresponde a uma atribuição de variáveis, obtida a partir dos caminhos extraídos dos ROBDDs. Para uma melhor visualização da tabela os valores não definidos para as variáveis de entrada não foram preenchidos com o valor *0*, conforme Algoritmo 2.

No quarto passo, para gerar a função *readInputs()* são definidas as atribuições para as variáveis de entrada que serão utilizadas para testar o sistema. Estas atribuições são definidas de acordo com a função f de cada saída do diagrama ISA. Observe que temos dois tipos de funções: as que não possuem elementos temporizados, como é o caso da função para a saída *S*, e as que possuem elementos temporizados, como é o caso da função para a saída *out_Timer1*. Para funções que não possuem elementos temporizados todas as atribuições devem ser consideradas. No caso de funções com temporização, as atribuições serão selecionadas de acordo com cada elemento temporizado que compõe a lógica de cada saída, conforme Algoritmo 3. Fazendo uso da Tabela 4.8, as seguintes atribuições para as variáveis de entrada são definidas:

1. Atribuições para testar o comportamento do temporizador *Timer1*:
   (a) Linha 1 da tabela - Ativa a aresta 1 do temporizador *Timer1*;
   (b) Linha 2 da tabela - Ativa a aresta 2 do temporizador *Timer1*;
   (c) Linha 1 da tabela - Ativa as aresta 1 e 3 do temporizador *Timer1*;
   (d) Linha 2 da tabela - Ativa a aresta 4 do temporizador *Timer1*;
2. Atribuições para testar o comportamento do temporizador *Timer2*:
   (a) Linha 3 da tabela - Ativa a aresta 1 do temporizador *Timer2*;
   (b) Linha 4 da tabela - Ativa a aresta 2 do temporizador *Timer2*;
   (c) Linha 3 da tabela - Ativa a aresta 3 do temporizador *Timer2*;
   (d) Linha 4 da tabela - Ativa as arestas 2 e 4 do temporizador *Timer2*;
3. Atribuições para testar o comportamento do temporizador *Timer3*:
   (a) Linha 5 da tabela - Ativa a aresta 1 do temporizador *Timer3*;

Tabela 4.8: Tabela para as variáveis de entrada do sistema de controle de nível de um tanque.

Atribuição para	NMB	D	A	NB	NMA	L	Ab	A2	F	O1	O2	O3	O4	O6	O7
1 - In_Timer1	1														
2 - In_Timer1	0														
3 - In_Timer2		1													
4 - In_Timer2		0													
5 - In_Timer3				1											
6 - In_Timer3				0											
7 - In_Timer4					1										
8 - In_Timer4					0										
9 - In_Timer5			1										1		
10 - In_Timer5			1										0		
11 - In_Timer5			1			0									
12 - In_Timer5			0			1									
13 - In_Timer6							1								
14 - In_Timer6							0								
15 - In_Timer7									1						
16 - In_Timer7									0						
17 - Desliga										1					
18 - Desliga										0	1				
19 - Desliga			0							0	0				
20 - Desliga			1							0	0	1			
21 - Desliga			1							0	0	0			
22 - S					1										
23 - S					0									0	
24 - S					0			0						1	
25 - S					0			1						1	
26 - R				1				1							
27 - R	1			1				0							
28 - R	0			1				0							1
29 - R	0			1				0							0
30 - R	1			0				0							
31 - R	0			0				0							0
32- R	0			0				0							1
33 - R	0			0				1							
34 - R	1			0				1							

(b) Linha 6 da tabela - Ativa a aresta 2 do temporizador *Timer3*;

(c) Linha 5 da tabela - Ativa as aresta 1 e 3 do temporizador *Timer3*;

(d) Linha 6 da tabela - Ativa a aresta 4 do temporizador *Timer3*;

4. Atribuições para testar a saída *Desliga*:

   (a) Linha 20 da tabela - deve ser selecionada após a saída do temporizador *Timer3* ser liberada com valor lógico *1*. Portanto, deve ser selecionada após a atribuição 3(c). De acordo com o algoritmo 2, o valor da variável *NB* deve ser *1*, para garantir que a saída do temporizador *Timer3* continuará ativada;

   (b) Linha 21 da tabela;

   (c) A linha 17 da tabela já foi selecionada, pois a saída de Timer1 (O1) com valor *1* já foi liberada quando a atribuição 1(d) foi selecionada. Da mesma forma, a linha 18 da tabela já foi selecionada, pois a saída de Timer2 (O2) com valor *1* já foi liberada quando a atribuição 2(b) foi selecionada. A linha 19 já foi selecionada, atribuição 1(b);

5. Atribuições para testar o comportamento do temporizador *Timer4*:

   (a) Linha 7 da tabela - Ativa a aresta 1 do temporizador *Timer4*;

   (b) Linha 8 da tabela - Ativa a aresta 2 do temporizador *Timer4*;

   (c) Linha 7 da tabela - Ativa as aresta 1 e 3 do temporizador *Timer4*;

   (d) Linha 8 da tabela - Ativa a aresta 4 do temporizador *Timer4*;

6. Atribuições para testar o comportamento do temporizador *Timer5* e a saída *Liga*: de acordo com o algoritmo 3, as atribuições utilizadas para testar este temporizador devem ser selecionadas após a saída do temporizador *Timer4* ser liberada com valor lógico *1*. Portanto, devem ser selecionadas após a atribuição 5(c). Veja:

   (a) Linha 9 da tabela - Ativa a aresta 1 do temporizador *Timer5*. De acordo com o algoritmo 2, o valor da variável *NMA* deve ser *1*, para garantir que a saída do temporizador *Timer4* continuará ativada;

   (b) Linha 10 da tabela - Ativa a aresta 2 do temporizador *Timer5*;

(c) A linha 11 da tabela já foi selecionada, pois é igual a atribuição 5(b);

(d) Linha 12 da tabela;

7. Atribuições para testar o comportamento do temporizador *Timer6*:

   (a) Linha 13 da tabela - Ativa a aresta 1 do temporizador *Timer6*;

   (b) Linha 14 da tabela - Ativa a aresta 2 do temporizador *Timer6*;

   (c) Linha 13 da tabela - Ativa a aresta 3 do temporizador *Timer6*;

   (d) Linha 14 da tabela - Ativa as arestas 2 e 4 do temporizador *Timer6*;

8. Atribuições para testar *S*:

   (a) Linha 22 da tabela - Já foi selecionada, pois é igual a atribuição 5(a). Da mesma forma, a linha 23 da tabela já foi selecionada, pois é igual a atribuição 1(b);

   (b) Linhas 24 e 25 da tabela - Devem ser selecionadas após a saída do temporizador *Timer6* ser liberada com valor lógico *1*. Portanto, deve ser selecionada após a atribuição 7(a). De acordo com o algoritmo 2, o valor da variável *Ab* deve ser *1*, para garantir que a saída do temporizador *Timer6* continuará ativada;

9. Atribuições para testar o comportamento do temporizador *Timer7*:

   (a) Linha 15 da tabela - Ativa a aresta 1 do temporizador *Timer7*;

   (b) Linha 16 da tabela - Ativa a aresta 2 do temporizador *Timer7*;

   (c) Linha 15 da tabela - Ativa a aresta 3 do temporizador *Timer7*;

   (d) Linha 16 da tabela - Ativa as arestas 2 e 4 do temporizador *Timer7*;

10. Atribuições para testar *R*:

    (a) Linhas 26, 27 e 34;

    (b) Linhas 28 e 32 da tabela - Devem ser selecionadas após a saída do temporizador *Timer7* ser liberada com valor lógico *1*. Portanto, devem ser selecionadas após a atribuição 9(a). De acordo com o algoritmo 2, o valor da variável *F* deve ser *1*, para garantir que a saída do temporizador *Timer7* continuará ativada;

(c) Linha 29 da tabela - Já foi selecionada, pois é igual a atribuição 3(a). Da mesma forma, as linhas 30, 31 e 33 da tabela já foram selecionada, pois, respectivamente, as atribuições 1(a), 1(b) e 6(b) (linha 25) já foram selecionadas;

Na Tabela 4.9 são apresentas as atribuições para as variáveis de entrada utilizadas para testar o sistema. Note que, de acordo com a Tabela 4.8, 34 atribuições de variáveis foram definidas para testar o comportamento do sistema, mas 36 foram utilizadas. Isto acontece porque, para avaliar o comportamento de cada temporizador do tipo DI ou DT é necessário utilizar 4 atribuições. Como foram definidas apenas duas atribuições para avaliar o comportamento dos temporizadores DI e DT, as mesmas terão que ser reutilizadas para avaliar o comportamento completo dos temporizadores. Desta forma, para testar o comportamento dos temporizadores *Timer1*, *Timer2*,*Timer3*, *Timer4*, *Timer6* e *Timer7* é necessário utilizar 24 atribuições e não apenas 12, como definido na Tabela 4.8. Portanto, o número total de atribuições definidas são 34 + 12 = 46.

Após gerado o conjunto com todas as atribuições para as variáveis de entrada é iniciada a execução da lógica do sistema. Desta forma, na função *updateOutputs()*, apresentada a seguir, cada atribuição é executada e as saídas do sistema são geradas. Cada atribuição é executada em um ciclo de varredura. O valor da variável *Liga* foi definido no modelo de autômato do temporizador *Timer5*.

```
void updateOutputs(){
  Desliga = ((out_Timer1) ∨ (out_Timer2 ∨ ∼Automatic) ∨
            (out_Timer3 ∧ Automatic));
  S = ((Nivel_Muito_Alto) ∨ (out_Timer6 ∧ ∼Automatic2));
  R = ((Nivel_Baixo ∧ Automatic2) ∨ (Nivel_Muito_Baixo) ∨
      (out_Timer7 ∧ ∼Automatic2));
  if (R == true) Saida = false;
  else Saida = S;
}
```

Tabela 4.9: Atribuições definidas para as variáveis de entrada do sistema de controle de nível de um tanque.

Atribuições para:	NMB	D	A	NB	NMA	L	Ab	A2	F
Timer1	1	0	0	0	0	0	0	0	0
Timer1	0	0	0	0	0	0	0	0	0
Timer1	1	0	0	0	0	0	0	0	0
Timer1	0	0	0	0	0	0	0	0	0
Timer2	0	1	0	0	0	0	0	0	0
Timer2	0	0	0	0	0	0	0	0	0
Timer2	0	1	0	0	0	0	0	0	0
Timer2	0	0	0	0	0	0	0	0	0
Timer3	0	0	0	1	0	0	0	0	0
Timer3	0	0	0	0	0	0	0	0	0
Timer3	0	0	0	1	0	0	0	0	0
Desliga	0	0	1	1	0	0	0	0	0
Timer3	0	0	0	0	0	0	0	0	0
Desliga	0	0	1	0	0	0	0	0	0
Timer4	0	0	0	0	1	0	0	0	0
Timer4	0	0	0	0	0	0	0	0	0
Timer4	0	0	0	0	1	0	0	0	0
Timer5	0	0	1	0	1	0	0	0	0
Timer5	0	0	1	0	0	0	0	0	0
Timer5	0	0	0	0	0	1	0	0	0
Timer4	0	0	0	0	0	0	0	0	0
Timer6	0	0	0	0	0	0	1	0	0
S	0	0	0	0	0	0	1	0	0
S	0	0	0	0	0	0	1	1	0
Timer6	0	0	0	0	0	0	0	0	0
Timer6	0	0	0	0	0	0	1	0	0
Timer6	0	0	0	0	0	0	0	0	0
Timer7	0	0	0	0	0	0	0	0	1
R	0	0	0	1	0	0	0	0	1
R	0	0	0	0	0	0	0	0	1
Timer7	0	0	0	0	0	0	0	0	0
Timer7	0	0	0	0	0	0	0	0	1
Timer7	0	0	0	0	0	0	0	0	0
R	0	0	0	1	0	0	0	1	0
R	1	0	0	1	0	0	0	0	0
R	1	0	0	0	0	0	0	1	0

4.4.2 Definição dos Casos de Teste

Na Tabela 4.10 são apresentado os casos de teste para o sistema de controle de nível de um tanque. Estes testes foram gerados a partir dos modelos de autômatos definidos na seção anterior. O valor do tempo de varredura, representado pela variável *tScan*, é de 25ms.

4.4.3 Execução dos Casos de Teste

Nesta seção são apresentados os resultados das execuções dos casos de teste no programa do sistema de controle de nível de um tanque. Para tanto, os seguintes testes foram realizados:

- teste na implementação correta: a implementação do sistema executada no CLP corresponde ao programa FBD apresentado na Figura 4.15;
- teste na implementação incorreta, programa FBD com os seguintes erros (ver Figura 4.22):
  - erro 1: retirar uma operação not do programa, neste caso, a operação not será retirada da variável *Automatico2*;
  - erro 2: trocar uma operação lógica *OR* por uma operação lógica *AND*;
  - erro 3: alterar o valor de *PT* de *Timer5* para 4 segundos.

Para o primeiro teste, implementação correta, o veredito de teste foi: "*A Implementação está em conformidade com a especificação*". Desta forma, nenhuma anormalidade relacionada ao comportamento da implementação do sistema foi detectada.

O veredito de teste para o programa FBD com o erro 1 foi: "*Erro no caso de teste 28 - Não há conformidade entre especificação e implementação. Para a combinação de entrada NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 1 era esperado que a saída* R *fosse* true *(saída obtida a partir da rede de AT), mas ela foi* false *(saída gerada pelo CLP).*

Caso de teste 28

E: NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 1

S: O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 1, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0

Tabela 4.10: Casos de Teste para o sistema de controle de nível de um tanque.

	E	S	T	t
1	{NMB = 1, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
2	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
3	{NMB = 1, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 1, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	start!,synchro!, update!, start!, ..., update!}	5s
4	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
5	{NMB = 0, D = 1, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 1, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
6	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 1, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
7	{NMB = 0, D = 1, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 1, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
8	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!, start!, ..., update!}	5s
9	{NMB = 0, D = 0, A = 0, NB = 1, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
10	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
11	{NMB = 0, D = 0, A = 0, NB = 1, NMA = 0, L = 0, Ab = 0, A2 = 0,F = 0}	{O1 = 0, O2 = 0, O3 = 1, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	start!,synchro!, update!, start!, ..., update!}	5s
12	{NMB = 0, D = 0, A = 1, NB = 1, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 1, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
13	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
14	{NMB = 0, D = 0, A = 1, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 0, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
15	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 1, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 1, Desliga = 1, Liga = 0, Saida = 1}	{start!,synchro!, update!}	tScan
16	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
17	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 1, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 1, O6 = 0, O7 = 0, R = 0, S = 1, Desliga = 1, Liga = 0, Saida = 1}	start!,synchro!, update!, start!, ..., update!}	5s
18	{NMB = 0, D = 0, A = 1, NB = 0, NMA = 1, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 1, O6 = 0, O7 = 0, R = 0, S = 1, Desliga = 0, Liga = 1, Saida = 1}	{start!,synchro!, update!}	tScan
19	{NMB = 0, D = 0, A = 1, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 0, Liga = 0, Saida = 0}	start!,synchro!, update!, start!, ..., update!}	5S
20	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 1, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
21	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
22	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 1, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 1, O7 = 0, R = 0, S = 1, Desliga = 1, Liga = 0, Saida = 1}	{start!,synchro!, update!}	tScan
23	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 1, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 1, O7 = 0, R = 0, S = 1, Desliga = 1, Liga = 0, Saida = 1}	{start!,synchro!, update!}	tScan
24	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 1, A2 = 1, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 1, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
25	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 1, O7 = 0, R = 0, S = 1, Desliga = 1, Liga = 0, Saida = 1}	{start!,synchro!, update!}	tScan
26	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 1, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 1, O7 = 0, R = 0, S = 1, Desliga = 1, Liga = 0, Saida = 1}	{start!,synchro!, update!}	tScan
27	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	start!,synchro!, update!, start!, ..., update!}	5S
28	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 1}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 1, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
29	{NMB = 0, D = 0, A = 0, NB = 1, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 1}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 1, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
30	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 1}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 1, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
31	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 1, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
32	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 1}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 1, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
33	{NMB = 0, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 1, Liga = 0, Saida = 0}	start!,synchro!, update!, start!, ..., update!}	5S
34	{NMB = 0, D = 0, A = 0, NB = 1, NMA = 0, L = 0, Ab = 0, A2 = 1, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
35	{NMB = 1, D = 0, A = 0, NB = 1, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0}	{O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan
36	{NMB = 1, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 1, F = 0}	{O1 = 0, O2 = 1, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0}	{start!,synchro!, update!}	tScan

T: start!, synchro!, update!

$t = 25ms$

Saídas geradas no programa do CLP

$O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 1, R = 0, S = 0, Desliga = 1,$

$Liga = 0, Saida = 0$

$Timestamp = 25ms$

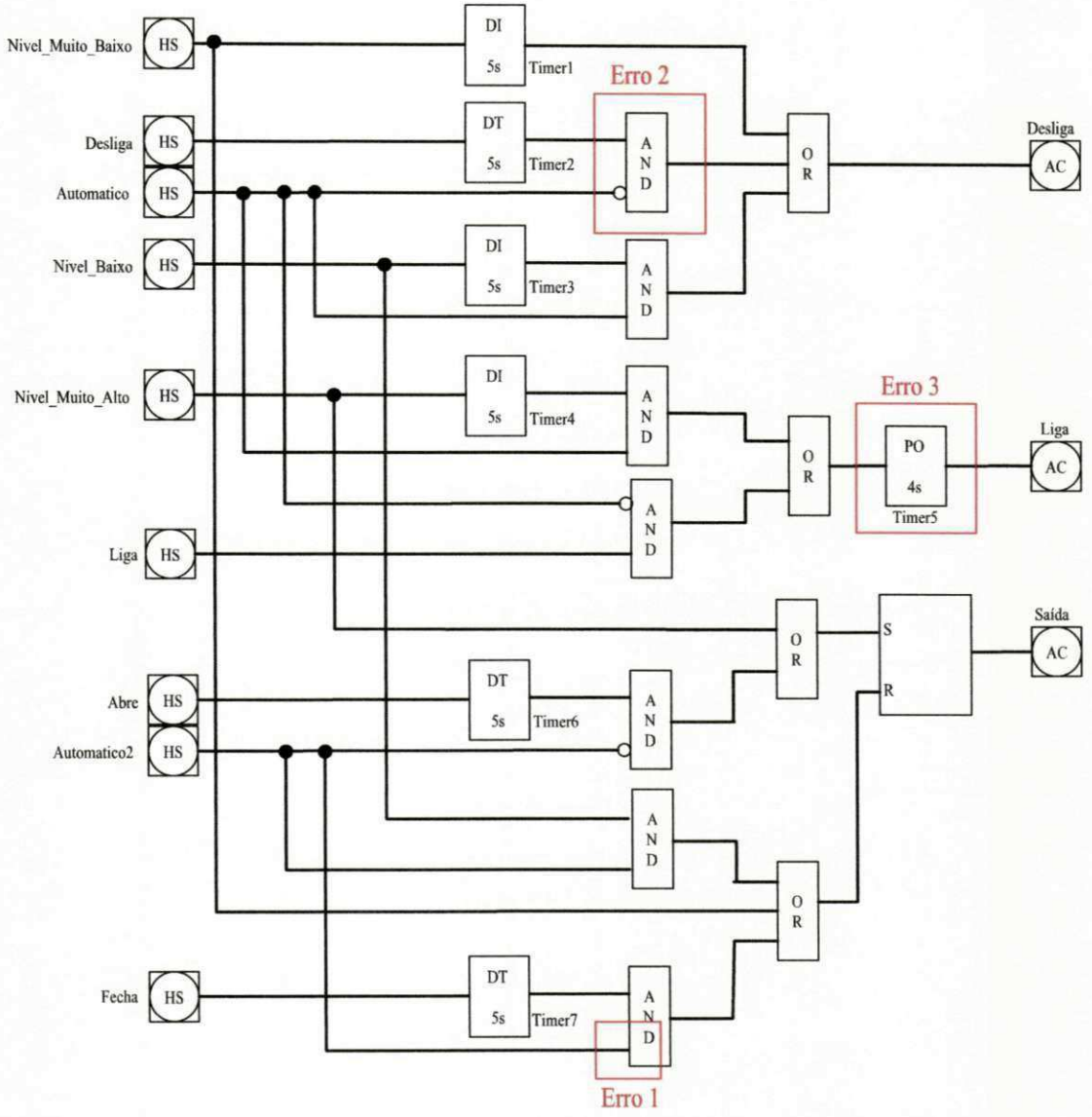


Figura 4.22: Erros no programa FBD para o sistema de controle de nível de um tanque.

O veredito de teste para o programa FBD com o erro 2 foi: "*Erro no caso de teste 1 - Não há conformidade entre especificação e implementação. Para a combinação de entrada NMB = 1, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0 era esperado que a saída* Desliga *fosse* true *(saída obtida a partir da rede de AT), mas ela foi* false *(saída gerada pelo CLP).*

Caso de teste 1

E: NMB = 1, D = 0, A = 0, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0

S: O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 1, S = 0, Desliga = 1, Liga = 0, Saida = 0

T: start!, synchro!, update!

t = 50ms

Saídas geradas no programa do CLP

O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 1, S = 0, Desliga = 0, Liga = 0, Saida = 0

Timestamp = 50ms

O veredito de teste para o programa FBD com o erro 3 foi: "*Erro no caso de teste 19 - Não há conformidade entre especificação e implementação. Para a combinação de entrada NMB = 0, D = 0, A = 1, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0 era esperado que o timestamp fosse 5 segundos, mas ele foi 4 segundos*". Este veredito informa que as saídas do programa foram geradas após 4 segundos e não após 5 segundos como descrito na especificação do sistema.

Caso de teste 19

E: NMB = 0, D = 0, A = 1, NB = 0, NMA = 0, L = 0, Ab = 0, A2 = 0, F = 0

S: O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 0, Liga = 0, Saida = 0

T: start!, synchro!, update!

t = 5s

Saídas geradas no programa do CLP

O1 = 0, O2 = 0, O3 = 0, O4 = 0, O6 = 0, O7 = 0, R = 0, S = 0, Desliga = 0, Liga = 0, Saida = 0

Timestamp = 4s

4.5 Avaliação dos Resultados dos Estudos de Caso

Nesta seção os resultados obtidos com a utilização do método proposto são avaliados. Primeiro será realizada uma comparação entre a rede de autômatos temporizados gerada neste trabalho com as redes geradas em (Barbosa *et al.* , 2007; Barbosa, 2008). Após isso, será feita uma avaliação dos casos de teste gerados bem como os vereditos fornecidos pela ferramenta de teste desenvolvida neste trabalho.

Comparando as redes de autômatos temporizados geradas neste trabalho com as redes geradas em (Barbosa *et al.* , 2007; Barbosa, 2008), para os mesmos estudos de caso, pode-se observar que:

- para o método proposto neste trabalho, 2, 8 e 22 localidades foram geradas, respectivamente, para o primeiro, o segundo e o terceiro estudo de caso, contra 20, 29 e 58 localidades geradas com a utilização do método proposto em (Barbosa *et al.* , 2007; Barbosa, 2008). Desta forma, podemos observar que com o uso do método proposto nesta tese o número de localidades, e consequentemente o número de estados gerados durante a geração dos casos de teste foram reduzidos;

- a rede de autômatos temporizados gerada em (Barbosa *et al.* , 2007; Barbosa, 2008), para os estudos de caso 1 e 3, possui um autômato temporizado que representa um flip−flop do tipo SR, aumentando assim a quantidade de localidades e, consequentemente o número de estados gerados. Para a rede de autômatos temporizados gerada neste trabalho este fato não ocorre, pois o funcionamento dos flip−flops são modelados por funções;

- todos os temporizadores foram modelados com variáveis de relógio com tempo contínuo.

Para os casos de teste gerados, nos três estudos de caso, pode-se observar que o comportamento de todos os temporizadores e os estados das saídas do sistema foram avaliados, pois todas as transições de todos os temporizadores foram cobertas e casos de teste que tornam cada saída do sistema ativada e desativada foram gerados e executados no programa do CLP para SIS. Erros, tanto a nível de operação lógica, como alteração do valor presente de um temporizador foram detectados na implementação do sistema. Além disso, casos de teste também foram gerados para avaliar a dependência de temporizadores, como apresentado no estudo de caso 3, sistema para controle de nível de um tanque.

Para cada estudo de caso pôde-se observar que nem todas as atribuições definidas na extração dos caminhos de cada ROBDD foram utilizadas para gerar os casos de teste. Para o primeiro estudo de caso 15 combinações de entrada foram geradas, mas apenas 11 foram utilizadas para gerar os casos de teste. Isto acontece porque atribuições iguais, testes redundantes, não são consideradas na etapa de geração dos casos de teste. Nos estudos de caso 2 e 3 pôde-se observar que também houve uma redução do número de casos de teste redundantes.

4.6 Considerações Finais do Capítulo

Neste capítulo foi apresentado os passos necessários para executar os experimentos descritos nesta tese. Estes experimentos servem para validar a aplicabilidade do método proposto em sistemas com ou sem elementos temporizados. Para tanto, três estudos de caso foram utilizados para avaliar o método proposto, um sistema para prevenção de incêndio, um SIS para detectar a presença de fogo e gás em uma zona e um SIS para controlar o nível de um tanque. De acordo com os resultados apresentados, para cada estudo de caso, foi possível avaliar tanto os estados das saídas como propriedades temporizadas do sistema com um conjunto reduzido de casos de teste.

Capítulo 5

Considerações Finais e Trabalhos Futuros

Neste capítulo são resumidos os principais resultados deste trabalho e apresentadas as sugestões de trabalhos futuros.

5.1 Considerações Finais

O trabalho aqui apresentado introduziu um método para aumentar a confiança e a segurança em programas de CLP para SIS através da abordagem HIL em conjunto com ROBDDs. Para tanto, casos de testes, que contemplam os estados das saídas bem como propriedades temporizadas do sistema são gerados automaticamente a partir da especificação do sistema no formato de diagramas ISA 5.2. Após gerados, os valores das entradas dos casos de teste são enviados para o CLP. Por fim, é fornecido um veredito informando a conformidade entre as saídas esperadas (saídas obtidas a partir da rede de autômatos temporizados) e as saídas geradas no CLP.

O principal diferencial do método proposto neste trabalho para os já conhecidos é que com o uso dele é possível aumentar a confiança e a segurança em programas de CLP para SIS, através da geração automática de casos de teste, fazendo uso de ROBDDs, para que propriedades temporizadas e os estados das saídas do sistema possam ser avaliadas. Além disso, os valores das saídas esperadas são comparadas com as saídas geradas pelo CLP, testando assim o comportamento do sistema na sua plataforma alvo.

A utilização do método possibilita a aplicação do processo de teste de forma paralela ao processo de desenvolvimento, minimizando o custo adicional para produção de artefatos específicos para teste. Além disso, a complexidade de construção dos modelos de autômatos temporizados gerados é transparente ao usuário, uma vez que o mesmo necessita apenas fornecer os arquivos com a descrição da especificação do sistema para obter o veredito dos testes que são de fácil interpretação.

Considerando as hipóteses definidas no Capítulo 1, os resultados obtidos foram:

Na seção 3.1 foi apresentada a arquitetura do sistema HIL proposto neste trabalho. Diferentemente das arquiteturas tradicionais, compostas por três módulos, a arquitetura proposta é formada apenas por dois módulos: unidade de controle, representada por um CLP, e uma interface homem-máquina (IHM). No módulo IHM utilizamos aplicações para modelar a especificação do sistema, gerar e executar os casos de teste de conformidade e controlar processos de tempo real. Neste trabalho, o modelo da planta não é utilizado, uma vez que estamos interessados apenas em verificar a conformidade do programa do SIS e não o funcionamento da planta industrial. Com o desenvolvimento desta arquitetura HIL a hipótese H1 (a arquitetura HIL pode ser adaptada no contexto de SIS de forma que a conformidade entre a especificação e a implementação do sistema possa ser avaliada) é comprovada.

Na seção 3.2.2, o algoritmo utilizado para definir as entradas para os casos de testes a partir das atribuições obtidas de um ROBDD, Algoritmo 3, é apresentado. Neste algoritmo, as atribuições para as variáveis de entrada são definidas de acordo com cada função que determina uma saída para o sistema sob teste. Caso a função não seja composta por temporizadores, então todas as atribuições obtidas a partir desta função serão utilizadas para testar o sistema. Caso a função seja composta por temporizadores, então de acordo com cada tipo de temporizador serão definidas atribuições para ativar as arestas dos autômatos que representam cada temporizador. Desta forma, com o algoritmo é possível testar os estados das saídas e propriedades temporizadas do sistema. No Capítulo 4, este algoritmo é executado para três estudos de caso e os casos de teste gerados realmente comprovaram a hipótese H2 (com o uso de ROBDDs é possível testar os estados das saídas e propriedades temporizadas do sistema). Porém, para outros estudos de caso, no contexto de SIS, só garantimos que a hipótese H2 é comprovada se cada função que determina uma saída para o sistema é representada por uma *fbf* que não é uma tautologia e uma *fbf* que não é uma contradição,

conforme descrito na seção 3.5.

No Algoritmo 2, apresentado na seção 3.2.2, os casos de teste gerados representam um conjunto reduzido das atribuições extraídas a partir dos ROBDDs que representam cada função Booleana que determina uma saída para o sistema. No Capítulo 4, para os três estudos de caso realizados pôde-se observar que houve uma redução do número de casos de teste redundantes, comprovando a hipótese H3 (com o uso de ROBDDs é possível reduzir o número de casos de teste redundantes). Na seção 3.5 é apresentada uma análise geral do Algoritmo 2 para verificar a redução do número de casos de teste gerados.

Neste documento é proposto um novo método para verificar programas de CLPs para SIS. Neste sentido, as principais contribuições são:

- um novo método para teste de programas para CLPs para SIS, em que a conformidade entre a especificação, representada por uma rede de autômatos temporizados, e a implementação do sistema pode ser verificada. O uso da arquitetura HIL, tendo como suporte o uso do protocolo OPC e ROBDDs possibilitou o desenvolvimento deste método;

- teste do comportamento de elementos temporizados. O comportamento de temporizadores, bem como a lógica completa do programa de um CLP para SIS é verificado em tempo de execução;

- mapeamento de um modelo referente à especificação do sistema para o formalismo de redes de autômatos temporizados, gerando assim uma especificação formal do sistema sob teste. Além disso, a rede de autômatos temporizados gerada possui um menor número de localidades e de variáveis quando comparada as redes apresentadas nos trabalhos de Barbosa et al. (2007; 2008);

- com o método de teste proposto neste trabalho é possível fornecer os meios necessários (arquitetura e algoritmos) para gerar e executar casos de testes e fornecer vereditos de teste que são de fácil interpretação, confiáveis e seguros;

- diferentemente de outros métodos, com o uso do método apresentado neste trabalho é possível gerar casos de teste, não redundantes, que contemplam os possíveis valores

para as saídas de um programa de CLP para SIS. O uso de ROBDDs possibilitou o desenvolvimento deste método. Com a utilização de ROBDDs é possível testar o comportamento de temporizadores e os estados das saídas do sistema bem como reduzir os casos de teste redundantes.

No que diz respeito à divulgação dos resultados desta pesquisa, enumeram-se na ordem cronológica um resumo das publicações como sendo:

- K. V. Oliveira, L. D. da Silva, A. Perkusich, K. Gorgônio and L. C. Silva. Utilização de Diagramas de Decisão Binária Ordenados para Geração de Casos de Teste em Sistemas Instrumentados de Segurança. Em Anais do XI Simpósio Brasileiro de Automação Inteligente, Fortaleza, Brasil, 2013. Sociedade Brasileira de Automática.

- K. V. Oliveira, A. Perkusich, K. C. Gorgônio, L. D. da Silva and A. F. Martins. Using equivalence classes for testing programs for safety instrumented systems. Em 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), páginas 1-7, Cagliari, Italy September 2013. IEEE.

- K. V. Oliveira, L. D. da Silva, A. Perkusich, and K. C. Gorgônio. Uma abordagem para a geração e execução de casos de teste em programa de sistemas instrumentados de segurança. Em Anais do XIX Congresso Brasileiro de Automática (CBA 2012), páginas 3372-3379, Campina Grande, Brasil, 2012. Sociedade Brasileira de Automática;

- K. V. Oliveira, K. Gorgônio, A. Perkusich, A. M. N. Lima, and L. D. da Silva. Automatic Timed Automata Extraction from Ladder Programs for Model-Based Analysis of Control Systems. H. Mouratidis, editor, Software Engineering for Secure Systems: Industrial and Research Perspectives, páginas 305-328. IGI Global, Hershey, USA, 2010;

- K. V. Oliveira, L. D. da Silva, A. Perkusich, A. M. N. Lima, and K. Gorgônio. Automatic timed automata extraction from ladder programs for model-based analysis of control systems. Em 2010 IEEE International Symposium on Industrial Electronics, páginas 90-95, Bari, Italy, July 2010. IEEE;

- K. V. Oliveira, L. D. da Silva, A. Perkusich, A. M. N. Lima, and K. Gorgônio. Geração Automática de Testes de Conformidade para Programas de Controladores Lógicos programáveis. Em Anais do CBA 2010, páginas 2995-3001, Bonito, Brasil, 2010. Sociedade Brasileira de Automática;

- K. V. Oliveira, A. Perkusich, A. M. N. Lima, K. Gorgônio, and L. D. da Silva. Standard-based formal validation of Programmable Logic Controller programs. Em 2010 IEEE International Conference on Industrial Technology, páginas 1655-1660, Viña del Mar - Valparaíso, Chile, 2010. IEEE;

- K. V. Oliveira, K. Gorgônio, A. Perkusich, A. M. N. Lima, and L. D. da Silva. Extração Automática de Autômatos Temporizados a Partir de Diagramas Ladder. Em Anais do IX Simpósio Brasileiro de Automação Inteligente, Brasília, Brasil, 2009. Sociedade Brasileira de Automática.

5.2 Trabalhos Futuros

Com a finalização do trabalho, tivemos a possibilidade de fazer uma análise do mesmo, a qual resultou no seguinte conjunto de propostas para a sua continuidade:

- modelar e avaliar blocos de funções e instruções de contagem, pois estas são muito comuns em programas de CLP para SIS;

- implementar um módulo na ferramenta de teste desenvolvida de forma que os possíveis erros apresentados nos vereditos de teste sejam reportados para os diagramas ISA;

- aperfeiçoar o método para que o usuário seja informado caso alguma função que determine uma saída não seja representada por uma *fbf* que é uma tautologia e uma *fbf* que é uma contradição;

- melhorar o método para que ele passe a trabalhar com programação multitarefa. Uma vez que o método apresentado neste trabalho lida apenas com a execução de um único programa;

- aprimorar o método para que mais de um CLP possa controlar o mesmo processo. Uma vez que o método apresentado neste trabalho lida apenas com modelagem de um CLP;

- submeter o trabalho a outros estudos de caso, tendo como objetivo obter uma maior confiança com relação ao procedimento proposto.

Bibliografia

Albuquerque, André Ribeiro Lins de. 2007. A Técnica de Hardware-in-the-Loop no Auxílio de Projetos Mecatrônicos. *Mecatrônica Atual*, **6**(35).

Alur, Rajeev, & Dill, David L. 1994. A Theory of Timed Automata. *Theoretical Computer Science*, **126**(2), 183–235.

Angelov, C., Ke, Xu, Guo, Yu, & Sierszecki, K. 2008. Reconfigurable State Machine Components for Embedded Applications. *Pages 51–58 of: 34th Euromicro Conference Software Engineering and Advanced Applications*.

Arnicane, Vineta. 2009. Complexity of Equivalence Class and Boundary Value Testing Methods. *International Journal of Computer Science and Information Technology*, **751**, 80–101.

Bacic, M. 2005. On hardware-in-the-loop simulation. *Pages 3194 – 3198 of: Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference*.

Barbosa, Luiz Paulo de Assis. 2008. Geração Automática de Modelos de Autômatos Temporizados para Realização de Testes de Sistemas Instrumentados de Segurança. Master´s Thesis, Universidade Federal de Campina Grande. Dezembro.

Barbosa, Luiz Paulo de Assis, Gorgonio, Kyller, Silva, Leandro Dias da, Lima, Antonio Marcus Nogueira, & Perkusich, Angelo. 2007. On the Automatic Generation of Timed Automata Models from ISA 5.2 Diagrams. *IEEE Conference on Emerging Technologies and Factory Automation. ETFA*, Setembro, 406–412.

Beizer, Boris. 1995. *Black-box Testing: techniques for functional testing of software and systems*. New York, NY, USA: John Wiley & Sons, Inc.

Bender, Darlam Fabio, Combemale, Benoît, Crégut, Xavier, Farines, Jean Marie, Berthomieu, Bernard, & Vernadat, François. 2008. Ladder Metamodeling and PLC Program Validation Through Time Petri Nets. *Pages 121–136 of: Proceedings of the 4th European Conference on Model Driven Architecture (ECMDA-FA '08)*. Berlin, Heidelberg: Springer-Verlag.

Bengtsson, Johan, & Yi, Wang. 2004. Timed Automata: Semantics, Algorithms and Tools. *Lectures on Concurrency and Petri Nets*, **1633**(Abril), 8–22.

Bohn, Jürgen, Damm, Werner, Wittke, Hartmut, Klose, Jochen, & Moik, Adam. 2002. Modeling and Validating Train System Applications Using Statemate and Live Sequence Charts. *In: Proceedings of the Conference on Integrated Design and Process Technology (IDPT2002), Society for Design and Process Science*.

Bollig, B., & Wegener, I. 1996. Improving the Variable Ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, **45**(9), 993–1002.

Borcsok, J., Chaaban, W., Schwarz, M., Sheng, H., Sheleh, O., & Batchuluun, B. 2009 (Outubro). An Automated Software Verification Tool for Model-Based Development of Embedded Systems with Simulink. *Pages 1 –6 of: XXII International Symposium on Information, Communication and Automation Technologies, ICAT*.

Brace, K.S., Rudell, R.L., & Bryant, R.E. 1990. Efficient Implementation of a BDD Package. *Pages 40–45 of: Proceedings of the 27th ACM/IEEE Conference on Design Automation*.

Bryan, Luis A., & Bryan, Eric A. 1997. *Programmable Controllers: Theory and Implementation*. Industrial Text Company.

Bryant, Randal E. 1992. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, **24**(3), 293–318.

Bryant, R.E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, **C-35**(8), 677–691.

Canet, Géraud, Couffin, Sandrine, Lesage, Jean-Jacques, Petit, Antoine, & Schnoebelen, Philippe. 2000. Towards the Automatic Verification of PLC Programs Written in Instruction List. *Pages 2449–2454 of: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC 2000)*. Nashville, Tennessee, USA: Argos Press.

Chaaban, W., Schwarz., M., Batchuluun, B., Sheng, H., & Borcsok, J. 2011a (dezembro). A HiL Test Bench for Verification and Validation Purposes of Model-Based Developed Applications Using Simulink and OPC DA Technology. *Pages 56–61 of: 7th International Conference on Electrical and Electronics Engineering (ELECO´2011)*.

Chaaban, W., Schwarz., M., Batchuluun, B., Sheng, H., & Borcsok, J. 2011b (outubro). A Partially Automated HiL Test Environment for Model-Based Development Using Simulink and OPC Technology. *Pages 1–6 of: XXIII International Symposium on Information, Communication and Automation Technologies (ICAT´2011)*.

Cooling, Jim. 2003. *Software Engineering for Real-Time Systems*. Addison Wesley.

Crãciun, Octavian, Florescu, Adrian, Bacha, Seddik, Munteanu, Iulian, & Bratcu, Antoneta Iuliana. 2010 (Setembro). Hardware-in-the-loop Testing of PV Control Systems Using RT-Lab Simulator. *Pages S2–1 –S2–6 of: 14th International Power Electronics and Motion Control Conference*.

En-Nouaary, Abdeslam, Dssouli, Rachida, & Khendek, Ferhat. 2002. Timed Wp-Method: Testing Real-Time Systems. *IEEE Transactions on Software Engineering*, **28**, 1023–1038.

Fang, Laihua, Wu, Zongzhi, Wei, Lijun, & Liu, Ji. 2008 (Setembro). Design and Development of Safety Instrumented System. *Pages 2685–2690 of: IEEE International Conference on Automation and Logistics (ICAL´2008)*.

Frey, George, & Litz, Leothar. 2000. Formal Methods in PLC Programming. *Pages 2431–2436 of: IEEE International Conference on Systems, Man, and Cybernetics*, vol. 4.

Fujita, M., Fujisawa, H., & Matsunaga, Y. 1993. Variable ordering algorithms for ordered binary decision diagrams and their evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **12**(1), 6–12.

Ge, Xi, Taneja, Kunal, Xie, Tao, & Tillmann, Nikolai. 2011 (Maio). DyTa: Dynamic Symbolic Execution Guided with Static Verification Results. *Pages 992–994 of: Procedures of 33rd International Conference on Software Engineering (ICSE´2011).*

Gergely, Eugen Ioan, Coroiu, Laura, & Gacsadi, Alexandru. 2010. Design of Safe PLC Programs by Using Petri Nets and Formal Methods. *Pages 86–91 of: Proceedings of the 11th International conference on Automation Robotics and Vision.* ICAI'10. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS).

Goble, William M., & Cheddie, Harry. 2005. *Safety Instrumented Systems Verification: Practical Probabilistic Calculations*. ISA.

Gruhn, Paul, & Cheddie, Harry L. 2006. *Safety Instrumented Systems: Design, Analysis, and Justification*. 2nd edn. ISA.

Hamelin, P., Bigras, P., Beaudry, J., Lemieux, S., & Blain, M. 2008 (Julho). Hardware-in-the-loop Simulation of an Impedance Controlled Robot Using a Direct-Drive Test Bench. *Pages 1281–1286 of: IEEE International Symposium on Industrial Electronics. ISIE.*

Hessel, Anders, & Pettersson, Paul. 2004. A Test Case Generation Algorithm for Real-Time Systems. *Pages 268–273 of: Fourth International Conference of the Quality Software.* QSIC´04. Washington, DC, USA: IEEE Computer Society.

Iacob, M., & Andreescu, G. 2011a (maio). Implementation of Hardware-In-the-Loop System for Drum-Boiler-Turbine Decoupled Multivariable Control. *Pages 45–50 of: 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI).*

Iacob, M., Andreescu, G.-D., Antal, R., & A.-M.Dan. 2011b (junho). Multivariable Adaptive Control with Hardware-In-the-Loop for a Drum-Type Boiler-Turbine System. *Pages 898–903 of: 19th Mediterranean Conference on Control Automation (MED).*

International Electrotechnical Commission. 1993. *International Standard 1131 Programmable Controllers. Part 3: Programming Languages.*

ISA. 1992 (July). *Binary Logic Diagrams for Process Operations*. ISA 5.2-1976 (R1992) edn. ISA - The Instrumentation, Systems, and Automation Society.

Isermann, R., Schaffnit, J., & Sinsel, S. 1999. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, **7**(5), 643–653.

Iwanitz, F., & Lange, J. 2002. *OPC: Fundamentals, Implementation, and Application*. 2nd edn. Hüthig.

Jacobs, P.H.M., Verbraeck, A., & Rengelink, W. 2005 (Dezembro). Emulation with DSOL. *Page 10 pp. of: Proceedings of the Winter Simulation Conference.*

Jee, Eunkyoung, Jeon, Seungjae, Bang, Hojung, Cha, Sungdeok, Yoo, Junbeom, Park, Geeyong, & Kwon, Keechoon. 2006 (Dezembro). Testing of Timer Function Blocks in FBD. *Pages 243 –250 of: 13th Asia Pacific Software Engineering Conference.*

John, Karl-Heinz, & Tiegelkamp, Michael. 2001. *IEC 61131-03: Programming Industrial Automation Systems*. Berlin, Germany: Springer-Verlag.

Kain, Sebastian, Schiller, Frank, & Dominka, Sven. 2011. Methodology for Reusing Real-time HiL Simulation Models in the Commissioning and Operation Phase of Industrial Production Plants. *In: Intelligent Mechatronics*. Ganesh Naik (Ed.) InTech.

Kalita, D., & Khargonekar, P.P. 2002. Formal Verification for Analysis and design of Logic Controllers for Reconfigurable Machining Systems. *IEEE Transactions on Robotics and Automation*, **18**(4), 463 – 474.

Kapasi, Chintan P., Jose, Vazhapilly Jeel, Sharma, Medha, & Warke, Nilima. 2011 (Dezembro). Safety Instrumented System and Alarm System for Heater. *Pages 512–516 of: International Conference on Recent Advancements in Electrical, Electronics and Control Engineering.*

Karimi, S., Poure, P., & Saadate, S. 2010. An HIL-Based Reconfigurable Platform for Design, Implementation, and Verification of Electrical System Digital Controllers. *IEEE Transactions on Industrial Electronics*, **57**(4), 1226–1236.

Katoen, Jooster-Pieter. 1999. *Concepts, Algoritms, and Tools for Model Checking*. Lecture Notes of the Course "Mechanised Validation of Parallel Systems".

Kim, Jung, De, Suvranu, & Srinivasan, M. A. 2002. Computationally Efficient Techniques for Real Time Surgical Simulation with Force Feedback. *Pages 51– of: Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*. Washington, DC, USA: IEEE Computer Society.

Kumar, A., Kumar, A., Choudhary, S., & Varde, P.V. 2010. Optimization of Binary Decision Diagram using Genetic algorithm. *Pages 168–175 of: 2nd International Conference on Reliability, Safety and Hazard (ICRESH)*.

Larsen, Kim G., Mikucionis, Marius, & Nielsen, Brian. 2004. Online Testing of Real-Time Systems Using uppaal. *Pages 79–94 of: Workshop on Formal Approaches to Testing of Software*. Springer Verlag.

Larsen, Kim G., Mikucionis, Marius, Nielsen, Brian, & Skou, Arne. 2005. Testing Real-Time Embedded Software Using UPPAAL-TRON: an Industrial Case Study. *Pages 299–306 of: Proceedings of the 5th ACM International Conference on Embedded Software*. New York, NY, USA: ACM.

Li, Liangming, Liu, Lei, Wang, Zhijian, & Tang, Yelong. 2008 (dezembro). Research on Interface Automata Testing. *Pages 743 –746 of: International Conference on Computer Science and Software Engineering*, vol. 2.

Li, Zhen, Kyte, M., & Johnson, B. 2004 (oct.). Hardware-in-the-loop real-time simulation interface software design. *Pages 1012 – 1017 of: The 7th International IEEE Conference on Intelligent Transportation Systems*.

Ljungkrantz, O., Akesson, K., Yuan, Chengyin, & Fabian, M. 2012. Towards Industrial Formal Specification of Programmable Safety Systems. *IEEE Transactions on Control Systems Technology*, **20**(6), 1567–1574.

Machado, J., & Seabra, E. 2013. HiL simulation workbench for testing and validating PLC programs. *Pages 230–235 of: 11th IEEE International Conference on Industrial Informatics (INDIN)*.

Mahnke, Wolfgang, Leitner, Stefan-Helmut, & Damm, Matthias. 2009. *OPC Unified Architecture*. Springer.

Mari, Federico, Melatti, Igor, Salvo, Ivano, & Tronci, Enrico. 2012. Synthesizing Control Software from Boolean Relations. *In: International Journal on Advances in Software*, vol. 5.

Martin, A., & Emami, M.R. 2006. An Architecture for Robotic Hardware-in-the-Loop Simulation. *Pages 2162–2167 of: Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation.*

Meinel, C., Sack, H., & Schillings, V. 2002. VisBDD - A Web-based Visualization Framework for OBDD Algorithms. *IWLS*, **2002**, 385–390.

Michalek, D., Gehsat, C., Trapp, R., & Bertram, T. 2005 (julho). Hardware-in-the-loop-simulation of a vehicle climate controller with a combined HVAC and passenger compartment model. *Pages 1065 –1070 of: International Conference on Advanced Intelligent Mechatronics.*

Minato, Shin-Ichi. 1996. *Binary decision diagrams and applications for VLSI CAD.* Kluwer International series in Engineering and Computer Science. Boston, Dordrecht, London: Kluwer academic publishers.

Miremadi, S., Akesson, K., & Lennartson, B. 2011. Symbolic Computation of Reduced Guards in Supervisory Control. *IEEE Transactions on Automation Science and Engineering*, **8**(4), 754–765.

Miremadi, S., Lennartson, B., & Akesson, K. 2012. A BDD-Based Approach for Modeling Plant and Supervisor by Extended Finite Automata. *IEEE Transactions on Control Systems Technology*, **20**(6), 1421–1435.

Mokadem, Houda Bel, Bérard, Béatrice, Gourcuff, Vincent, Smet, Olivier De, & Roussel, Jean-Marc. 2010. Verification of a Timed Multitask System With Uppaal. *IEEE Transactions on Automation Science and Engineering*, **7**(4), 921 –932.

Nidhra, J., & Dondeti, S. 2012. Black Box and White Box Testing Techiniques - A Literature Review. *International Journal of Embedded Systems and Applications (IJESA)*, **2**(6), 29–50.

Oliveira, Kézia de Vasconcelos, Gorgônio, Kyller, Perkusich, Angelo, Lima, Antonio Marcus Nogueira, & Silva, Leandro Dias da. 2010a. Automatic Timed Automata Extraction from Ladder Programs for Model-Based Analysis of Control Systems. *Pages 305–328 of:* Mouratidis, Haralambos (ed), *Software Engineering for Secure Systems: Industrial and Research Perspectives*. Hershey, USA: IGI Global.

Oliveira, Kézia de Vasconcelos, Perkusich, Angelo, e Lima, Antonio Marcus Nogueira, Gorgonio, Kyller, & Silva, Leandro Dias da. 2010b (Março). Standard-based Formal Validation of Programmable Logic Controller Programs.

Oliveira, Kezia de Vasconcelos, Perkusich, Angelo, Gorgonio, Kyller Costa, Dias da Silva, Leandro, & Martins, Aldenor Falcao. 2013. Using Equivalence Classes for Testing Programs for Safety Instrumented Systems. *Pages 1–7 of: IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*.

OPC Foundation. 1998. *OPC Overview v1.0. Disponível em: http://www.opcfoundation.org*.

OPC Foundation. 2002. *OPC Alarms and Events Custom Interface Standard Specification v.1.10. Disponível em: http://www.opcfoundation.org*.

OPC Foundation. 2003a. *OPC Data Access Specification v.3.00. Disponível em: http://www.opcfoundation.org*.

OPC Foundation. 2003b. *OPC Historical Data Access Custom Interface Specification 1.0. Disponível em: http://www.opcfoundation.org*.

OPC Foundation. 2006. *OPC Unified Architecture v1.0. Disponível em: http://www.opcfoundation.org*.

Otto, Andreas, & Hellmann, Klas. 2009. IEC 61131: A General Overview and Emerging Trends. *IEEE Industrial Electronics Magazine*, **3**(4), 27 –31.

Parr, Andrew. 2003. *Programmable Controllers an Engineer´s Guide*. 3nd edn. Newnes.

Patil, M.M., Subbaraman, S., & Joshi, S. 2011 (dezembro). Exploring Integrated Circuit Verification Methodology for Verification and Validation of PLC Systems. *Pages 88 –93 of: International Symposium on Electronic System Design (ISED)*.

Peixoto, R. J. S., da Silva, L. D., Perkusich, A., Lima, A. M. N., & Gorgônio, K. 2010. GUNGNIR - Uma ferramenta para geração e execução de testes de conformidade utilizando autômatos temporizados. *Pages 3211–3216 of: Anais do CBA 2010.* Bonito, Brasil: Sociedade Brasileira de Automática.

Pressman, Roger S. 2006. *Engenharia de Software.* 6 edn. São Paulo: McGraw-Hill.

Pu, Fei, & Zhang, Wenhui. 2007 (junho). Partition Refinement in Abstract Model Checking. *Pages 209 –218 of: Symposium on Theoretical Aspects of Software Engineering.*

Rankin, D.J., & Jiang, Jin. 2011. A Hardware-in-the-Loop Simulation Platform for the Verification and Validation of Safety Control Systems. *IEEE Transactions on Nuclear Science*, **58**(2), 468 –478.

Rice, Michael, & Kulhari, Sanjay. 2008. *A Survey of Static Variable Ordering Heuristics for Efficient BDD/MDD Construction.* Techinical. University of California.

Rudas, Imre J., Fodor, Janos, & Kacprzyk, Janusz (eds). 2010. *Computational Intelligence and Informatics: Principles and Practice.* 1nd edn. New York: Springer-Verlag New York, LLC.

Rudell, Richard. 1993. Dynamic variable ordering for ordered binary decision diagrams. *Pages 42–47 of: Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design.* ICCAD '93. Los Alamitos, CA, USA: IEEE Computer Society Press.

Schaffnit, J., Sinsel, S., & Isermann, R. 1998 (junho). Hardware-in-the-loop simulation for the investigation of truck diesel injection systems. *Pages 498 –502 vol.1 of: Proceedings of the American Control Conference*, vol. 1.

Schiffmann, Jessica, & Steinbach, Bernd. 2012. Utilization of BDDs for Test-Case-Specifications of GUI-Applications. *Pages 41–48 of: Proceedings of the 10th International Workshops on Boolean Problems, Freiberg University of Mining and Technology.*

Schneider, K., & Brandt, J. 2008 (junho). Performing Causality Analysis by Bounded Model Checking. *Pages 78 –87 of: 8th International Conference on Application of Concurrency to System Design.*

Sensarma, Debajit, Banerjee, Subhashis, Basuli, Krishnendu, Naskar, Saptarshi, & Sen-Sarma, Samar. 2012. On an Optimization Technique Using Binary Decision Diagram. *CoRR*, **abs/1203.2505**.

Shah, S.M., & Irfan, M. 2005. Embedded Hardware/Software verification and Validation Using Hardware-in-the-Loop Simulation. *Pages 494–498 of: IEEE Symposium on Emerging Technologies.*

Shannon, Claude E. 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal*, **27**(July, October), 379–423, 623–656.

Shi-peng, Fan, De-fu, Lin, Yu-long, Lu, & Hong, Zhang. 2011 (junho). Research and Simulation of Rolling Missile's Control System. *Pages 195 –198 of: IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, vol. 1.

Short, M., & Pont, M.J. 2005 (setembro). Hardware in the Loop Simulation of Embedded Automotive Control System. *Pages 426 – 431 of: Proceedings IEEE Intelligent Transportation Systems.*

Silva, Leandro Dias da, Barbosa, Luiz Paulo de Assis, Gorgônio, Kyller, Perkusich, Angelo, & Lima, Antonio Marcus Nogueira. 2008. On the Automatic Generation of Timed Automata Models from Function Block Diagrams for Safety Instrumented Systems. *Pages 291–296 of: 34th Annual Conference of the IEEE Industrial Electronics Society (IECON'2008).* Orlando, USA: IEEE Industrial Electronics Society.

Sisle, Mitchell E., & McCarthy, Edward D. 1982. Hardware-in-the-Loop Simulation for an Active Missile. *SIMULATION*, **39**(5), 159–167.

Skoldstam, M., Akesson, K., & Fabian, M. 2007. Modeling of Discrete Event Systems Using Finite Automata with Variables. *Pages 3387–3392 of: 46th IEEE Conference on Decision and Control.*

Sommerville, Ian. 2007. *Engenharia de Software*. 8 edn. Addison Wesley.

Stergiou, Stergios. 2011. Implicit Permutation Enumeration Networks and Binary Decision Diagrams Reordering. *Pages 615–620 of: Proceedings of the 48th Design Automation Conference*. DAC '11. New York, NY, USA: ACM.

Sugai, Masahito, Teruya, Akira, Iwata, Eiichiro, Zakaria, Nurul Azma, Matsumoto, Noriko, & Yoshida, Norihiko. 2008. Assertion-Based Dynamic Verification for Executable UML Specifications. *Pages 181–186 of: Proceedings of the 8th Conference on Applied Computer Science*. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS).

Timo, Omer Nguena, & Rollet, Antoine. 2010. Conformance Testing of Variable Driven Automata. *Pages 241–248 of: 8th IEEE International Workshop on Factory Communication Systems (WFCS)*.

Tschannen, Julian, Furia, Carlo A., Nordio, Martin, & Meyer, Bertrand. 2011. Usable Verification of Object-Oriented Programs by Combining Static and Dynamic Techniques. *Pages 382–398 of: Proceedings of the 9th International Conference on Software Engineering and Formal Methods*. SEFM'11. Berlin, Heidelberg: Springer-Verlag.

Utting, Mark, & Legeard, Bruno. 2006. *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Verwer, A.A. 1996 (março). Teaching Open Standards for PLC Programming. *Pages 8/1 –8/7 of: IEE Colloquium on Mechatronics in Education: Delivery of a New Engineering Discipline into the Workplace*.

Williams, B.C., Ingham, M.D., Chung, S.H., & Elliott, P.H. 2003. Model-Based Programming of Intelligent Embedded Systems and Robotic Space Explorers. *Proceedings of the IEEE*, **91**(1), 212–237.

Young, Michal, & Pezze, Mauro. 2005. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley & Sons.

Zhanyuan, Bai, Aidong, Xu, Mingzhe, Liu, & Yan, Song. 2010 (dezembro). A Novel Comparator with Hamming Code Correction for Safety Programmable Logic Controller. *Pages 410 –412 of: International Conference on Computational Problem-Solving (ICCP).*

Zhou, Qing, Liu, Bin, wei Yu, Zheng, & yu Feng, Shi. 2011 (junho). The Applied Technology of a Hardware-In-Loop Simulation Testing Platform for Reactive System Based on the Data-Playback. *Pages 1438 –1442 of: 6th IEEE Conference on Industrial Electronics and Applications (ICIEA).*