UNIVERSIDADE ESTADUAL DA PARAÍBA CAMPUS VII – GOVERNADOR ANTÔNIO MARIZ CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

AUGUSTO BEZERRA DE OLIVEIRA

GERAÇÃO AUTOMÁTICA DE AUTÔMATOS TEMPORIZADOS PARA ESPECIFICAÇÕES DE SISTEMAS INSTRUMENTADOS DE SEGURANÇA

PATOS – PB

2018

GERAÇÃO AUTOMÁTICA DE AUTÔMATOS TEMPORIZADOS PARA ESPECIFICAÇÕES DE SISTEMAS INSTRUMENTADOS DE SEGURANÇA

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de bacharel em Computação.

Orientador (A): Prof. Dra. Kézia de Vasconcelos Oliveira Dantas

PATOS – PB

2018

O48g Oliveira, Augusto Bezerra de.

Geração automática de autômatos temporizados para especificações de sistemas instrumentados de segurança [manuscrito] / Augusto Bezerra de Oliveira. - 2018.

50 p.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas , 2018.

"Orientação : Profa. Dra. Kézia de Vasconcelos Oliveira Dantas , Departamento de Computação - CCT."

1. Autômatos. 2. Sistemas de segurança. 3. Geração automática. I. Título

21. ed. CDD 004.2

Elaborada por Kênia O. de Araújo - CRB - 15/649 BC/UEPB

AUGUSTO BEZERRA DE OLIVEIRA

GERAÇÃO AUTOMÁTICA DE AUTÔMATOS TEMPORIZADOS PARA ESPECIFICAÇÕES DE SISTEMAS INSTRUMENTADOS DE SEGURANÇA

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciências da Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Bacharel em Computação

Aprovado em 14 de novembro de 2018

BANCA EXAMINADORA

Kézia de V. O. Dantas

Prof.ª Kézia Vasconcelos Oliveira Dantas
(Orientadora)

Cheyenne Ribeiro G. I. Abílio

Prof.ª Cheyenne Ribeiro Guedes Isidro Abílio
(Examinadora)

Rodrigo Alves Costa

Prof. Rodrigo Alves Costa
(Examinador)

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado saúde e forças para concluir este trabalho.

A Universidade Estadual da Paraíba pela direção e administração que permitiram concluir o curso de Bacharelado em Ciência da Computação.

A minha orientadora Prof. Dra. Kézia de Vasconcelos Oliveira Dantas por ter me guiado com todas as orientações, correções e incentivos.

Aos meus pais, pelo amor, incentivo e todo apoio necessário.

Aos meus amigos que incentivaram a continuar.

Aos professores que contribuíram com minha formação acadêmica.

E por fim, a todos que contribuíram direta ou indiretamente com o desenvolvimento deste trabalho.

Resumo

Os Sistemas Instrumentados de Segurança (SIS) são utilizados nas indústrias, eles têm o objetivo de garantir a segurança e o estado operacional dos sistemas industriais. Os SIS são sistemas críticos, pois, defeitos no hardware ou software podem ocasionar danos ao meio ambiente, instalações e nos funcionários que trabalham perto das áreas afetadas por uma possível situação indesejada. Para isto, faz-se necessário aplicar técnicas de verificação e validação para validar esses sistemas. Neste trabalho foi utilizada a técnica de testes baseados em modelos (*Model Based Testing* - MBT) que necessita de uma representação formal da especificação do sistema para gerar os testes. Este trabalho tem como objetivo a proposta de um método para gerar, de forma automática, uma rede de autômatos temporizados que representem formalmente a especificação do sistema. Autômatos que representam contadores incrementais (CTU), contadores decrementais (DTU), elementos de memória (MS) e (LS) foram gerados de forma automática. O método proposto foi aplicado em um estudo de caso para validar sua efetividade. Os resultados foram positivos e o método consegue informar um veredito da conformidade entre a especificação e a implementação do sistema.

Palavras-chave: Segurança, Testes, Autômatos.

Abstract

Safety systems (SIS) are used in industries, aiming to ensure the safety and operational functioning of industrial systems. SIS are critical systems because, defects no hardware or software can destroy the environment, facilities and employees who work near the areas affected by an undesirable situation. For this, we use verification and validation techniques to validate such systems, in this work we applied a model based testing technique (MBT), which consisted of a formal representation of the system specification to generate the tests. This work proposes a method to automatically generate a network of timed domains that formally represent a system specification. Automata representing incremental counters (CTU), decreasing counters (DTU), memory elements (MS) and (LS) were generated automatically. The same method was applied in a case study to validate its effectiveness. The results were positive and the method of obtaining an indicator of conformity between the implementation and the specification of the system.

Key words: Security, Test, Authomatons.

Lista de Ilustrações

Lista de reduções

CLP	Controlador Lógico Programável
CTC	Contador de Caixas
CTU	Contador Incremental
DTU	Contador Decremental
EG	Enchedor de Garrafas
HIL	Hardware In The Loop
LS	Elemento de Memória LS
MBT	Testes Baseados em Modelos
MS	Elemento de Memória MS
RCG	Reset de Esteira
ROBDD	Diagramas de Decisão Binária Ordenada e Reduzida
SIS	Sistemas Instrumentados de Segurança
SN	Sensor de Nível
SP	Sensor de Proximidade
STG	Sensor de Tipo de Garrafa
TAF	Teste de Aceitação de Fabrica
TP	Embalador de Garrafas
XML	Extensible Markup Language

Sumário

1. INTRODUÇÃO

Neste Capítulo é apresentada uma visão geral deste trabalho, de modo a descrever a contextualização do problema, objetivo e justificativa deste estudo.

1.1. Cenário Técnico Científico

Os Sistemas Instrumentados de Segurança (SIS) são utilizados nas indústrias, eles têm o objetivo de garantir a segurança e o estado operacional dos sistemas industriais. Os SIS são sistemas críticos, pois, defeitos no hardware ou software podem ocasionar danos ao meio ambiente, instalações e funcionários que ali trabalham. As operações dos SIS são executadas em Controladores Lógicos Programáveis (CLP) de Segurança (LJUNGKRANTZ *et al.,* 2012), uma classe de CLP projetada exclusivamente para áreas de segurança (BRYAN & BRYAN, 1997).

O processo de desenvolvimento de um programa de CLP para SIS consiste nos seguintes passos: primeiramente o projetista do sistema gera a especificação, a especificação deve estar no formato de diagramas de lógica binária ISA 5.2, o segundo passo é o desenvolvimento e teste do programa a partir da especificação, o programa pode ser testado á medida que é desenvolvido, e no terceiro passo o programa é enviado para o CLP.

Atualmente, a forma que as empresas testam os programas restringe-se em testar módulos de programas conforme os mesmos são desenvolvidos. Uma forma utilizada para detecção de erros que não foram encontrados durante o processo de teste é o comissionamento. É no comissionamento que ocorre o Teste de Aceitação de Fabrica (TAF); No decorrer do TAF as entradas das matrizes de causa/efeito são reproduzidas em campo com o intuito de comparar as entradas com as entradas especificadas nas matrizes. Utilizar o TAF para validar programas para CLP é uma prática onerosa, pois, ao combinar uma grande quantidade de variáveis o processo pode levar muito tempo para ser executado.

Para garantir a confiança e a segurança de um SIS faz-se necessário a utilização de técnicas de verificação e validação no decorrer do desenvolvimento de um programa de CLP para SIS.

1.2. Técnicas para verificação e validação de SIS

Segundo (GOBLE & CHEDDIE, 2005), o SIS é essencial para garantir a segurança e confiança no funcionamento do software, pois defeitos no hardware e/ou software podem provocar danos tanto dentro quanto fora das instalações. A vista disso torna-se essencial a utilização de técnicas de verificação e validação de programas de CLP para SIS.

Dentre as várias técnicas de verificação e validação de programas de CLP para SIS, podemos destacar as técnicas estáticas e dinâmicas (PATIL *et al.,* 2011); (PRESSMAN, 2006). Um exemplo de técnica de verificação estática é a verificação de modelos (KATOEN, 1999; FREY & LITZ, 2000); (TSCHANNEN *et al.*, 2011); Já ás técnicas dinâmicas fundamentam-se na execução de um programa com o intuito de encontrar erros (SOMMERVILLE, 2007). Um bom exemplo de técnicas de verificação dinâmicas são os testes e avaliação de asserção no código (GE *et al.,* 2011); (SUGAI *et al.,* 2008).

Com base na revisão bibliográfica realizada, pode-se constatar que a técnica de verificação de modelos é mais utilizada no cenário de programas para CLP. (BENDER *et al*., 2008) transforma programas Ladder em redes de Petri e efetua o teste de propriedades na ferramenta Tina. (CANER *et al*., 2000) mostra como transformar um programa desenvolvido em IL em uma estrutura de Kripke e, com o teste de propriedades, segundo a lógica temporal LTL pode ser analisada. (MOKADEM *et al*., 2010) utiliza o formalismo de redes de autômatos temporizados para modelar programas para CLP e a verificação dos modelos é executado na ferramenta Uppal-TRON. (GERGELY *et al*., 2010) estabelece o formalismo de redes de Petri e a verificação dos modelos para avaliar os programas para CLP. Entretanto, devido a um problema conhecido como explosão do espaço de dados, o uso desta técnica para verificar os programas torna-se improprio para sistemas mais complexos (KALITA & KHARGONEKAR, 2002); (PU & ZHANG, 2007); (SCHNEIDER & BRANDT, 2008).

O uso de testes é uma prática bastante utilizada para analisar programas de CLP para SIS. Teste é uma prática para analisar a segurança, qualidade e confiança das aplicações, e é geralmente utilizado para validar e verificar os programas, pois,

segundo (SOMMERVILLE, 2007) com o uso de testes é possível validar tanto requisitos funcionais quanto requisitos não funcionais de um sistema. No cenário de programas para CLP e sistemas em tempo real (COOLING, 2003) consideram-se dois tipos de técnicas para testes: testes baseados em modelos (Model Based Testing – MBT) e (UTTING & LEGEARD, 2006) e testes no código fonte.

A diferença entre os trabalhos onde a técnica de verificação MBT é utilizada está na forma de como as métricas de cobertura do modelo são estabelecidas; Em (LARSEN *et al.*, 2005); (OLIVEIRA *et al.*, 2010ª); (SILVA *et al.*, 2008) foi utilizado a escolha de variáveis de entrada aleatoriamente como métrica, já em (LI *et al.*, 2008) foi utilizado a métrica conjuntos de ações e estados; Nestes casos nem sempre é possível encontrar erros na implementação com os casos de teste gerados. Trabalhos que utilizam as métricas de fluxo de evento (TIMO & ROLLET, 2010) e cobertura de falhas Em (NOUAARY *et al*., 2002) não geram casos de teste para avaliar a forma de comportamento dos elementos temporizados. Nos trabalhos de (PEIXOTO *et al*., 2010) e (HESSEL & PETTERSSON, 2004); (JEE *et al*., 2006) os casos de teste que foram gerados tem um grau considerável de redundância, pois não encontram novos erros e não tem uma melhor cobertura, entre si ao usar respectivamente as métricas de heurística do fator determinante e cobertura de transições

O foco neste trabalho é nas técnicas de testes baseados em modelos, mais especificamente 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.

Em (OLIVEIRA *et al*., 2009, 2010 a, b, c) é mostrado um método que aumenta a confiança e segurança na automação de processos executados por CLPs. Para isto, casos de teste de conformidade são executados com o intuito de validar se o programa foi desenvolvido conforme a sua especificação. No método, modelos de autômatos temporizados são gerados de forma automática conforme os requisitos da ferramenta Uppal-TRON, a partir dos seguintes itens: Diagramas ISA 5.2, que representam a especificação do sistema; E softwares, desenvolvidos em Ladder, que representam a implementação do sistema. Os testes de conformidade são realizados na ferramenta Uppal-TRON logo após a ser executada a geração dos modelos.

Utilizando a técnica de teste sob o código fonte, a validação de propriedades particulares é efetuada diretamente no código fonte do sistema que está sendo testado. Nesta técnica o hardware dedicado é introduzido no processo de teste, através da técnica HIL (do inglês, *Hardware in the loop*) (MICHALEK *et al*., 2005); (BACIC, 2005); (CRACIUN *et al*., 2010); (RANKIN & JIANG, 2011) onde o sistema tem o seu comportamento testado em tempo real.

Em (OLIVEIRA *et al*., 2013) é utilizado classes de equivalência para criar os casos de testes com base em diagramas ISA 5.2. A validação entre a especificação e a implementação do programa é executada com os casos de teste que foram gerados. Em sua tese de doutorado, Oliveira, utilizando a estrutura de diagramas de decisão binária ordenada e reduzida (ROBDD) e o formalismo de redes de autômatos temporizados, gerou os casos de teste de conformidade. Porém, apenas elementos temporizados em conjunto com a lógica booleana do sistema foram modelados e testados. Neste trabalho busca-se a modelagem automática de mais elementos ISA, para que no futuro casos de testes possam ser gerados de forma automática.

1.3. Problemática

Para validar os Sistemas Instrumentados de Segurança utilizando a técnica de validação de testes baseados em modelos (MBT) faz-se necessário um modelo formal para a representação da especificação do sistema, para tanto, será necessário modelar os diagramas de lógica binária ISA 5.2.

Diante do exposto, evidenciamos o seguinte problema de pesquisa: como validar programas de Controladores Lógicos Programáveis para Sistemas Instrumentados de Segurança utilizando uma representação formal da especificação?

1.4. Justificativa

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 tanto, para realizar a

verificação é necessário ter um modelo que represente a especificação de maneira formal. Neste trabalho pretende-se gerar de forma automática, modelos de autômatos temporizados a partir de diagramas ISA 5.2.

1.5. Objetivos

1.5.1. Objetivo Geral

O objetivo geral neste trabalho enquadra-se na formação de bases formais para a elaboração de métodos para a modelagem da especificação de sistemas instrumentados de segurança. Neste trabalho, a especificação do sistema é dada na forma de diagramas de lógica binária ISA 5.2.

1.5.2. Objetivos Específicos

No contexto deste trabalho de modelagem de bases formais para sistemas instrumentados de segurança, buscam-se os seguintes objetivos específicos:

- Estudar os diagramas de lógica binária ISA 5.2;
- Estudar o formalismo de rede de autômatos temporizados;
- Identificar um conjunto significativo de elementos ISA 5.2, que possam representar a especificação do sistema, para serem modelados;
- Desenvolver um método para gerar autômatos temporizados a partir de diagramas ISA 5.2;
- Desenvolver uma ferramenta para gerar de forma automática os modelos de autômatos temporizados;
- Modelar estudos de caso reais.

1.6. Metodologia

- **Revisões Sistemáticas** na literatura referente ao tema do trabalho. Neste ponto foram utilizadas ferramentas como o portal periódico da Capes, IEEE e ACM; também foram utilizados livros, teses e dissertação;

- **Identificação de protocolos e conceitos a serem analisados**. Nesta etapa foram utilizados estudos com relação as linguagens e formalismos da especificação utilizada para representar o SIS;

- **Desenvolvimento do método para geração automática dos autômatos**. Nesta etapa foi desenvolvido o método para gerar, de forma automática, uma rede de autômatos temporizados a partir da especificação do sistema.

- **Desenvolvimento de uma ferramenta para modelar a especificação do sistema e gerar a rede de autômatos**. Nesta etapa foi desenvolvida uma ferramenta para modelar os diagramas de lógica binária ISA 5.2 e a partir dos diagramas ISA, a ferramenta consegue gerar, de forma automática, uma rede de autômatos temporizados que representam formalmente a especificação do sistema.

2. REFERENCIAL TEÓRICO

O objetivo desta sessão é prover um principio teórico fundamental para o entendimento deste trabalho. São retratados os principais conceitos pertencentes a controladores lógicos programáveis (CLP), diagramas de lógica binária ISA 5.2 e uma rede de autômatos temporizados.

2.1. Controladores Lógicos Programáveis

O Controlador Lógico Programável, também conhecido por suas siglas (CLP), surgiu no final da década de 1960, na empresa General Motors, com o objetivo de suceder os relés, pois, a cada nova modificação na linha de montagem desta empresa, a lógica de controle dos painéis tinha que ser reprogramada, fazendo com que tal encargo fosse oneroso (Parr, 2003).Os CLPs melhoraram e passaram a ser mais versáteis e de simples aplicação tornando-se uma das ferramentas mais utilizadas na automação industrial. Segundo a ABNT (Associação Brasileira de Normas e Técnicas), um controlador lógico programável é um equipamento eletrônico digital com hardware e software conciliáveis com aplicações industriais.

Uma das vantagens fundamentais da utilização dos controladores lógicos programáveis, quando comparados a outros dispositivos de gerenciamento industrial, são (JOHN & TIEGELKAMP, 2001); (PARR, 2003):

- Menor espaço ocupado
- Menor potência elétrica necessária;
- Programação e manutenção executada de uma forma simples;
- Maior flexibilidade
- Apresenta uma interface de comunicação com outros CLPs e computadores de controle

A seguir, na Figura 1 é apresentado o esquema de um CLP, o CLP é dividido em três partes: Entradas, Saídas e a Unidade Central de Processamento (CPU).

Figura 1: Esquema de um CLP Fonte: (Bryan & Bryan, 1997).

O CLP tem um funcionamento sequencial, onde ciclos de varreduras são executados em etapas, apenas uma etapa é executada por vez. O tempo para executar cada ciclo é denominado tempo de varredura.

Quando o ciclo de varredura de um CLP é iniciado, o seu funcionamento segue os seguintes passos: Primeiramente as entradas do sistema são lidas, no segundo passo o programa é executado e na terceira etapa os valores das saídas são liberados. Na Figura 2 é apresentada a sequência de etapas do ciclo de varredura de um CLP

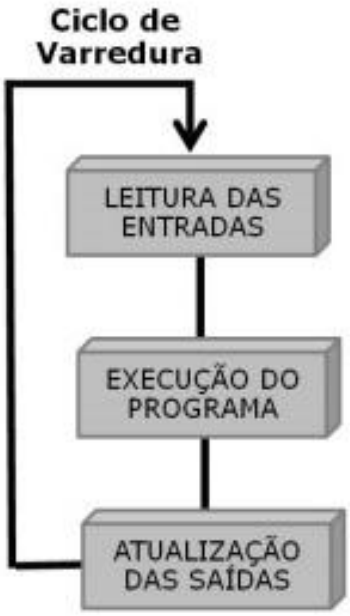


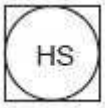
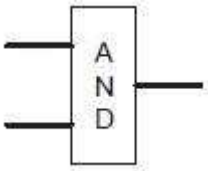
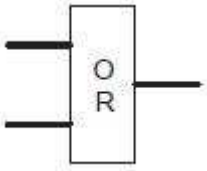
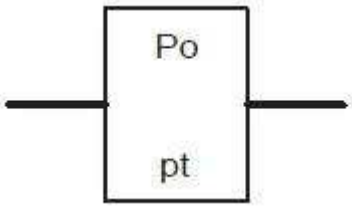
Figura 2: Etapas do ciclo de varredura de um CLP. Fonte: (Oliveira, 2014)

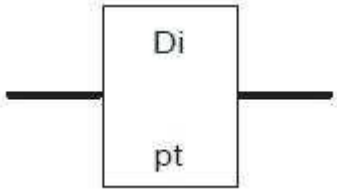
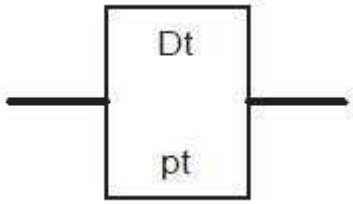
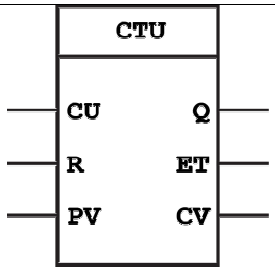
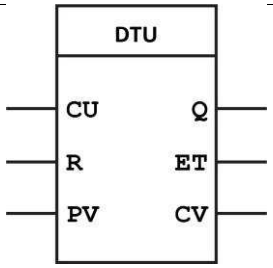
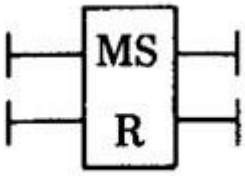
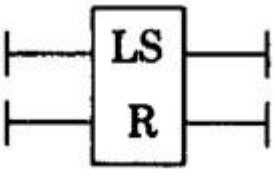
2.2. Diagramas de Lógica Binária ISA 5.2

O padrão ISA 5.2 foi desenvolvido com o intuito de prover um método de diagramação de lógica de entre ligamento binário e sequenciamento de sistemas. Este padrão foi projetado para simplificar a clareza do funcionamento de sistemas binários e facilitar a comunicação entre os técnicos, gerentes, projetistas e as pessoas responsáveis por operar e manter os sistemas (ISA, 1992).

O padrão ISA 5.2 fornece símbolos para descrever as operações dos processos por meio de funções operacionais binárias. Alguns destes símbolos são retratados na Tabela 1.

Tabela 1: Elementos 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 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.

	Temporizador DI (Delay Initiation of output): a saída é ativada depois de um período de tempo quando existe lógica 1 na entrada
	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
	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).
	Contador decremental. Pulsos positivos em CU são contados até que o valor mínimo 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).
	Representa o elemento de memória *Maintain Supply* (MS), assim que a entrada for ativada a saída é ativada, caso a entrada seja desativada, a saída se mantem ativa
	Representa o elemento de memória *Lost Memory* (LS), assim que a entrada for ativada a saída é ativada, caso a entrada seja desativada, a saída também é desativada

Fonte: (Oliveira, 2014).

O diagrama ISA 5.2 de um exemplo representado por (Parr, 2003) é demonstrado na Figura 3. No exemplo é exposto o funcionamento de um motor onde o mesmo é ligado e desligado por meio dos respectivos botões *Iniciar* e *Parar*. Este sistema também é composto por um contato auxiliar, *AuxIniciar*, que é empregado para energizar o motor. No caso de um erro ser cometido, por causa de uma sobrecarga, uma parada de emergência ser solicitada, ou se ocorrer algum tipo de falha no abastecimento, o sinal do contato auxiliar será perdido. Este contato só pode ser verificado 5 segundos após *Motor* ter sido energizado. A entrada *Manual* simboliza a ativação de um alarme caso alguma falha aconteça.

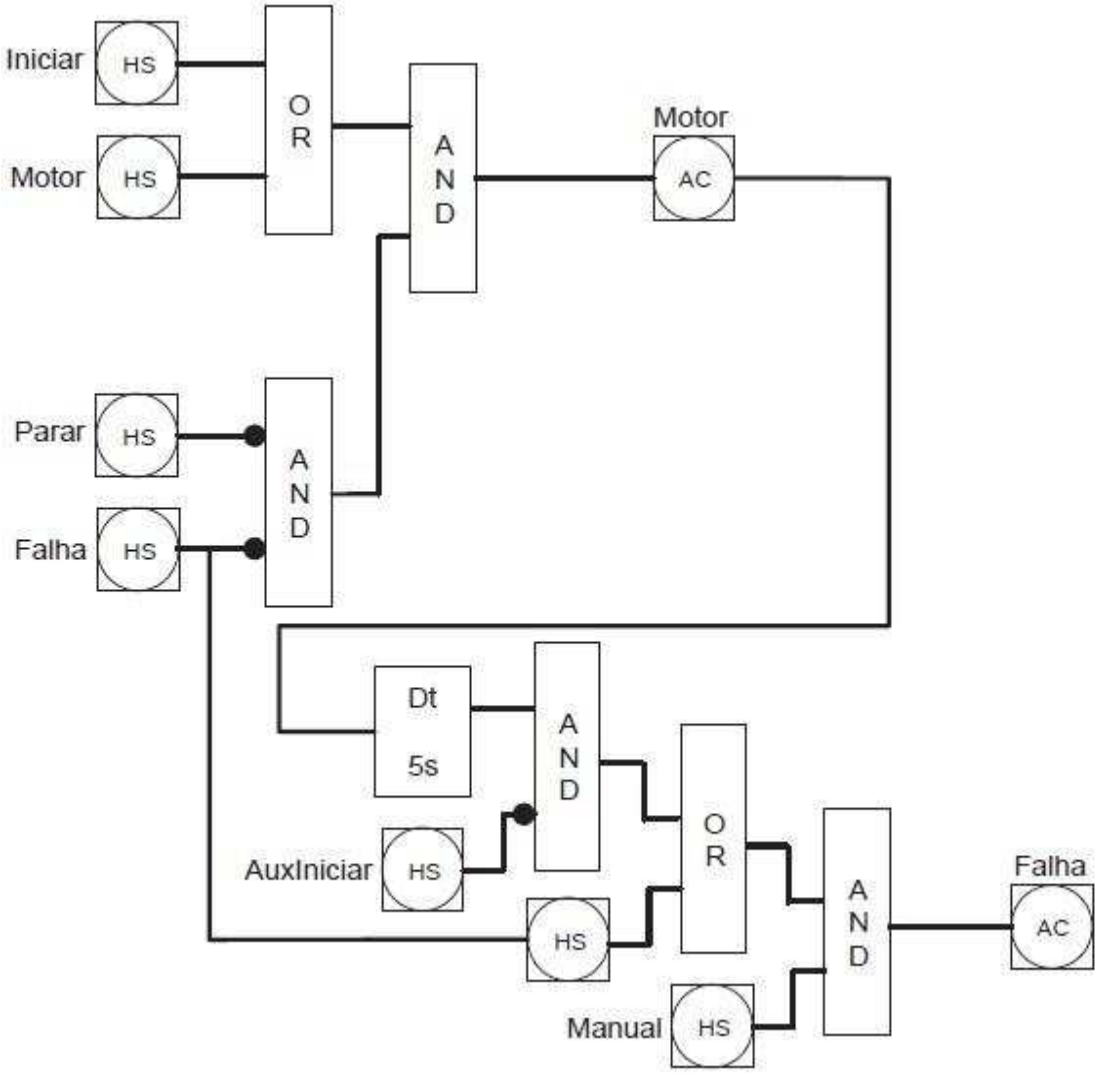


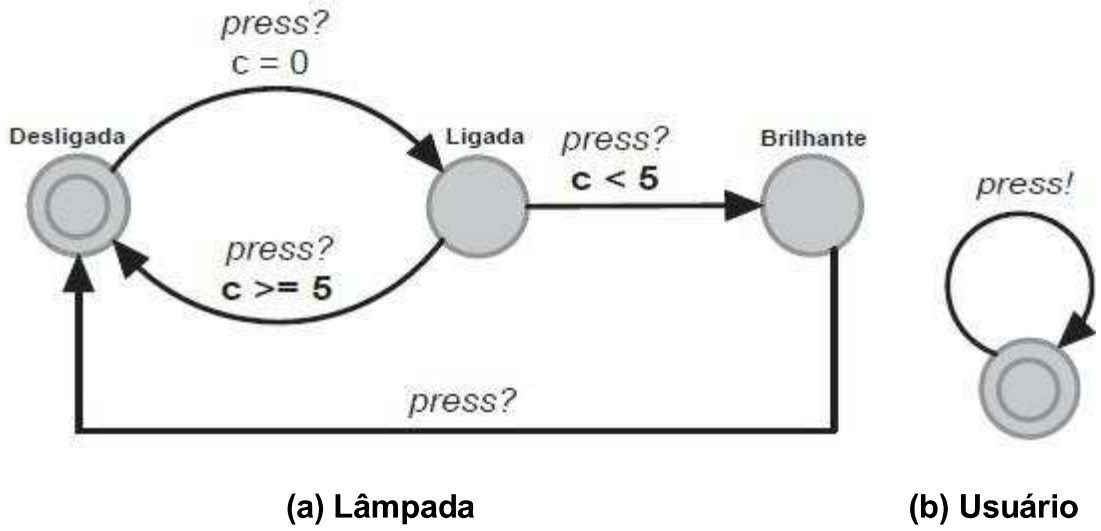
Figura 3: Diagrama ISA 5.2 para o funcionamento de um motor. Fonte: (Oliveira, 2014).

2.3. Autômatos Temporizados

Autômatos temporizados são autômatos de estados finitos com restrições de temporização relacionadas as suas arestas e localidades, são utilizados para projetar o comportamento de sistemas de tempo real (Alur & Dill, 1994; Bengtsson & yi, 2004). As restrições dos autômatos são compostas segundo variáveis de gerenciamento de tempo intituladas de relógios.

Na Figura 4 uma simples colocação da sintaxe e semântica de um conjunto de autômatos temporizados na ferramenta Uppal-TRON é apresentada. Para isso, a atividade de uma lâmpada é modelada. O autômato que simboliza a lâmpada, é

demonstrado na Figura 4(a), é constituído por três localidades: ligada, desligada e brilhante. As expressões *c = 0* simbolizam a atualização de variáveis. As sincronias são demonstradas em itálico. Esta rede de autômatos funciona da seguinte forma: Caso o interruptor seja pressionado, ou seja, sincronizar com *press?*, O estado da lâmpada muda para ligado, Caso o usuário acione o interruptor outra vez, a lâmpada muda para o estado desligado; Caso o usuário seja rápido, e pressione o interruptor duas vezes, a lâmpada fica ligada e torna-se brilhante. O modelo de usuário é representado na Figura 4(b), O usuário tem liberdade para acionar o interruptor da forma que quiser ou nunca o pressionar. O relógio *c* é usado para detectar se o usuário pressionou de forma rápida (*c < 5)* ou se o usuário pressionou de forma lenta (c>=5).



(a) Lâmpada **(b) Usuário**

Figura 4: Exemplo simples de uma lâmpada. Fonte: (Oliveira, 2014).

A seguir serão demonstradas algumas descrições formais da sintaxe e semântica dos autômatos temporizados. As notações a seguir serão utilizadas: C representa o conjunto de relógios e B(C) um conjunto de conjunções sobre condições simples da forma x $\triangleleft\triangleright$ c ou x-y $\triangleleft\triangleright$ c, de modo que x, y ∈ C, c ∈ N e $\triangleleft\triangleright$ ∈ {<, ≤, =, ≥, > }.

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 ∈ 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. Seja *Rc* Ser o conjunto de todas as valorações dos relógios. Façamos $u0(x) = 0$ 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 $u \in g$, $u' = [r \rightarrow 0]u$, 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 $[r \rightarrow 0]u$ denota a valoração do relógio que leva cada relógio de r até 0, ou seja, todos os relógios são restaurados.

Os autômatos temporizados são geralmente compostos por uma rede de autômatos temporizados sobre um conjunto comum de relógios e ações, constituindo *n* autômatos temporizados, $Ai = (Li, l0\ i , C, A, Ei, Ii)$, em que $1 \leq i \leq n$. Um vetor de localidade é um $\_l = (l1, ..., ln)$. As funções invariantes de cada autômato da rede são compostas numa função comum sobre os vetores de posição $I(\_l) = \wedge\, ili(li)$. Denota-

se por *_l[l′ i / li]* o vetor onde o i-ésimo elemento *li de*_*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^0*, C, A,* E_i*,* l_i*)* uma rede de n autômatos temporizados. E seja $\bar{l}_0$ = *(*l^0*, ...,* l^0 *)* o vetor de localidade inicial. A semântica de uma rede de autômatos temporizados é definida como um sistema de transição ⟨*S, s*0, →⟩, 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 →⊆ *S x S* é a relação de transição definida por:

- *(l, u)* → (*l*, u + d) se ∀ d´: 0 ≤ d´ ≤ d ⇒ u + d´∈ I(*l*), e
- *(l, u)* → (*l* [li′ / li], u') se existe li →rgr li′ tal que u ∈ g, u' = [r ›→ 0]u e u'∈ I(*l*)
- *(l, u)* → (*l* [lj′ / lj , li′ / li], u') se existe li →c?giri li′ e lj →c!gj rj lj′ tal que u ∈ (gi ∧ gj), u´=´[ri ∪ rj ›→ 0]u e u'∈ I(*l*)

Os autômatos temporizados na ferramenta Uppal-TRON são extensões dos autômatos temporizados com as propriedades adicionais a seguir:

- Templates: São um conjunto de parâmetros que podem ser de vários tipos(int, chan). Estes parâmetros são trocados por um argumento na definição de processos;
- Constantes: São variáveis que são definidas como *const nome valor*. As constantes tem valor inicial do tipo inteiro que é declarado na definição e não pode ser modificado.
- Arrays: São utilizados para canais, constantes, variáveis inteiras e relógios. São definidos adicionando um tamanho ao nome da variável, por exemplo, int a[10];

- Inicializadores: São utilizados para iniciar as variáveis inteiras e arrays de variáveis inteiras. Por exemplo, int j = 5 ou int j[2] = 3,4;
- Variáveis inteiras limitadas: São definidas como *int[min, max] nome*, onde os termos *min* e *max* definem os limites inferiores e superiores respectivamente. As guardas, invariantes e atribuições podem ser criadas por expressões que variam sobre as variáveis inteiras limitadas.
- Sincronização binária: São declarados como *chan syn*. Uma aresta com o atributo *syn!* é sincronizada com outra aresta com o atributo *syn?*. Um par de sincronizações é definido de forma não determinística se forem permitidas diversas combinações;
- Canal de broadcast: É declarado como *broadcast chan bc*. Em uma sincronização um remetente *syn!* pode ser sincronizado com um número qualquer de receptores *syn?*. Caso não tenha nenhum receptor, então a ação *syn!* pode ser executada, a transmissão não é bloqueada;

3. MÉTODO PARA GERAÇÃO AUTOMÁTICA DE AUTÔMATOS TEMPORIZADOS

3.1. Processo de Desenvolvimento de um Programa de CLP para SIS

Os SIS utilizam uma metodologia de desenvolvimento *top-down*, que conta com os seguintes atores:

- Projetista do sistema: Ator responsável por gerar a especificação do sistema, esta especificação deve estar na forma de diagramas de lógica binária ISA 5.2.
- Programador do sistema: Ator responsável por desenvolver o programa; O programa deve ser desenvolvido em uma das linguagens do padrão IEC61131-3.

- Testador: Ator responsável pelos testes no sistema que foi desenvolvido pelo programador. O testador do sistema irá utilizar o método proposto neste trabalho.

Esta metodologia de desenvolvimento, conforme é demonstrado na Figura 5, também conta com as seguintes etapas:

1. Coleta de requisitos;
2. Geração da especificação do sistema;
3. Desenvolvimento e teste do código por uma equipe especialista em automação industrial
4. Comissionamento, onde irá decidir se o código será aceito ou se precisa de ajustes;

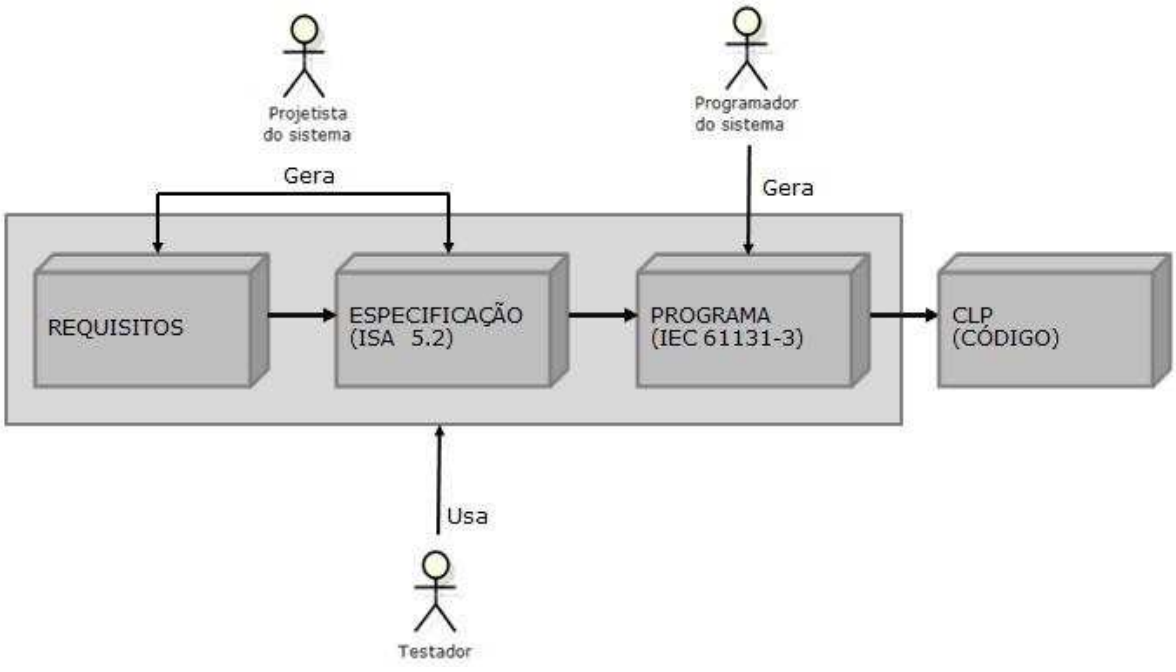


Figura 5: Processo de desenvolvimento de um programa de CLP para SIS. Fonte: (Oliveira, 2014).

O método proposto funciona da seguinte maneira: Após o projetista do sistema fornecer a especificação do sistema, a especificação é convertida de forma automática em uma rede de autômatos temporizados. Com a rede de autômatos temporizados é possível gerar os casos de teste. Para esta tarefa, foi desenvolvida uma ferramenta que é capaz de gerar um conjunto de casos de teste onde contadores, elementos de memória e a lógica do sistema pode ser avaliada. Após esta tarefa ser executada, as entradas dos casos de teste são enviadas para o CLP, onde estará a implementação do sistema que vai ser avaliado. E por fim, é dado um veredito sobre a conformidade entre a implementação e a especificação do sistema, existem dois tipos de veredito a serem obtidas: *conforme*, onde as saídas esperadas foram obtidas, e *não conforme*, onde as saídas esperadas não foram obtidas.

A especificação do sistema permite que o testador possa, utilizando o método proposto, testar o sistema conforme o mesmo ainda estiver sendo desenvolvido. A formação dos modelos de autômatos temporizados e dos casos de teste é transparente para o testador, pois ele consegue o veredito dos testes apenas informando a especificação e a implementação do sistema.

3.2. Processo de Envio e Recebimento de Dados para o CLP

Na Figura 6 é apresentado o fluxograma para o processo de envio e recebimento de dados entre os modelos de autômatos e o programa de CLP para SIS. Este fluxograma tem as seguintes etapas:

1. Leitura da especificação do sistema, esta especificação deve estar em formato de diagramas de lógica binária ISA 5.2;
2. Geração de uma rede de autômatos temporizados com base na especificação;
3. Geração dos casos de teste a partir da rede de autômatos;
4. Envio das entradas dos casos de teste para o CLP;
5. Recebe os valores das saídas do programa;

6. Compara os valores das saídas recebidas do programa com as saídas geradas nos casos de teste e informa o veredito

O foco neste trabalho é na geração automática da rede de autômatos temporizados, passo dois.

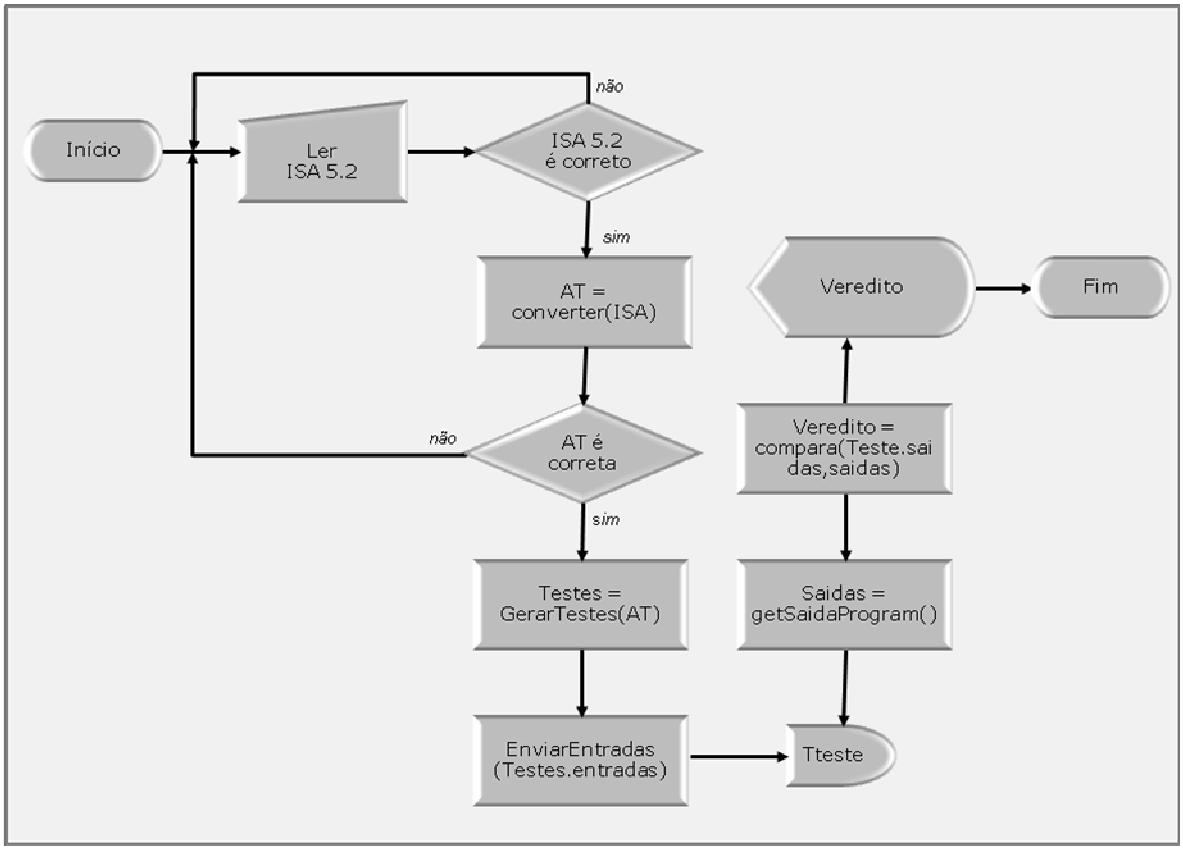


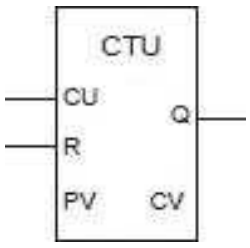
Figura 6: Fluxograma para processo de envio e recebimento de dados para o CLP. Fonte: (Oliveira, 2014)

3.3. Modelos de Autômatos Temporizados

Os modelos de autômatos temporizados são gerados com base nos diagramas ISA 5.2. Para tanto, foram utilizados os seguintes modelos: autômato que representa um contador de Incremento (CTU), autômato que representa um contador de decremento (DTU), autômato que representa um elemento de memória (MS), autômato que representa um elemento de memória (LS) e autômato que representa o ciclo de varredura de um CLP.

Na Figura 7(a) é apresentado um contador incremental. Neste contador, pulsos positivos em CU são contados até que o valor máximo PV seja atingido. Neste instante

o valor de contagem (CV) é pausado e a saída do contador (Q) é ativada (Q = 1). Quando a entrada R (Reset) for ativada (R = 1) o contador e a saída são restaurados. Na Figura 7(b) é apresentada a tabela verdade para a saída deste contador.



CU	PV	Q
1	>= CV	0
1	< CV	1
0	-	0

Figura 7(a): Elemento ISA 5.2 Figura 7(b): Tabela Verdade para saída Q

Figura 7: Contador Incremental

Para definirmos o autômato para contadores incrementais vamos fazer as seguintes considerações. O autômato é composto por três localidades: *Inicio_L1, L2* e *L3*. Estas localidades representam as condições correspondentes do contador. A tabela apresentada na Figura 7(b) será utilizada para determinar a mudança entre as localidades do autômato.

Para cada mudança de localidade um arco será gerado, no qual as entradas representarão as guardas e as saídas representarão as atribuições. A primeira linha será utilizada para o contador mover da localidade *Inicio_L1* para a localidade L2, A segunda linha será utilizada para o contador mover da localidade L2 para a localidade L3. A terceira linha será utilizada para o contador mover da localidade L3 para a localidade *Inicio_L1*, As funções *Addition(numCont)* e *Reset_cont(numCont)* são utilizadas para atualizar e voltar o contador ao seu estado inicial, respectivamente, A função *ReadInput()* é utilizada para ativar a saída do contador (Q). A seguir, apresenta-se a definição formal do autômato que representa o comportamento de um contador incremental CTU. O sufixo e a variável numCont são utilizados para determinar a ordem de execução lógica dos autômatos que representam cada contador.

Definição formal para um contador incremental: Um autômato temporizado para um contador incremental é a sêxtupla (L, L0, C, Ac, E, I), tal que:

- L = {Inicio_L1, L2, L3};
- L_0 = Inicio_L1;
- C =;
- Ac = {synchro};
- E = {E1,E2,E3,E4,E5};Tal que:
    - E1 = {Inicio_L1, synchro?, executou*NumCont* && checkExecution(numCont),{Addition(numCont),executou_indice(numCont)},L2};
    - E2 = {L2, synchro?, cv_ numCont < PV_ numCont && !executou_ numCont && checkExecution(numCont),{ Addition(numCont),executou_indice(numCont)},L2};
    - E3 = {L2, synchro?, cv_ numCont >= PV_ numCont && !executou0,{ Output_IN(numCont),executou_indice(numCont)},L3};
    - E4 = {L2, synchro?, !executou_ numCont && BRESET == true, { Reset_Cont(numCont),executou_indice(numCont)}, L1};
    - E5 = {L3, synchro?, !executou_ numCont && BRESET, { Reset_Cont(numCont),executou_indice(numCont)}, L1};

Na Figura 8 apresenta-se o autômato para um contador incremental. A execução deste autômato é da seguinte forma: Quando o autômato que representa o ciclo de varredura do CLP for sincronizado com *synchro?* e a variável *executou_numCont* for verdadeira (*executou_numCont = true*), então, o arco com origem na localidade *inicio_L1* e destino na localidade *L2* é ativado e a variável *executou_numCont* é atualizada para *true*. A partir deste ponto podem ocorrer três situações. A primeira ocorre quando a variável *executou* for *false* (*!executou_numCont*) e a entrada do contador estiver desativada (*!checkExecution(numCont)*), desta forma, o arco com origem na localidade *L2* e como destino na localidade *inicio_L1* é ativado e valor de *CV* é restaurado (*CV = 0*). A segunda ocorre quando a entrada do temporizador for verdadeira, a variável *executou_numCont* tiver valor *false* e a contagem de pulsos do contador for menor que *PV* (*cv_ numCont < PV_ numCont && !executou_ numCont && checkExecution(numCont)),* desta forma o contador entra em loop na localidade *L2* e a cada valor de entrada verdadeiro o valor de *cv* é incrementado na função

Addition(numCont) e a variável *executou_numCont* é atualizada para *true*. A terceira ocorre quando a entrada do temporizador for verdadeira, a variável *executou_numCont* tiver valor *false* e a contagem de pulsos do contador for maior ou igual a *PV* (*cv_numCont >= PV_numCont && !executou_ numCont*), desta forma a saída do contador será ativada (*Output_In(*numCont*)*). Na localidade L3, quando a saída do contador for falsa, o valor de CV é restaurado e a saída do temporizador é desativada (*Reset_Cont(numCont))*. Detalhes sobre as funções do autômato que representa um contador incremental podem ser encontradas no Apêndice A.

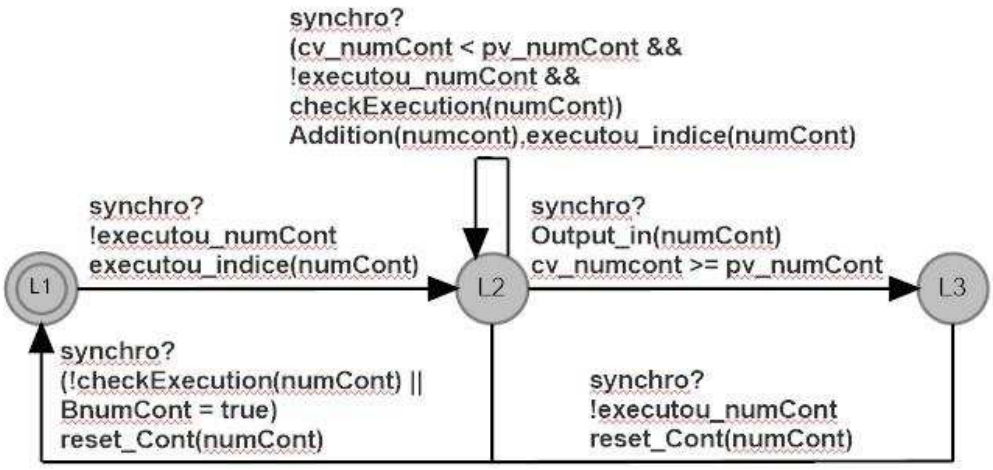
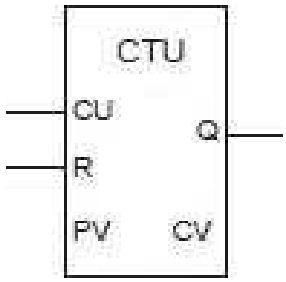


Figura 8: Autômato que representa um contador incremental

Na Figura 9(a) é apresentado um contador de decremento. Para este, pulsos positivos em CU são contados e a variável CV é decrementada até que o seu valor seja igual ao valor de PV. Neste instante o valor de contagem CV é pausado e a saída Q é ativada (Q = 1). Quando a entrada R (*Reset)* for ativada (R = 1) o contador e a saída são restaurados. Na Figura 9(b) é apresentada a tabela verdade para a saída deste contador



CU	PV	Q
1	<= CV	1
1	> CV	0
0	-	0

Figura 9(a): Elemento ISA 5.2 Figura 9(b): Tabela Verdade para a saída Q

Figura 9: Contador Decremental.

Para definirmos o autômato para contadores decrementais devemos fazer as seguintes considerações. O contador é composto por três localidades: *Inicio_L1*, L2 e L3. Estas localidades representam as condições correspondentes do contador. A tabela apresentada na Figura 9(b) será utilizada para determinar a mudança entre as localidades do autômato.

Para cada mudança de localidade um arco será gerado, no qual as entradas representarão as guardas e as saídas representarão as atribuições. A primeira linha será utilizada para o contador mover da localidade *Inicio_L1* para a localidade L2. A segunda linha será utilizada para o contador mover da localidade L2 para a localidade L3. A terceira linha será utilizada para o contador mover da localidade L3 para a localidade *Inicio_L1*. As funções *Decremento_cont(numCont)* e *Reset_Cont(numCont)* são utilizadas, respectivamente, para decrementar e voltar o contador ao seu estado inicial. A função *OutPut_IN(numCont)* é utilizada para controle. A seguir, apresenta-se a definição formal do autômato que representa o comportamento de um contador decremental. O sufixo e a variável numCont são utilizados para determinar a ordem de execução lógica dos autômatos que representam cada contador.

Definição formal do contador de decremento: Um autômato temporizado para um contador decremental é a sêxtupla (L, L0, C, Ac, E, I), tal que:

- L = {Inicio_L1,L2,L3};

- L_0 = Inicio_L1;
- C =;
- Ac = {synchro);
- E = {E1,E2,E3,E4,E5};Tal que:
  - E1 = {Inicio_L1, synchro?, !executou0 && checkExecution(*numCont*),{Decremento_cont(*numCont*),executou_indice(*numCont*)},L2};
  - E2 = {L2, synchro?, cv_*numCont* > PV_*numCont* && !executou_*numCont* && checkExecution(*numCont*),{ Decremento_cont(*numCont*),executou_indice(*numCont*)},L2};
  - E3 = {L2, synchro?, cv_*numCont* <= PV_*numCont* && !executou_*numCont* ,{ Output_IN(*numCont*),executou_indice(*numCont*)},L3}; ○ E4 = {L2, synchro?, !executou_*numCont* && BRESET == true, { Reset_Cont(*numCont*),executou_indice(*numCont*)}, L1};
  - E5 = {L3, synchro?, !executou_*numCont* && BRESET, { Reset_Cont(*numCont*),executou_indice(*numCont*)}, L1};

Na Figura 10 apresenta-se um autômato para um contador decremental. A execução deste autômato é da seguinte forma: quando o autômato que representa o ciclo de varredura do CLP for sincronizado com *synchro?* E a variável *executou_numCont* for falsa (*executou_numCont* = *false*), então, o arco com origem na localidade *inicio_L1* e destino na localidade L2 é ativado, então a função *executou_indice(numCont)* atualiza a variável *executou* para *true*. A partir deste ponto podem ocorrer três situações. A primeira ocorre a variável *executou_numCont* for *false* (*!executou_numCont*) e a entrada do contador estiver desativada (!checkExecution(*numCont*)), desta forma, o arco com origem na localidade L2 e como destino na localidade L1 é ativado e o valor de *CV* é restaurado (*CV* = *PV*). A segunda ocorre quando a entrada do temporizador for verdadeira, a variável *executou* tiver valor *false* e a contagem de pulsos do contador for maior que *PV* (*cv_numCont* > *PV_numCont && !executou_numCont && checkExecution(numCont)*), desta forma o

contador entra em loop na localidade *L2* e a cada valor de entrada verdadeiro o valor de *cv* é decrementado na função *Decremento_cont(numCont)* e a variável *executou_numCont* é atualizada para *true* na função *executou_indice(numCont).* A terceira ocorre quando a entrada do temporizador for verdadeira, a variável *executou* tiver valor *false* e a contagem de pulsos do contador for menor ou igual a *PV* (*cv_numCont* <= *PV_numCont* && *!executou_numCont*), desta forma a saída do contador será ativada (*Output_In(numCont)*). Na localidade L3, quando a saída do contador for falsa, o valor de CV é restaurado e a saída do temporizador é desativada (*Reset_Cont(numCont)*). Detalhes sobre as funções do autômato que representa um contador decremental podem ser encontradas no Apêndice B.

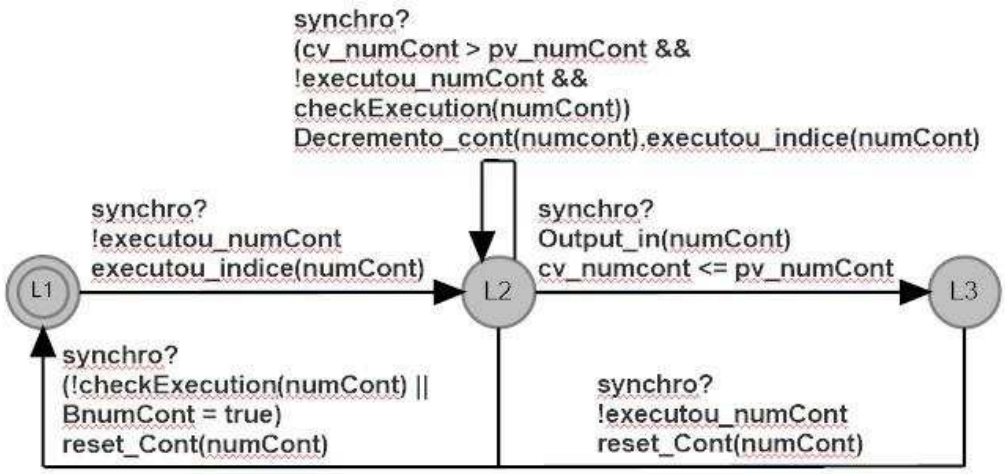


Figura 10: Autômato temporizado para um contador decremental

Na Figura 11 é apresentado um autômato que representa o elemento de memória MS (*Maintain Supply)*. Para este elemento, a saída lógica Q existe assim que a entrada for verdadeira e o evento *reset* não esteja ativado. Mesmo que a entrada tenha seu valor modificado para *false* ou o evento *Reset* seja verdadeiro, o autômato MS consegue manter a saída ativada independentemente do estado subsequente da entrada.

Para definirmos um autômato que representa o elemento de memória MS devemos fazer as seguintes considerações, o autômato é composto por duas localidades, L1 e L2. Estas localidades representam as condições correspondentes do elemento de memória MS. A função *SetMemory()* é utilizada para ativar a saída do

elemento MS (Q = *true)*, o canal de sincronização *synchro?* É utilizado para sincronizar este autômato com o autômato que representa o ciclo de varredura. As variáveis *executou* e *onFirstTime* são utilizadas para controle. A variável *A* representa a entrada do elemento MS e *B* representa o reset do elemento MS. A seguir, apresenta-se a definição formal do autômato que representa o comportamento de um elemento de memória MS.

Definição formal do autômato que representa o elemento de memória MS: Um autômato temporizado para um autômato MS é a sêxtupla (L, L0, C, Ac, E, I), tal que:

- *L = { L1, L2};*
- *L0 = L1;*
- *C = ;*
- *Ac = synchro?;*
- *E = { E1, E2, E3 }; tal que:*
  - *E1 = (L1, synchro?, (A && executou == false)* || *(onFirstTime == true && B && executou == false), { SetMemory(), executou_indice(), onFirstTime = true }, L2);*
  - *E2 = (L2, synchro?, !B && executou == false, { SetMemory(),executou_indice()}, L2);*
  - *E3 = (L2, synchro?, B && executou == false, { SetMemory(),executou_indice()}, L1).*

A execução do autômato que representa o elemento de memória MS é da seguinte forma: quando o autômato que representa o ciclo de varredura do CLP for sincronizado com o *synchro?*, a variável *executou* estiver desativada *(!executou)* ou a variável *onFirstTime* estiver ativada (*onFirstTime = true*) o arco com origem na localidade L1 e destino na localidade L2 é ativado a função *SetMemory()* é ativada, a função *executou_indice()* atualiza a variável *executou* para *true* e a variável *onFirstTime* é atualizada para *true* (*onFirstTime = true*). A partir deste ponto podem ocorrer duas situações. A primeira ocorre quando *B* e *executou* estiverem desativados (!B && executou == false), desta forma, o arco com origem e destino na localidade *L2* é ativado. Isto irá ocorre até que o valor de *B* seja *true*. A segunda situação ocorre quando a variável *B* tem valor *true* e *executou* tiver valor *false* (*B && executou == false)*, então o arco com origem na localidade *L2* e destino na localidade *L1* e a função

SetMemory() são ativados, garantindo que mesmo que o autômato seja restaurado a saída Q seja mantida.

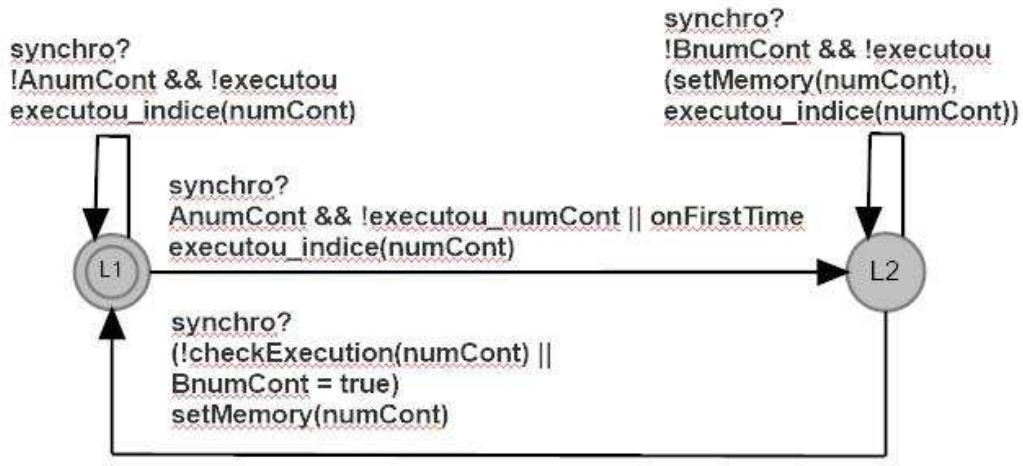


Figura 11: Autômato temporizado para um elemento de memória MS

Na Figura 12 é apresentado um autômato que representa o elemento de memória LS (*Lost Supply)*. Para este elemento, a saída lógica *Q* existe assim que a entrada for verdadeira e o evento *reset* não esteja ativado. A saída *Q* permanece ativa, se e somente se, a entrada do autômato for verdadeira e o autômato não for restaurado. Caso o autômato seja restaurado, a saída *Q* é desativada, pois o autômato não consegue manter a saída ativada em caso de restauração.

Para definirmos um autômato que representa o elemento de memória LS devemos fazer as seguintes considerações, o autômato é composto por três localidades inicio_L1, L2, e L3. Estas localidades representam as condições correspondentes do autômato. Para cada mudança de localidade um arco será gerado, no qual as entradas representarão as guardas e as saídas representarão as atribuições. A função *SetMemory()* é utilizada para ativar a saída do elemento LS (Q = *true*), a função *Reset()* é utilizada para desativar a saída do elemento LS (*Q* = *false*). A variável *A* representa a entrada do elemento LS e *B* representa o reset do elemento LS. A variável *executou* é utilizada para controle. A seguir, apresenta-se a definição formal do autômato que representa o comportamento de um elemento de memória LS.

Definição formal do autômato que representa um elemento de memória LS: Um autômato tmeporizado para um autômato LS é a sêxtupla(L, l0, C, Ac, E, I), tal que:

- L= {*inicio_l1,* L2, L3}
- L0 = inicio_L1 ;
- C = ;
- Ac = { synchro? } ;
- E = { E1, E2, E3, E4, E5, E6 } ; tal que:
    - E1 = (inicio_L1, synchro?, A && executou == false, { SetMemory(),executou = true }, L2) ; ○ E2 = (inicio_L1, synchro?, B && executou == false && !A, { Reset(),executou = true }, L3) ; ○ E3 = (L2, synchro?, !B && executou == false, { SetMemory(),executou = true }, L2) ; ○ E4 = (L3, synchro?, !A && executou == false, { Reset(),executou = true }, L3) ;
    - E5 = (L2, synchro?, B && executou == false, { Reset(),executou = true}, inicio_L1) ;
    - E6 = (L3, synchro?, A && executou == false, { SetMemory(),executou = true }, inicio_L1) ;

A execução do autômato que representa o elemento de memória LS é da seguinte forma: quando o autômato que representa o ciclo de varredura do CLP for sincronizado com *synchro?,* a variável *executou* estiver desativada e a entrada do elemento de memória estiver ativada (*A && executou== false)* então, o arco com origem na localidade *inicio_L1* e destino na localidade *L2* é ativado, a função *SetMemory()* é ativada e a variável *executou* é atualizada para *true*. A partir deste ponto podem ocorrer duas situações. A primeira ocorre quando o autômato entra em loop na localidade L2, sempre que a variável *B* e a variável *executou* estiverem com

valor *false* (!B && executou == false) a saída é ativada (*Q = 1*). A segunda ocorre quando o valor *B* é *true*, então o arco com origem na localidade *L2* e destino na localidade *inicio_L1* é ativado, restaurando a saída do elemento de memória (Q = 0). Na localidade inicio_L1, se a variável *B* estiver com valor *true* e a variável *A* estiver com valor *false* o autômato ficará em loop com a saída desativada (*Q = 0*) até que a variável *A* obtenha o valor *true,* ativando a saída do elemento de memória com a função *SetMemory()*.

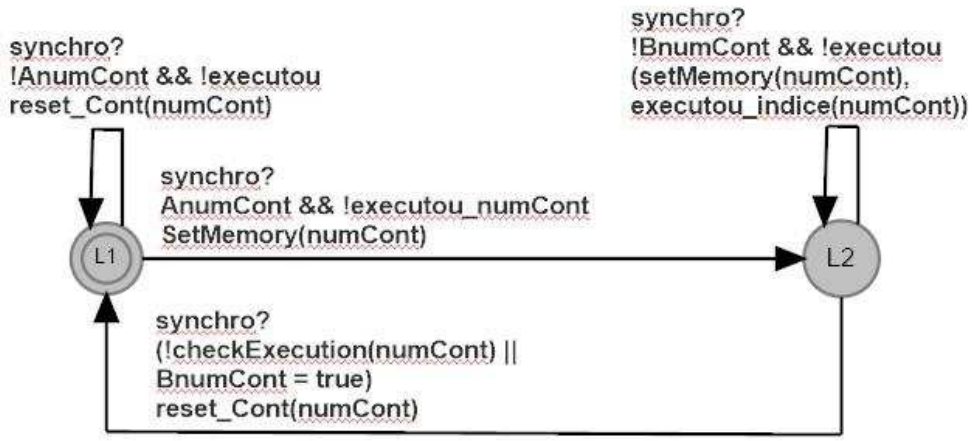


Figura 12: Autômato temporizado para um elemento de memória LS

A seguir 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 13 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()* e os valores para

as variáveis específicas de cada elemento de memória serem restaurados, função *atualiza_executou()*, a execução lógica do sistema é iniciada. Após a execução da lógica do sistema, as saídas são liberadas, a função *UpdateOutputs()*.

O estado inicial deste autômato é *committed* porque a evolução dos relógios não deve ser considerada durante a atualização das variáveis de entrada. A seguir é apresentada a definição formal deste autômato.

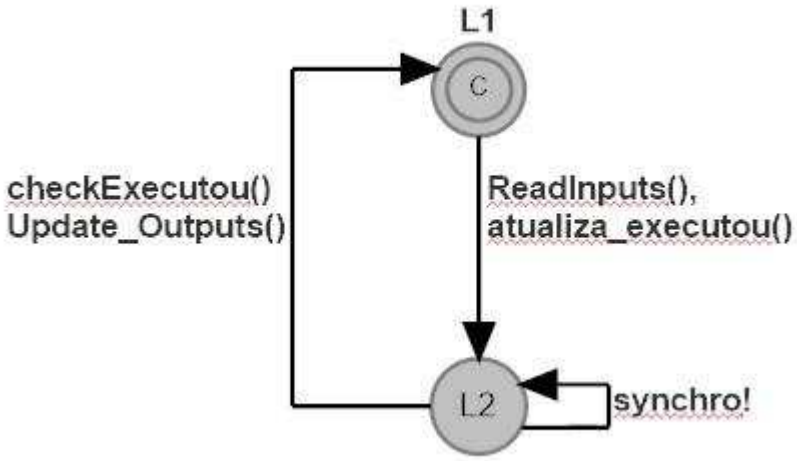


Figura 13: Autômato temporizado para o ciclo de varredura de um CLP.

Definição formal do autômato para o ciclo de varredura: Um autômato temporizado para o ciclo de varredura de um CLP é a sêxtupla (L, L0, C, Ac, E, I), tal que:

- L = {L1,L2}, L1 é committed;
- L_0 = L1;
- C = ;
- Ac = {synchro!};
- E = {E1,E2,E3};Tal que:
  - E1 = {L1,,,{readInputs(),atualiza_executou()},L2};
  - E2 = {L2,executou!,!executou,,L2};
  - E3 = {L2,,checkExecutou(),{Update_Outputs()},L1};
  - I = { I(*L2*) →time ≤ tScan}.

3.4. Ferramenta desenvolvida

Nesta seção é apresentada a ferramenta que foi desenvolvida para modelagem da especificação do SIS, a ferramenta consegue gerar, de forma automática uma rede de autômatos temporizados a partir da especificação do SIS.

Na Figura 14 é apresentada a ferramenta desenvolvida para gerar, de forma automática, modelos de autômatos temporizados. A ferramenta dispõe de uma barra de ferramentas que está localizada na parte superior. Com o uso desta é possível: escolher quais elementos ISA deseja modelar, salvar uma imagem do diagrama ISA 5.2 modelado e gerar os modelos de autômatos de forma automática segundo a sintaxe e semântica da ferramenta UPPAL.

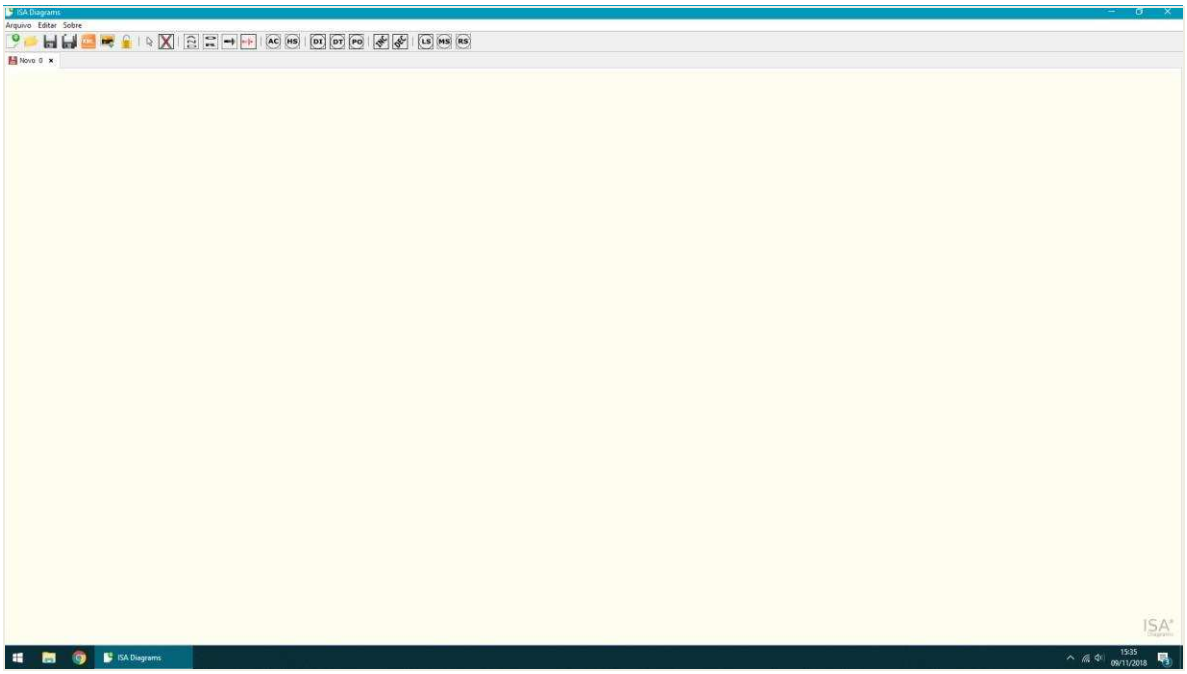


Figura 14: Ferramenta para geração e edição de diagramas de lógica binária ISA 5.2

A seguir, são listadas as funcionalidades da ferramenta:

- Criar um novo projeto
- Abrir um projeto existente
- Salvar projeto atual

- Salvar projeto atual como
- Gerar *XML* (rede de autômatos)
- Exportar imagem do diagrama
- Copiar
- Colar
- Recortar
- Adicionar nova aba no projeto

Na Figura 15 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.

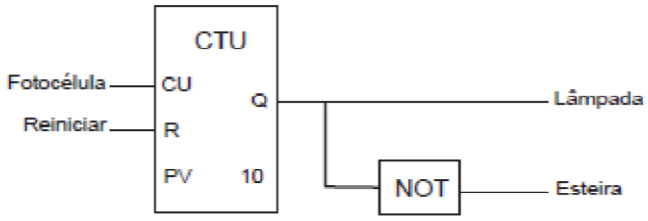


Figura 15: Exemplo de um programa FBD com contador (PARR, 2003)

Na Figura 16 é apresentado o exemplo, citado anteriormente, modelado na ferramenta que foi desenvolvida. Para o exemplo foi escolhido o valor de *PV = 10,* que representa o total de lotes da esteira; O mesmo conta com duas entradas do tipo *HS* que representam respectivamente a fotocélula e o botão de reiniciar, também conta com duas saídas *AC* que representam a lâmpada da esteira que é ligada quando o numero de itens atinge o valor máximo permitido e a saída negativa representada pelo botão *AC*, nomeado por “!Saída

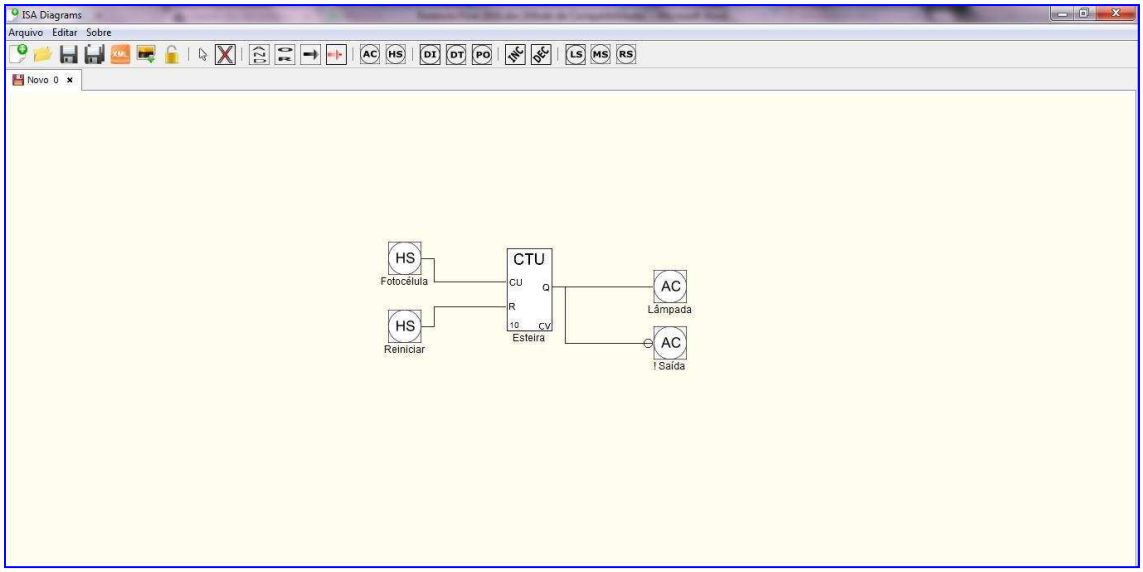


Figura 16: Exemplo da esteira modelado na fer ramenta desenvolvida

A partir deste ponto o usuário pode gerar automaticamente a rede de autômatos que representa o diagrama ISA modelado na ferramenta. Para gerar automaticamente o usuário deve clicar no botão *gerar xml*, botão em destaque na Figura 17.

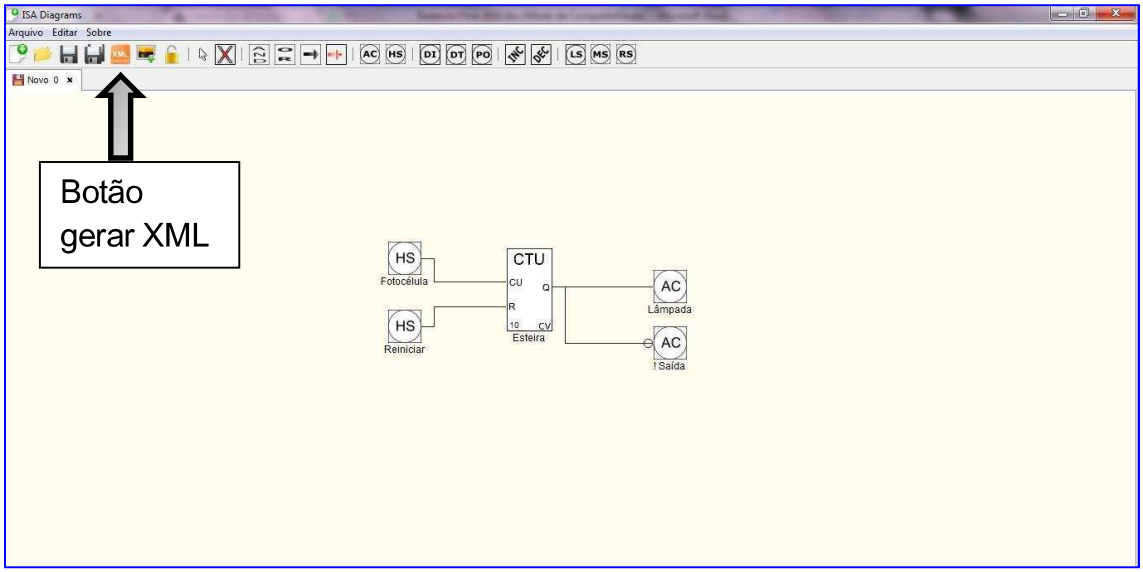


Figura 17: Botão gerar xml em destaque

Após o usuário clicar no botão “gerar xml”, o xml será gerado. Assim que o xml for gerado, o usuário poderá visualizar a rede de autômatos na ferramenta UPPAL (ver Figura 18)

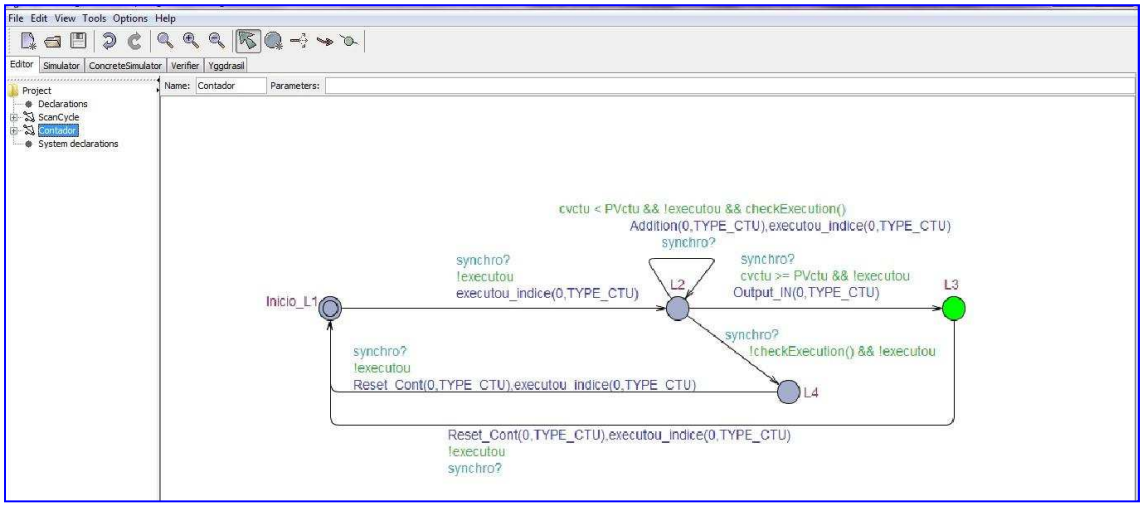


Figura 18: Rede de autômatos gerada a partir do modelo ISA modelado

4. ESTUDO DE CASO

Nesta seção o método proposto neste trabalho é avaliado através de um estudo de caso. O SIS utilizado é formado por três esteiras, estas esteiras possuem três motores (*M1, M2* e *M3)*, que as movimentam, sendo um motor para cada esteira. Este sistema tem a função de selecionar, encher e esvaziar as garrafas. Ao longo das três esteiras estão os sensores, responsáveis por verificar a presença das garrafas, ao passo em que elas são levadas pelas esteiras.

O sistema é composto por: *STG* (sensor de tipo de garrafa). *MS* (motor de separação das garrafas), *SP1*(sensor de proximidade 1 da esteira 1), *SN1*(sensor de nível das garrafas da esteira 1), *EG1*(enchedor de garrafas da esteira 1), *TP1*(embalador de garrafas da esteira 1), *CTG1* (contador de garrafas da esteira 1), CTC1(contador de caixas da esteira 1), *RCG1*(reset da esteira 1), *ALARME_CX_CHEIA(*alarme que indica quando uma caixa está cheia)*, BSTART (*botão que inicia as esteiras), *BSTOP(*botão que para o processo),*BEMERG(*botão de emergência), *B_G_RESET*(botão de reset geral).

Para iniciar o sistema é necessário pressionar o botão *BSTART*. Além disso, os botões *BTOP* e *BEMERG*, não devem estar ativados. Quando o processo é iniciado, as garrafas são levadas pela esteira 3, sendo divididas em dois tipos, *T1* e *T2*. Caso a garrafa seja do tipo *T1*,então o sensor *STG* emitirá um sinal positivo, para que o motor *MS* seja ativo, e posteriormente a garrafa será levada para a esteira 1; Quando a garrafa está em movimento na esteira 2, ao chegar na posição de *SP1,* os motores param, e *EG1* torna-se ativo. Quando a garrafa está com a quantidade X do produto, o sensor *SN1* desativa *EG1* e libera os motores. Mais adiante na esteira, o sensor *SP2,* é ativado com a presença da garrafa, e este ativa o *TP1,* o qual tampa a garrafa. Neste momento ocorre também a contagem de garrafas no contador *CTG1.* Seguindo o processo, a garrafa é depositada em uma caixa. Quando a caixa estiver com 10 garrafas, o alarme *ALARME_CX_CHEIA,* será ativado e o processo interrompido; Neste momento também ocorre a contagem de caixas, ou seja, *CTC1* conta mais uma caixa. Depois de substituída a caixa por outra vazia, o operador deve pressionar o botão *B_reset*. E assim iniciar novamente os motores das esteiras, retornando o ciclo

de seleção, enchimento, envasamento, contagem e armazenamento. Na Figura 19 é apresentada a implementação do sistema que é descrita na linguagem Ladder.

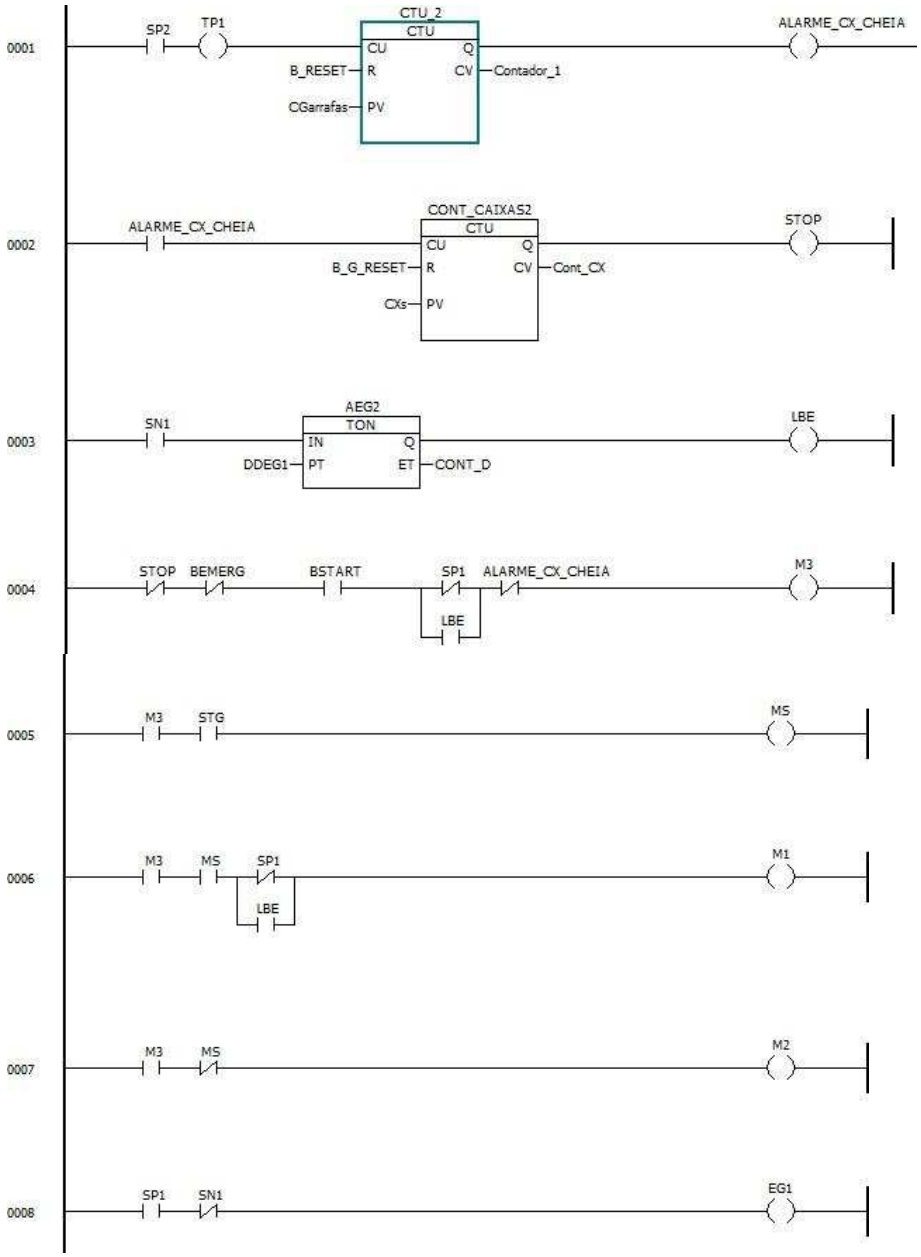


Figura 19: Programa Ladder para o sistema de enchimento de garrafas

Na Figura 20 é apresentado o modelo de autômato temporizado extraído a partir da especificação do sistema. De acordo com o método proposto, primeiro são gerados os modelos de autômatos temporizados para contadores. Desta forma. De acordo com a definição formal do autômato para um contador incremental, o modelo de autômato temporizado para contador *CTU_2* é apresentado na Figura 20, Para este autômato temporizado que a entrada do contador (*CU*) será representada pela variável *SP2*, a

variável *R* (reset do contador) será representada pela variável *B_RESETI* e a variável *PV* será inicializada com a constante 10.

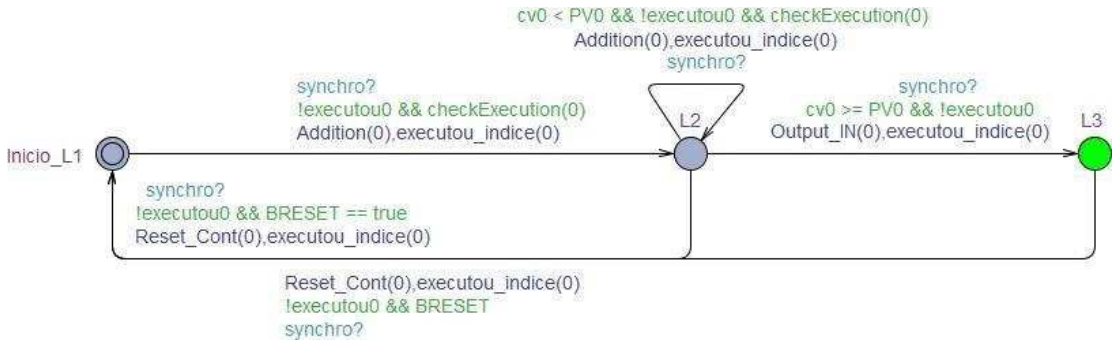


Figura 20: Autômato temporizado para o contador CTU_2

Após definidos os autômatos para os contadores, o modelo de autômato que representa o ciclo de varredura do CLP é gerado. Para tanto, as funções *readInputs()* e *updateOutputs()* devem ser definidas. Na função *readInputs()* são definidas as entradas dos casos de teste, apresentadas a seguir, na Tabela 2. A definição destas entradas não faz parte do escopo deste trabalho.

Tabela 2: Tabela para as variáveis de entrada do sistema de enchimento de garrafas.

#	Atribuições	SP2	B_RESET	STOP	BEMERG	BSTART	SP1	LBE	ALARME_CX_CHEIA	...
2	CU (CTU_2)	1								
2	ALARME_CX_CHEIA		1	4	ALARME_CX_CHEIA					
3	CU (CTU_2)	1								
4	CU (CTU_2)	0								
5	CU (CTU_2)	1								
6	CU (CTU_2)	0								
7	CU (CTU_2)	1								
8	CU (CTU_2)	0								
9	CU (CTU_2)	1								
10	CU (CTU_2)	0								
11	CU (CTU_2)	1								
12	CU (CTU_2)	0								
13	CU (CTU_2)	1								
14	CU (CTU_2)	0								
15	CU (CTU_2)	1								

16	CU (CTU_2)	0								
17	CU (CTU_2)	1								
18	CU (CTU_2)	0								
19	CU (CTU_2)	1								
20	CU (CTU_2)	0								
21	CU (CTU_2)	1								
22	CU (CTUobjet_2)	0								
23	ALARME_CX_CHEIA		1							
24	CU (CTU_2)	1								
25	M3			0	0	1	0		1	

Após gerado o conjunto com todas as atribuições para as variáveis de entrada é iniciada a execução 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.

```
void updateOutputs(){

        ALARME_CX_CHEIA = Q
        M3 = (!STOP and !BEMERG and BSTART) and (!SP1 or LBE) and
        !ALARME_CX_CHEIA;
}
```

5. CONCLUSÃO

O trabalho aqui apresentado introduziu bases formais para o desenvolvimento de métodos técnicas e ferramentas para modelagem da especificação de SIS, esta especificação é descrita no formato de diagramas de lógica binária ISA 5.2. Uma rede de autômatos temporizados foi gerada, de forma automática a partir dos diagramas ISA, gerando uma especificação formal para o SIS. Templates de autômatos temporizados para contadores incrementais e decrementais elementos de memória MS e LS e o ciclo de varredura foram apresentados.

O método proposto foi utilizado em um estudo de caso e o mesmo foi capaz de informar vereditos sobre a conformidade entre a especificação e a implementação do sistema. Uma ferramenta foi desenvolvida para modelagem dos diagramas ISA 5.2. Um método para gerar, de forma automática, uma rede de autômatos temporizados foi apresentada.

O próximo passo deste trabalho consiste submeter o método proposto a outros estudos de caso e modelar blocos de funções, pois, os mesmos são muito comuns em diagramas ISA.

REFERÊNCIAS

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

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.

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 Pro- gram Validation Through Time Petri Nets. Pages 121–136 of: Proceedings of the 4th European Conference on Model Driven Architecture (ECMDAFA '08). Berlin, Heidel- berg: Springer-Verlag.

Bryan, Luis A., & Bryan, Eric A. 1997. Programmable Controllers: Theory and Implemen- tation. Industrial Text Company.

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

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

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

Ge, Xi, Taneja, Kunal, Xie, Tao, & Tillmann, Nikolai. 2011 (Maio). DyTa: Dynamic Sym- bolic 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 Pro- grams 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.

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

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

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.

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

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

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.

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

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

Oliveira, Kézia de Vasconcelos, Oliveira, Augusto bezerra, Santos, Victor Fortunato, 2014, Geração Automática de Autômatos Temporizados para Especificações de Sistemas Instrumentados de Segurança.

Oliveira, Kézia de Vasconcelos, Gorgônio, Kyller, Perkusich, Angelo, Lima, Antonio Marcus Nogueira, & Silva, Leandro Dias da. 2009. Extração Automática de Autômatos Temporizados a Partir de Diagramas Ladder. In: Anais do IX Simpósio Brasileiro de Automação Inteligente. Brasília, Brasil: Sociedade Brasileira de Automática.

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, Silva, Leandro Dias da, Perkusich, Angelo, Lima, Antonio Marcus Nogueira, & Gorgônio, Kyller. 2010b (July). Automatic timed automata extraction from ladder programs for model-based analysis of control systems. Pages 90 – 95 of: IEEE International Symposium on Industrial Electronics (ISIE´2010).

Oliveira, Kézia de Vasconcelos, Silva, Leandro Dias da, Perkusich, Angelo, Lima, Antonio Marcus Nogueira, & Gorgônio, Kyller. 2010c. Geração Automática de Testes de Conformidade para Programas de Controladores Lógicos programáveis. Pages 2995–3001 of: Anais do CBA 2010. Bonito, MS, Brasil: Sociedade Brasileira de Automática.

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

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).

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

PLCopen. 2004. IEC 61131-3: a Standard Programming Resource. In http://www.plcopen.org/. PLCopen For Efficiency in Automation.

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

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

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.

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.

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

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).

Tschannen, Julian, Furia, Carlo A., Nordio, Martin, & Meyer, Bertrand. 2011. Usable Verifi- cation 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: SpringerVerlag.

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

APENDICE A – Funções do autômato incremental

```
void executou_indice(int nctu){
   executou = true;
}

void Output_IN(int nctu){
   Qctu = true;
}

void Reset_Cont(int nctu){ // esse reset é depois de liberar o Q e reseta tudo

   if(nctu == 0){
      Qctu = false;
      cv_numCont = 0;
   }
}

bool checkExecution(int numCont){
   if(Actu == true && Bctu == false){
      return true;
   }else{
      return false;
   }
}

void Addition (int nctu){
   cv_numCont = cv_numCont + 1;
}
```

APÊNDICE B – Funções do autômato Decremental

```
void executou_indice(int ndtu){
    executou = true;
}

void Output_IN(int ndtu){
    Qdtu = true;
}

void Reset_Cont(int ndtu){
    Qdtu = false;
    cv_numCont = 10;
}

bool checkExecution(int numCont){
    if(Actu == true && Bctu == false){
        return true;
    }else{
        return false;
    }
}

void Decremento_cont (int ndtu){
    cv_numCont = cv_numCont - 1;
}
```