UTILIZAÇÃO DE DIAGRAMAS DE DECISÃO BINÁRIA ORDENADOS PARA GERAÇÃO DE CASOS DE TESTE EM SISTEMAS INSTRUMENTADOS DE SEGURANÇA

Kézia de Vasconcelos Oliveira*, Leandro Dias da Silva†, Angelo Perkusich*, Kyller Costa Gorgônio*, Lenardo Chaves e Silva*

* *Universidade Federal de Campina Grande - UFCG*
Campina Grande, Paraíba, Brasil

† *Universidade Federal de Alagoas - UFAL*
Maceió, Alagoas, Brasil

Emails: kezia@copin.ufcg.edu.br, {leandro, perkusic, kyller}@dee.ufcg.edu.br, lenardosilva@copin.ufcg.edu.br

Abstract— The goal of this work is to present a method in which test cases are generated automatically, and the output states and timed properties of the system are evaluated from the system specification given in ISA 5.2 diagrams, and that they are executed in a CLP program. For this purpose, we use timed automata networks and OBDD approach.

Keywords— Safety Instrumented Systems, Programmable Logic Controller, ISA 5.2 Diagrams, Timed Automata Network, Ordered Binary Decision Diagram.

Resumo— O objetivo neste trabalho é apresentar um método no qual casos de teste são gerados automaticamente, e os estados das saídas e propriedades temporizadas do sistema são avaliados, a partir da especificação do sistema dada em diagramas ISA 5.2, e que os mesmos sejam executados em um programa para CLP. Para este propósito faremos uso de redes de autômatos temporizados e da abordagem OBDD.

Palavras-chave— Sistemas Instrumentados de Segurança, Controlador Lógico Programável, Diagramas ISA 5.2, Rede de Autômatos Temporizados, Diagramas de Decisão Binária Ordenados.

1 Inrodução

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 (Gruhn and Cheddie, 2006). A automação destes sistemas é realizada por Controladores Lógicos Programáveis (CLP) de segurança, uma classe de CLP especialmente projetada para trabalhar com o conceito de falha segura e ser tolerante a falhas.

Para SIS é fundamental garantir a confiança e a segurança no funcionamento, pois defeitos no hardware ou no software podem ocasionar danos às instalações, aos seres humanos e ao meio ambiente. 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 um programa de CLP para SIS, desde técnicas estáticas a dinâmicas (Patil et al., 2011). Teste de caixa preta é uma técnica de verificação dinâmica bastante utilizada para verificar programas de CLP (Oliveira et al., 2010; Chaaban et al., 2011). 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 and 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 para avaliar variáveis do tipo booleana são (Schiffmann and Steinbach, 2012): particionamento em classes de equivalência; grafo de causa-efeito; tabela de decisão e diagrama de decisão binária (BDD). Neste trabalho apenas variáveis booleanas são consideradas.

BDDs são estruturas utilizadas para representar funções booleanas através de grafos direcionados acíclicos, onde os nós intermediários representam as variáveis e os nós folha representam os valores da função. Um BDD é ordenado (OBDD) se as variáveis aparecem sempre na mesma ordem ao longo de qualquer caminho desde a raíz.

O objetivo neste trabalho é apresentar um método que aumente a confiança e a segurança em programas de CLP para SIS. Para tanto, geração e execução automática de casos de teste de caixa-preta, que contemplam os estados das saídas e propriedades temporizadas do sistema, são utilizadas para avaliar se o programa do SIS, em tempo de execução, está em conformidade com sua especificação dada no formato de diagramas ISA 5.2. Para este propósito, faremos uso de redes de autômatos temporizados e da abordagem OBDD. Um estudo de caso real será apresentado.

O restante deste artigo está organizado da seguinte forma. Na Seção II o método proposto é introduzido. Na Seção III um estudo de caso é apresentado. Na Seção IV este trabalho é concluído e sugestões para trabalhos futuros são discutidas.

2 Método para geração dos casos de teste

Na Figura 1 é apresentado o método proposto neste trabalho. Primeiro o arquivo que representa a especificação do sistema (diagramas ISA 5.2) é transformado em uma rede de autômatos temporizados (AT). Após a geração desta rede de autômatos temporizados, casos de testes de conformidade são gerados, de forma automática, a partir desta rede de autômatos. Para tal tarefa, uma ferramenta de teste foi desenvolvida com o objetivo de gerar um conjunto de casos de testes em que os estados das saídas do sistema e propriedades temporizadas são avaliados. Após isso, os valores das entradas gerados a partir da rede de autômatos temporizados são enviados para o CLP, onde está presente a implementação do sistema a ser avaliada. 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. Neste trabalho apenas as etapas de geração dos modelos de autômatos temporizados e dos casos de teste serão apresentadas.

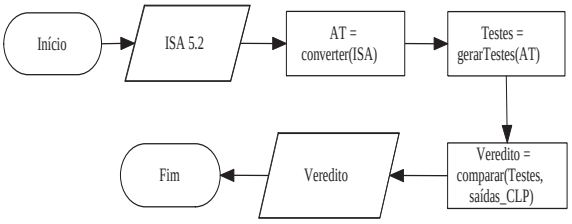


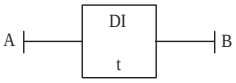
Figura 1: Método utilizado para validar o SIS

Formalmente, um autômato temporizado é um sêxtuplo (L, l_0, C, A, E, I), onde: 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, co-ações e ações internas; 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).

Neste trabalho, dois tipos de autômatos temporizados são gerados: 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.

2.1 Autômatos para Temporizadores

Na Figura 2(a) é apresentado um temporizador do tipo DI (*Delay Initiation of output*). Em um temporizador DI a saída (B) é energizada depois de um período de tempo (t) quando a entrada é verdadeira (A = *true*). Quando a entrada for falsa a saída será desativada. Na Figura 2(b) é apresentada a tabela verdade para a variável B. Considere a variável *Time* como sendo o tempo decorrido.



(a) Símbolo

A	Time	B
true	< t	false
true	> = t	true
false	-	false

(b) Tabela Verdade

Figura 2: Temporizador DI

Para definirmos o autômato temporizado que representa o comportamento de um temporizador DI, conforme Figura 3, vamos fazer as seguintes considerações. O temporizador DI pode está em uma das três possíveis localidades: *timer enable* (L_1), *timer timing* (L_2) and *timer done* (L_3). Estas localidades representam as condições correspondentes do temporizador. A tabela apresentada na Figura 2(b) será utilizada para determinar a mudança entre as localidades do autômato. Para cada mudança de localidade será gerada uma aresta, na 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*, *acc* e o canal de sincronização *sync* são utilizados para sincronizar a rede de autômatos temporizados. A seguir é apresentada a definição formal para este autômato.

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

- L = $\{L_1, L_2, L_3\}$
- $l_0 = L_1$
- C = {time}
- Ac = {sync?}
- E = $\{E_1, E_2, E_3, E_4\}$, tal que:
  - $E_1 = (L_1$, sync?, A, {B = false, time = 0, acc = t, control = 1}, $L_2)$;
  - $E_2 = (L_2$, sync?, !A, {B = false, acc = 0, control = 0}, $L_1)$;
  - $E_3 = (L_2$, sync?, A && time $\geq$ t, {B = true, acc = 0}, $L_3)$;
  - $E_4 = (L_3$, sync?, !A, {B = false, acc = 0}, $L_1)$;
- I = {}, *não possui invariantes*

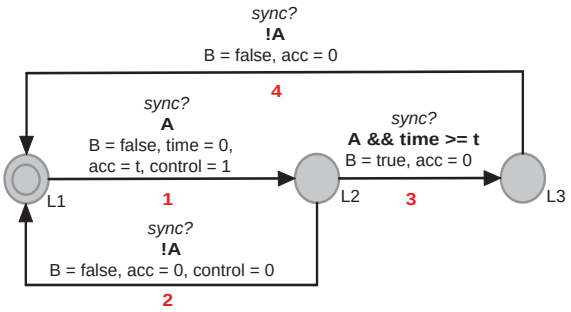


Figura 3: Autômato para o temporizador DI

2.2 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 necessárias três arestas, uma para executar cada etapa do ciclo. O autômato pode está em uma das duas possíveis localidades: *inicialization mode* (L_1) e *run mode*(L_2). 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()*.

A seguir, o algoritmo utilizado para definir os valores para as variáveis de entrada é apresentado. Estes valores representam as entradas dos casos de teste. Para o algoritmo, considere um diagrama ISA como sendo uma tripla *(T, E, S)*, onde $T = \{t_1,...,t_n\}$ é o conjunto de temporizadores, $E = \{e_1,...,e_n\}$ é o conjunto de variáveis de entrada e S é o conjunto de saídas do diagrama ISA. Cada saída $s \in S$ é composta por um conjunto de variáveis de entrada Y e uma função booleana f (lógica do bloco). A entrada para o algoritmo é um diagrama ISA e a saída é um conjunto de valores $\{B_1,...,B_m\}$, onde $B_i = (b_1,...,b_n)$ 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. Desta forma, teremos a execução de m ciclos de varredura.

Na linha 2 do algoritmo é construído um OBDD para cada função booleana de S. 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 OBDD. Neste trabalho, a ordem das variáveis é pré-estabelecida, 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 OBDDs 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. Pesquisas sobre como encontrar a melhor ordem das variáveis têm sido feitas nos últimos anos e vários métodos de ordenação têm sido apresentados (Rudell, 1993; Sensarma et al., 2012).

Algorithm 1 GenerateInputs()

Input: Diagrama ISA 5.2, no qual ISA = (T,E,S)
Output: Um conjunto B = $\{B_1,...,B_m\}$, no qual B_i = $(b_1,...,b_n)$ e $1 \leq i \leq m$

```
1: for all s ∈ S do
2:    obdd ← generateOBDD(Y,f(y1,..., yn))
3:    listPaths ← extractPaths(obdd,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 3 do algoritmo, todos os caminhos cujo valor da função f é $f(y_1,...,y_n) = 1$ ou $f(y_1,...,y_n) = 0$ são extraídos. Para calcular o valor de uma função f dada em termos de um OBDD 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 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 é criada uma tabela 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 OBDD gerado). Caso alguma variável de entrada e_i não seja definida em l, então consideramos $b_i = 0$.

Na linha 7 do algoritmo é gerado um conjunto com todas as atribuições para as variáveis entrada do sistema. 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. Caso a função f não possua temporizadores, então todas atribuições $(b_1,...,b_n)$ obtidas a partir da função f serão utilizadas para testar o sistema. Caso f possua temporizadores, então serão necessárias quatro atribuições da forma $(b_1,...,b_n)$ para testar o comportamento do temporizador. Estas atribições serão utilizadas para ativar as quatro arestas do autômato que representa um temporizador DI, conforme apresentado na Figura 3. Desta forma, as seguintes atribuições serão utilizadas: uma atribuição em que $f(b_1,...,b_n) = 1$ (a entrada do temporizador, representada pela função f, 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). 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 casos de testes redundantes são reduzidos.

Na função *updateOutputs()*, cujo algoritmo é apresentado a seguir, são processadas as saídas do diagrama ISA. Estas saídas representam as saídas dos casos de teste. 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).

Algorithm 2 updateOutputs(ISA)

Input: Uma atribuição (b_1,...,b_n), com $n = |E|$
Output: Uma atribuição A = (a_1,...,a_j), com $j = |S|$
1: **for all** s ∈ S **do**
2: A ← A ∪ execute(f(b_1,...,b_n))
3: **end for**
4: **return** A

Na Figura 4 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. Detalhes de como gerar as funções *checkExecutionTimers()*, *evaluate()* e *updateVariables()* são apresentados em (Oliveira et al., 2010). A seguir, é apresentada a definição formal para o autômato.

Definição 2: Um autômato temporizado para o ciclo de varredura de um CLP é uma sêxtupla (L, l_0, C, A, E, I), onde:

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

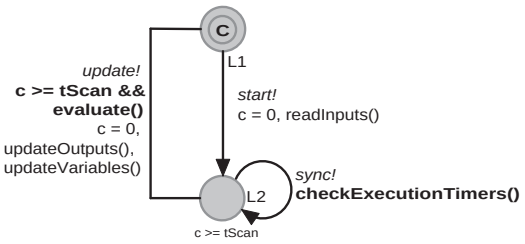


Figura 4: Autômato para o ciclo de varredura

3 Case Study

Para apresentar o método, foi utilizado um exemplo de um sistema de segurança, cujo diagrama ISA 5.2 é apresentado na Figura 5. Este sistema é utilizado para manter a segurança de uma zona após a detecção de fogo ou gás na mesma.

Para o método primeiro são gerados os modelos de autômatos temporizados para temporizadores. Desta forma, de acordo com a definição 1, o modelo de autômato temporizado para o temporizador *Timer1* é apresentado na Figura 6. A variável *A* será representada pela função booleana *SF1 or SF2*, a variável *t* será inicializada com a constante *2* e a variável *B* será representada por *DispCO2*. De forma análoga, o modelo de autômato temporizado para *Timer2* pode ser gerado.

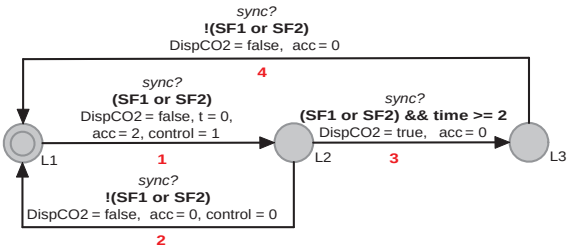


Figura 6: Autômato temporizado para Timer1

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 5 temos: T = {Timer1, Timer2}; E = {SF1, SF2, SG1, SG2, SG3} e S = {SF1, SF2, SG1, SG2, SG3}.

A função *readInput()* é gerada em quatro passos. No primeiro passo são gerados OBDDs para cada função booleana que determina uma saída do diagrama ISA. Na Figura 7 estes OBDDs são apresentados. Observe que apenas 3 OBDDs foram gerados para 5 saídas. Isto acontece porque a função

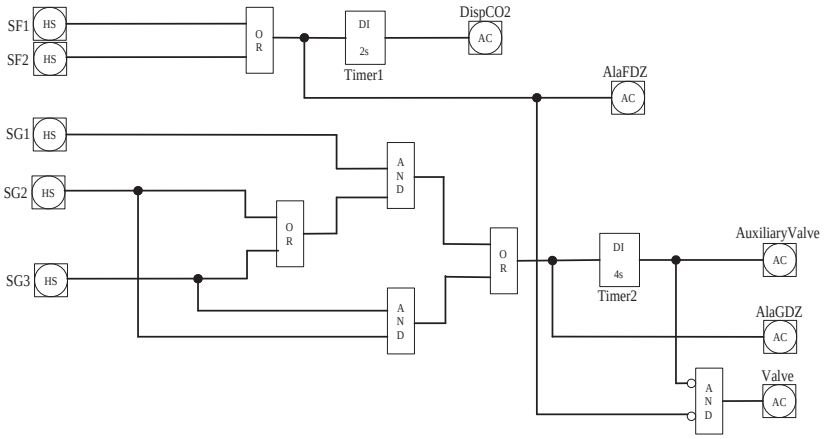
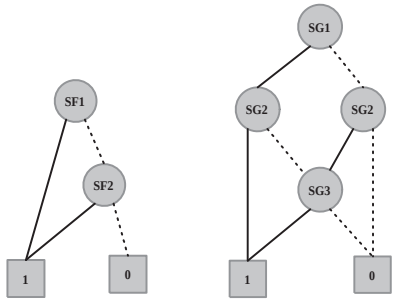


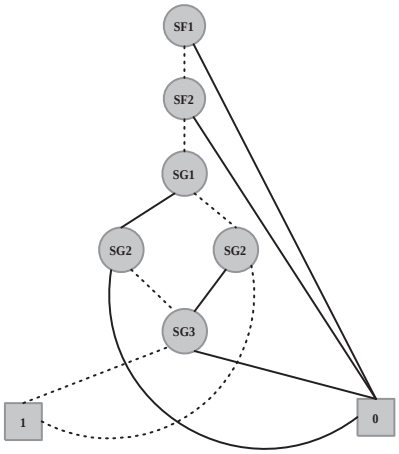
Figura 5: Diagrama ISA 5.2 para o sistema de segurança

booleana para as saídas *DispCO2* e *AlaFDZ* é a mesma. O mesmo ocorre com a função para as saídas *AuxiliaryValve* e *AlaGDZ*.



(a) DispCO2 e AlaFDZ

(b) AuxiliaryValve e AlaGDZ



(c) Valve

Figura 7: OBDDs para as saídas do SIS

No segundo passo para gerar a função *readInput()* são extraídos todos os caminhos para cada função *f*. Na Tabela 1 são apresentados os caminhos extraídos para a função *f(SF1,SF2,SG1,SG2,SG3)*. Observe que para esta função foram extraídos 9 caminhos: f(1,-,-,-.-) = 0, ..., f(0,0,0,1,1) = 0. No primeiro caminho os valores para as variável *SF2*, *SG1*, *SG2* e *SG3* não foram definidos porque independente dos seus valores (*0* ou *1*) a saída para a função booleana seria a mesma (valor *1*).

Tabela 1: Caminhos para OBDD de Valve

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

No terceiro passo é criada uma tabela, conforme Tabela 2, com 5 colunas cada uma representada por uma variável de entrada. Cada linha da tabela corresponde a uma atribuição de variáveis. Como 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 o algoritmo 1, 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.

No quarto passo 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 aresta do autômato que representa o temporizador, são consideradas. Fazendo uso da Tabela 2, 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 7 da tabela - Ativa a aresta 4 do temporizador Timer2;
- Atribuições para AlaGDZ: apenas as atribuições das linhas 8 e 9 da tabela deve ser selecionada, pois as demais atribuições, linhas 4-7 já foram selecionadas;
- Atribuições para Valve: todas as atribuições definidas para esta saída, linhas 12-17 já foram selecionadas. Desta forma, nenhuma nova atribuição é selecionada.

Tabela 2: Atribuições para as variáveis de entrada

	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

Note que, de acordo com a Tabela 2, 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()* cada uma das m atribuições, onde m = 9, são executadas e as saídas são geradas. Cada atribuição é executada em um ciclo de varredura.

4 Conclusões

Neste trabalho nós introduzimos um método para aumentar a confiança e a segurança no funcionamento de SIS através da geração automática de casos de teste utilizando a teoria de autômatos temporizados e a abordagem OBDD. Nos casos de teste gerados os estados das saídas bem como propriedades temporizadas do sistema são avaliados. A próxima etapa deste trabalho consiste na aplicação de técnicas para determinar a melhor ordem das variáveis com o objetivo de minimizar ainda mais o número de casos teste gerado.

Agradecimentos

Os autores gostariam de agradecer o apoio financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Referências

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

Gruhn, P. and Cheddie, H. L. (2006). *Safety Instrumented Systems: Design, Analysis, and Justification*, 2nd edn, ISA.

Oliveira, K. d. V., Gorgônio, K., Perkusich, A., Lima, A. M. N. and Silva, L. D. d. (2010). Automatic Timed Automata Extraction from Ladder Programs for Model-Based Analysis of Control Systems, *in* H. Mouratidis (ed.), *Software Engineering for Secure Systems: Industrial and Research Perspectives*, IGI Global, Hershey, USA, pp. 305–328.

Patil, M., Subbaraman, S. and Joshi, S. (2011). Exploring integrated circuit verification methodology for verification and validation of plc systems, *International Symposium on Electronic System Design*, pp. 88 –93.

Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams, *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, ICCAD '93, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 42–47.

Schiffmann, J. and Steinbach, B. (2012). Utilization of BDDs for Test-Case-Specifications of GUI-Applications, *Proceedings of the 10th International Workshops on Boolean Problems, Freiberg University of Mining and Technology*, Freiberg,Germany, pp. 41–48.

Sensarma, D., Banerjee, S., Basuli, K., Naskar, S. and Sen-Sarma, S. (2012). On an optimization technique using binary decision diagram, *CoRR* **abs/1203.2505**.