FORÇAMENTO DE OPACIDADE DE ESTADO ATUAL POR MEIO DE TROCAS DAS OBSERVAÇÕES DE EVENTOS

Raphael Julio Barcelos*, João Carlos Basilio*

*COPPE - Programa de Engenharia Elétrica
Universidade Federal do Rio de Janeiro
Rio de Janeiro, RJ, Brasil

raphajulio@gmail.com, basilio@poli.ufrj.br

Abstract— Opacity is a property that ensures that a secret behavior of the system is kept hidden from an Intruder. In this paper, we deal with current-state opacity, and propose an Opacity-Enforcer that is able to change, in an appropriate way, the order of observation in the event occurrences in the system, so as to mislead the Intruder to wrongly estimate at least one non-secret state. A necessary and sufficient condition for the feasibility of the Opacity-Enforcer synthesis and also two algorithms to build the automaton that realizes such an enforcement are presented. An example taken from the literature is used to illustrate the results of the paper.

Keywords— Discrete event systems, opacity, opacity-enforcement, event observation delay.

Resumo— Opacidade é uma propriedade que garante que qualquer comportamento secreto do sistema permaneça escondido de um Intruso. Neste trabalho será considerado o problema da opacidade de estado atual e será proposto um Forçador de Opacidade capaz de permutar adequadamente a ordem de observação dos eventos ocorridos no sistema de forma que o Intruso seja enganado e sempre estime, erroneamente, pelo menos um estado não secreto. Condições necessárias e suficientes para verificar se a síntese do Forçador de Opacidade é factível e dois algorítimos para construção do autômato que implementa a estratégia usada pelo Forçador de Opacidade são propostos. Os resultados do artigo são ilustrados utilizando-se um exemplo retirado da literatura.

Palavras-chave— Sistemas a eventos discretos, opacidade, executor de opacidade, observação de eventos com atraso.

1 Introdução

Recentemente, a opacidade surgiu no contexto de Sistemas a Eventos Discretos (SED) (Bryans, Koutny e Ryan 2005) como uma forma de lidar com problemas relacionados a segurança. Opacidade é uma propriedade que garante que um comportamento secreto do sistema permaneça escondido de observadores externos, usualmente chamados de Intruso. Diferentes noções de opacidade, assim como suas características, foram propostas na literatura (Lin 2011, Wu e Lafortune 2013).

Dentre os diversos problemas relacionados à opacidade, o forçamento de opacidade vem recebendo bastante atenção. Dubreil, Darondeau e Marchand (2010), Saboori e Hadjicostis (2008), e Yin e Lafortune (2016) utilizaram da teoria de controle supervisório para restringir o comportamento do sistema ao comportamento não secreto, sendo, portanto, uma abordagem conservadora. Cassez, Dubreil e Marchand (2012) propõem uma estratégia para forçar opacidade por meio de máscaras dinâmicas e Wu e Lafortune (2014) aborda o mesmo problema usando funções de inserção; a ideia, em ambos os casos, é permitir que o sistema evolua livremente, porém, manipulando as observações e com o objetivo de enganar o Intruso.

Este artigo considera o problema da opacidade de estado atual, *current-state opacity* (CSO). Neste contexto, um sistema é dito opaco em relação ao estado atual, se para todo estado secreto alcançado a partir de um estado inicial por uma sequência, existe um estado não secreto alcançado a partir de um estado inicial por uma outra sequência, em que ambas possuem a mesma observação no ponto de vista do Intruso. Neste artigo será proposta uma nova estratégia para forçar opacidade baseada na troca da ordem da observação dos eventos visando que o Intruso estime erroneamente pelo menos um estado não secreto.

O Forçador de Opacidade a ser proposto funciona da seguinte forma: ao receber um sinal associado à ocorrência de um evento, ele escolhe, baseado em regras preestabelecidas, se o libera ou o segura até a chegada de outro evento, visando trocar a ordem de observação dos eventos. As observações de eventos liberadas pelo Forçador de Opacidade são transmitidas para um Receptor por uma rede não segura, suscetível a vazar informação para o Intruso. A ideia por trás do Forçador de Opacidade proposto é usar o mesmo autômato desenvolvido por Nunes, Moreira, Alves, Carvalho e Basilio (2018), que modela atrasos nos canais de comunicação entre o sistema e o diagnosticador, para sincronizar todas as possíveis trocas de ordem de observação conforme as propriedades do Forçador de Opacidade de forma que as sequências com observações permutadas permaneçam dentro da linguagem gerada pelo sistema. Outra condição que o Forçador de Opacidade deve satisfazer é que o Intruso deve sempre acreditar que ele está estimando um estado não secreto do autômato observador.

Este artigo é organizado da seguinte forma. A seção 2 apresenta uma breve revisão sobre SED e introduz as notações usadas no artigo. A seção 3 apresenta a formulação do problema e as hipóteses adotadas. A seção 4 explica como funciona a

estratégia para forçar opacidade. A seção 5 apresenta dois algoritmos, o primeiro que verifica se a opacidade pode ser forcada, e em caso positivo, implementa o Forçador de Opacidade e o segundo que, dado um limite, achará uma configuração mínima para forçar opacidade. A seção 6 ilustra a estratégia proposta com um problema denominado "rato no labirinto" (Ramadge e Wonham 1989). A seção 7 resume todas as contribuições deste artigo e sugere possíveis extensões.

2 Conceitos preliminares e notação

O formalismo aqui utilizado para modelar SED é o autômato de estados finitos, que é definido pela quíntupla $G := (X, \Sigma, f, \Gamma, X_0)$, em que X é o conjunto finito de estados, Σ é o conjunto finito de eventos, $f : X \times \Sigma \to X$ é a função de transição, $\Gamma : X \to 2^{\Sigma}$, definido por $\Gamma(x) := \{\sigma \in \Sigma : f(x, \sigma)!\}$ é o conjunto de eventos ativos, em que $f(x, \sigma)!$ significa que $f(x, \sigma)$ é definido, e $X_0 \subseteq X$ é o conjunto de estados iniciais (Cassandras e Lafortune 2008). A função de transição pode ser estendida para $f : X \times \Sigma^* \to X$, em que Σ^* representa o Fecho-de-Kleene de Σ, pela recursão $f(x, s\sigma) = f(f(x, s), \sigma)$; $f(x, \epsilon) = x$; $\sigma \in \Sigma, s \in \Sigma^*$. A linguagem gerada por G é definida como $\mathcal{L}(G) := \{s \in \Sigma^* : f(X_0, s)!\}$ e será representada por L. Supõe-se que o conjunto de eventos Σ seja particionado em $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, em que Σ_o e Σ_{uo} são, respectivamente, o conjunto de eventos observáveis e não observáveis.

Ao longo do texto, o *fecho do prefixo* de uma sequência s será representado por $Pre(s)$, sendo definido como $Pre(s) := \{u \in \Sigma^* : (\exists v \in \Sigma^*)[uv = s]\}$. A *pós-linguagem* de L após uma sequência s é definida por $L/s := \{t \in \Sigma^* : st \in L\}$. Dados dois conjuntos de eventos, Σ_a e Σ_b, em que $\Sigma_b \subseteq \Sigma_a$, a *projeção natural*, ou apenas projeção, é definida por $P_{a,b} : \Sigma_a^* \to \Sigma_b^*$ em que $P_{a,b}(s\sigma) = P_{a,b}(s)P_{a,b}(\sigma)$; $P_{a,b}(\sigma) = \sigma$ se $\sigma \in \Sigma_b$ e $P_{a,b}(\sigma) = \varepsilon$ se $\sigma \in \Sigma_a \setminus \Sigma_b$ e $P_{a,b}(\varepsilon) = \varepsilon$, em que $s \in \Sigma_a^*$, $\sigma \in \Sigma_a$ e ε é a sequência nula. A *projeção inversa* é definida por $P_{a,b}^{-1} : \Sigma_b^* \to \Sigma_a^*$, em que $P_{a,b}^{-1}(t) := \{s \in \Sigma_a^* : P_{a,b}(s) = t\}$. O *fecho do prefixo*, a *projeção* e a *projeção inversa* podem ser estendidas sobre uma linguagem L aplicando-as a cada uma das sequências de L. A operação *parte acessível* de um autômato remove todos os estados que não são alcançados a partir de algum estado inicial por meio de sequências, e é definida por $Ac(G) := (X_{Ac}, \Sigma, f_{Ac}, \Gamma_{Ac}, X_0)$, em que $X_{Ac} := \{x \in X : (\exists s \in \Sigma^* \wedge \exists x_0 \in X_0)[f(x_0, s) = x]\}$ e $f_{Ac} : X_{Ac} \times \Sigma^* \to X_{Ac}$ em que $f_{Ac}(x, \sigma) = f(x, \sigma)$ se $f(x, \sigma) \in X_{Ac}$ e não definida, caso contrário. A *composição paralela* entre autômatos cria um novo autômato em que o comportamento comum dos autômatos anteriores são sincronizados e seus comportamentos particulares ocorrem livremente, sendo formalmente definida por $G_1||G_2 := Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1||2}, \Gamma_{1||2}, X_{0,1} \times X_{0,2})$, em que $f_{1||2}((x_1, x_2), \sigma) := (f_1(x_1, \sigma), x_2)$ se $\sigma \in \Gamma_1(x_1) \setminus \Sigma_2$; $(x_1, f_2(x_2, \sigma))$ se $\sigma \in \Gamma_2(x_2) \setminus \Sigma_1$; $(f_1(x_1, \sigma), f_2(x_2, \sigma))$ se $\sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2)$, e não definida, caso contrário.

O *alcance não observável* é definido por $UR(x, \Sigma_o) := \{x' \in X : (\exists s \in \Sigma_{uo}^*)[f(x, s) = x']\}$. Sua extensão para um conjunto de estados $Y \in 2^X$ é dada por: $UR(Y, \Sigma_o) := \cup_{x \in Y} UR(x, \Sigma_o)$. Dessa forma, o *observador* do autômato G é dado por $G_{obs} = O_{bs}(G, \Sigma_o) := (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, X_{0,obs})$, em que $X_{obs} \in 2^X$, $f_{obs}(x_{obs}, \sigma) := \cup_{x \in x_{obs} \wedge f(x,\sigma)!} UR(f(x, \sigma), \Sigma_o)$, $\Gamma_{obs}(x_{obs}) := \cup_{x \in x_{obs}} \Gamma(x) \cap \Sigma_o$, $X_{0,obs} := UR(X_0, \Sigma_o)$. Finalmente, $\mathcal{L}(G_{obs}) = P_o(\mathcal{L}(G_{obs}))$ será representado por L_{obs}, em que $P_o : \Sigma^* \to \Sigma_o^*$.

A definição formal de CSO é retomada de Wu e Lafortune (2013).

Definição 1 (Opacidade de Estado Atual) *Dado um sistema modelado por $G = (X, \Sigma, f, \Gamma, X_0)$, a projeção $P_o : \Sigma^* \to \Sigma_o^*$ e um conjunto de estados secretos X_s, diz-se que o sistema G é opaco com relação ao estado atual se $\forall x_0 \in X_0$ e $\forall s \in L$ tal que $f(x_0, s) \in X_s$, $\exists \tilde{x}_0 \in X_0$, $\tilde{x}_0$ não necessariamente diferente de x_0, e $\exists \tilde{s} \in L$ tal que $f(\tilde{x}_0, \tilde{s}) \in X \setminus X_s$ e $P_o(s) = P_o(\tilde{s})$.*

Uma forma intuitiva e fácil de verificar se um sistema G é CSO é construindo-se seu autômato observador G_{obs} e verificando se seus estados são compostos simultaneamente por estados secretos e não secretos de G. Como o conjunto de estados secretos com relação a G_{obs} é dado por $X_{s,obs} := X_{obs} \cap 2^{X_s}$, o sistema será CSO se e somente se $X_{s,obs} = \emptyset$.

3 Formulação do Problema

A arquitetura considerada neste artigo é mostrada na Figura 1 e é composta por um sistema físico que contém diversos pontos de medição, que são responsáveis por gravar a ocorrência dos eventos, e canais de transmissão, que transmitem os sinais dos sensores ao Forçador de Opacidade. Supõe-se que esses canais sejam seguros. Os eventos liberados pelo Forçador de Opacidade são transmitidos ao Receptor por meio de uma rede não segura, existindo a possibilidade de um Intruso capturar suas informações; note que somente os eventos observáveis são transmitidos por esses canais. Embora o Intruso e o Receptor possuam o mesmo poder de observação, somente o Receptor conhece as regras utilizadas pelo Forçador de Opacidade para garantir tal propriedade. Supõe-se que o Intruso possua total conhecimento do modelo do sistema. Os segredos do sistema são representados por estados secretos, que o Intruso nunca deve inferir.

O Forçador de Opacidade aqui proposto funciona da seguinte forma. Quando ele recebe um sinal associado à ocorrência de um evento, ele pode

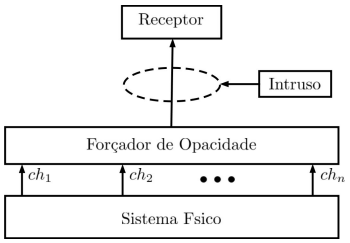

Figura 1: Arquitetura do problema.

liberar esse sinal imediatamente ou segurá-lo até que um outro evento observável ocorra, *i.e.*, o Forçador de Opacidade atrasa a liberação dos eventos por um certo número de passos, em que a contagem de passos é dada pela chegada de eventos no Forçador de Opacidade.

As seguintes hipóteses sobre a capacidade do Intruso são feitas:

I1. O Intruso possui uma cópia completa do autômato que modela o sistema.

I2. O Intruso possui acesso completo aos sinais transmitidos pelo Forçador de Opacidade para o Receptor, *i.e.*, ele é capaz de observar qualquer evento observável.

I3. O Intruso não tem conhecimento sobre a existência do Forçador de Opacidade, portanto, ele não sabe que a informação pode ter sido alterada.

I4. O Intruso sempre espera estimar estados dentro do comportamento do seu modelo ao observar um evento.

São realizadas também as seguintes hipóteses sobre os canais:

A1. Cada canal que conecta o sistema físico e o Forçador de Opacidade transmite um único evento específico de cada vez.

A2. Não há atrasos na transmissão de eventos.

A3. Os canais que ligam o sistema físico e o Forçador de Opacidade são seguros.

A4. Os canais que ligam o Forçador de Opacidade e o Receptor são não seguros.

4 A estratégia para forçar CSO

A estratégia aqui proposta tira proveito da possibilidade de atrasar-se a liberação de eventos para causar trocas na ordem de observação, induzindo o Intruso a fazer estimações erradas do estado atual do sistema. Para tanto, o Forçador de Opacidade verifica se, para toda sequência que atinge um estado secreto e suas continuações, existe alguma possível troca na ordem da observação de eventos de tal modo que a sequência observada seja idêntica a outra cujos prefixos e continuações levem somente a estados não secretos do observador. Para este fim, definem-se:

Definição 2 (Tupla de uma sequência) *Seja* $s = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma^*$ *uma sequência finita. A tupla de uma sequência* $T(s)$ *é o mapeamento* $T : \Sigma^* \to \Sigma^n$*, em que* $n = |s|$*,* $\Sigma^n = \Sigma \times \Sigma \cdots \times \Sigma$*, n vezes, e é definida pela n-upla* $T(s) := (\sigma_1, \sigma_2, \ldots, \sigma_n)$ *formada pelos eventos de s. Dada a n-upla* $t = (t_1, t_2, \ldots, t_n) \in \Sigma^n$*, o mapeamento inverso* $T^{-1} : \Sigma^n \to \Sigma^*$ *é definido pela sequência* $T^{-1}(t) := t_1t_2\ldots t_n$ *formada pela concatenação dos elementos da n-upla t, conforme sua ordem de aparição em t.*

Note que a Definição 2 pode ser estendida a um conjunto de tuplas $T^{-1} : 2^{\Sigma^n} \to 2^{\Sigma^*}$, tal que $T^{-1}(A) := \{s \in \Sigma^* : (\exists t \in A)[s = T^{-1}(t)]\}$.

Definição 3 (Permutação de tuplas) *Seja* $t = (t_1, t_2, \ldots, t_n) \in \Sigma^n$ *uma tupla de ordem n. A permutação de tuplas é o mapeamento* $\mathcal{P} : \Sigma^n \to 2^{\Sigma^n}$*, que associa a cada n-upla t um conjunto* $\mathcal{P}(t)$ *de n-uplas compostas por todas as permutações na ordem de seus elementos, i.e.,* $\mathcal{P}(t) := \{(t_1, t_2, \ldots, t_n), (t_2, t_1, \ldots, t_n), \ldots, (t_n, \ldots, t_2, t_1)\}$.

Note que, dado uma n-upla $t = (t_1, t_2, \ldots, t_n)$, $|\mathcal{P}(t)| \leq n!$.

Definição 4 (Permutação de sequência) *Seja* $s = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma^*$ *e seja* $T(s) = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ *a n-upla formada a partir de s. A permutação de sequência* S_p *é um mapeamento* $S_p : \Sigma^* \to 2^{\Sigma^*}$*, em que, para cada* $s = \sigma_1\sigma_2\ldots\sigma_n$*, associa-se um conjunto* $S_p(s) := T^{-1}(\mathcal{P}(T(s)))$

Por exemplo, para $s = aba$, obtém-se $t = T(s) = (a, b, a)$ e $\mathcal{P}(t) = \{(a, b, a), (a, a, b), (b, a, a)\}$, portanto, $S_p(s) = T^{-1}(\mathcal{P}(t)) = \{aba, aab, baa\}$.

Definição 5 (Forçabilidade de opacidade) *Um sistema modelado pelo autômato* $G = (X, \Sigma, f, \Gamma, x_0)$ *pode ser forçado a tornar-se CSO por meio de trocas na ordem da observação de eventos com respeito a* Σ_o *e* X_s *se* $\forall s : f_{obs}(x_{0,obs}, s) \in X_{s,obs}$*,* $\forall t \in L_{obs}/s$*,* $\exists n \in \mathbb{Z}_+$*, e* $\exists u \in L_{obs}/st : |u| = n$*, e* $\exists v \in S_p(stu) \cap L_{obs}$ *tal que* $f_{obs}(x_{0,obs}, w) \in X_{obs} \backslash X_{s,obs}, \forall w \in Pre(v)$.

Para se definir a estratégia forçadora de opacidade, a ocorrência de um evento no sistema deve ser diferenciada de sua observação pelo Intruso. Para tal distinção, define-se Σ_s como uma cópia de Σ e rotulam-se todos os seus eventos com o subscrito s. Define-se, então, a seguinte operação para rotular os eventos de uma sequência $s \in \Sigma^*$.

Definição 6 (Rotulação) *A operação rotulação é definida por* $R : \Sigma^* \to \Sigma_s^*$*, em que:* $R(\varepsilon) = \varepsilon$*,* $R(\sigma) = \sigma_s$ *e* $R(s\sigma) = (s\sigma)_s = R(s)R(\sigma) = R(s)\sigma_s$*, em que* $\sigma \in \Sigma$ *e* $s \in \Sigma^*$*. A operação inversa* $R^{-1} : \Sigma_s^* \to \Sigma^*$ *é definida por* $R^{-1}(\varepsilon) = \varepsilon$*,* $R^{-1}(\sigma_s) = \sigma$ *e* $R^{-1}((s\sigma)_s) = R^{-1}(s_s\sigma_s) = R^{-1}(s_s)R^{-1}(\sigma_s) = R^{-1}(s_s)\sigma$*.*

Com base na operação de rotulação, o conjunto de observação de eventos é definido por $\Sigma_{o,s} := \{\sigma_s : (\sigma_s = R(\sigma)) \wedge (\sigma \in \Sigma_o)\}$, e o conjunto estendido de eventos por $\Sigma_e := \Sigma_o \cup \Sigma_{o,s}$.

Suponha que a CSO possa ser forçada em um sistema por meio de trocas na ordem da observação de eventos. Portanto, existe uma estratégia forçadora de opacidade OE que, para toda sequência gerada pelo sistema, após a chegada de cada evento, decide entre: (i) liberar imediatamente sua observação, (ii) segurá-lo sem liberar observação alguma ou, (iii) segurá-lo e liberar a observação de um evento retido anteriormente. A fim de se obter a função que traduz a estratégia para forçar opacidade, primeiro define-se a função $\mathcal{N} : \Sigma_e^* \times \Sigma_e \to \mathbb{Z}_+$, em que $\mathcal{N}(s, \sigma)$ representa o número de ocorrências do evento σ na sequência s, e então define-se:

Definição 7 ***(Função Forçadora de opacidade)*** *Seja* $P_{e,s} : \Sigma_e^* \to \Sigma_{o,s}^*$*. A função forçadora de opacidade* $OE : \Sigma_e^* \to 2^{\Sigma_{o,s}} \cup \{\varepsilon\}$ *é definida por* $OE(s_e) = \sigma_s \in \Sigma_{o,s}$ *se* $\mathcal{N}(s_e, \sigma_s) < \mathcal{N}(s_e, R^{-1}(\sigma_s)) \wedge f_{obs}(x_{0,obs}, R^{-1}(P_{e,s}(s_e)\sigma_s)) \in X_{obs} \backslash X_{s,obs}$*, e* $OE(s_e) = \varepsilon$*, caso contrário.*

Quando a função forçadora de opacidade é executada, as sequências s_e, obtidas pela ação do forçador de opacidade, geram uma linguagem estendida L_e, conforme a linguagem gerada pelo sistema e as observações liberadas pelo forçador de opacidade. Esta linguagem é definida recursivamente por:

1. $\varepsilon \in L_e$, pois $OE(\varepsilon) = \varepsilon$, logo, $s_e = \varepsilon \in L_e$
2. $s_e\sigma \in L_e \iff (s_e \in L_e) \wedge (P_{e,o}(s_e\sigma) \in L_{obs})$
3. $s_e\sigma_s \in L_e \iff (s_e \in L_e) \wedge (\sigma_s \in OE(s_e))$

em que $P_{e,o} : \Sigma_e^* \to \Sigma_o^*$, $\sigma \in \Sigma_o$ é a ocorrência de um evento e $\sigma_s \in \Sigma_{o,s} \cup \{\varepsilon\}$ é uma observação liberada pelo forçador de opacidade ou ε, caso nenhuma observação seja liberada.

De acordo com a Definição 7, é possível que $|OE(s_e)| > 1$. Se esse for o caso, então alguma política adicional com relação à liberação de eventos deve ser adotada. A mais intuitiva seria liberar o evento que ocorreu primeiro. Se $OE(s_e) = \varepsilon$, então nenhum evento será liberado. Portanto, uma forma trivial de forçar CSO seria reter todas as observações de eventos, *i.e.*, $OE(s_e) = \varepsilon, \forall s_e$. Tal solução não é viável do ponto de vista prático, e portanto, a quantidade de passos que um evento pode ser retido é limitada para tornar realístico o problema. Assim, as ações do Forçador de Opacidade devem ser restritas a um número máximo de atraso associado a cada evento. Esta propriedade do Forçador de Opacidade é dada por $SD(k) = \{(\sigma_1, k_1), (\sigma_2, k_2), \ldots, (\sigma_n, k_n)\}$, em que $k = [k_1, k_2, \ldots, k_n]$ é um vetor cuja i-ésima componente representa o número máximo de passos que o evento σ_i pode ser retido. Por exemplo, se $k = [1, 0, 0]$, então $SD(k) = \{(\alpha, 1), (\beta, 0), (\gamma, 0)\}$. Portanto, se a sequência $\alpha\beta\gamma$ chegar, o Forçador de Opacidade pode imediatamente liberar α, criando a sequência $\alpha\alpha_s\beta\beta_s\gamma\gamma_s$, ou reter α por um passo, criando a sequência $\alpha\beta\beta_s\alpha_s\gamma\gamma_s$. Portanto, duas sequências de observações, $\alpha_s\beta_s\gamma_s$ ou $\beta_s\alpha_s\gamma_s$, poderiam ser escolhidas dependendo da política adotada para forçar CSO. Dada essa restrição, serão apresentadas as condição necessária e suficiente para que se possa forçar CSO.

Teorema 1 *Seja* $P_{e,o} : \Sigma_e^* \to \Sigma_o^*$*. Então,* G *pode tornar-se CSO com relação a* Σ_o*,* X_s *e* SD *se e somente se* $\forall s : f_{obs}(x_{0,obs}, s) \in X_{s,obs}$*, e* $\forall t \in L_{obs}/s$*,* $\exists n \in \mathbb{Z}_+$ *e* $\exists u \in L_{obs}/st : |u| = n$*,* $\exists s_e \in L_e$ *tal que :* (i) $P_{e,o}(s_e) = stu$*;* (ii) $\forall \sigma \in \Sigma_o$*,* $\mathcal{N}(s_e, \sigma) = \mathcal{N}(s_e, R(\sigma)) + \mathcal{N}(s_e, D(\sigma))$*; e* (iii) $SD(k)$ *é sempre satisfeita.*

5 Algoritmos

Dois algoritmos serão apresentados nesta seção. O primeiro algoritmo verifica se o sistema pode ser forçado a CSO por meio de trocas na ordem de observação de eventos e retorna o autômato que implementa a função OE, enquanto o segundo algoritmo encontra uma solução mínima com respeito a quantos passos cada evento deve ser retido para que seja possível forçar CSO. A permutação da observação dos eventos é realizada pelo autômato D, proposto por Nunes et al. (2018).

5.1 Algoritmo para verificar a factibilidade do forçamento de CSO

O seguinte algoritmo (Algoritmo 1) permite verificar se um dado sistema pode ser forçado a ser CSO para um dado limite de atraso $SD(k)$.

Algoritmo 1 *Verificação do forçamento de CSO*

Entradas: G_{obs}*,* X_s*,* $SD(k)$*.*
Saídas: (True, R_{oe}*) ou False.*

Passo 1. *Defina* $G_{sys} = (X_{sys}, \Sigma_o, f_{sys}, \Gamma_{sys}, x_{0,sys}) = G_{obs}$

Passo 2. *Defina* $G_{int} = (X_{sys}, R(\Sigma_o), f_{int}, \Gamma_{int}, x_{0,sys})$*, em que* $f_{int}(x, \sigma_s) = f_{sys}(x, \sigma)$ *e* $\Gamma_{int}(x) = R(\Gamma_{sys}(x))$

Passo 3. *Construa o autômato* $D = (X_D, \Sigma_e, f_D, \Gamma_D, \nu)$ *conforme Nunes et al. (2018), usando* $SD(k)$ *como entrada*

Passo 4. *Construa* $V := G_{sys}||D||G_{int} = (X_V, \Sigma_e, f_V, \Gamma_V, x_{0,V})$

Passo 5. *Para cada* $x_V = (x_{sys}, x_D, x_{int}) \in X_V$*:*

Passo 5.1. *Se* ***(i)*** $x_{int} \in 2^{X_s}$*, ou;* ***(ii)*** $x_D \neq \nu$ *e* $\Gamma_V(x_V) = \emptyset$*, ou;* ***(iii)*** $[\Gamma_{sys}(x_{sys}) \cap \Gamma_D(x_D)] \setminus \Gamma_V(x_V) \neq \emptyset$*; então remova* x_V*, defina* $V = Ac(V)$ *e comece* Passo 5 *novamente*

Passo 6. *Para cada* $x_V \in X_V$*:*

Passo 6.1. *Se* $\Gamma_V(x_V) \cap \Sigma_s \neq \emptyset$*, então remova todos os eventos* $\sigma \in \Sigma_e \setminus \Sigma_s$ *associados a* x_V *e defina* $V = Ac(V)$

Passo 7. *Defina* $V_{obs} = O_{bs}(V, \Sigma_o)$

Passo 8. *Se* $\mathcal{L}(V_{obs}) = \mathcal{L}(G_{sys})$*, defina* $R_{oe} = V$ *e retorne (True,* R_{oe}*). Se não, retorne False*

No Passo 1 do Algoritmo 1, computa-se o autômato G_{sys}, que modela a ocorrência de eventos observáveis no sistema, sendo igual ao observador de G com respeito a Σ_o. No Passo 2, computa-se o autômato G_{int}, que modela as estimativas do Intruso, como uma cópia de G_{sys} cujos seus eventos foram rotulados com o subscrito s. No Passo 3 constrói-se o autômato D dada a configuração de $SD(k)$ atual, conforme Nunes et al. (2018). No Passo 4 calcula-se o autômato V como a composição paralela entre os autômatos G_{sys}, D e G_{int}. A ideia por trás do cálculo de V é que o autômato D possui todas as possíveis sequências de ocorrências dos eventos e todas as permutações de suas observações com respeito aos limites de $SD(k)$. Desta forma, a primeira composição paralela sincroniza somente as sequências cuja ordem de ocorrência pertence a $\mathcal{L}(G_{sys})$ e a segunda composição paralela elimina as sequências cujas ordens de observação dos eventos não pertencem a $\mathcal{L}(G_{int})$; portanto, somente as sequências s_D tal que $P_{e,o}(s_D) \in \mathcal{L}(G_{sys})$ e $P_{e,s}(s_D) \in \mathcal{L}(G_{int})$ permanecerão. Note que os estados de V, $x_V = (x_{sys}, x_D, x_{int})$, possuem, respectivamente, informação sobre o estado atual da planta, os eventos que estão sendo retidos pelo Forçador de Opacidade, e a estimativa do Intruso.

No passo seguinte do Algoritmo 1 (Passo 5) removem-se todos os estados x_{rem} do autômato V que satisfazem a pelo menos uma das condições: (i) sua terceira componente $x_{int} \in X_{s,obs}$, ou seja, o Intruso estimou com clareza algum estado secreto; (ii) $\Gamma_V((x_{sys}, x_D, x_{int})) = \emptyset$ e $x_D \neq \nu$, em que ν representa "nenhum evento a ser observado", ou seja, uma observação foi retida indefinidamente; (iii) o conjunto de eventos ativos de x_{rem} e o de sua primeira componente x_{sys} são diferentes; portanto, algum evento $\sigma \in \Sigma_o$ que ocorreria em x_{sys} foi inibido em x_{rem}. Repete-se o Passo 5 até que nenhum estado seja removido.

A estratégia de embaralhar as ocorrências dos eventos com suas observações pode causar conflitos de decisões em V, representados por estados que possuem, em seu conjunto de eventos ativos, eventos de Σ_o e Σ_s. Isso significa que o Forçador de Opacidade possui duas ações possíveis: esperar a chegada de uma nova ocorrência ou liberar a observação de um evento. Tais conflitos são removidos no Passo 6, uma vez que para cada estado x_V de V que possui algum evento de observação $\sigma_s \in \Sigma_s$ em seu conjunto de eventos ativos, todas as transições associadas a eventos $\sigma \notin \Sigma_s$ que partem desse estado serão removidos, e em seguida, será tomada a parte acessível de V. Esse passo é justificado fato do Forçador de Opacidade ter que liberar a observação de eventos sempre que possível. No Passo 7 calcula-se o autômato V_{obs} como o observador de V em relação a Σ_o a fim de que a linguagem gerada pelo autômato V_{obs} seja comparada com a linguagem gerada pelo autômato G_{sys} no Passo 8. Essa comparação de linguagens garante que o modelo do Forçador de Opacidade, dado pelo autômato V, não restringe o comportamento do sistema. Caso $\mathcal{L}(V_{obs}) = \mathcal{L}(G_{sys})$, então o Passo 8 calcula o autômato forçador de opacidade $R_{oe} = V$ e retorna ($True$, R_{oe}), mostrando que o forçamento de CSO pode ser realizado com o SD dado; caso contrário, ela retorna *False*.

Com base nos argumentos acima, será proposta a condição necessária e suficiente para forçar CSO por meio de trocas na ordem de observação de eventos.

Teorema 2 *O sistema* G *pode ser forçado a CSO por meio de trocas na ordem da observação de eventos com respeito a* Σ_o*,* X_s *e* $SD(k) = \{(\sigma_1, k_1), (\sigma_2, k_2), \ldots, (\sigma_n, k_n)\}$ *se e somente se* $\mathcal{L}(V_{obs}) = \mathcal{L}(G_{sys})$*, em que* V_{obs} *é o autômato observador calculado no Algoritmo 1.*

5.2 Um algoritmo para minimização dos passos de atrasos

Dado um vetor k^{max}, o Algoritmo 2 calcula um vetor k^{min} cujas componentes são os números mínimos de passos que os eventos podem ser retidos para que seja possível forçar CSO.

Algoritmo 2 *Otimização dos passos de atrasos*

Entradas: G_{obs}*,* X_s*,* $k^{max} = [k_1^{max}, \ldots, k_n^{max}]$.
Saídas: $SD(k^{min}) = \{(\sigma_1, k_1^{min}), (\sigma_2, k_2^{min}), \ldots, (\sigma_n, k_n^{min})\}$ *ou Failure.*

Passo 1. *Crie um vetor* $k = [k_1, \ldots, k_n]$ *e defina* $k_i = 0$ $i = 1, \ldots, n$

PASSO 2. *Defina* $SD(k) = \{(\sigma_1, k_1), (\sigma_2, k_2), \ldots, (\sigma_n, k_n)\}$

PASSO 3. *Se* $X_{obs} \cap 2^{X_s} = \emptyset$*, então retorne* $SD(k)$*, caso contrário:*

PASSO 3.1. *Defina* $k_i = k_i + 1,\ \forall i \in \{1, 2, \ldots, n\} : k_i \neq k_i^{max}$ *e atualize* $SD(k)$

PASSO 3.2. *Enquanto o Algoritmo 1 retornar False e* $k \neq k^{max}$*:*

PASSO 3.2.1. *Defina* $k_i = k_i + 1,\ \forall i \in \{1, 2, \ldots, n\} : k_i \neq k_i^{max}$ *e atualize* $SD(k)$

PASSO 4. *Se o Algoritmo 1 retornar False, então retorne Failure. Caso contrário:*

PASSO 4.1. *Para* $i = 1, 2, \ldots, n$*:*

PASSO 4.1.1. *Defina* $k_i = k_i - 1$ *e atualize* $SD(k)$

PASSO 4.1.2. *Enquanto o Algoritmo 1 retornar True e* $k_i \geq 0$*, defina* $k_i = k_i - 1$ *e atualize* $SD(k)$

PASSO 4.1.3. *Defina* $k_i = k_i + 1$ *e atualize* $SD(k)$

PASSO 5. *Retorne* $SD(k)$

No PASSO 1 do Algoritmo 2 cria-se um vetor de atrasos $k = [k_1, k_2, \ldots, k_n]$ e atribui-se 0 a todos os seus elementos. No PASSO 2 calcula-se $SD(k)$ para forçar opacidade a partir de Σ_o e k. O PASSO 3 verifica se o sistema já é opaco com respeito a Σ_o, X_s e $SD(k)$. Se o sistema já for opaco, então retorne $SD(k) = \{(\sigma_1, 0), \ldots, (\sigma_n, 0)\}$. Por sua vez, se G não é CSO, então o Algoritmo 2 avançará para um loop que procura $SD(k)$ tal que o sistema possa ser forçado a ser CSO por meio de trocas na ordem da observação de eventos. Esse loop funciona da seguinte forma: executa-se o Algoritmo 1 a fim de checar se CSO pode ser forçado com a atual configuração de atrasos máximos $SD(k)$, e caso a resposta seja negativa, cada k_i é incrementado em 1 unidade, a menos que $k_i = k_i^{max}$. O PASSO 3.2 termina quando o forçamento de CSO se torna factível com um dado $SD(k)$ ou quando o vetor k atingir seu valor máximo $k = k^{max}$. No PASSO 4 acha-se uma solução mínima ao se checar o menor valor de cada k_i de forma que o forçamento de opacidade permaneça possível, ou retorna *Failure*. Note que o PASSO 4 seleciona um k_i e decrementa seu valor em 1 unidade até que a opacidade não possa mais ser forçada ou k_i atinja valores negativos, e então volta k_i para o valor anterior, adicionando-se 1. Por último, o PASSO 5 retorna uma configuração mínima para os atrasos $SD(k)$ tal que a CSO possa ser forçado em G por meio de trocas na ordem da observação de eventos.

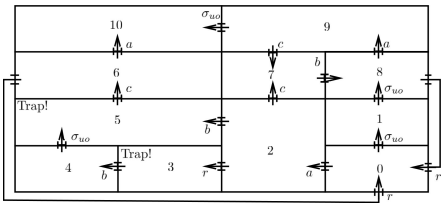

Figura 2: Modelo do "Rato no Labirinto".

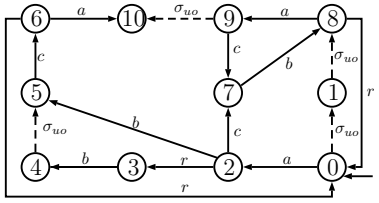

Figura 3: Autômato G que modela o sistema.

6 Exemplo

A fim de ilustrar a estratégia para forçar opacidade proposta neste trabalho, será apresentada uma adaptação de um problema clássico chamado "Rato no labirinto" (Ramadge e Wonham 1989), em que o labirinto é composto por diversas salas conectadas por portas que permitem passagem unidirecional, as quais podem ou não ter sensores instalados, que disparam quando o rato passa por elas. Algumas dessas salas possuem armadilhas, que podem ser ativadas remotamente pelo Intruso. Tanto o dono do sistema quanto o Intruso sabem em quais salas as armadilhas foram postas, ditas "Salas Secretas". O objetivo é impedir que o Intruso estime com clareza a atual posição do rato quando ele estiver nas salas secretas.

O labirinto e seus sensores são mostrados na Figura 2, e o autômato G que modela o sistema é mostrado na Figura 3. O estado inicial representa a sala em que o rato é colocado inicialmente, *i.e.*, $X_0 = \{0\}$, e o conjunto de estados secretos é $X_s = \{3, 5\}$ e representa as salas com armadilhas. O conjunto de eventos é particionado em $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, em que $\Sigma_o = \{a, b, c, r\}$ e $\Sigma_{uo} = \{\sigma_{uo}\}$, que representam, respectivamente, os sinais disparados pelos sensores em algumas portas (note que sensores diferentes podem disparar o mesmo sinal) e a passagem por portas que não possuem sensor. Note que o Receptor e o Intruso realizam suas estimativas de estados conforme o observador de G, representado na Figura 4. Desta forma, o Intruso somente terá certeza de que o rato chegou em um estado secreto quando os estados $\{3\}$ ou $\{5\}$ forem estimados. Dois tubos, que permitem a passagem em uma só direção, foram acoplados por fora do labirinto para conectar as salas 6 e 8 com a sala inicial 0.

Suponha que se queira achar uma política para forçar CSO tal que as observações dos eventos a e b possam ser retidas por no máximo um

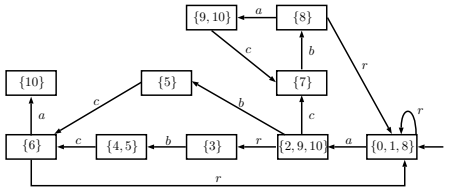

Figura 4: Observador $G_{obs} = O_{bs}(G, \Sigma_o)$.

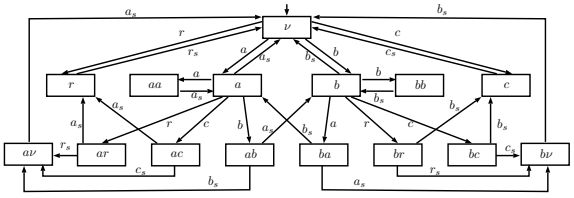

Figura 5: Autômato D.

passo e que os eventos c e r tenham sua observação liberada assim que ocorram. Portanto, define-se $SD(k) = \{(a, 1), (b, 1), (c, 0), (r, 0)\}$.

Para solucionar tal problema, executa-se o Algoritmo 1 a fim de verificar se um Forçador de Opacidade com tais limitações pode ser construído. Para realizar tal verificação, primeiro constrói-se o autômato G_{sys}, conforme o Passo 1 do Algoritmo 1, que é idêntico a G_{obs}. Acompanhando o Algoritmo 1, constrói-se então o autômato G_{int}, idêntico a G_{obs} exceto pelo fato do subscrito s ser adicionado a todos os seus eventos. Em seguida, constrói-se o autômato D, mostrado na Figura 5, conforme Nunes et al. (2018), utilizando os atrasos SD do Forçador de Opacidade. No próximo passo, calcula-se o autômato $V = G_{sys}||D||G_{int}$ e então segue-se para o Passo 5, em que os estados ilegais de V são removidos até que não reste nenhum estado que satisfaça as condições *(i)*, *(ii)* ou *(iii)*. A Figura 6 mostra o trecho do autômato V cujos estados (pintados de cinza) satisfazem a condição de remoção *(i)*. Após a remoção desses estados, deve-se remover os estados gradeados, uma vez que não são mais acessíveis, e, então, repete-se o Passo 5.

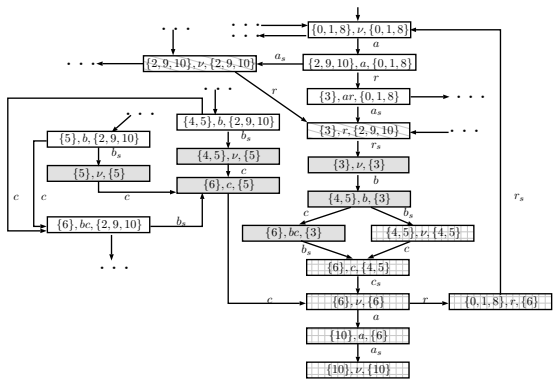

Figura 6: Parte de V mostrando estados ilegais.

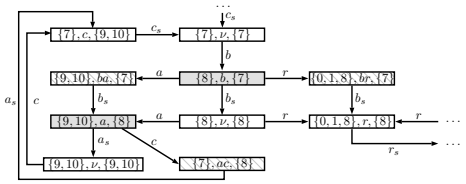

Figura 7: Parte de V com conflitos de decisão.

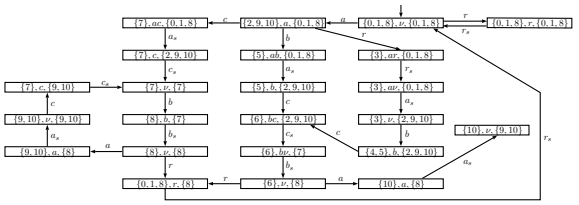

Figura 8: Autômato R_{oe} que implementa a estratégia forçadora de opacidade.

Depois dos estados cinzas e gradeados serem eliminados, o estado hachurado $(\{3\}, r, \{5\})$ satisfará a condição de remoção *(ii)* e também será removido, seguido pela operação de parte acessível e a execução do Passo 5 novamente. O estado hachurado $(\{2, 9, 10\}, \nu, \{2, 9, 10\})$ também será removido, pois passou a satisfazer a condição *(iii)*. Finalizando essas remoções, não haverá estados que satisfaçam as condições de remoção e, consequentemente, o Algoritmo 1 avançará para os próximos passos. No Passo 6, os conflitos de decisão entre esperar a chegada de um evento ocorrido e liberar a observação de um evento serão removidos do autômato V. A Figura 7 apresenta os estados pintados, $(\{8\}, b, \{7\})$ e $(\{9, 10\}, a, \{8\})$, que possuem conflitos de decisão no autômato V, *e.g*, quando o Forçador de Opacidade está no estado $(\{8\}, b, \{7\})$, ele pode esperar a chegada dos eventos ocorridos a ou r, ou pode liberar a observação b_s. Estes conflitos de decisão são eliminados no Passo 6.1 quando as transições relacionadas a Σ_o são removidas dos estados $(\{8\}, b, \{7\})$ e $(\{9, 10\}, a, \{8\})$. Consequentemente, os estados hachurados tornam-se não alcançáveis e deverão ser removidos.

O cálculo do observador V_{obs} do autômato V com relação a Σ_o, $V_{obs} = O_{bs}(V, \Sigma_o)$ é realizado no Passo 7 e a comparação de linguagem $\mathcal{L}(V_{obs}) = \mathcal{L}(G_{sys})$, no Passo 8. Como V_{obs} e G_{sys} possuem a mesma linguagem gerada, o Algoritmo 2 retorna *True* e implementa o autômato forçador de opacidade R_{oe}, conforme a Figura 8.

É importante ressaltar que se o limite máximo de atraso não for suficiente para forçar opacidade, eventualmente o Forçador de Opacidade não liberará nenhuma observação de evento (as trocas na ordem da observação de eventos realizadas não garantiram que a opacidade pudesse ser forçada). Consequentemente, os ramos de V com esse comportamento serão eliminados, pois reter um evento

indefinidamente não é plausível e foge das hipóteses adotadas, portanto $\mathcal{L}(V_{obs}) \neq \mathcal{L}(G_{sys})$.

Note que o autômato R_{oe}, mostrado na Figure 8, possui informações tanto do atual local do sistema como das observações enviadas ao Intruso. Por exemplo, se G gera a sequência $s = rab$, então o rato está na sala secreta 5, porém o Forçador de Opacidade liberará as observações r_s e a_s, reterá b, esperará o evento c ocorrer para liberar $c_s b_s$, gerando $r_s a_s c_s b_s$; desta forma, o Intruso estimará as salas 2, 9 ou 10 para a localização do rato.

Será ilustrado agora o Algoritmo 2. Se forem utilizados como entrada os mesmos G_{obs} e X_s usados anteriormente, e definindo-se $k^{max} = [2, 2, 2, 0]$, o Passo 1 do Algoritmo 2 cria um ventor k e define $k = [0, 0, 0, 0]$. No Passo 2, atualizam-se os atrasos para $SD(k) = \{(a, 0), (b, 0), (c, 0), (r, 0)\}$. No Passo 3, como $X_{obs} \cap 2^{X_s} = \{\{3\}, \{5\}\}$, segue-se para o Passo 3.1, em que incrementa-se cada k_i e atualizam-se os atrasos para $SD(k) = \{(a, 1), (b, 1), (c, 1), (r, 0)\}$. No Passo 3.2, executa-se o Algoritmo 1, que retornará *True*. Note que k_4 não foi incrementado, visto que seu valor máximo já foi alcançado. Como o Algoritmo 1 retornou *True*, então passa-se para o Passo 4 do Algoritmo 2, em que tenta-se achar k^{min}. Primeiro, o Passo 4.1.1 decrementa k_1 e atualiza os atrasos para $SD(k) = \{(a, 0), (b, 1), (c, 1), (r, 0)\}$. O Passo 4.1.2 do Algoritmo 1 retornará *False*, prosseguindo dessa forma para o Passo 4.1.3, em que k_1 é incrementado. Repete-se o Passo 4.1 para $i = 2$, 3 e 4. Por último, o Passo 5 do Algoritmo 2 retorna $SD(k) = \{(a, 1), (b, 1), (c, 0), (r, 0)\}$.

7 Conclusão e trablhos futuros

Neste artigo foi apresentada uma nova estratégia para forçar CSO em SED modelados por autômatos, que tira proveito da possibilidade de atrasar as observações dos eventos ocorridos para permutá-las, fazendo com que o Intruso sempre estime pelo menos um estado não secreto.

O Forçador de Opacidade possui informação sobre os eventos executados pelo sistema e também sobre a liberação de seus sinais. Tal característica pode ser considerada como um avanço nas estratégias de forçamento de opacidade, uma vez que, até o presente momento, para se confundir o Intruso, o Receptor deve ser confundido também. Isto sugere que pode existir um autômato inversor capaz de determinar o estado exato do sistema pelo Receptor.

Por último, a estratégia proposta neste trabalho pode ser aprimorada ao permitir que as observações de eventos possam também ser apagadas, o que pode ser obtido realizando-se a operação de dilatação (Carvalho, Basilio e Moreira 2012) sobre o autômato D.

Referências

Bryans, J. W., Koutny, M. e Ryan, P. Y. (2005). Modelling opacity using petri nets, *Electronic Notes in Theoretical Computer Science* **121**(Supplement C): 101–115.

Carvalho, L. K., Basilio, J. C. e Moreira, M. V. (2012). Robust diagnosis of discrete event systems against intermittent loss of observations, *Automatica* **48**: 2068–2078.

Cassandras, C. G. e Lafortune, S. (2008). *Introduction to Discrete Events Systems*, 2nd edn, Springer, New York, NY : USA.

Cassez, F., Dubreil, J. e Marchand, H. (2012). Synthesis of opaque systems with static and dynamic masks, *Formal Methods in System Design* **40**(1): 88–115.

Dubreil, J., Darondeau, P. e Marchand, H. (2010). Supervisory control for opacity, *IEEE Transactions on Automatic Control* **55**(5): 1089–1100.

Lin, F. (2011). Opacity of discrete event systems and its applications, *Automatica* **47**: 496–503.

Nunes, C. E. V., Moreira, M. V., Alves, M. V. S., Carvalho, L. K. e Basilio, J. C. (2018). Codiagnosability of networked discrete event systems subject to communication delays and intermittent loss of observation, *Discrete Event Dynamic Systems: Theory and Applications* pp. 1–32.

Ramadge, P. J. e Wonham, W. M. (1989). The control of discrete event systems, *Proceedings of the IEEE* **77**(1): 81–98.

Saboori, A. e Hadjicostis, C. N. (2008). Opacity-enforcing supervisory strategies for secure discrete event systems, *47th IEEE Conference on Decision and Control (CDC)*, pp. 889–894.

Wu, Y.-C. e Lafortune, S. (2013). Comparative analysis of related notions of opacity in centralized and coordinated architectures, *Discrete Event Dynamic Systems: Theory and Applications* **23**(3): 307–339.

Wu, Y.-C. e Lafortune, S. (2014). Synthesis of insertion functions for enforcement of opacity security properties, *Automatica* **50**(5): 1336–1348.

Yin, X. e Lafortune, S. (2016). A uniform approach for synthesizing property-enforcing supervisors forpartially-observed discrete-event systems, *IEEE Transactions on Automatic Control* **61**(8): 2140–2154.