Verification of the Observer Property in Discrete Event Systems

P. N. Pena, H. J. Bravo, A. E. C. da Cunha, R. Malik,
S. Lafortune, *Fellow, IEEE,* and J. E. R. Cury, *Member, IEEE*

***Abstract*—The observer property is an important condition to be satisfied by abstractions of Discrete Event System (DES) models. This paper presents a new algorithm that tests if an abstraction of a DES obtained through natural projection has the observer property. The procedure, called *OP-Verifier*, can be applied to (potentially nondeterministic) automata, with no restriction on the existence of cycles of "non-relevant" events. This procedure has quadratic complexity in the number of states. The performance of the algorithm is illustrated by a set of experiments.**

***Index Terms*—Discrete Event Systems, Natural Projections, Observer Property.**

I. Introduction

Natural projections play a central role in the computation of abstractions for Discrete Event Systems (DES) models. Abstractions obtained by natural projections have been extensively used in the Supervisory Control Theory of DES [1] as, for example, in control with partial observation, in hierarchical control [2]–[6], in modular synthesis [2], [7], [8], and in compositional verification of the nonblocking property [2], [9], [10], among many problem domains. In several of the above cited works, the *observer property* is an important condition to be satisfied by the abstracted models. Abstractions satisfying this property are called *OP-abstractions* [11].

The observer property, or simply OP hereafter, was first introduced in the context of hierarchical control of DES. In [12], the abstraction is obtained in the form of a *reporter map*, that projects strings of events of the original (low-level) model, built from a set Σ, into high-level strings built from an independent set of events, denoted by T. Due to some difficulties with the use of reporter maps [13], most of the approaches subsequent to [12] focus on abstractions obtained by the *natural projection*, which maps strings of the original model into strings of the abstraction, by erasing events of Σ that are not contained in a given subset of relevant events, denoted by Σ_r, with $\Sigma_r \subseteq \Sigma$; see [3]–[5], [14], [15].

The structure called *OP-Verifier* was first presented in [11]; it was inspired by an algorithm for testing diagnosability presented in [16]. Given an input automaton G, defined on the alphabet Σ, a set of relevant events $\Sigma_r \subseteq \Sigma$, and a natural projection θ from strings in Σ to strings in Σ_r, the *OP-Verifier* algorithm checks whether the projection $\theta(\mathcal{L}_m(G))$ is an *OP-abstraction*. The *OP-Verifier* algorithm does not require explicitly computing the abstraction to check for the OP and has been shown to have better computational performance when compared to other similar procedures [13], [17]–[19]. It runs in quadratic complexity in the number of states. A limitation of the *OP-Verifier* algorithm as proposed in [11], however, is that it can only be applied to automata that do not have cycles of non-relevant events.

A different algorithm to test the OP is proposed in [13], [17]. This algorithm relies on the computation of a coarsest observation equivalence relation and runs in cubic complexity in the number of states. Yet another algorithm for testing so-called "observerness" for a system G and a mask M is presented in [18]. This procedure may give false negatives as stated and needs to be modified to address this problem [20].

This paper presents a modified version of the *OP-Verifier* algorithm of [11] that subsumes the preliminary results in [21]. This algorithm can be applied to automata with no restriction on the existence of cycles of non-relevant events. The algorithm operates on a modified automaton G_{nr}, obtained from the input automaton G, by aggregating states connected by cycles of non-relevant events. It overcomes the limitations of the previously proposed verifier [11], [21], while retaining its quadratic complexity. The modified *OP-Verifier* algorithm has been implemented in Supremica [22].

This paper is organized as follows. Section II introduces the necessary background. Section III describes the construction of the *OP-Verifier* automaton and its properties. Then Section IV presents an algorithm to construct the OP-Verifier and check the observer property. This section also contains a complexity analysis and experimental results to demonstrate the performance of the algorithm in comparison with [17]. Finally, concluding remarks are given in Section V.

II. Preliminaries

This paper is set in the supervisory control framework. The reader is referred to [1] for a detailed introduction to the theory. Behaviors of DES are modeled using strings of *events*

P. N. Pena is with the Departamento de Engenharia Eletrônica, Universidade Federal de Minas Gerais, Brazil (e-mail: ppena@ufmg.br).

H. J. Bravo and A. E. C. da Cunha are with the Seção de Engenharia Elétrica, Instituto Militar de Engenharia, Brazil (email: hugobravoc@gmail.com, carrilho@ime.eb.br).

R. Malik is with the Department of Computer Science, The University of Waikato, New Zealand (e-mail: robi@waikato.ac.nz).

S. Lafortune is with the Department of Electrical Engineering and Computer Science, The University of Michigan, USA (e-mail: stephane@eecs.umich.edu).

J. E. R. Cury is with the Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina, Brazil (e-mail: jose.cury@ufsc.br).

The first, second, third, and sixth authors are supported in part by CAPES (PROCAD 102/2007). The first and sixth authors are supported in part by FAPEMIG and CNPq grant 300953/93-3, respectively. The research of the fifth author is supported in part by NSF grant CNS-0930081.

taken from a finite alphabet Σ. Σ^* is the set of all finite strings of events in Σ, including the empty string ε. The *concatenation* of strings $s, u \in \Sigma^*$ is written as su. A string $s \in \Sigma^*$ is called a *prefix* of $t \in \Sigma^*$, written $s \leq t$, if there exists $u \in \Sigma^*$ such that $su = t$. A subset $L \subseteq \Sigma^*$ is called a *language*. The *prefix-closure* $\overline{L}$ of a language $L \subseteq \Sigma^*$ is the set of all prefixes of strings in L, i.e., $\overline{L} = \{\, s \in \Sigma^* \mid s \leq t \text{ for some } t \in L \,\}$. Regular languages are represented by (possibly nondeterministic) *finite-state automata* as follows.

Definition 1: A (nondeterministic) finite-state automaton is a tuple $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$, where Σ is a finite set of events, Q is a finite set of *states*, $\rightarrow \,\subseteq Q \times \Sigma \times Q$ is the *state transition relation*, and $Q^\circ \subseteq Q$ is the set of *initial states*. G is *deterministic*, if $|Q^\circ| \leq 1$ and $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces in Σ^* by letting $x \xrightarrow{\varepsilon} x$ for all $x \in Q$, and $x \xrightarrow{s\sigma} z$ if $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$. Furthermore, $x \xrightarrow{s}$ means $x \xrightarrow{s} y$ for some $y \in Q$, and $x \rightarrow y$ means $x \xrightarrow{s} y$ for some $s \in \Sigma^*$. These notations also apply to state sets: $X \xrightarrow{s} Y$ for $X, Y \subseteq Q$ means $x \xrightarrow{s} y$ for some states $x \in X$ and $y \in Y$. Also, if G is an automaton, then $G \xrightarrow{s} x$, $G \xrightarrow{s} X$, and $G \xrightarrow{s}$ stand for $Q^\circ \xrightarrow{s} x$, $Q^\circ \xrightarrow{s} X$, and $Q^\circ \xrightarrow{s}$, respectively. For example, $G \xrightarrow{s} X$ means that the automaton G can reach some state in the set $X \subseteq Q$ on execution of trace $s \in \Sigma^*$. Finally, the *generated language* of automaton G is $\mathcal{L}(G) = \{\, s \in \Sigma^* \mid G \xrightarrow{s} \,\}$.

To express the marking of strings, the alphabet Σ is assumed to contain the *marking event* $\omega \in \Sigma$, which may only appear on self-loops, i.e., $x \xrightarrow{\omega} y$ always implies $x = y$. In this notation, the *marked language* of G is defined as $\mathcal{L}_{\mathrm{m}}(G) = \{\, s \in (\Sigma \setminus \{\omega\})^* \mid s\omega \in \mathcal{L}(G) \,\}$. This paper uses the marking event ω instead of the more conventional set of marking, or final, states, because it simplifies the presentation by associating the marking of strings to a special case of transition.

Given an automaton $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$, a state $x \in Q$ is called *reachable* if $G \rightarrow x$, and *coreachable* if $x \xrightarrow{t\omega}$ for some $t \in \Sigma^*$. The automaton G is called *reachable* if every state $x \in Q$ is reachable, and *nonblocking* if every reachable state $x \in Q$ is coreachable.

A common automaton operation is the *quotient* modulo an equivalence relation on the state set.

Definition 2: Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton and let $\sim \,\subseteq Q \times Q$ be an equivalence relation. The *quotient automaton* of G modulo $\sim$ is

$$G/{\sim} = \langle \Sigma, Q/{\sim}, \rightarrow/{\sim}, \tilde{Q}^\circ \rangle \ , \tag{1}$$

where $\rightarrow/{\sim} = \{\, ([x], \sigma, [y]) \mid x' \xrightarrow{\sigma} y' \text{ for some } x' \in [x] \text{ and } y' \in [y] \,\}$ and $\tilde{Q}^\circ = \{\, [x^\circ] \mid x^\circ \in Q^\circ \,\}$. Here, $[x] = \{\, x' \in Q \mid x \sim x' \,\}$ denotes the *equivalence class* of $x \in Q$, and $Q/{\sim} = \{\, [x] \mid x \in Q \,\}$ is the set of all equivalence classes.

An operation over languages that is very important for abstraction is *natural projection*. For this purpose, the event alphabet is partitioned into $\Sigma = \Sigma_{\mathrm{r}} \,\dot\cup\, \Sigma_{\mathrm{nr}}$, where Σ_{r} denotes the set of *relevant* events, while Σ_{nr} denotes the set of *non-relevant* events. For $\Sigma_{\mathrm{r}} \subseteq \Sigma$, the *natural projection* $\theta\colon \Sigma^* \rightarrow \Sigma_{\mathrm{r}}^*$ maps strings in Σ^* to strings in Σ_{r}^* by erasing all events

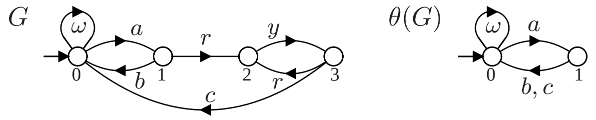

Fig. 1. Example to demonstrate the observer property.

not contained in Σ_{r}. The concept is extended to languages by defining $\theta(L) = \{\, t \in \Sigma_{\mathrm{r}}^* \mid t = \theta(s) \text{ for some } s \in L \,\}$.

This paper is concerned with the property of projections known as the *observer property*, which was first introduced in the context of reporter maps in [12] and [14]. In the context of natural projections, it is written as follows.

Definition 3: [14] Let $L \subseteq \Sigma^*$ be a language, let $\Sigma_{\mathrm{r}} \subseteq \Sigma$, and let $\theta\colon \Sigma^* \rightarrow \Sigma_{\mathrm{r}}^*$ be the natural projection. If for all $s \in \overline{L}$ and all $t \in \Sigma_{\mathrm{r}}^*$ such that $\theta(s)t \in \theta(L)$, there exists $t' \in \Sigma^*$ such that $\theta(st') = \theta(s)t$ and $st' \in L$, then $\theta(L)$ has the *observer property*.

The observer property ensures that, if two states can be reached by traces with the same projection, i.e., $G \xrightarrow{s_1} x_1$ and $G \xrightarrow{s_2} x_2$ with $\theta(s_1) = \theta(s_2)$, then these states can also achieve termination by traces with equal projection, i.e., $x_1 \xrightarrow{t_1\omega}$ implies $x_2 \xrightarrow{t_2\omega}$ with $\theta(t_1) = \theta(t_2)$. If the observer property is satisfied for an automaton, then its natural projection is "observation equivalent" to that automaton, which means that all branching in the automaton remains visible in its projection [12].

Projections can also be applied to automata. Given a deterministic and nonblocking automaton G, its projection $\theta(G)$ is the minimal deterministic recognizer of the language $\theta(\mathcal{L}_{\mathrm{m}}(G))$ [23]. Then it is said that $\theta(G)$ has the observer property if $\theta(\mathcal{L}_{\mathrm{m}}(G))$ has the observer property. In this case $\theta(G)$ is also called an *OP-abstraction*.

Example 1: Automaton G in Fig. 1 models the behavior of a simple manufacturing transfer line with material feedback, adapted from [21], [24]. After starting to manufacture a workpiece (a), the transfer line can either finish production sucessfully (b), or decide to retain the workpiece (r) for one or more rework cycles (y), and eventually finish production with a reworked workpiece (c). Assume that, in some hierarchical control approach, as in [3], [5], [6], one is concerned only with the input-output behavior of the line. Then it is of interest to construct the abstraction $\theta(G)$ with respect to relevant events $\Sigma_{\mathrm{r}} = \{a, b, c, \omega\}$ and non-relevant events $\Sigma_{\mathrm{nr}} = \{r, y\}$, which is shown in Fig. 1. In this case, $\theta(G)$ is not an OP-abstraction. To see this, let $s = ar$ and $t = b$ in Definition 3. Then $\theta(s)t = ab \in \theta(\mathcal{L}_{\mathrm{m}}(G))$, but there is no trace $t' \in \Sigma^*$ such that $st' = art' \in \mathcal{L}_{\mathrm{m}}(G)$ and $\theta(st') = \theta(s)t = ab$.

The *OP-Verifier* algorithm [11] can check for certain projections whether or not they satisfy the observer property. This algorithm, which was inspired by the verifier [16] for testing the property of diagnosability, can only be applied to deterministic automata that do not have cycles of non-relevant events. The automaton G in Fig. 1 has a cycle of non-relevant events involving states 1, 2, and 3. Because of this cycle, the example cannot be classified correctly by the algorithm [11].

III. Verification of the Observer Property

In this section, the *OP-Verifier* algorithm is presented. It extends the algorithm in [11] by adding the ability to handle cycles of non-relevant events.

A. Strongly $\Sigma_{\rm nr}$-Connected Components Automaton $G_{\rm nr}$

In order to deal with cycles of non-relevant events, a *strongly* $\Sigma_{\rm nr}$*-connected components automaton* is introduced. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton, and let $\Sigma_{\rm nr} \subseteq \Sigma$ be a set of non-relevant events. Define the following relations on the state set Q:

$$x \overset{\rm nr}{\rightarrow} y \iff x \overset{s}{\rightarrow} y \text{ for some } s \in \Sigma_{\rm nr}^* \ ; \tag{2}$$

$$x \overset{\rm nr}{\leftrightarrow} y \iff x \overset{\rm nr}{\rightarrow} y \text{ and } y \overset{\rm nr}{\rightarrow} x \ . \tag{3}$$

If $x \overset{\rm nr}{\leftrightarrow} y$, then the states x and y are called *strongly* $\Sigma_{\rm nr}$*-connected* ($\Sigma_{\rm nr}$-SC), because it is possible to reach each state from the other using only non-relevant events. If G does not contain two distinct $\Sigma_{\rm nr}$-SC states it is said to be $\Sigma_{\rm nr}$*-acyclic*.

A set of $\Sigma_{\rm nr}$-SC states is called a *strongly* $\Sigma_{\rm nr}$*-connected component* ($\Sigma_{\rm nr}$-SCC). If each $\Sigma_{\rm nr}$-SCC is contracted to a single state, the resulting automaton is $\Sigma_{\rm nr}$-acyclic. This contracted automaton is called the strongly $\Sigma_{\rm nr}$-connected components automaton ($\Sigma_{\rm nr}$-SCC automaton) of G in the following. Formally, the $\Sigma_{\rm nr}$-SCC of state $x \in Q$ is

$$[x] = \{\, y \in Q \mid x \overset{\rm nr}{\leftrightarrow} y \,\} \ , \tag{4}$$

and the $\Sigma_{\rm nr}$-SCC automaton of G is the quotient automaton constructed by merging the $\Sigma_{\rm nr}$-SCCs in G,

$$G_{\rm nr} = G/\overset{\rm nr}{\leftrightarrow} \ . \tag{5}$$

Remark 1: In graph theory, the $\Sigma_{\rm nr}$-SCC automaton is called a *condensation graph*, which is known to be *acyclic* [25], i.e., it does not contain any cycles of non-relevant events except for self-loops. For a finite state set, it follows that for every state $x \in Q$, there exists a state $y \in Q$ such that $x \overset{\rm nr}{\rightarrow} y$, with $[y]$ a *terminal* component, i.e., a component with no further $\Sigma_{\rm nr}$-transitions outgoing to other components.

Definition 4: Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton, and let $\Sigma_{\rm nr} \subseteq \Sigma$. For $y \in Q$, the component $[y]$ is $\Sigma_{\rm nr}$*-terminal* if, for all $\sigma \in \Sigma_{\rm nr}$ and all $z \in Q$ such that $[y] \overset{\sigma}{\rightarrow} [z]$, it holds that $[y] = [z]$.

The strongly connected components of a graph can be computed efficiently using *Tarjan's Algorithm* [26]. This algorithm has a worst-case time complexity of $O(|\!\rightarrow\!|)$, i.e., it is linear in the number of transitions. Tarjan's Algorithm can be easily adapted to compute the $\Sigma_{\rm nr}$-SCC automaton.

B. OP-Verifier V_G

Based on the $\Sigma_{\rm nr}$-SCC automaton, the *OP-Verifier* V_G is constructed. The *OP-Verifier* is a nondeterministic automaton that is used to determine whether or not the observer property is satisfied for the original automaton G and non-relevant events $\Sigma_{\rm nr}$. It is constructed in a similar way to the previous *OP-Verifier* for $\Sigma_{\rm nr}$-acyclic automata in [11], except that it is based on the $\Sigma_{\rm nr}$-SCC automaton $G_{\rm nr}$ instead of G.

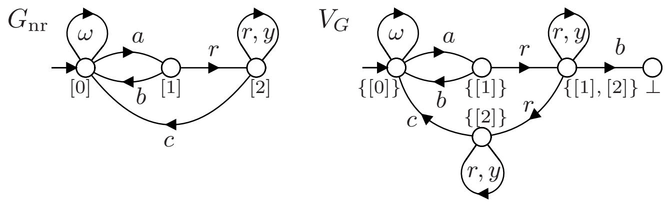

Fig. 2. Example of OP-Verifier construction.

Definition 5: Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton with $\Sigma_{\rm nr}$-SCC automaton $G_{\rm nr} = \langle \Sigma, Q_{\rm nr}, \rightarrow_{\rm nr}, \tilde{Q}^\circ \rangle$, and let $\Sigma = \Sigma_{\rm r} \,\dot{\cup}\, \Sigma_{\rm nr}$. The *OP-Verifier* V_G for G is

$$V_G = \langle \Sigma, Q_V, \rightarrow_V, Q_V^\circ \rangle \tag{6}$$

where

- $Q_V = \{\, P \subseteq Q/\overset{\rm nr}{\leftrightarrow} \mid 1 \le |P| \le 2 \,\} \cup \{\perp\}$.
  The state set of the verifier consists of sets of $\Sigma_{\rm nr}$-SCCs of G of cardinality one or two, i.e, a single $\Sigma_{\rm nr}$-SCC and pairs of $\Sigma_{\rm nr}$-SCCs, plus the special state $\perp$.
- $\rightarrow_V$ consists of the following transitions:

$$\{[x],[y]\} \overset{\sigma}{\rightarrow} \{[x'],[y']\} \quad \text{if } \sigma \in \Sigma_{\rm r},\ [x] \overset{\sigma}{\rightarrow}_{\rm nr} [x'], \text{ and } [y] \overset{\sigma}{\rightarrow}_{\rm nr} [y']; \tag{7}$$

$$\{[x],[y]\} \overset{\sigma}{\rightarrow} \{[x'],[y]\} \quad \text{if } \sigma \in \Sigma_{\rm nr} \text{ and } [x] \overset{\sigma}{\rightarrow}_{\rm nr} [x']; \tag{8}$$

$$\{[x],[y]\} \overset{\sigma}{\rightarrow} \perp \quad \text{if } \sigma \in \Sigma_{\rm r},\ [x] \overset{\sigma}{\rightarrow}_{\rm nr},\ [y] \text{ is terminal, and } [y] \overset{\sigma}{\not\rightarrow}_{\rm nr}. \tag{9}$$

- $Q_V^\circ = \{\, \{[x^\circ],[y^\circ]\} \mid x^\circ, y^\circ \in Q^\circ \,\}$.
  The initial state set of the verifier contains all pairs of $\Sigma_{\rm nr}$-SCCs of initial states of G.

Example 2: The $\Sigma_{\rm nr}$-SCC automaton corresponding to G in Example 1 is $G_{\rm nr}$ shown in Fig. 2. The $\Sigma_{\rm nr}$-SCCs are $[0] = \{0\}$, $[1] = \{1\}$ and $[2] = \{2,3\}$. Notice that $[2]$ is $\Sigma_{\rm nr}$-terminal. The verifier V_G, shown in Fig. 2, contains the following transitions: from (7), $\{[0]\} = \{[0],[0]\} \overset{a}{\rightarrow} \{[1],[1]\} = \{[1]\}$, $\{[0]\} \overset{\omega}{\rightarrow} \{[0]\}$, $\{[1]\} \overset{b}{\rightarrow} \{[0]\}$, and $\{[2]\} \overset{c}{\rightarrow} \{[0]\}$; from (8), $\{[1]\} \overset{r}{\rightarrow} \{[1,2]\}$, $\{[1,2]\} \overset{r}{\rightarrow} \{[1,2]\}$, $\{[1,2]\} \overset{r}{\rightarrow} \{[2]\}$, $\{[1,2]\} \overset{y}{\rightarrow} \{[1,2]\}$, $\{[2]\} \overset{r}{\rightarrow} \{[2]\}$, and $\{[2]\} \overset{y}{\rightarrow} \{[2]\}$; and, from (9), $\{[1],[2]\} \overset{b}{\rightarrow} \perp$, since $[1] \overset{b}{\rightarrow}_{\rm nr} [0]$, $[2]$ is $\Sigma_{\rm nr}$-terminal, and $[2] \overset{b}{\rightarrow}_{\rm nr}$ does not hold. Note that state $\perp$ is reachable in V_G. It is shown below that this is a necessary and sufficient condition to confirm that $\theta(G)$ is not an OP-abstraction.

C. Properties of the OP-Verifier

This section establishes a key property of the OP-Verifier. The special state $\perp$ is reachable in the OP-Verifier if and only if the observer property is *not* satisfied. The main result in Theorem 3 depends on two lemmas to relate traces with the same projection to the states of the verifier: the OP-Verifier contains all pairs of $\Sigma_{\rm nr}$-SCCs that can be reached by traces that project to the same relevant events.

Lemma 1: Let V_G be the verifier for automaton G. Let $a, b \in \Sigma^*$ such that $\theta(a) = \theta(b)$ and $G \overset{a}{\rightarrow} x_a$ and $G \overset{b}{\rightarrow} x_b$. Then there exists $s \in \Sigma^*$ such that $\theta(a) = \theta(b) = \theta(s)$ and $V_G \overset{s}{\rightarrow} \{[x_a],[x_b]\}$.

Proof: The claim is shown by induction on $n = |a| + |b|$.

In the base case, $n = 0$ and thus $a = b = \varepsilon$. Then $x_a, x_b \in Q^\circ$ and thus $\{[x_a], [x_b]\} \in Q_V^\circ$, i.e., $V_G \xrightarrow{\varepsilon} \{[x_a], [x_b]\}$.

Now assume the claim has been shown for all $a, b \in \Sigma^*$ such that $\theta(a) = \theta(b)$ and $|a| + |b| \le n$. Consider $a, b \in \Sigma^*$ such that $\theta(a) = \theta(b)$ and $|a| + |b| = n + 1$ and $G \xrightarrow{a} x_a$ and $G \xrightarrow{b} x_b$. As $\theta(a) = \theta(b)$ and $|a| + |b| > 0$, either at least one of the traces a or b ends with an event in $\Sigma_{\rm nr}$, or both end with the same event in $\Sigma_{\rm r}$.

In the first case, assume without loss of generality $a = a'\sigma$ for some $\sigma \in \Sigma_{\rm nr}$. Then $G \xrightarrow{a'} x'_a \xrightarrow{\sigma} x_a$ and $\theta(a') = \theta(a'\sigma) = \theta(a) = \theta(b)$, and by inductive assumption there exists $s \in \Sigma^*$ such that $\theta(s) = \theta(a') = \theta(b)$ and $V_G \xrightarrow{s} \{[x'_a], [x_b]\}$. Given $x'_a \xrightarrow{\sigma} x_a$ with $\sigma \in \Sigma_{\rm nr}$, it follows by construction of V_G (8) that $V_G \xrightarrow{s} \{[x'_a], [x_b]\} \xrightarrow{\sigma} \{[x_a], [x_b]\}$, i.e., $V_G \xrightarrow{s\sigma} \{[x_a], [x_b]\}$ with $\theta(s\sigma) = \theta(s) = \theta(a') = \theta(a) = \theta(b)$.

In the second case, $a = a'\sigma$ and $b = b'\sigma$ for some $\sigma \in \Sigma_{\rm r}$. Then $G \xrightarrow{a'} x'_a \xrightarrow{\sigma} x_a$ and $G \xrightarrow{b'} x'_b \xrightarrow{\sigma} x_b$ and $\theta(a') = \theta(b')$. By inductive assumption there exists $s' \in \Sigma^*$ such that $\theta(s') = \theta(a') = \theta(b')$ and $V_G \xrightarrow{s'} \{[x'_a], [x'_b]\}$. Given $x'_a \xrightarrow{\sigma} x_a$ and $x'_b \xrightarrow{\sigma} x_b$ with $\sigma \in \Sigma_{\rm r}$, it follows by construction of V_G (7) that $V_G \xrightarrow{s'} \{[x'_a], [x'_b]\} \xrightarrow{\sigma} \{[x_a], [x_b]\}$, i.e., $V_G \xrightarrow{s'\sigma} \{[x_a], [x_b]\}$ with $\theta(s'\sigma) = \theta(a'\sigma) = \theta(a)$ and $\theta(s'\sigma) = \theta(b'\sigma) = \theta(b)$. ■

Lemma 2: Let $V_G = \langle \Sigma, Q_V, \rightarrow_V, Q_V^\circ \rangle$ be the verifier for automaton G. Let $s \in \Sigma^*$ and $\{A, B\} \in Q_V$ such that $V_G \xrightarrow{s} \{A, B\}$. Then there exist $a, b \in \Sigma^*$ such that $\theta(a) = \theta(b) = \theta(s)$ and $G \xrightarrow{a} A$ and $G \xrightarrow{b} B$.

Proof: The claim is shown by induction on $n = |s|$.

In the base case, $n = 0$ and thus $s = \varepsilon$. Note that $A, B \in Q_V^\circ$. By Def. 5 there exist $x_a^\circ \in A$ such that $x_a^\circ \in Q^\circ$ and $x_b^\circ \in A$ such that $x_b^\circ \in Q^\circ$, which is enough to show $G \xrightarrow{\varepsilon} A$ and $G \xrightarrow{\varepsilon} B$.

Now consider $s = s'\sigma$ such that $V_G \xrightarrow{s'} \{A', B'\} \xrightarrow{\sigma} \{A, B\}$, and assume by inductive assumption that there exist $a', b' \in \Sigma^*$ such that $\theta(a') = \theta(b') = \theta(s')$ and $G \xrightarrow{a'} A'$ and $G \xrightarrow{b'} B'$. Consider two cases.

If $\sigma \in \Sigma_{\rm nr}$, then by construction of V_G (8), without loss of generality, there exist $x'_a \in A'$ and $x_a \in A$ such that $x'_a \xrightarrow{\sigma} x_a$, and $B' = B$. As $G \xrightarrow{a'} A'$, there exists $y'_a \in A'$ such that $G \xrightarrow{a'} y'_a$. Furthermore, $x'_a, y'_a \in A'$ implies $x'_a \overset{\rm nr}{\leftrightarrow} y'_a$, i.e., there exists $t \in \Sigma_{\rm nr}^*$ such that $y'_a \xrightarrow{t} x'_a$. Thus $G \xrightarrow{a'} y'_a \xrightarrow{t} x'_a \xrightarrow{\sigma} x_a \in A$. It follows that $\theta(a't\sigma) = \theta(a'\sigma) = \theta(s'\sigma) = \theta(s)$, $\theta(b') = \theta(s') = \theta(s'\sigma) = \theta(s)$, $G \xrightarrow{a't\sigma} A$, and $G \xrightarrow{b'} B' = B$.

If $\sigma \in \Sigma_{\rm r}$, then by construction of V_G (7) there exist $x'_a \in A'$, $x_a \in A$, $x'_b \in B'$, and $x_b \in B$ such that $x'_a \xrightarrow{\sigma} x_a$ and $x'_b \xrightarrow{\sigma} x_b$. As $G \xrightarrow{a'} A'$, there exists $y'_a \in A'$ such that $G \xrightarrow{a'} y'_a$. Furthermore, $x'_a, y'_a \in A'$ implies $x'_a \overset{\rm nr}{\leftrightarrow} y'_a$, i.e., there exists $t_a \in \Sigma_{\rm nr}^*$ such that $y'_a \xrightarrow{t_a} x'_a$. Thus $G \xrightarrow{a'} y'_a \xrightarrow{t_a} x'_a \xrightarrow{\sigma} x_a \in A$ and $\theta(a't_a\sigma) = \theta(a'\sigma) = \theta(s'\sigma) = \theta(s)$. Likewise, there exists $y'_b \in B'$ and $t_b \in \Sigma_{\rm nr}^*$ such that $G \xrightarrow{b'} y'_b \xrightarrow{t_b} x'_b \xrightarrow{\sigma} x_b \in B$ and $\theta(b't_b\sigma) = \theta(b'\sigma) = \theta(s'\sigma) = \theta(s)$. ■

Theorem 3: Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic nonblocking automaton. The special state $\perp$ is reachable in V_G if and only if $\theta(G)$ is not an OP-abstraction.

Proof: First assume that $\perp$ is reachable in V_G. By construction of V_G (9), this means that there exists a reachable state $\{A, B\} \in Q_V$ such that $V_G \xrightarrow{s} \{A, B\} \xrightarrow{\sigma} \perp$, where $\sigma \in \Sigma_{\rm r}$, $x_a \xrightarrow{\sigma} y_a$ for some $x_a \in A$ and $y_a \in Q$, and B is a terminal component such that $B \xrightarrow{\sigma}$ does not hold. By Lemma 2, there exists $a, b \in \Sigma^*$ such that $\theta(a) = \theta(b) = \theta(s)$ and $G \xrightarrow{a} A$ and $G \xrightarrow{b} B$. Then there exists $z_a \in A$ and $t_a \in \Sigma_{\rm nr}^*$ such that $G \xrightarrow{a} z_a \xrightarrow{t_a} x_a \xrightarrow{\sigma} y_a$, and since G is nonblocking, there exists $u \in \Sigma^*$ such that $y_a \xrightarrow{u\omega}$. Thus,

$$G \xrightarrow{a} z_a \xrightarrow{t_a} x_a \xrightarrow{\sigma} y_a \xrightarrow{u\omega} \ . \tag{10}$$

Now let $s_0 = b$ and $t_0 = \sigma\theta(u)$. Then $\theta(s_0)t_0 = \theta(b)\sigma\theta(u) = \theta(a)\sigma\theta(u) = \theta(at_a\sigma u) \in \theta(\mathcal{L}_{\rm m}(G))$. However, there does not exist $t' \in \Sigma^*$ such that $\theta(s_0t') = \theta(s_0)t_0$ and $s_0t' \in \mathcal{L}_{\rm m}(G)$, because if such t' exists then $\theta(t') = t_0 = \sigma\theta(u)$ and then as $s_0 = b$ and $G \xrightarrow{b} B$ and B is a terminal component, also $G \xrightarrow{s_0} B \xrightarrow{\sigma}$. However, B was chosen such that $B \xrightarrow{\sigma}$ does not hold.

Conversely, assume $\theta(G)$ is not an OP-abstraction. Then there exist $s \in \mathcal{L}(G)$ and $t \in \Sigma_{\rm r}^*$ such that $\theta(s)t \in \theta(\mathcal{L}_{\rm m}(G))$, and there does not exist $t' \in \Sigma^*$ such that $\theta(st') = \theta(s)t$ and $st' \in \mathcal{L}_{\rm m}(G)$. Let $u \le t\omega$ be the longest prefix of $t\omega$ such that there exists $u' \in \Sigma^*$ such that $\theta(u') = u$ and $su' \in \mathcal{L}(G)$. Clearly $u \ne t\omega$, since otherwise t' as above exists.

So let $t = u\sigma v$ with $\sigma \in \Sigma_{\rm r}$ and $v \in \Sigma_{\rm r}^*$. Then $\theta(su)\sigma v = \theta(s)u\sigma v = \theta(s)t \in \theta(\mathcal{L}_{\rm m}(G))$, so there exists $a \in \Sigma^*$ such that $\theta(a) = \theta(su)$ and $G \xrightarrow{a} x_a \xrightarrow{\sigma}$. Since $su' \in \mathcal{L}(G)$, there exists $x'_b \in Q$ such that $G \xrightarrow{su'} x'_b$. By Remark 1, there exist $t_b \in \Sigma_{\rm nr}^*$ and $x_b \in Q$ such that $x'_b \xrightarrow{t_b} x_b$ and $[x_b]$ is a terminal component. Let $b = su't_b$. Note that $\theta(a) = \theta(su) = \theta(s)u = \theta(su') = \theta(su't_b) = \theta(b)$ and $G \xrightarrow{a} x_a$ and $G \xrightarrow{b} x_b$. By Lemma 1, there exists $s_{ab} \in \Sigma^*$ such that $\theta(s_{ab}) = \theta(a) = \theta(b)$ and $V_G \xrightarrow{s_{ab}} \{[x_a], [x_b]\}$. Here it holds that $[x_a] \xrightarrow{\sigma}$ and $[x_b]$ is a terminal component and $[x_b] \xrightarrow{\sigma}$ does not hold, because otherwise $G \xrightarrow{b} x_b \xrightarrow{z\sigma}$ for some $z \in \Sigma_{\rm nr}^*$, and thus $bz\sigma = su't_bz\sigma \in \mathcal{L}(G)$ in contradiction to the maximal choice of u above. It follows by construction of V_G (9) that $V_G \xrightarrow{s_{ab}} \{[x_a], [x_b]\} \xrightarrow{\sigma} \perp$. ■

Based on Theorem 3, the observer property can be checked by constructing the verifier automaton and checking whether it contains the state $\perp$.

IV. Implementation

A. The OP-Verifier Algorithm

Algorithm 1 shows the pseudo-code of the *OP-Verifier* algorithm; this pseudo-code is the basis of the implementation of the OP-verifier algorithm within Supremica [22], which is further discussed in Section IV-C. The algorithm explores the state space of the verifier until a transition to $\perp$ is encountered, or until all possible verifier states have been constructed.

Verifier states are represented as ordered pairs $([x], [y])$ to represent a set $\{[x], [y]\} \in Q/\overset{\rm nr}{\leftrightarrow}$, with singletons $\{x\}$ represented as $([x], [x])$. To exploit the symmetry, all pairs are ordered such that $[x] < [y]$ based on a fixed but arbitrary ordering of the $\Sigma_{\rm nr}$-SCC. The algorithm maintains the set

Algorithm 1 OP-Verifier algorithm

1: **input** $G = \langle \Sigma, Q, \to, Q^\circ \rangle$
2: calculate $G_{\mathrm{nr}} = \langle \Sigma, Q/\overset{\mathrm{nr}}{\leftrightarrow}, \to_{\mathrm{nr}}, Q^\circ_{\mathrm{nr}} \rangle$
3: $queue \leftarrow \langle$empty queue$\rangle$
4: $visited \leftarrow \langle$empty hash set$\rangle$
5: **for all** $[x] \in Q/\overset{\mathrm{nr}}{\leftrightarrow}$ **do**
6: $\quad \mathrm{expand}([x],[x])$
7: **end for**
8: **while** $queue$ not empty **do**
9: $\quad$ remove $([x],[y])$ from $queue$
10: $\quad \mathrm{expand}([x],[y])$
11: **end while**
12: **stop** "The observer property is satisfied."

13: **procedure** $\mathrm{expand}([x],[y])$
14: **for all** $\sigma \in \Sigma_{\mathrm{r}}$ **do**
15: $\quad$ **if** $[x] \xrightarrow{\sigma}_{\mathrm{nr}}$ and $[y] \xrightarrow{\sigma}_{\mathrm{nr}}$ **then**
16: $\quad\quad$ **for all** $[x] \xrightarrow{\sigma}_{\mathrm{nr}} [x']$ and $[y] \xrightarrow{\sigma}_{\mathrm{nr}} [y']$ **do**
17: $\quad\quad\quad \mathrm{enqueue}([x'],[y'])$
18: $\quad\quad$ **end for**
19: $\quad$ **else if** $[x] \xrightarrow{\sigma}_{\mathrm{nr}}$ and $[y] \not\xrightarrow{\sigma}_{\mathrm{nr}}$ and y is terminal **or** $[y] \xrightarrow{\sigma}_{\mathrm{nr}}$ and $[x] \not\xrightarrow{\sigma}_{\mathrm{nr}}$ and x is terminal **then**
20: $\quad\quad$ **stop** "The observer property is not satisfied."
21: $\quad$ **end if**
22: **end for**
23: **for all** $\sigma \in \Sigma_{\mathrm{nr}}$ **do**
24: $\quad$ **for all** $[x] \xrightarrow{\sigma}_{\mathrm{nr}} [x']$ **do**
25: $\quad\quad \mathrm{enqueue}([x'],[y])$
26: $\quad$ **end for**
27: $\quad$ **for all** $[y] \xrightarrow{\sigma}_{\mathrm{nr}} [y']$ **do**
28: $\quad\quad \mathrm{enqueue}([x],[y'])$
29: $\quad$ **end for**
30: **end for**

31: **procedure** $\mathrm{enqueue}([x],[y])$
32: **if** $[x] = [y]$ **then**
33: $\quad$ **return**
34: **else if** $[x] > [y]$ **then**
35: $\quad \mathrm{enqueue}([y],[x])$
36: **else if** $([x],[y]) \notin visited$ **then**
37: $\quad$ add $([x],[y])$ to $visited$
38: $\quad$ add $([x],[y])$ to $queue$
39: **end if**

$visited$ containing all pairs $([x],[y])$ discovered so far, and a $queue$ containing those pairs that still need to be explored.

After construction of the Σ_{nr}-SCC automaton using Tarjan's Algorithm [26], the loop in line 5 examines every Σ_{nr}-SCC $[x] \in Q/\overset{\mathrm{nr}}{\leftrightarrow}$ and records the successors of the verifier state $\{[x]\}$ according to (7) and (8). This step assumes that all states are reachable. Afterwards, the loop in line 8 visits and expands all verifier states $\{[x],[y]\}$ resulting from the previous loop in line 5, and their successors.

Procedure expand checks for transitions originating from a verifier state $\{[x],[y]\}$. For relevant events, the loop in line 14 checks in line 15 for successor pairs according to (7), and then checks in line 19 whether condition (9) is satisfied. If so, the verifier clearly contains the state $\perp$, so the algorithm terminates and reports that the observer property is not satisfied. For non-relevant events, the loop in line 23 constructs successor pairs according to (8). Procedure $\mathrm{enqueue}$ adds new state pairs to the set $visited$ and to the $queue$, ensuring the ordering and using the hash set $visited$ to prevent any pairs from being processed more than once.

B. Complexity

The complexity of the *OP-Verifier* algorithm is determined by the complexity to construct the verifier. If the input is a deterministic automaton $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ then the number of reachable states of V_G is bounded by $|Q|^2 + |Q| + 1 = O(|Q|^2)$. To estimate the number of transitions of V_G, consider a transition $x \xrightarrow{\sigma} x'$ in the input automaton G, and let $y \in Q$ be an arbitrary state. If $\sigma \in \Sigma_{\mathrm{r}}$, then this produces at most one transition $\{[x],[y]\} \xrightarrow{\sigma} \{[x'],[y']\}$ or $\{[x],[y]\} \xrightarrow{\sigma} \perp$ according to (7) or (9), and if $\sigma \in \Sigma_{\mathrm{nr}}$, then there is one transition $\{[x],[y]\} \xrightarrow{\sigma} \{[x'],[y]\}$ according to (8). That is, every transition of G produces up to $|Q|$ transitions in V_G. The deterministic automaton G has up to $|\Sigma||Q|$ transitions, so the total number of transitions of V_G is bounded by

$$|{\to}||Q| \le |\Sigma||Q|^2 = O(|\Sigma||Q|^2) \ . \tag{11}$$

Tarjan's algorithm to identify the Σ_{nr}-SCC runs in $O(|{\to}|) = O(|\Sigma||Q|)$ time [26], so it is dominated by the verifier construction. Therefore, (11) gives the worst-case time complexity of the OP-Verifier algorithm.

C. Experimental Results

The OP-Verifier algorithm has been implemented in Java and integrated in the discrete event systems tool Supremica [22]. Table I shows some experimental results to demonstrate the performance of the implementation. All experiments were run on a standard desktop computer using a single core 2.33 GHz CPU and 3 GB of RAM.

The test suite consists of 23 automata obtained as intermediate results during compositional nonblocking verification [9], and variations of such automata. The table shows for each automaton that was checked, the number of states $|Q|$, the number of events $|\Sigma|$, the total number of transitions $|{\to}|$, and the number of non-relevant transitions $|\overset{\mathrm{nr}}{\to}|$. Then the table shows the number of states $|Q_V|$ constructed by the OP-Verifier algorithm, and the time taken to check the observer property. Furthermore, the time taken by Supremica [22] to compute a coarsest observation equivalence relation using the method in [27] is shown. This is the crucial step of the observer property verification algorithm proposed in [17] and is indicative of its performance. Finally, the table shows the verdict whether or not the given automaton satisfies the observer property.

Table I shows that the OP-Verifier algorithm can easily check automata with more than 100,000 states in a few seconds. With one exception, the number of verifier states is of the same order of magnitude as the number of states of the automaton, and the OP-Verifier algorithm runs significantly faster than observation equivalence. This is particularly true

TABLE I
EXPERIMENTAL RESULTS

Automaton				OP-Verifier		Obseq.	
$\lvert Q\rvert$	$\lvert\Sigma\rvert$	$\lvert\rightarrow\rvert$	$\lvert\overset{\text{nr}}{\rightarrow}\rvert$	$\lvert Q_V\rvert$	Time	Time	Verdict
14934	38	118793	16087	31023	0.01 s	0.21 s	false
14934	8985	118793	1272	16207	0.01 s	0.09 s	false
18816	42	532812	82656	101484	0.05 s	7.29 s	false
19538	38	115369	41105	60807	0.02 s	0.94 s	false
19538	6028	114952	23616	43373	0.02 s	0.73 s	false
21867	30	67058	34661	56530	0.02 s	0.23 s	false
21867	16777	67058	2	21870	0.01 s	0.08 s	true
22599	22	156361	35667	58287	0.02 s	1.04 s	false
22599	19062	156361	2187	24786	0.02 s	0.14 s	true
22634	26	67100	35655	58291	0.02 s	0.32 s	false
22634	9462	67100	2110	26030	0.01 s	0.11 s	true
23313	40	232014	23784	47099	0.02 s	0.49 s	false
23313	14366	232014	648	23961	0.02 s	0.22 s	true
24938	32	75399	39305	64245	0.02 s	0.27 s	false
24938	16265	75399	1932	27676	0.01 s	0.09 s	true
31216	40	278251	33122	64340	0.03 s	0.57 s	false
31216	14139	278251	4614	36750	0.03 s	0.25 s	true
36277	18	288995	54049	90928	0.04 s	3.38 s	false
36277	34013	288995	4926	41205	0.02 s	0.27 s	false
49152	44	1546504	319488	368663	0.18 s	93.89 s	false
49152	434	1446152	122880	5147232	27.71 s	2.17 s	true
105619	20	680591	326173	1231667	1.01 s	18.38 s	false
136656	36	1273580	249928	387611	0.17 s	9.37 s	false

when the observer property is not satisfied, because the OP-Verifier algorithm can terminate early as soon as the state $\perp$ is encountered during construction of the verifier. The case where the OP-Verifier is slower than observation equivalence has the largest number of non-relevant transitions among the examples that satisfy the observer property, while the observation equivalence algorithm quickly finds a good partition in this case. In all other cases, the OP-Verifier algorithm gives an answer in less than two seconds.

V. CONCLUSIONS

The *OP-Verifier* algorithm presented in this paper allows to efficiently check whether an abstraction obtained by a natural projection has the observer property. The procedure is a modified version of a previous one [11], which removes a restriction on the existence of cycles of non-relevant events while still ensuring quadratic complexity in the number of states. The new version of the verifier first merges all states connected by cycles of non-relevant events. The resulting (non-deterministic) automaton is then translated into a transition structure, in which the observer property is checked by verifying the reachability of a specific state. We are currently investigating how the *OP-Verifier* can be used to improve the OP-Search algorithm [19] in order to help computing reduced *OP-abstractions*.

REFERENCES

[1] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.

[2] R. C. Hill and D. M. Tilbury, "Modular Supervisory Control of Discrete Event Systems with Abstraction and Incremental Hierarchical Construction," in *8th Int. Workshop on Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 399–406.

[3] A. E. C. da Cunha and J. E. R. Cury, "Hierarchical Supervisory Control Based on Discrete Event Systems with Flexible Marking," *IEEE Trans. Autom. Control*, vol. 52, no. 12, pp. 2242–2253, Dec. 2007.

[4] L. Feng and W. M. Wonham, "Supervisory control architecture for discrete-event systems," *IEEE Trans. Autom. Control*, vol. 53, no. 6, pp. 1449–1461, July 2008.

[5] K. Schmidt, T. Moor, and S. Perk., "Nonblocking hierarchical control of decentralized discrete event systems," *IEEE Trans. Autom. Control*, vol. 53, no. 10, pp. 2252–2265, Nov. 2008.

[6] K. Schmidt and C. Breindl, "Maximally Permissive Hierarchical Control of Decentralized Discrete Event Systems," *IEEE Trans. Autom. Control*, vol. 56, no. 4, pp. 723–737, April 2011.

[7] K. Schmidt, H. Marchand, and B. Gaudin, "Modular and Decentralizd Supervisory Control of Concurrent Discrete Event Systems Using Reduced Systems Models," in *8th Int. Workshop on Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 149–154.

[8] L. Feng and W. M. Wonham, "Computationally Efficient Supervisor Design: Abstraction and Modularity," in *8th Int. Workshop on Discrete Event Systems, WODES' 06*, Ann Arbor, MI, USA, July 2006, pp. 3–8.

[9] H. Flordal and R. Malik, "Compositional verification in supervisory control," *SIAM J. Control and Optimization*, vol. 48, no. 3, pp. 1914–1938, 2009.

[10] P. N. Pena, J. E. R. Cury, and S. Lafortune, "Verification of Nonconflict of Supervisors Using Abstractions," *IEEE Trans. Autom. Control*, vol. 54, no. 12, pp. 2803–2815, 2009.

[11] ——, "Polynomial-Time Verification of the Observer Property in Abstractions," in *2008 American Control Conference, ACC '08*, Seattle, USA, June 2008, pp. 465–470.

[12] K. C. Wong and W. M. Wonham, "Hierarchical Control of Discrete-Event Systems," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 6, no. 3, pp. 241–273, 1996.

[13] L. Feng and W. M. Wonham, "On the computation of natural observers in discrete-event systems," *Discrete Event Dynamic Systems*, vol. 20, no. 1, pp. 63–102, 2010.

[14] K. C. Wong, J. G. Thistle, R. P. Malhamé, and H.-H. Hoang, "Supervisory Control of Distributed Systems: Conflict Resolution," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 10, pp. 131–186, 2000.

[15] K. Schmidt and C. Breindl, "On maximal permissiveness of hierarchical and modular supervisory control approaches for discrete event systems," in *9th Int. Workshop on Discrete Event Systems, WODES '08*, Göteborg, Sweden, 2008, pp. 462–467.

[16] T. Yoo and S. Lafortune, "Polynomial-time verification of diagnosability of partially observed discrete-event systems," *IEEE Trans. Autom. Control*, vol. 47, no. 9, pp. 1491– 1495, Sep. 2002.

[17] K. C. Wong and W. M. Wonham, "On the Computation of Observers in Discrete-Event Systems," *Discrete Event Dynamical Systems*, vol. 14, no. 1, pp. 55–107, Jan. 2004.

[18] S. Jiang, R. Kumar, and H. E. Garcia, "Optimal sensor selection for discrete-event systems with partial observation," *IEEE Trans. Autom. Control*, vol. 48, no. 3, pp. 369–381, March 2003.

[19] P. N. Pena, J. E. R. Cury, R. Malik, and S. Lafortune, "Efficient Computation of Observer Projections using OP-Verifiers," in *10th Int. Workshop on Discrete Event Systems, WODES '10*, Berlin, Germany, Aug. 2010, pp. 416–421.

[20] R. Kumar and P. N. Pena, "Private communication," Oct. 2012.

[21] H. J. Bravo, A. E. C. da Cunha, P. Pena, R. Malik, and J. E. R. Cury, "Generalised verification of the observer property in discrete event systems," in *11th Int. Workshop of Discrete Event Systems, WODES '12*, Guadalajara, Mexico, 2012, pp. 337–342.

[22] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems," in *8th Int. Workshop of Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, 2006, pp. 384–385.

[23] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.

[24] H. Zhong and W. M. Wonham, "On the Consistency of Hierarchical Supervision in Discrete-Event Systems," *IEEE Trans. Autom. Control*, vol. 35, no. 10, pp. 1125–1134, Oct. 1990.

[25] L. R. Foulds, *Graph theory applications*. Universitext, 1992.

[26] R. Tarjan, "Depth first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, Jun. 1972.

[27] T. Bolognesi and S. A. Smolka, "Fundamental results for the verification of observational equivalence: a survey," in *Protocol Specification, Testing and Verification VII: IFIP WG6.1 7th Int. Conf. on Protocol Specification, Testing and Verification*, H. Rudin and C. H. West, Eds. Amsterdam, The Netherlands: North-Holland, 1987, pp. 165–179.