2008 American Control Conference
Westin Seattle Hotel, Seattle, Washington, USA
June 11-13, 2008

Polynomial-Time Verification of the Observer Property in Abstractions

Patrícia N. Pena, José E.R. Cury and Stéphane Lafortune

Abstract— **This paper presents an algorithm to test if an abstraction obtained through natural projection has the observer property, without having to compute the abstraction. The original automaton and the set of events to be kept by the projection are inputs to the algorithm. An automaton, the verifier, is built such that the verification of the property becomes a verification of reachability of a special state. The complexity of the algorithm is polynomial in the size of the state space of the automaton. Two examples are presented to illustrate the algorithm.**

I. INTRODUCTION

The observer property characterizes languages obtained by natural projection. It appears in many works in supervisory control of discrete event systems, such as [1], [2], [3], [4], [5], [6], [7].

It is well known that the complexity of obtaining natural projections is exponential (with the number of states of the non-deterministic automaton), in the worst case. It is also well known that the state space of the minimal automaton that generates the projected language may grow exponentially with the number of states of the original minimal automaton [8]. However, if the projection has the observer property, the number of states is at worst of the same size as the original automaton [8]. The observer property was first introduced in the context of hierarchical control of discrete-event systems [1] for prefix-closed languages and was extended to non-prefix-closed languages in [2].

Reference [9] discusses the complexity of obtaining abstractions with the observer property and finds that the problem of obtaining minimal abstractions (in terms of the number of states and transitions) is NP-hard. Reference [9] also presents a polynomial algorithm that gives, from an initial set of relevant events, a reasonable extension for this set such that the projection has the observer property. This algorithm is an extension of previous work [8], where an algorithm that renames events that cause the projection to fail the observer property is presented. For the authors of reference [10], it is not clear what should be the initial set of relevant events in [8]. So, they propose that this set be composed of the shared events of the decentralized subsystems.

In this paper, we propose a polynomial algorithm to test if, given an automaton and a set of events to be kept in the projection, the resulting natural projection has the observer property, without having to compute it. To the best of our knowledge, this is the first polynomial test for the observer property. Our algorithm makes use of a *verifier* to help reasoning about the property. It is inspired by the verifier introduced in [11], for the purpose of testing the property of diagnosability of languages. When searching for a projection with the observer property, if we use the algorithm presented in this paper, the projection need only be computed after knowing that it will have the property. This avoids performing unnecessary calculations. Calculating intermediate projections, iteratively obtained until the projection with the observer property is found, may increase the complexity of the overall problem exponentially.

In previous work of the authors, the observer property was used in the context of local modular control of discrete event systems [12]. The idea was to apply the nonconflict test over abstractions instead of applying it over the set of supervisors [6], [7]. For this new test to work, the abstractions, obtained through natural projections, have to obey certain conditions. Among the conditions, there is one that states that the abstractions have to be OP-abstractions, namely, their languages have the observer property.

In Section II, a review of some basic concepts of languages and automata theory is presented, along with the definition of the observer property. In Section III, we first introduce an auxiliary automaton denoted by M. We characterize the observer property in terms of the language of M. Then we present an algorithm that constructs the verifier that is used for checking the observer property. Two theorems that support the algorithm and the conclusions drawn from it are established. Section IV presents two examples that illustrate the testing procedure. The last section presents the conclusions.

II. PRELIMINARIES

The paper is set in the supervisory control framework of Ramadge and Wonham [13]. We refer the reader to [14] or [15] for a detailed introduction to the theory. In this framework, a DES is modeled as an automaton $G = (Q, \Sigma, \delta, q_0, F)$, where Q is the set of states, Σ is the set of events, δ is the transition function, q_0 is the initial state, and F is the set of marked states. Σ^* is the set of all finite traces of elements in Σ, including the empty trace ϵ. A language is a subset of Σ^*. The behavior of G, modeled as a language $\mathcal{L}(G) \subseteq \Sigma^*$, is the set of finite traces that G can generate. G can model a second language, $\mathcal{L}_m(G) \subseteq \mathcal{L}(G)$, that is the set of traces that represent completed tasks (or, equivalently, that end in marked states).

P. Pena - Department of Electronics Engineering (DELT) - Federal University of Minas Gerais, Belo Horizone, MG, Brazil. ppena@ufmg.br
J.E.R. Cury - Department of Automation and Systems (DAS) - Federal University of Santa Catarina, Florianópolis, SC, Brazil. cury@das.ufsc.br
S. Lafortune - Department of Electrical Engineering and Computer Science (EECS) - The University of Michigan, Ann Arbor, MI, USA. stephane@eecs.umich.edu

The natural projection $\theta : \Sigma^* \to \Sigma_r^*$ is an operation over languages defined over two sets of events, Σ and Σ_r, where $\Sigma_r \subseteq \Sigma$. The set Σ_r is sometimes referred to as the relevant event set. The natural projection maps traces in Σ to traces in Σ_r by erasing the occurrences of events in $\Sigma \setminus \Sigma_r$ and is defined as follows:

$$\begin{aligned}\theta(\epsilon) &= \epsilon \\ \theta(\sigma) &= \begin{cases} \epsilon & \text{if } \sigma \notin \Sigma_r \\ \sigma & \text{if } \sigma \in \Sigma_r \end{cases} \\ \theta(s\sigma) &= \theta(s)\theta(\sigma) \text{ with } s \in \Sigma^*, \sigma \in \Sigma.\end{aligned}$$

The concept of natural projection can be extended to languages as follows:

$$\theta(L) = \{u_i \in \Sigma_r^* \,|\, u_i = \theta(u) \text{ for some } u \in L\}.$$

The property of projections known as the *observer property* is presented in Definition II.1.

Definition II.1 [2] *Let $L \subseteq \Sigma^*$ be a language, $\Sigma_r \subseteq \Sigma$ an event set and $\theta : \Sigma^* \to \Sigma_r^*$ the natural projection of traces in Σ^* to traces in Σ_r^*. If $(\forall m \in \overline{L})(\forall n \in \Sigma_r^*)\ \ \theta(m)n \in \theta(L) \implies (\exists p \in \Sigma^*)\ \theta(mp) = \theta(m)n$ and $mp \in L$ then $\theta(L)$ has the observer property.*

We will use the same notation θ for projections of languages, as in $\theta(L)$, or for projections of automata, as in $\theta(G)$. In the latter case, $\theta(G)$ will denote the trim deterministic automaton that marks the projected language $\theta(\mathcal{L}_m(G))$. We will say that $\theta(G)$ is an "OP-abstraction" if the corresponding language $\theta(\mathcal{L}_m(G))$ has the observer property. So, we can assume without loss of generality that G is a trim automaton. We present an example where the projection violates the observer property.

Example II.1 *Let the automaton G be the one presented in Fig. 1(a). The deterministic automaton associated with the natural projection erasing events $\Sigma_u = \{x, z\}$ is presented in Fig. 1(b).*

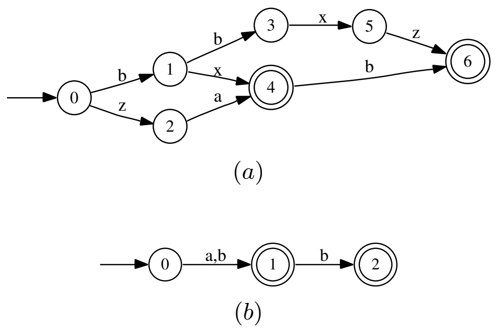

Fig. 1. Example II.1: (a) G; (b) $\theta(G)$.

To show that this projection does not have the observer property, we choose $m = z$ and $n = b$, such that $\theta(z)b = b \in \theta(\mathcal{L}_m(G))$. However, there is no $p \in \Sigma^*$ such that $\theta(zp) = \theta(z)b$ and $zp \in \mathcal{L}_m(G)$. That means that for this choice of Σ_r, the projection $\theta(\mathcal{L}_m(G))$ does not have the observer property; stated differently, $\theta(G)$ is not an OP-abstraction.

In the next section we present the main results of the paper.

III. Main Results

The main result of this paper consists of a procedure to be applied over a automaton $G = (Q^G, \Sigma, \delta^G, q_0^G, F^G)$, given the set of relevant events Σ_r, to check if $\theta(G)$, where $\theta : \Sigma^* \to \Sigma_r^*$, is an OP-abstraction. To get rid of the marking of G, we use an auxiliary M where marked states in G are replaced by unmarked states with self-loops labeled with an special event τ (Section III-A). Using M, a trace in the marked language $\mathcal{L}_m(G)$ is represented as a trace that ends with this special event in $\mathcal{L}(M)$. Then, the observer property is translated into a test, termed OP-Test, over the language $\mathcal{L}(M)$ (Section III-B). In the following subsection (Section III-C), we introduce the OP-verifier, a transition structure that translates the OP-Test into the verification of the reachability of a specific state. The failure of the OP-Test causes a state named $Dead$ in the OP-verifier to be reached.

A. Auxiliary Transition Structure M

The first step is to obtain $M = (Q^M, \Sigma^\tau, \delta^\tau, q_0^M)$ from the original automaton G. Q^M is the set of states and $Q^M = Q^G$. The set of events of M is denoted by $\Sigma^\tau = \Sigma \cup \{\tau\}$ and τ is considered relevant, namely, $\tau \in \Sigma_r$. The transition function δ^τ is defined as:

$$\delta^\tau(q,\sigma) = \begin{cases} \delta^G(q,\sigma) & \text{if } q \in Q^M, \sigma \neq \tau \\ q & \text{if } q \in F^G, \sigma = \tau \end{cases}$$

namely, the transition function δ^τ of M has the same transitions as G plus self-loops labeled with τ in the states corresponding to marked states in G. The initial state q_0^M of M corresponds to the initial state of G, q_0^G. This representation is convenient because it allows to represent a marked state of G as a state in M where a special transition, labeled by τ, is defined.

The language generated by M is:

$$\mathcal{L}(M) = \{s' \in \Sigma^{\tau *} | \delta^\tau(q_0^M, s') \text{ is defined}\}.$$

The language $\mathcal{L}_m(G)$ can be re-defined in terms of the traces of $\mathcal{L}(M)$ as:

$$\mathcal{L}_m(G) = \{s \in \Sigma^* | s' = s\tau \text{ with } s' \in \mathcal{L}(M)\}.$$

We define the language $N \subseteq \mathcal{L}(M)$ as the sublanguage of $\mathcal{L}(M)$ composed of traces in Σ^*, namely,

$$N = \mathcal{L}(M) \cap \Sigma^*.$$

It is trivial to show that $N = \mathcal{L}(G)$.

We define a function $T : \Sigma^* \to 2^{\Sigma^*}$ as:

$$T(s) = \{t \in \Sigma^* | st\tau \in \mathcal{L}(M)\}$$

namely, $T(s)$ returns the set of traces $t \in \Sigma^*$ such that $st\tau \in \mathcal{L}(M)$.

A restriction imposed on the automaton G, for the purpose of the results in this paper, is that G does not have cycles of non-relevant events.

B. OP-Test

In this section, we present a test to be applied over M, such that it is possible to determine if $\theta(\mathcal{L}_m(G))$ has the observer property. The OP-Test is as follows:

OP-Test If $s_1, s_2 \in N$ such that $\theta(s_1) = \theta(s_2)$ then $\theta(T(s_2)) = \theta(T(s_1))$.

Namely, for all $s_1, s_2 \in N$ with $\theta(s_1) = \theta(s_2)$, $\forall t \in \Sigma^*$ such that $s_2 t\tau \in \mathcal{L}(M)$, $\exists v \in \Sigma^*$ such that $s_1 v\tau \in \mathcal{L}(M)$ and $\theta(t) = \theta(v)$. In such a case, we say that M satisfies the OP-Test.

Theorem III.1 shows that $\theta(G)$ is an OP-abstraction if and only if the condition presented above can be verified for all pairs $s_1, s_2 \in N$ with $\theta(s_1) = \theta(s_2)$.

Theorem III.1 *Let the automaton G, the transition structure M, the languages $\mathcal{L}(M)$ and N, the natural projection θ and the OP-Test be as presented before. We have that:*

M satisfies the OP-Test $\Leftrightarrow \theta(G)$ is an ***OP-abstraction****.*

This proof uses the definition of the observer property and the OP-Test. It is omitted due to lack of space. Complete details can be found in [16].

Example III.1 uses the G and Σ_r of Example II.1 to illustrate how the OP-Test detects the violation of the observer property.

Example III.1 *The automaton G and the transition structure M are presented in Figures 1(a) and 2, respectively. Let N be as defined before. To show that M does not*

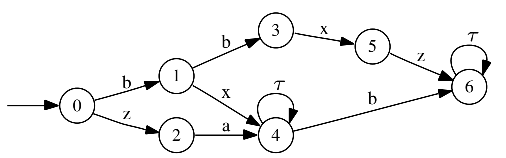

Fig. 2. Example III.1: M.

satisfy the OP-Test, we choose $s_2 = \epsilon$, $s_1 = z$, such that $\theta(s_2) = \theta(s_1)$ and $s_1, s_2 \in N$. The trace $bx \in T(s_2)$ but $\theta(bx) = b \notin \theta(T(s_1))$. Notice that those traces are the same ones that violated the observer property, in Example II.1.

The OP-Test, as stated, does not provide an effective way of verifying the observer property. In the next section, we introduce a transition structure, named OP-verifier, that translates the OP-Test into a problem of reachability of a state named *Dead*.

C. OP-verifier

The OP-verifier, denoted by V_G, is a nondeterministic transition structure obtained from M whose states represent pairs of states of M that have the same projection. We present in this section an algorithm to build the OP-verifier. This algorithm uses as inputs the transition structure M and the set of relevant events Σ_r.

Let $M = (Q^M, \Sigma^\tau, \delta^\tau, q_0^M)$ be as defined before and $En^M(q) = \{\sigma \in \Sigma^\tau | \delta^\tau(q, \sigma)!\}$ be the set of events defined at state q of M. The execution of the algorithm results in a structure $V_G = (Q, \Sigma^\tau, \delta, q_0)$, where:

- $Q \in Q^M \times Q^M \cup \{Dead\}$ is the set of states;
- $\Sigma^\tau = \Sigma \cup \{\tau\}$ is the event set;
- $q_0 = (q_0^M, q_0^M)$ is the initial state;
- $\delta : Q \times \Sigma \rightarrow Q$ is the extended transition function. It is defined later, as procedure $\delta(q)$.

As in the verifier introduced in [11], the OP-verifier is a nondeterministic automaton. At first, the only known state is the initial one. The other states are enumerated as they are reached from the initial state of V_G. The procedure to build V_G is presented Fig. 3, where $\Sigma_u = \Sigma - \Sigma_r$. The transition function δ, presented in Fig.4, is applied over each state of Q_{T+1} that is not in Q_T, to generate the new set of reached states to be included in Q_{T+1}. This procedure is applied iteratively, until $Q_{T+1} = Q_T$. At the end, we have the set of reachable states of V_G. If the state *Dead* is included in Q, then we can say that the observer property is violated in $\mathcal{L}_m(G)$. Actually, in the algorithm, we halt the execution once state *Dead* is reached from any state in Q_{T+1}.

```
MainAlgorithm
$Q_{T+1} = \{(0,0)\}$
$Q_T = \{\ \}$
$\forall q \in (Q_{T+1} - Q_T)$
$Q_T = Q_T \cup q$
$\delta(q)$
if $Dead \in Q_{T+1}$
    $quit$
end
end
$V_G = (Q_T, \Sigma^\tau, \delta, q_0)$
```

Fig. 3. Main Algorithm.

Like the verifier presented in [11], the OP-verifier has its transition function defined differently for each type of event: σ relevant ($\sigma \in \Sigma_r$) and σ non-relevant ($\sigma \in \Sigma_u$).

Let $s_1, s_2 \in N$, with $\theta(s_1) = \theta(s_2)$, be traces that take M from the initial state to states q_1 and q_2, respectively. The observer property, restated in terms of traces with the same projection and their suffixes in what we called OP-Test before, establishes that $\theta(T(s_2)) = \theta(T(s_1))$, namely, that the set of projections of the suffixes of s_2 must be equal to set of projections of the suffixes of s_1.

The algorithm builds the OP-verifier, whose states represent pairs of states of M, so that the traces that reach those states of V_G are traces of M with the same projection. A state *Dead* is reached when a relevant event σ is defined in q_2 ($s_2\sigma \in N$) of a state (q_1, q_2) and neither σ is defined in q_1 ($s_1\sigma \notin N$) nor there is a non-relevant event α defined in q_1 ($\nexists \alpha \in \Sigma_u$ such that $s_1\alpha \notin N$) or if the dual is true, namely, when a relevant event σ is defined in q_1 ($s_1\sigma \in N$) of a state (q_1, q_2) and neither σ is defined in q_2 ($s_2\sigma \notin N$)

```
$\delta(q)$
$q_1 = q(1)$
$q_2 = q(2)$
$\forall \sigma \in En(q) = En^M(q_1) \cup En^M(q_2)$
    if $\sigma \in \Sigma_r$
        if $\delta^\tau(q_1,\sigma)!$ & $\delta^\tau(q_2,\sigma)!$
            $\delta((q_1,q_2),\sigma) = (\delta^\tau(q_1,\sigma), \delta^\tau(q_2,\sigma))$
            $Q_{T+1} = Q_{T+1} \cup \{(\delta^\tau(q_1,\sigma), \delta^\tau(q_2,\sigma))\}$
        elseif $\delta^\tau(q_1,\sigma)! \,\&\, En^M(q_2) \cap \Sigma_u = \emptyset$ or $\delta^\tau(q_2,\sigma)! \,\&\, En^M(q_1) \cap \Sigma_u = \emptyset$
            $\delta((q_1,q_2),\sigma) = (Dead)$
            $Q_{T+1} = Q_{T+1} \cup \{(Dead)\}$
        end
    else
        if $\delta^\tau(q_1,\sigma)!$
            $\delta((q_1,q_2),\sigma) = (\delta^\tau(q_1,\sigma), q_2)$
            $Q_{T+1} = Q_{T+1} \cup \{(\delta^\tau(q_1,\sigma), q_2)\}$
        end
        if $\delta^\tau(q_2,\sigma)!$
            $\delta((q_1,q_2),\sigma) = (q_1, \delta^\tau(q_2,\sigma))$
            $Q_{T+1} = Q_{T+1} \cup \{(q_1, \delta^\tau(q_2,\sigma))\}$
        end
    end
end
```

Fig. 4. Transition Function Algorithm $\delta(q)$

nor there is a non-relevant event α defined in q_2 ($\nexists\alpha \in \Sigma_u$ such that $s_2\alpha \notin N$).

This situation characterizes that $\theta(T(s_2)) \neq \theta(T(s_1))$ namely,

- for at least one suffix $v \in \Sigma^*$ of s_1 ($s_1 v\tau \in \mathcal{L}(M)$) it is true that $\nexists t \in \Sigma^*$ with $s_2 t\tau \in \mathcal{L}(M)$ and $\theta(t) = \theta(v)$ or
- for at least one suffix $t \in \Sigma^*$ of s_2 ($s_2 t\tau \in \mathcal{L}(M)$) it is true that $\nexists v \in \Sigma^*$ with $s_1 v\tau \in \mathcal{L}(M)$ and $\theta(t) = \theta(v)$.

The first state to be analyzed is state $(0,0)$, where $q_1 = 0 \in Q^M$ and $q_2 = 0 \in Q^M$. Based on the events defined in state $q_2 \in Q^M$ we find which states of V_G are reached from the state q. The new states reached are included in Q_{T+1} and state $(0,0)$ is copied to set Q_T. This procedure is iteratively performed until $Q_{T+1} = Q_T$.

State $Dead$ is reached if and only if $\theta(\mathcal{L}(M))$ does not have the observer property. This result is presented in Theorem III.2.

Theorem III.2 *Let M, V_G, θ and the OP-Test be as previously defined. State Dead is not reachable in V_G if and only if M satisfies the OP-Test.*

The proof of Theorem III.2 can be found in [16].

Corollary III.2.1 follows from Theorems III.1 and III.2.

Corolary III.2.1 *$\theta(G)$ is an OP-abstraction $\Leftrightarrow$ state $Dead$ is not reached in the OP-verifier V_G.*

Overall, the procedure to verify the observer property is done in three steps, given G and Σ_r:

- Obtain the transition structure M from G, by the introduction of self-loops labeled with τ in all marked states of G and remove the marking of the states;
- Apply the algorithm to build the OP-verifier V_G from M;
- Classify $\theta(G)$ as an OP-abstraction in case state $Dead$ is not reachable in V_G or as not being an OP-abstraction otherwise.

The complexity involved in this algorithm is polynomial in the number of states of M, in the worst case. The worst condition would be when the number of states of the OP-verifier is equal to the number of pairs of states of M, that is n^2 (n is the number of states of M). In general, the number of states of V_G will be less than n^2. Moreover, in case the observer property is violated, meaning that state $Dead$ is reached, the algorithm halts and the OP-verifier may not be fully constructed.

The next section presents two examples that illustrate the procedure of building V_G.

IV. Examples

Two examples are presented, one that has the observer property and another that does not have the observer property.

Example IV.1 *Let $G = (Q^G, \Sigma, \delta^G, q_0^G, F^G)$ be the automaton and its projection as presented in Figures 5(a) and 5(b). The set of non-relevant events is $\Sigma_u = \Sigma - \Sigma_r = \{x, z\}$.*

To obtain M, self-loops labeled with event τ are to the marked states of G, as shown in Fig.6.

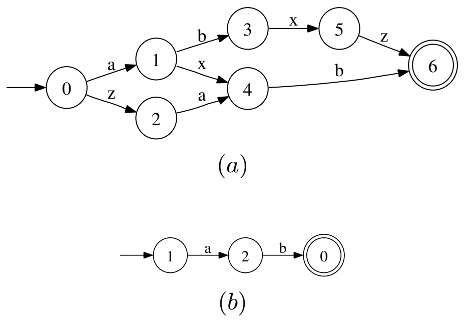

Fig. 5. Example IV.1: (a) G; (b) $\theta(G)$.

The execution of the algorithm presented in Figures 3 and 4 for the construction of V_G from M is shown in the following.

$(0,0).a.(1,1) \rightarrow$ Line 15
$(0,0).z.(2,0) \rightarrow$ Line 23
$(0,0).z.(0,2) \rightarrow$ Line 27
$(0,2).a.(1,4) \rightarrow$ Line 15
$(0,2).z.(2,2) \rightarrow$ Line 23
$(2,2).a.(4,4) \rightarrow$ Line 15
$(4,4).b.(6,6) \rightarrow$ Line 15
$(6,6).\tau.(6,6) \rightarrow$ Line 15
$(1,4).b.(3,6) \rightarrow$ Line 15
$(1,4).x.(4,4) \rightarrow$ Line 23
$(3,6).x.(5,6) \rightarrow$ Line 23
$(5,6).z.(6,6) \rightarrow$ Line 23
$(2,0).a.(4,1) \rightarrow$ Line 15
$(2,0).z.(2,2) \rightarrow$ Line 27
$(4,1).b.(6,3) \rightarrow$ Line 15
$(4,1).x.(4,4) \rightarrow$ Line 27
$(6,3).x.(6,5) \rightarrow$ Line 27
$(6,5).z.(6,6) \rightarrow$ Line 27
$(1,1).b.(3,3) \rightarrow$ Line 15
$(1,1).x.(4,1) \rightarrow$ Line 23
$(1,1).x.(1,4) \rightarrow$ Line 27
$(3,3).x.(5,3) \rightarrow$ Line 23
$(3,3).x.(3,5) \rightarrow$ Line 27
$(5,3).x.(5,5) \rightarrow$ Line 27
$(5,3).z.(6,3) \rightarrow$ Line 23
$(3,5).x.(5,5) \rightarrow$ Line 23
$(3,5).z.(3,6) \rightarrow$ Line 27
$(5,5).z.(6,5) \rightarrow$ Line 23
$(5,5).z.(5,6) \rightarrow$ Line 27

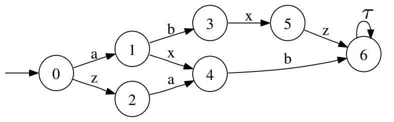

Fig. 6. Example IV.1: M.

The verifier V_G is presented in Figure 7.

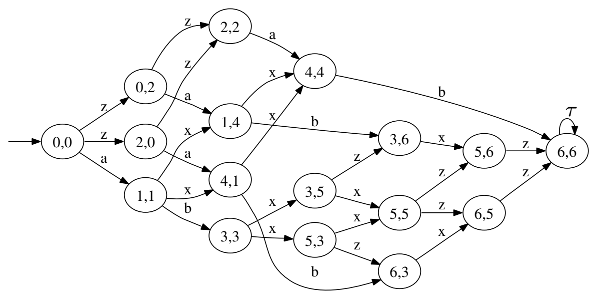

Fig. 7. Example IV.1. Verifier V_G obtained from the algorithm.

State $Dead$ is not reached so, we can conclude that $\theta(G)$ is an OP-abstraction.

Now, we present and example in which the same choice of relevant events leads to a projection that is not an OP-abstraction.

Example IV.2 *Let G_b be as in Figure 8(a). Notice that now we have states 4 and 6 marked and a transition labeled with b from state 0 to 1. The projection in shown in Figure 8(b).*

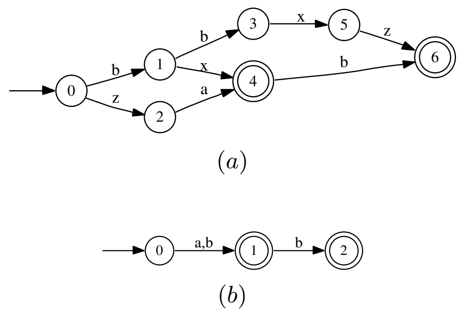

Fig. 8. Example IV.2: (a) G_b; (b) $\theta(G_b)$.

The automaton M_b is not shown. It can be obtained by replacing every marking state of G_b by an unmarked one with a selfloop labeled with τ.

The automaton V_{G_b} is presented in Fig. 9. It can be noticed that state $Dead$ is reachable, indicating that $\theta(G_b)$ is not an OP-abstraction. The verifier presented is completely built just for the sake of illustration. Once state $Dead$ is reached, the algorithm halts. So, in this example, after the fourth step of the execution, the algorithm halts.

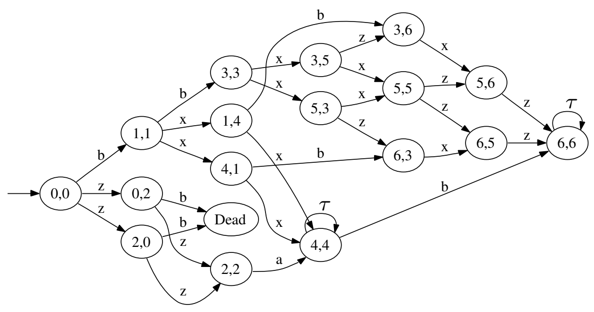

Fig. 9. Example IV.2: Verifier V_{G_b}.

The violation of the observer property can be shown by making the choices $m = z \in \mathcal{L}(G_b)$ and $n = bx$, so that $\theta(m)n = \theta(z)bx \in \theta(\mathcal{L}_m(G_b))$. There is no continuation $p \in \Sigma^*$ such that $zp \in \mathcal{L}_m(G_b)$ and $\theta(p) = \theta(bx)$, which violates the observer property.

V. Conclusions

In this paper, we presented a test that is necessary and sufficient to classify if an abstraction obtained by the natural projection operation is an OP-abstraction without computing the projection. A transition structure V_G is built such that the application of the OP-Test translates into the verification of the reachability of state $Dead$ in the verifier V_G. This structure is constructed from M and the set of relevant events. The reachability of state $Dead$ implies the violation of the observer property. The complexity of this algorithm is polynomial in the number of states of M, in the worst case. In general, the number of states of V_G will be less than n^2.

Using the method presented in this paper, we avoid the calculation of projections that do not have the property, providing a reduction of the complexity of the intermediate steps in obtaining OP-abstractions. However, there is no automatic procedure to help defining the set of relevant events. The OP-verifier can be used to reason about the best choices to be made in terms of the relevant set. Some preliminary investigations in that regard are reported in [17].

Acknowledgments

The first author is supported by FAPEMIG (TEC 00138/07). The second author is supported in part by CNPq grant 300953/93-3. The research of the third author is supported in part by NSF grant ECS-0624821.

References

[1] K. 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.

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

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

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

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

[6] P. Pena, J. Cury, and S. Lafortune, "Testing Modularity of Local Supervisors: An Approach Based on Abstractions," in *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, MI, USA, July 2006, pp. 107–112.

[7] ——, "New Results on Testing Modularity of Local Supervisors using Abstractions," in *11th IEEE International Conference on Emerging Technologies and Factory Automation*, Sep. 2006, pp. 950–956.

[8] K. Wong, "On the Complexity of Projections of Discrete-Event Systems," in *Proceedings of the 4th Workshop on Discrete Event Systems, WODES'98*, Aug. 1998.

[9] L. Feng, "On the Computation of Natural Observers in Discrete-Event Systems," Systems Control Group Report, University of Toronto, Tech. Rep., Jan. 2006.

[10] K. Schmidt and T. Moor, "Marked-String Accepting Observers for the Hierarchical and Decentralized Control of Discrete Event Systems," in *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, MI, USA, July 2006, pp. 413–418.

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

[12] M. de Queiroz and J. Cury, "Modular Control of Composed Systems," in *Proceedings of the American Control Conference, ACC'00*, June.

[13] P. Ramadge and W. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM Journal Control and Optimization*, pp. 206–230, Jan. 1987.

[14] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems - Second Edition*. Springer, 2007.

[15] W. Wonham, "Supervisory Control of Discrete-Event Systems," Dept. of Electrical & Computer Engineering, University of Toronto, 2005.

[16] P. Pena, J. Cury, and S. Lafortune, "Polynomial-Time Verification of the Observer Property in Abstractions," Discrete Event Systems Group Report, Department of Automation and Systems (DAS) - Federal University of Santa Catarina, Tech. Rep., Sep. 2007.

[17] P. Pena, A. da Cunha, J. Cury, and S. Lafortune, "New Results on the Nonconflict Test of Modular Supervisors," in *Proceedings of the 9th International Workshop on Discrete Event Systems, WODES'08*, Göteborg, Sweden, May 2008.