Preprints of WODES 2012
October 3-5, 2012
Guadalajara, México

Generalised Verification of the Observer Property in Discrete Event Systems [1]

H. J. Bravo * **A. E. C. da Cunha** * **P. N. Pena** **
R. Malik *** **J. E. R. Cury** ****

* *Seção de Engenharia Elétrica, Instituto Militar de Engenharia, Brazil (email: hugobravoc@gmail.com, carrilho@ime.eb.br)*
** *Departamento de Engenharia Eletrônica, Universidade Federal de Minas Gerais, Brazil (e-mail: ppena@ufmg.br)*
*** *Department of Computer Science, The University of Waikato, New Zealand, (e-mail: robi@waikato.ac.nz)*
**** *Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina, Brazil (e-mail: cury@das.ufsc.br)*

Abstract: The observer property is an important condition to be satisfied by abstractions of Discrete Event Systems (DES) models. This paper presents a generalised version of a previous algorithm which tests if an abstraction of a DES obtained through natural projection has the observer property. The procedure called *OP-verifier II* overcomes the limitations of the previously proposed verifier while keeping its computational complexity. Results are illustrated by a case study of a transfer line system.

Keywords: Discrete Event Systems, Natural Projections, Observer Property.

1. 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 (Ramadge and Wonham, 1989), as for example in control with partial observation of events (Wonham, 2011), in hierarchical control (Hill and Tilbury, 2006; Cunha and Cury, 2007; Feng and Wonham, 2008; Schmidt *et al.*, 2008; Schmidt and Breindl, 2011), in modular synthesis (Hill and Tilbury, 2006; Schmidt *et al.*, 2006; Feng and Wonham, 2006), in compositional verification of the nonblocking property (Hill and Tilbury, 2006; Flordal and Malik, 2009; Pena *et al.*, 2009; Pena *et al.*, 2010*b*), among others.

In several of the above cited works, the *observer property* (OP) is an important condition to be satisfied by the abstracted models. Abstractions satisfying this property are called *OP-abstractions* (Pena *et al.*, 2008).

OP was first introduced in the context of hierarchical control (Wong and Wonham, 1996). In (Wong and Wonham, 1996), the abstraction is obtained in the form of a *reporter map*, which projects strings of events of the original (low-level) model, built from a set Σ, into high-level strings built from an independent set T of events. Due to some difficulties with the use of reporter maps (Feng and Wonham, 2010), most of the approaches subsequent to (Wong and Wonham, 1996) 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 which are not contained in a given subset of relevant events $\Sigma_r \subseteq \Sigma$, like in (Wong *et al.*, 2000; Cunha and Cury, 2007; Feng and Wonham, 2008; Schmidt *et al.*, 2008; Schmidt and Breindl, 2008).

The structure called *OP-verifier* was presented in (Pena *et al.*, 2008), inspired by the verifier introduced in (Yoo and Lafortune, 2002). Given an input automaton M, 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 checks whether the projection $\theta(M)$ is an *OP-abstraction*.

The *OP-verifier* 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 (Wong and Wonham, 2004; Jiang *et al.*, 2003; Feng and Wonham, 2010; Pena *et al.*, 2010*a*). Nevertheless, the *OP-verifier* algorithm as proposed in (Pena *et al.*, 2008) can only be applied to automata that do not have cycles of non-relevant events.

This paper presents a generalised version of the OP-verifier that can be applied to automata with no restriction on the existence of cycles of non-relevant events. The new version called *OP-verifier II* operates on a modified automaton $\overline{M}$, obtained from the input automaton M, by aggregating states connected by cycles of non-relevant events. The *OP-verifier II* overcomes the limitations of the previously proposed verifier while keeping its computational complexity. The *OP-verifier II* algorithm was implemented in Supremica (Åkesson *et al.*, 2006).

[1] The first, second, third, and fifth authors are partially supported by CAPES (PROCAD 102/2007). The research of the third and fifth authors are supported in part by FAPEMIG and CNPq grant 300953/93-3, respectively.

This paper is organised as follows. Section 2 introduces the necessary background. Section 3 describes the *OP-verifier II*. Then, Section 4 presents the results that assure the correctness of the *OP-verifier II*. In Section 5, a complexity analysis is presented. Section 6 illustrates the results by an example of a transfer line. Finally, concluding remarks are presented in Section 7.

2. PRELIMINARIES

This paper is set in the supervisory control framework initiated by (Ramadge and Wonham, 1989). The reader is referred to (Cassandras and Lafortune, 2007) for a detailed introduction to the theory.

Discrete system behaviours are modelled using strings of *events* taken from a finite alphabet Σ. Then 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., $\overline{L} = \{ s \in \Sigma^* \mid s \leq t \text{ for some } t \in L \}$.

For $\Sigma_r \subseteq \Sigma$, the *natural projection* $\theta\colon \Sigma^* \to \Sigma_r^*$ maps strings in Σ^* to strings in Σ_r^* by erasing all events not contained in Σ_r. Σ_r denotes the set of *relevant* events, while Σ_{nr} denotes the set of *non-relevant* events. The concept is extended to languages by defining $\theta(L) = \{ t \in \Sigma_r^* \mid t = \theta(s) \text{ for some } s \in L \}$.

This paper is concerned about the property of projections known as the *observer property*, which is introduced in (Wong and Wonham, 1996) for prefix-closed languages and extended to general languages in (Wong *et al.*, 2000).

Definition 1. (Wong *et al.*, 2000) Let $L \subseteq \Sigma^*$ be a language, let $\Sigma_r \subseteq \Sigma$, and let $\theta\colon \Sigma^* \to \Sigma_r^*$ be the natural projection. If for all $s \in \overline{L}$ and all $t \in \Sigma_r^*$ 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*.

Discrete event systems are modelled as (nondeterministic) *finite-state automata* $M = (Q^M, \Sigma, \delta^M, q_0^M)$, where Q^M is the set of *states*, Σ is the alphabet of *events*, $\delta^M\colon Q^M \times \Sigma \to 2^{Q^M}$ is the *transition function*, and $q_0^M \in Q^M$ is the *initial state*. M is deterministic when $|\delta^M(x,\sigma)| \leq 1$ for all $x \in Q^M$ and $\sigma \in \Sigma$. For $x \in Q^M$, the set of *enabled* events at state x of M is $\mathrm{En}^M(x) = \{ \sigma \in \Sigma \mid \delta^M(x,\sigma) \neq \emptyset \}$.

The transition function δ^M is extended to strings in Σ^* by letting $\delta^M(x,\varepsilon) = \{x\}$ for all $x \in Q^M$ and, for $x, z \in Q^M$, $s \in \Sigma^*$ and $\sigma \in \Sigma$, $z \in \delta^M(x, s\sigma)$ if $y \in \delta^M(x,s)$ and $z \in \delta^M(y,\sigma)$ for some $y \in Q^M$. The *behaviour* of M, modelled as a language $\mathcal{L}(M) = \{ s \in \Sigma^* \mid \delta^M(q_0^M, s) \neq \emptyset \}$, represents the set of all finite strings that M can generate.

To express termination, the alphabet Σ is assumed to contain the special event $\tau \in \Sigma$, which may only appear on selfloops, i.e., $\delta^M(x,\tau) = \{x\}$ whenever $\delta^M(x,\tau) \neq \emptyset$. In this notation, the *marked behaviour* of M is defined as $\mathcal{L}_m(M) = \{ s \in (\Sigma - \{\tau\})^* \mid s\tau \in \mathcal{L}(M) \}$.

This paper uses the termination event in favour of the more conventional set of terminal states, because it simplifies presentation by unifying termination with ordinary events. A traditional automaton with terminal states, $G = (Q^G, \Sigma^G, \delta^G, q_0^G, F^G)$ with $\tau \notin \Sigma^G$, can be converted into this paper's $M = (Q^M, \Sigma, \delta^M, q_0^M)$ by adding τ to the alphabet, $\Sigma = \Sigma^G \dot\cup \{\tau\}$, letting $Q^M = Q^G$ and $q_0^M = q_0^G$, and adding τ-selfloops to every terminal state in F^G, i.e., $\delta^M(x,\sigma) = \delta^G(x,\sigma)$ for $\sigma \neq \tau$, $\delta^M(x,\tau) = \{x\}$ for all $x \in F^G$, and $\delta^M(x,\tau) = \emptyset$ for all $x \in Q^G - F^G$.

We define the language $N \subseteq \mathcal{L}(M)$ as the sublanguage of $\mathcal{L}(M)$ composed of strings in $(\Sigma - \{\tau\})^*$, namely, $N = \mathcal{L}(M) \cap (\Sigma - \{\tau\})^*$. For the automata M and G, related as above, it is trivial to show that $N = \overline{\mathcal{L}(G)}$ and $\mathcal{L}_m(M) = \mathcal{L}_m(G)$. M is nonblocking when $N = \overline{\mathcal{L}_m(M)}$.

Projections can also be applied to automata, and in the following, given a nonblocking M, it will be said that $\theta(M)$ has the observer property if $\theta(\mathcal{L}_m(M))$ has the observer property. In this case $\theta(M)$ is also called an *OP-abstraction*.

Example 1. In order to illustrate the abstractions and the observer property, let M be the automaton shown in Fig. 1 and $\Sigma_r = \{a, \tau\}$. The abstraction $\theta(M)$ is shown in Fig. 2. It can be shown that $\theta(M)$ is not an OP-abstraction by making, as in Definition 1, $s = aa$, $t = \varepsilon$. Therefore $\theta(s)t = aa \in \theta(\mathcal{L}_m(M))$ but there is no $t' \in \Sigma^*$ such that $st' \in \mathcal{L}_m(M)$ and $\theta(st') = \theta(s)t$.

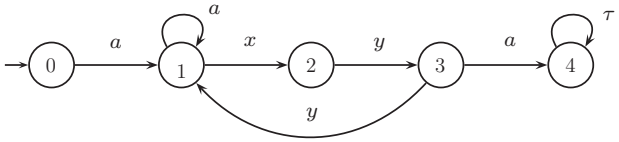

Fig. 1. Automaton M.

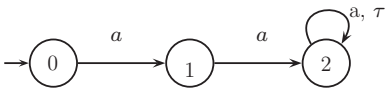

Fig. 2. Automaton $\theta(M)$.

In order to check if a given projection presents the observer property, an algorithm named *OP-Verifier* was proposed in (Pena *et al.*, 2008). The OP-Verifier was inspired by the verifier introduced in (Yoo and Lafortune, 2002). Given an input automaton M and an alphabet partition $\Sigma = \Sigma_r \dot\cup \Sigma_{nr}$, it checks whether $\theta(M)$ has the observer property. The OP-verifier algorithm can only be applied to deterministic automata that do not have cycles of non-relevant events.

The automaton M in Fig. 1 has a cycle of non-relevant events involving states 1, 2 and 3. It can be shown that, although $\theta(M)$ is not OP, the OP-verifier algorithm in (Pena *et al.*, 2008) erroneously classifies it as OP.

3. VERIFICATION OF THE OBSERVER PROPERTY

In this section, a generalised OP-Verifier algorithm is presented. It derives from a previous version of the OP-Verifier (Pena *et al.*, 2008), and adds to it the ability to handle cycles of non-relevant events.

3.1 Strongly Connected Components Automaton $\overline{M}$

In order to be able to deal with cycles of non-relevant events, a *strongly connected components automaton* $\overline{M}$ is introduced. The automaton $\overline{M} = (Q^{\overline{M}}, \overline{\Sigma}, \delta^{\overline{M}}, A_0^{\overline{M}})$ is obtained from $M = (Q^M, \Sigma, \delta^M, q_0^M)$ where:

- the set of states $Q^{\overline{M}}$ is defined as the set of strongly connected components, in terms only of non-relevant events, of M. Tarjan's algorithm (Nuutila and Soisalon-Soininen, 1994) can be used to determine those components. In this sense, $Q^{\overline{M}}$ is a partition of Q^M. Each component $A_t \in Q^{\overline{M}}$, such that $A_t \subseteq Q^M$, is named, in this paper, a macro-state.
- The event set $\overline{\Sigma} \subseteq \Sigma$ is partitioned into $\overline{\Sigma} = \overline{\Sigma}_r \cup \overline{\Sigma}_{nr}$, such that $\overline{\Sigma}_r = \Sigma_r$ and $\overline{\Sigma}_{nr} \subseteq \Sigma_{nr}$;
- $A_t = A_0^{\overline{M}} \in Q^{\overline{M}}$ is named the initial macro-state of $\overline{M}$ if $q_0^M \in A_0^M$;
- The transition function $\delta^{\overline{M}} : Q^{\overline{M}} \times \overline{\Sigma} \rightarrow 2^{Q^{\overline{M}}}$ is defined below:
  - Let $A_t, A_k \in Q^{\overline{M}}$ and $\sigma \in \overline{\Sigma}_r$. Then, $A_k \in \delta^{\overline{M}}(A_t, \sigma)$ if and only if $\exists q_i \in A_t$ and $\exists q_j \in A_k$ such that $q_j \in \delta^M(q_i, \sigma)$.
  - Let $A_t, A_k \in Q^{\overline{M}}$ and $\sigma \in \overline{\Sigma}_{nr}$. Then, $A_k \in \delta^{\overline{M}}(A_t, \sigma)$ if and only if: (i) $A_t \neq A_k$ and (ii) $\exists q_i \in A_t$ and $\exists q_j \in A_k$ such that $q_j \in \delta^M(q_i, \sigma)$.

By construction, the strongly connected components automaton does not contain cycles of non-relevant events.

Example 2. The strongly connected components automaton $\overline{M}$ obtained from M in Fig. 1 is shown in Fig. 3. A strongly connected component in M can be identified, composed of states 1, 2 and 3, forming macro-state $A_1 = \{1, 2, 3\}$ in $\overline{M}$. The macro-states $A_0 = \{0\}$ and $A_2 = \{4\}$ consist of only one state each.

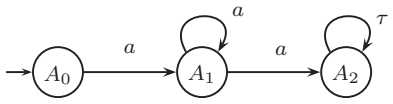

Fig. 3. Strongly connected components automaton $\overline{M}$.

3.2 Generalised Verifier $\overline{V}$

A nondeterministic automaton, named *generalised verifier*, is defined in this section.

The generalised verifier $\overline{V} = (\overline{Q}, \overline{\Sigma}, \overline{\delta}, \{A_0^M\})$ is obtained by execution of Algorithm OP-Verifier II, Algorithm 3.1, where:

- $\overline{Q}$ is the set of states, where each state is either a set of cardinality 1 or 2, namely, $1 \leq |\{A_t, A_k\}| \leq 2$, where $A_t, A_k \in Q^{\overline{M}}$, or a special state *Dead*.
- $\overline{\Sigma}$ is the event set;
- $\{A_0^{\overline{M}}\} \in \overline{Q}$ is the initial state;
- $\overline{\delta} : \overline{Q} \times \overline{\Sigma} \rightarrow 2^{\overline{Q}}$ is a transition function defined by the subroutine $\overline{Delta}(\overline{X})$ (Algorithm 3.2).

The OP-Verifier II builds $\overline{V}$, from $\overline{M}$ and $\overline{\Sigma}_r$. As shown in Section 4, if *Dead* is reachable in $\overline{V}$, then $\theta(M)$ is not an OP-abstraction.

Algorithm 3.1. OP-Verifier II

```
1  Input M̄ = (Q^M̄, Σ̄, δ^M̄, A_0^M̄);
2  Let Q̄ ← ∅ and Q̄_T ← {{A_0^M̄}};
3  while Q̄_T − Q̄ ≠ ∅ do
4     Select X̄ ∈ Q̄_T − Q̄;
5     Let Q̄_T ← Q̄_T − {X̄} and Q̄ ← Q̄ ∪ {X̄};
6     Delta(X̄); (Algorithm 3.2)
7  end
8  Return V̄ = (Q̄, Σ̄, δ̄, {A_0^M̄})
```

Algorithm 3.2. Transition function algorithm $\overline{Delta}(\overline{X})$

```
9  Input X̄ = {A_t, A_k}, (where possibly A_t = A_k);
10 for each σ ∈ En(X̄) = En^M̄(A_t) ∪ En^M̄(A_k) do
11    if σ ∈ Σ̄_r then
12       if σ ∈ En^M̄(A_t) and σ ∈ En^M̄(A_k) then
13          for each A'_t ∈ δ^M̄(A_t, σ) and A'_k ∈ δ^M̄(A_k, σ) do
14             δ̄({A_t, A_k}, σ) ← δ̄({A_t, A_k}, σ) ∪ {{A'_t, A'_k}};
15             Q̄_T ← Q̄_T ∪ {{A'_t, A'_k}};
16          end
17       elseif (σ ∈ En^M̄(A_t) and En^M̄(A_k) ∩ Σ̄_nr = ∅) or
            (σ ∈ En^M̄(A_k) and En^M̄(A_t) ∩ Σ̄_nr = ∅) then
18          δ̄({A_t, A_k}, σ) ← δ̄({A_t, A_k}, σ) ∪ {Dead};
19          Q̄ ← Q̄ ∪ {Dead};
20       end
21    else
22       if σ ∈ En^M̄(A_t) then
23          for each A'_t ∈ δ^M̄(A_t, σ) do
24             δ̄({A_t, A_k}, σ) ← δ̄({A_t, A_k}, σ) ∪ {{A'_t, A_k}};
25             Q̄_T ← Q̄_T ∪ {{A'_t, A_k}};
26          end
27       end
28       if σ ∈ En^M̄(A_k) then
29          for each A'_k ∈ δ^M̄(A_t, σ) do
30             δ̄({A_t, A_k}, σ) ← δ̄({A_t, A_k}, σ) ∪ {{A_t, A'_k}};
31             Q̄_T ← Q̄_T ∪ {{A_t, A'_k}};
32          end
33       end
34    end
35 end
```

The structure of the OP-Verifier II is similar to the original OP-Verifier. For that reason, only the differences will be highlighted. Algorithm $\overline{Delta}(\overline{X})$ carries the main differences.

The transition function $\overline{\delta}$ is initialised empty and is constructed iteratively by the execution of lines 14, 18, 24 and 30 of the subroutine $\overline{Delta}(\overline{X})$.

Like in the original OP-Verifier, the transition structure $\overline{Delta}(\overline{X})$ of the OP-Verifier II is also defined distinctly for each type of event. Lines 11 to 20 describe the case where $\sigma \in \overline{\Sigma}_r$, and lines 21 to 34 present the case where $\sigma \in \overline{\Sigma}_{nr}$.

If $\sigma \in \overline{\Sigma}_r$ (line 11), verify if $\sigma \in En^{\overline{M}}(A_t)$ and $\sigma \in En^{\overline{M}}(A_k)$ (line 12). If so, then A_t and A_k move synchronously trough σ, generating sets of states $\{A'_t, A'_k\}$ (lines 13–14). If not, jump to line 17. Add this new set of states to the set $\overline{Q}_T$ (line 15). To verify reachability of *Dead* in $\overline{V}$ (lines 17–20), the same procedure of the original OP-Verifier is used.

If $\sigma \in \overline{\Sigma}_{nr}$ (line 21), then A_t and A_k move asynchronously trough σ. There are two possibilities: (i) $\sigma \in En^{\overline{M}}(A_t)$ (line 22), then a new set of states $\{A'_t, A_k\}$ is

generated (lines 23–24), and this set is included in $\overline{Q}_T$ (line 25). (*ii*) $\sigma \in En^{\overline{M}}(A_k)$ (line 28), then a new set of states $\{A_t, A'_k\}$ is reached (lines 29–30) and added to Q^T (line 31).

Example 3. Consider M and $\overline{M}$ (figures 1 and 3, respectively). The relevant event set of $\overline{M}$ is $\overline{\Sigma}_r = \{a, \tau\}$. The execution of the algorithm OP-Verifier II over $\overline{M}$ and $\overline{\Sigma_r}$ generates the following transitions:

$$\begin{array}{lcll} \{A_0\} & \xrightarrow{a} & \{A_1\} & \textbf{(Line 12)} \\ \{A_1\} & \xrightarrow{a} & \{A_1\} & \textbf{(Line 12)} \\ \{A_1\} & \xrightarrow{a} & \{A_2\} & \textbf{(Line 12)} \\ \{A_1\} & \xrightarrow{a} & \{A_1, A_2\} & \textbf{(Line 12)} \\ \{A_1, A_2\} & \xrightarrow{a} & \mathit{Dead} & \textbf{(Line 17)} \\ \{A_1, A_2\} & \xrightarrow{\tau} & \mathit{Dead} & \textbf{(Line 17)} \\ \{A_2\} & \xrightarrow{\tau} & \{A_2\} & \textbf{(Line 12)} \end{array}$$

The generalised verifier $\overline{V}$ is shown in Fig. 4. In the verifier $\overline{V}$, *Dead* is reachable. Therefore, $\theta(M)$ is not an OP-Abstraction.

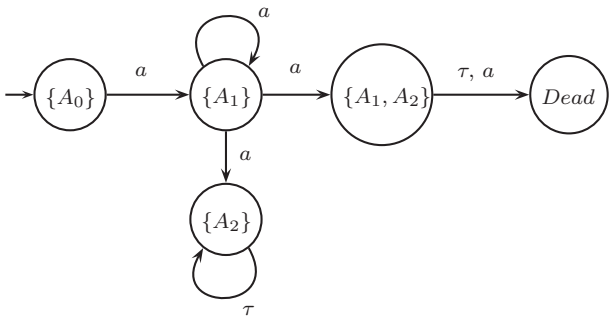

Fig. 4. Generalised verifier $\overline{V}$.

4. PROPERTIES OF THE GENERALISED VERIFIER

In this section, some properties regarding the generalised OP-Verifier are presented.

The following two lemmas bring some results relating strings with the same projection to states of $\overline{V}$. The proofs of Lemmas 1 and 2 are omitted due to space constraints.

Lemma 1. Let $a, b \in \Sigma^*$ such that $\theta(a) = \theta(b)$, $x_a \in \delta^M(q_0^M, a)$ and $x_b \in \delta^M(q_0^M, b)$. Then there is $s \in \Sigma^*$ such that $\theta(a) = \theta(b) = \theta(s)$ and $\{A_a, A_b\} \in \overline{\delta}(\{A_0^{\overline{M}}\}, s)$, with $x_a \in A_a$ and $x_b \in A_b$.

Lemma 2. Let $s \in \Sigma^*$ and $\{A_a, A_b\} \in \overline{\delta}(\{A_0^{\overline{M}}\}, s)$. Then there exist $a, b \in \Sigma^*$ such that $\theta(a) = \theta(b) = \theta(s)$, $\delta^M(q_0^M, a) \in A_a$ and $\delta^M(q_0^M, b) \in A_b$.

Theorem 3 brings the main result regarding the correctness of the generalised OP-verifier, and its proof, also omitted in this paper, is based on Lemmas 1 and 2.

Theorem 3. Dead is reachable in $\overline{V}$ if, and only if, $\theta(M)$ is not an OP-abstraction.

5. COMPLEXITY

The complexity of the OP-Verifier II algorithm is determined by the complexity to construct the generalised verifier. If the input is a deterministic automaton $M = (Q^M, \Sigma, \delta^M, q_0^M)$ then the number of reachable states of $\overline{V}$ is bounded by

$$|Q^M|^2 + |Q^M| + 1 = O(|Q^M|^2) \ . \tag{1}$$

To estimate the number of transitions of $\overline{V}$, consider a transition (x, σ, x') in the input automaton M, and let $y \in Q^M$ be an arbitrary state. If $\sigma \in \Sigma_r$, then this produces at most one transition

$$(\{A_x, A_y\}, \sigma, \{A_{x'}, A_{y'}\}) \quad \text{or} \quad (\{A_x, A_y\}, \sigma, \mathit{Dead}) \tag{2}$$

according to lines 14 or 18 of Algorithm 3.2, were $A_x \in Q^{\overline{M}}$ denotes the macro-state containing state $x \in Q^M$. If $\sigma \in \Sigma_{nr}$, then there is one transition

$$(\{A_x, A_y\}, \sigma, \{A_{x'}, A_y\}) \tag{3}$$

according to lines 24 or 30 of Algorithm 3.2. That is, every transition of M produces up to $|Q^M|$ transitions in $\overline{V}$. The deterministic automaton M has up to $|\Sigma||Q^M|$ transitions, so the total number of transitions of $\overline{V}$ is bounded by

$$|\delta^M||Q^M| \leq |\Sigma||Q^M|^2 = O(|\Sigma||Q^M|^2) \ . \tag{4}$$

Tarjan's algorithm to identify the strongly connected components runs in $O(|\delta^M|) = O(|\Sigma||Q|)$ time, so it is dominated by the verifier construction. Therefore, (4) gives the worst-case time complexity of the OP-Verifier II algorithm.

6. CASE STUDY

In this section, the case study of obtaining an abstraction for the behaviour of a transfer line in the context of hierarchical control is presented. It is shown that cycles of non-relevant events arise naturally from the problem and that the original OP-Verifier algorithm in (Pena *et al.*, 2008) fails to detect that the abstraction is not OP, while the generalised OP-Verifier algorithm presented in this paper detects it. Moreover, a modification is shown for the abstracted behaviour that turns it into OP and that the generalised OP-Verifier also detects it.

Consider the example of a manufacturing transfer line with material feedback in Fig. 5, adapted from (Zhong and Wonham, 1990). The system is composed of two machines, M_1 and M_2, and a test unit TU, linked by intermediate buffers of unitary capacity. Machines M_1 and M_2 produce workpieces that are tested in TU, which determines their approval or rework in M_2. The input and output events for M_1, M_2 and TU are indicated in the workflow lines in Fig. 5, therefore $\Sigma = \{x, y, z, w, v_1, v_2, a_1, a_2, \tau\}$. There are two kinds of input events for TU, event v_1 is the test of a workpiece for the first time, and event v_2 is the test of a reworked workpiece. There are also three different output events for TU, event a_1 is the approval of a workpiece after the first test, event a_2 is the approval of a workpiece that has been reworked, and event r the redirection of a workpiece that was not approved to rework in M_2.

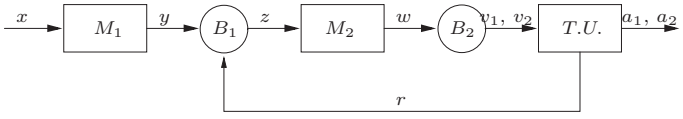

Fig. 5. Transfer Line.

Consider that the low-level control has already been developed, solving the problems of overflow and underflow of the unitary buffers and the problem of performing different

tests for reworked and non-reworked workpieces. The low-level controlled behaviour, automaton M, is shown in Fig. 6. Also consider that, in some hierarchical control approach one is concerned with the input-output behaviour of the workcell for coordination purposes with other workcells (Cunha and Cury, 2007; Schmidt *et al.*, 2008; Schmidt and Breindl, 2011). Thus, the relevant events are the input event for the workcell x and the two output events a_1 and a_2, therefore $\Sigma_r = \{x, a_1, a_2, \tau\}$. The abstraction $\theta(M)$ is then shown in Fig. 7.

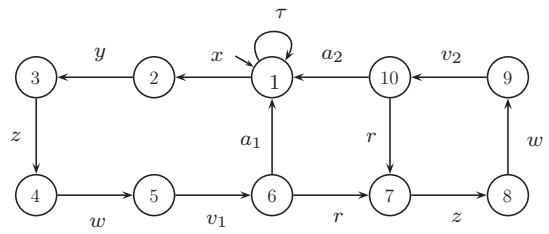

Fig. 6. Automaton M.

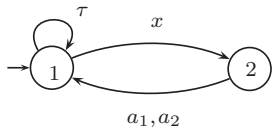

Fig. 7. Abstraction $\theta(M)$.

One of the most important conditions for hierarchical control approaches like (Cunha and Cury, 2007), (Schmidt *et al.*, 2008), or (Schmidt and Breindl, 2011) is that $\theta(M)$ has to be an OP-abstraction. From definition 1, it can be shown that $\theta(M)$ is not an OP-abstraction by making $s = xyzwv_1rzwv_2 \in N$ and $t = a_1 \in \Sigma_r^*$ such that $\theta(s)t = xa_1 \in \mathcal{L}_m(\theta(M))$, but there is no $t' \in \Sigma^*$ such that $st' \in \mathcal{L}_m(M)$ and $\theta(st') = xa_1$. Moreover, when applying the OP-Verifier algorithm of (Pena *et al.*, 2008) for M and Σ_r, a verifier automaton can be obtained, with 46 states and 81 transitions, where the state *Dead* is not reached. It can be shown that the reason that makes the verifier not reach the state *Dead* is the cycle of non-relevant events formed by states 7, 8, 9 and 10 of M, Fig. 6.

The strongly connected components automaton $\overline{M}$ is shown in Fig. 8. The labels for the macro-states of $\overline{M}$ are $A_j = \{j\}$, for $j = 1...6$, and $A_7 = \{7, 8, 9, 10\}$. By applying the OP-Verifier II algorithm, a generalised verifier $\overline{V}$, with 23 states and 35 transitions, is obtained, Fig. 9. Observe that $\overline{V}$ reaches the state *Dead* confirming that $\theta(M)$ is not an OP-abstraction.

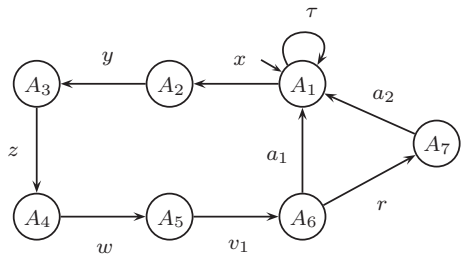

Fig. 8. Automaton $\overline{M}$.

To obtain an OP-Abstraction for this problem, one can follow the lines of (Pena *et al.*, 2010*a*) and exploit

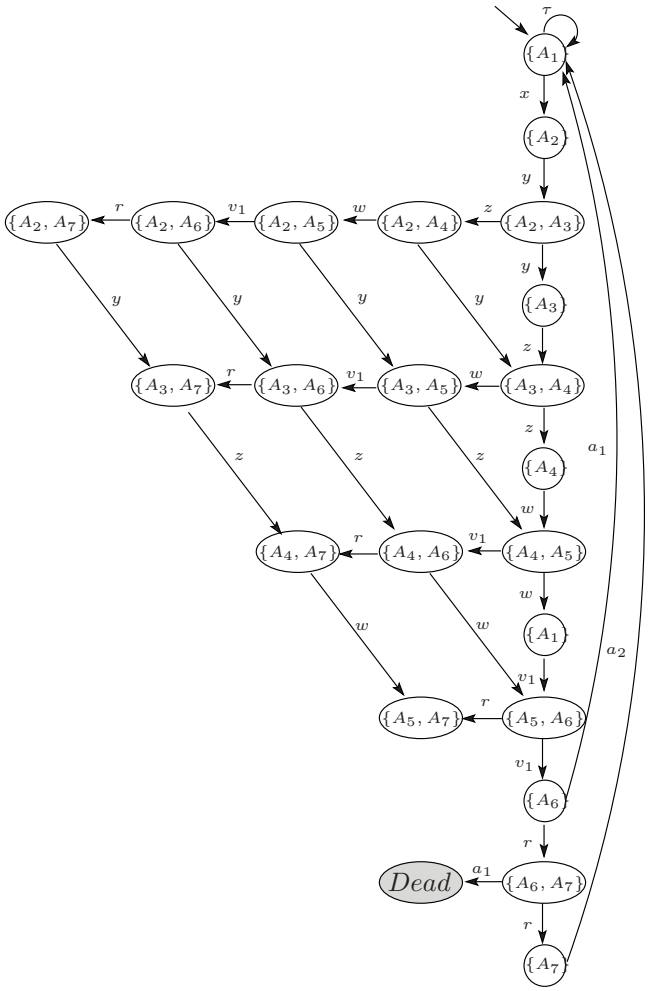

Fig. 9. Verifier $\overline{V}$.

the properties of the *unsafe* path $(\{A_1\}, r, \{A_6, A_7\})$ $(\{A_6, A_7\}, a_1, Dead)$ in $\overline{V}$, Fig. 9. Therefore, consider the relabelling of the transition $(6, r, 7)$ of M in Fig. 6 to $(6, r_1, 7)$, creating the new relevant event r_1 that corresponds to the first rework of a workpiece. Let M' be the modified automaton and $\theta'(M')$ the new abstraction, shown in Fig. 10. It can be proved that $\theta'(M')$ is an OP-abstraction. Moreover, when applying the OP-Verifier II algorithm in this case, a verifier $\overline{V'}$ is obtained, with 17 states and 24 transitions, where state *Dead* is not reachable.

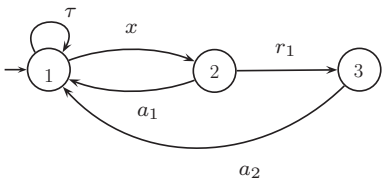

Fig. 10. Abstraction $\theta'(M')$.

7. CONCLUSIONS

The *OP-verifier II* 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, where a restriction on the existence of cycles of non-relevant events in the original automaton model M was overcome. The new version of the verifier builds upon the encapsulation of states in M connected by cycles of non-relevant events. The resulting (non-deterministic) automaton $\overline{M}$ is then translated into a transition structure $\overline{V}$ in which the OP is checked by verifying the reachability of a state Dead. The authors are currently investigating how the *OP-verifier II* could be used to improve the OP-Search proposed in (Pena *et al.*, 2010*a*) in order to help computing reduced *OP-abstractions*.

REFERENCES

Cassandras, C.G. and S. Lafortune (2007). *Introduction to Discrete Event Systems - Second Edition*. Springer.

Cunha, A.E.C. and J.E.R. Cury (2007). Hierarchical Supervisory Control Based on Discrete Event Systems with Flexible Marking. *IEEE Transactions on Automatic Control* **52**(12), 2242–2253.

Feng, L. and W.M Wonham (2006). Computationally Efficient Supervisor Design: Abstraction and Modularity. In: *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*. Ann Arbor, MI, USA. pp. 3–8.

Feng, L. and W.M. Wonham (2008). Supervisory control architecture for discrete-event systems. *Automatic Control, IEEE Transactions on* **53**(6), 1449–1461.

Feng, Lei and W. M. Wonham (2010). On the computation of natural observers in discrete-event systems. *Discrete Event Dynamic Systems* **20**(1), 63–102.

Flordal, Hugo and Robi Malik (2009). Compositional verification in supervisory control. *SIAM J. Control and Optimization* **48**(3), 1914–1938.

Hill, R.C. and D.M. Tilbury (2006). 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. pp. 399–406.

Jiang, S., R. Kumar and H.E. Garcia (2003). Optimal sensor selection for discrete-event systems with partial observation. *IEEE Transactions Automatic Control* **48**(3), 369–381.

Åkesson, Knut, Martin Fabian, Hugo Flordal and Robi Malik (2006). Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems. In: *Proceedings of the 8th Internationsl Workshop of Discrete Event Systems, WODES'06*. Ann Arbor, MI, USA. pp. 384–385.

Nuutila, Esko and Eljas Soisalon-Soininen (1994). On Finding the Strongly Connected Components in a Directed Graph. *Information Processing Letters* **49**, 14.

Pena, P. N., J.E.R. Cury, R. Malik and S. Lafortune (2010*a*). Efficient Computation of Observer Projections using OP-Verifiers. In: *Proceedings of the 10th International Workshop on Discrete Event Systems, WODES'10*. Berlin, Germany. pp. 416–421.

Pena, P.N., J.E.R. Cury, A.E.C. da Cunha and S. Lafortune (2010*b*). Metodologia e Ferramenta de Apoio ao Teste de Não-Conflito no Controle Modular de Sistemas a Eventos Discretos. *SBA Controle & Automação* **21**(1), 58–68.

Pena, P.N., J.E.R. Cury and S. Lafortune (2008). Polynomial-Time Verification of the Observer Property in Abstractions. In: *Proceedings of the 2008 American Control Conference, ACC'08*. Seattle, USA. pp. 465–470.

Pena, P.N., J.E.R. Cury and S. Lafortune (2009). Verification of Nonconflict of Supervisors Using Abstractions. *IEEE Transactions on Automatic Control* **54**(12), 2803–2815.

Ramadge, P. J. G. and W. M. Wonham (1989). The Control of Discrete Event Systems. *Proc. of the IEEE* **77**(1), 81–98.

Schmidt, K. and C. Breindl (2008). On maximal permissiveness of hierarchical and modular supervisory control approaches for discrete event systems. In: *Discrete Event Systems, International Workshop on*. pp. 462–467.

Schmidt, K., H. Marchand and B. Gaudin (2006). 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. pp. 149–154.

Schmidt, K., T. Moor and S. Perk. (2008). Nonblocking hierarchical control of decentralized discrete event systems. *Automatic Control, IEEE Transactions on* **53**(10), 2252–2265.

Schmidt, Klaus and Christian Breindl (2011). Maximally Permissive Hierarchical Control of Decentralized Discrete Event Systems. *IEEE Transactions Automatic Control* **56**(4), 723–737.

Wong, K.C. and W. M. Wonham (1996). Hierarchical Control of Discrete-Event Systems. *Discrete Event Dynamic Systems: Theory and Applications* **6**(3), 241–273.

Wong, K.C. and W.M Wonham (2004). On the Computation of Observers in Discrete-Event Systems. *Discrete Event Dynamical Systems* **14**(1), 55–107.

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

Wonham, W. M. (2011). *Supervisory Control of Discrete-Event Systems*. Systems Control Group, Department of Electrical & Computer Engineering, University of Toronto. Toronto, Canada. Updates posted annually at http://www.control.utoronto.ca/DES.

Yoo, T. and S. Lafortune (2002). Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control* **47**(9), 1491– 1495.

Zhong, H. and W.M. Wonham (1990). On the Consistency of Hierarchical Supervision in Discrete-Event Systems. *IEEE Transactions on Automatic Control* **35**(10), 1125–1134.