Preprints of WODES 2018
May 30 - June 1, 2018
Sorrento Coast, Italy

Synthesis of Dynamic Masks for Infinite-Step Opacity

Xiang Yin * **Shaoyuan Li** *

* *Department of Automation, Shanghai Jiao Tong University, Shanghai, China. (e-mail: yinxiang@sjtu.edu.cn, syli@sjtu.edu.cn)*

Abstract: We investigate the problem of synthesizing *dynamic masks* that preserve *infinite-step opacity* in the context of discrete-event systems. A system equipped with a dynamic mask, which acquires information dynamically by turning sensors on/off, is said to be infinite-step opaque if an outside intruder that can access all information acquired can never infer that the system was at some secret state for some specific previous instant. Existing works on the dynamic mask synthesis problem only consider current-state opacity. However, synthesizing dynamic masks for infinite-step opacity, a notion stronger than current-state opacity, is much more challenging. The main reason is that delayed information is involved in this problem and whether or not a current secret will be revealed depends on sensing decisions to be synthesized in the future. In this paper, a new type of information state is proposed to capture all delayed information in the infinite-step opacity problem. An effective algorithm is then presented to solve the synthesis problem. Our result extends existing dynamic mask synthesis techniques from current-state opacity to infinite-step opacity.

Keywords: Discrete Event Systems; Supervisory Control and Automata; Partial Observation.

1. INTRODUCTION

In this paper, we investigate an information-flow property called opacity in the context of Discrete-Event Systems (DES) Cassandras and Lafortune [2008]. Opacity is a confidentiality property that captures the *plausible deniability* of the system's "secret behavior", i.e., the secret of the system should can never be revealed to a passive observer (intruder) that is potentially malicious. In the context of DES, opacity has drawn considerable attention in the past decade; see, e.g., Badouel et al. [2007], Bryans et al. [2008], Lin [2011], Saboori and Hadjicostis [2012], Chédor et al. [2015], Yin and Lafortune [2017], Tong et al. [2017]. The reader is referred to Jacob et al. [2016] for a more comprehensive literature review.

When the original system is not opaque, one important problem in opacity is to *synthesize* an opaque system. This problem has recently been widely studied in the literature and several different synthesis mechanisms have been investigated. For example, in Takai and Oka [2008], Dubreil et al. [2010], Yin and Lafortune [2015b, 2016b], the authors investigated how to synthesize a supervisory controller such that the closed-loop system is opaque. In Wu and Lafortune [2014], the problem of synthesizing an insertion function that enforces opacity was studied. In Falcone and Marchand [2014], runtime mechanism is used to enforce opacity.

In this paper, we consider the problem of synthesizing *dynamic masks* that preserve opacity. Dynamic mask is an information acquisition mechanism that acquires information from the system by dynamically turning on/off the associated sensors. The dynamic mask synthesis problem is also referred to as the dynamic sensor activation problem in the DES literature; see, e.g., Cassez and Tripakis [2008], Shu et al. [2013], Yin and Lafortune [2015a], Zhang et al. [2015], Sears and Rudie [2016]. In particular, in Cassez et al. [2012], the authors investigated the problem of synthesizing dynamic masks that preserve *current-state opacity*, where an algorithm with exponential complexity was provided. Note that current-state opacity only requires that the intruder should never know that the system is currently at a secret state. As a stronger security requirement, *infinite-step opacity* Saboori and Hadjicostis [2012] requires that the intruder should never know that the system was at a secret state for any specific previous instant.

We tackle the problem of synthesizing dynamic masks for infinite-step opacity by addressing the above mentioned key difficulty. Specifically, the main contributions of this paper are as follows. First, we propose a new type of information state in order to capture all possible delayed information in this problem. The proposed information state is general than the subset-based information state that is widely used in partially-observed synthesis problems related to current information. Based on the novel information state proposed, an effectively algorithm is then presented to solve the dynamic mask synthesis problem.

2. PRELIMINARY

2.1 System Model

Let Σ be a finite set of events; a string $s = \sigma_1 \dots \sigma_n, \sigma_i \in \Sigma$ is a finite sequence of events, where $|s|$ denotes its length. We denote by Σ^* the set of all strings over Σ including the empty string ϵ. A language $L \subseteq \Sigma^*$ is a set of strings; $\overline{L} = \{s \in \Sigma^* : \exists t \in \Sigma^* \text{ s.t. } st \in L\}$ denotes its prefix-closure.

$$T_j = T'_j \ , \quad \text{if } |T'_j| \leq K+1 \ ,$$
$$T_j \subset T'_j, |T_j| = K+1 \ , \quad \text{if } |T'_j| > K+1 \ .$$

Clearly, if $T_j = T'_j$ (when $|T'_j| \leq K+1$), we have $T_j \cap (X \setminus S) = T'_j \cap (X \setminus S) \neq \emptyset$. If $T_j \subset T'_j$ (i.e., when $|T'_j| > K+1$), then $|T_j| = K+1$; thus, since we take $K \geq |S|$, T_j has at least one element outside S and we can conclude that $T_j \cap (X \setminus S) \neq \emptyset$. Thus, we conclude that if a solution exists (i.e., we can find a path in the digraph of DFA G_{do}), we can also find a path in the digraph of NFA G_{dkd}. □

Remark 4. If, in the proof of the above theorem, we use a construction of G_{dkd} where $K = 1$ (i.e., if we use the standard detector, which relates to a standard verifier or twin-plant construction), it will not be possible to obtain a solution to the problem of interest (not unless S has only one element). To see this, consider the following: if a path can be found using the standard detector (i.e., if we can find a path that involves nodes (i.e., states of G_{dkd}) where each detector component is associated with a pair of system states, of which at least one state is in $X \setminus S$, the corresponding sequence of inputs along this path will indeed be a solution to the problem of interest. The problem is the reverse direction: if a solution using G_{dkd} with $K = 1$ is not found, that does not imply that a solution does not exist. The reason is that an appropriate sequence of inputs t does not necessarily need to be "protected" by the *same* sequence of inputs s at all instants (multiple sequences could be providing "protection" at different points during the observation process). This should be contrasted with the situation in fault diagnosis where a fault event at a certain point implies that continuing state trajectories will always be associated with a fault condition.

Remark 5. Indirectly, since the sequence of inputs induces a path in the digraph of G_{dkd}, which can be made acyclic (by removing any cycles), the above theorem says that if there exists a solution, namely a sequence of n inputs with the desirable properties, that sequence does not have to be longer than $|X_{dkd}| - 1 \leq |X|^{K+1} - 1$ (where $|X_{dkd}|$ is the number of states of the product of the system and the detector). This was not evident from the solution using Algorithm 1, which would suggest that the sequence would not have to be longer than $|X_{do}| - 1 \leq |X|2^{|X|} - 1$.

It should be evident that weak current state opacity with respect to a set of secret states S (see Definition 3) can be verified by constructing a K-detector as long as $K \geq |S|$. The key observation is that if there is an (infinite) sequence of events that causes a sequence of observations that satisfy current-state opacity constraints at all times, this will also be reflected in the K-detector.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we discussed how controlled finite automata can be steered from one state (initial location) to another state (target location), while maintaining, in the process, certain privacy guarantees. In the future, we plan to consider settings where control may not be deterministic (or there may be uncontrollable events, as well as extensions to problems that involve the enforcement of other types of opacity constraints.

ACKNOWLEDGEMENTS

The author would like to acknowledge fruitful discussions with Bengt Lennartson.

REFERENCES

Bryans, J.W., Koutny, M., Mazare, L., and Ryan, P. (2005). Opacity generalised to transition systems. In *Selected Papers of the 3rd Int. Workshop on Formal Aspects in Security and Trust*, 421–435.

Cassandras, C.G. and Lafortune, S. (2009). *Introduction to Discrete Event Systems*. Springer Science & Business Media.

Focardi, R. and Gorrieri, R. (1994). A taxonomy of trace–based security properties for CCS. In *Proc. of the 7th Workshop on Computer Security Foundations*, 126–136.

Hadjicostis, C.N. (2010). Supervisory control strategies for enhancing system security and privacy. In *Proc. of 48th Annual Allerton Conf. on Communication, Control, and Computing (Allerton)*, 1622–1627.

Hadjicostis, C.N. and Seatzu, C. (2016). K-Detectability in discrete event systems. In *Proc. of 55th IEEE Conf. on Decision and Control (CDC)*, 420–425.

Jacob, R., Lesage, J.J., and Faure, J.M. (2015). Opacity of discrete event systems: models, validation and quantification. *IFAC-PapersOnLine*, 48(7), 174–181. Proc. of 5th IFAC Int. Workshop on Dependable Control of Discrete Systems.

Lakhnech, Y. and Mazaré, L. (2005). Probabilistic opacity for a passive adversary and its application to Chaum's voting scheme. *IACR Cryptology ePrint Archive*, 2005, 98.

Lin, F. (2011). Opacity of discrete event systems and its applications. *Automatica*, 47(3), 496–503.

Saboori, A. and Hadjicostis, C.N. (2007). Notions of security and opacity in discrete event systems. In *Proc. of 46th IEEE Conf. on Decision and Control (CDC)*, 5056–5061.

Saboori, A. and Hadjicostis, C.N. (2008). Verification of initial-state opacity in security applications of DES. In *Proc. of 9th Int. Workshop on Discrete Event Systems (WODES)*, 328–333.

Saboori, A. and Hadjicostis, C.N. (2011a). Verification of K-step opacity and analysis of its complexity. *IEEE Transactions on Automation Science and Engineering*, 8(3), 549–559.

Saboori, A. and Hadjicostis, C.N. (2013). Verification of initial-state opacity in security applications of DES. *Information Sciences*, 246, 115–132.

Saboori, A. and Hadjicostis, C.N. (2011b). Coverage analysis of mobile agent trajectory via state-based opacity formulations. *Control Engineering Practice*, 19(9), 967–977.

Saboori, A. and Hadjicostis, C.N. (2014). Current-state opacity formulations in probabilistic finite automata. *IEEE Transactions on Automatic Control*, 59(1), 120–133.

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

Wu, Y.C., Sankararaman, K.A., and Lafortune, S. (2014). Ensuring privacy in location-based services: An approach based on opacity enforcement. *IFAC Proceedings Volumes*, 47(2), 33–38.

a given (possibly non-deterministic) finite automaton. Though strong K-detectability is not of interest in this paper, the construction of the K-detector will prove useful if we take $K = |S|$ or larger (where $|S|$ is the number of secret states). Please note that the work in (Hadjicostis and Seatzu, 2016) established several important relationships between the observer and the K-detector of a given system (see the proof of Theorem 4 in that paper).

Definition 4. (K-Detector (Hadjicostis and Seatzu, 2016)) Consider DFA $G_d = (X, \Sigma, \delta, x_0)$ under an output mapping $\lambda : \Sigma \rightarrow Y \cup \{\epsilon\}$ (assuming initial uncertainty $\hat{X}_0$ with $x_0 \in \hat{X}_0$ at the observer and taking $X_{0,obs} = UR(\hat{X}_0)$). The K-detector $G_{kd} = (X_{kd}, Y, \delta_{kd}, X_{0,kd})$ is a non-deterministic finite automaton, where

(1) $X_{kd} = \{X_{0,kd}\} \cup X_s \cup X_p$ with
(i) $X_{0,kd} = UR(\hat{X}_0)$ being the set of all possible initial states for DFA G_d before any observation is made;
(ii) $X_s = \{T_s \mid T_s \subseteq X \wedge |T_s| \leq K\}$; and
(iii) $X_p = \{T_p \mid T_p \subseteq X \wedge |T_p| = K + 1\}$.
(2) $\delta_{kd} : X_{kd} \times Y \rightarrow X_{kd}$ captures the state transitions and is defined, for $x_{kd} \in X_{kd}$ and $y \in Y$, as follows:

$$\delta_{kd}(x_{kd}, y) = \begin{cases} \{T_s \in X_s \mid T_s = R(x_{kd}, y)\}, & \text{if } |R(x_{kd}, y)| \leq K, \\ \{T_p \in X_p \mid T_p \subseteq R(x_{kd}, y)\}, & \text{if } |R(x_{kd}, y)| > K. \end{cases}$$

To solve the problem formulated in Section 3, we can follow a similar approach as in Section 4.1, to track the state of the system and the knowledge (state estimate) at the observer/intruder. The difference is that this time, instead of using an observer, we use a K-detector (with $K \geq |S|$). More specifically, to simultaneously track the state of the system and the state of the K-detector, we use a parallel-like composition of G_d and G_{kd}, which is an NFA $G_{dkd} = AC(X_{dkd}, \Sigma, \delta_{dkd}, X_{0,dkd})$ constructed as follows:

(1) $X_{dkd} = X \times X_{kd}$
(2) $X_{0,dkd} = (x_0, X_{0,kd})$
(3) The function δ_{dkd} is defined as follows:
(i) For $x \in X$ and $x_{kd} \in X_{kd}$, and all $\sigma \in \Sigma$ such that $\lambda(\sigma) \in Y$ (observable σ), we have

$$\delta_{dkd}((x, x_{kd}), \sigma) = \{\delta(x, \sigma)\} \times \delta_{kd}(x_{kd}, \lambda(\sigma))$$

(ii) For $x \in X$ and $x_{kd} \in X_{kd}$, and all $\sigma \in \Sigma$ such that $\lambda(\sigma) = \epsilon$ (unobservable), we have

$$\delta_{dkd}((x, x_{kd}), \sigma) = \{(\delta(x, \sigma), x_{kd})\}$$

(Note that $\delta_{dkd}((x, x_{kd}), \sigma)$ is taken to be the empty set if $\delta(x, \sigma)$ is undefined.)

Algorithm 2 below can be advantageous when the set of secret states S is of relatively small cardinality.

Algorithm 2: Consider the problem formulation in Section 3, where we would like to find (if possible) a sequence of inputs $t \in \Sigma^*$, such that $\delta(x_0, t) = x_f$ and for each $s \in \overline{t}$, we have $\hat{X}(\lambda(s)) \cap (X \setminus S) \neq \emptyset$.

Step 1: We construct the K-detector (for $K \geq |S|$) $G_{kd} = (X_{kd}, Y, \delta_{kd}, X_{0,kd})$.
Step 2: We construct the parallel-like composition DFA $G_{dkd} = AC(X_{dkd}, \Sigma, \delta_{dkd}, X_{0,dkd})$.
Step 3: In the transition digraph of G_{dkd}, we eliminate nodes (states) of the form (x, x_{dkd}) for which $x_{dkd} \cap (X \setminus S) = \emptyset$.
Step 4: In what remains of the transition digraph of G_{dkd}, we (i) mark states (nodes) of the form (x_f, x_{dkd}) (for some x_{dkd}) as potential final states; and (ii) look for a path that starts from state $X_{0,dkd} = (x_0, X_{0,kd})$ and ends at a marked state. If a path can be found, then we have one possible sequence of inputs that solves the problem of interest; if no path can be found, then no solution exists.

Theorem 3. Consider DFA $G_d = (X, \Sigma, \delta, x_0)$ under an output mapping $\lambda : \Sigma \rightarrow Y \cup \{\epsilon\}$. Given a target state $x_f \in X$, a set of secret states S, $S \subseteq X$, and initial state uncertainty $\hat{X}_0$ at the observer, Algorithm 2 solves the problem formulated in Section 3.

Proof. The first direction is straightforward: if Algorithm 2 finds, in the digraph of G_{dkd}, a path, say of length n, from the initial state $X_{0,dkd} = (x_0, X_{0,kd})$ to some marked state (x_f, x_{kd}), this path corresponds to a sequence of inputs $t = \sigma_{i_1}\sigma_{i_2}...\sigma_{i_n} \in \Sigma^*$ such that $\delta(x_0, t) = x_f$. Furthermore, the path in the digraph of G_{dkd} defines a specific sequence of states in the K-detector, say

$$T_0 = X_{0,kd}, T_1, T_2, ..., T_n = x_{kd} \ ,$$

where T_j is the state of the K-detector following the occurrence of σ_{i_j}. [Note that if a certain σ_{i_j} is unobservable (i.e., $\lambda(\sigma_{i_j}) = \epsilon$), then $T_{j-1} = T_j$ (as expected, the K-detector does not change state).] If we view the states of the K-detector as subsets of X, we have that

$$T_j \cap (X \setminus S) \neq \emptyset \ , j = 0, 1, ..., n \ .$$

We can easily establish (e.g., by induction) that

$$T_j \subseteq R(\hat{X}_0, \lambda(\sigma_{i_1}\sigma_{i_2}...\sigma_{i_j})) \ , j = 0, 1, ..., n \ ,$$

where for $j = 0$ we have $T_0 = X_{0,kd} = UR(\hat{X}_0) = R(\hat{X}_0, \epsilon)$. Thus, since $T_j \cap (X \setminus S) \neq \emptyset$, we also have $R(\hat{X}_0, \lambda(\sigma_{i_1}\sigma_{i_2}...\sigma_{i_j})) \cap (X \setminus S) \neq \emptyset$ for $j = 0, 1, ..., n$. We conclude that that the sequence of inputs $t = \sigma_{i_1}\sigma_{i_2}...\sigma_{i_n}$ is a solution to the problem of interest.

To establish the reverse direction, we need to argue that if Algorithm 2 returns no solution (no path with the desirable properties is found in the digraph of G_{dkd}), then no solution to the problem formulation of Section 3 exists. We prove the contrapositive: if there is a solution to the problem of interest, which means that there is a path, say of length n, in the digraph of DFA G_{do} of Algorithm 1, then we necessarily have a path of length n in the digraph of the G_{dkd} of Algorithm 2.

Suppose that the sequence of inputs $t = \sigma_{i_1}\sigma_{i_2}...\sigma_{i_n}$ in the digraph of DFA G_{do} is such that $\delta(x_0, t) = x_f$ and the sequence of states visited in the digraph of G_{do} is associated with state estimates

$$T'_0 = X_{0,obs}, T'_1, ..., T'_2, ..., T'_n = x_{kd} \ ,$$

such that $T'_j \cap (X \setminus S) \neq \emptyset$ for $j = 1, 2, ..., n$. [Note that if a certain σ_{i_j} is unobservable (i.e., $\lambda(\sigma_{i_j}) = \epsilon$), then $T'_{j-1} = T'_j$ (this would not be a transition in the observer construction).]

From the property of the K-detector outlined Theorem 4 of (Hadjicostis and Seatzu, 2016), we have that the sequence t in the digraph of NFA G_{dkd} induces at least one sequence of states $T_0 = X_{0,kd}, T_1, T_2, ..., T_n = x_{kd}$ in the detector that satisfies for $j = 1, ..., n$:

(starting location) x_0, $x_0 \in X$, we want to get to some target state (destination) x_f, $x_f \in X$, by a applying a sequence of inputs (events) $t \in \Sigma^*$ (i.e., we need to choose t such that $\delta(x_0, t) = x_f$). In addition, we assume that there is an output mapping $\lambda : \Sigma \to Y \cup \{\varepsilon\}$ with respect to certain outputs Y (sensor information) that become available at an external observer. We would like the automaton (vehicle) controller to choose t so that, after each observation, the current state (present location) of the vehicle is kept opaque with respect to a given set S, $S \subseteq X$, of secret/critical states. If such a sequence of inputs t exists, we would like to find one. More formally, the problem can be expressed as follows.

Problem Formulation: Consider DFA $G_d = (X, \Sigma, \delta, x_0)$ under an output mapping $\lambda : \Sigma \to Y \cup \{\epsilon\}$. Given a target state $x_f \in X$, a set of secret states S, $S \subseteq X$, and initial state uncertainty $\hat{X}_0$ at the observer (we assume $x_0 \in \hat{X}_0$), find (if possible) a sequence of inputs $t \in \Sigma^*$, such that (i) $\delta(x_0, t) = x_f$, and (ii) for each $s \in \overline{t}$, we have $\hat{X}(\lambda(s)) \cap (X \setminus S) \neq \emptyset$ (i.e., at no point in time does the external observer know that the state of the system lies strictly within S.

Remark 1. Given the set of secret states S, the following question would be directly related to current (or delayed, or initial) state opacity: are we guaranteed that, no matter how the vehicle moves, the external observer will never be able to isolate the current (or delayed or initial) location of the vehicle within the set of critical locations S? The answer to this question is essentially equivalent to the question of verifying current (or delayed or initial) state opacity in a given NFA, problem that have been shown to be NP-hard. In this paper, we ask the question from the point of the vehicle controller: can the vehicle be navigated in such a way so that its current (more generally, delayed, or initial) location cannot be isolated within the set of critical locations S? This question is related to weak current-state opacity (see Definition 3) and in the next section we develop two algorithms to solve this problem.

Remark 2. An alternative to the above problem formulation is to modify the observations that are generated by the system so as to keep the observer/intruder uncertain (see, for example, the strategies in Wu and Lafortune (2014) which insert label among the actual labels).

4. ALGORITHMIC SOLUTIONS

We describe two possible algorithmic solutions to the problem formulated in the previous section. One approach is via the use of a parallel construction that involves the given system and its observer, in order to capture simultaneously the (actual) state of the system and the set of system states perceived (estimated) at the observer/intruder; this approach has complexity $O(|X|2^{|X|})$ (where X is the set of states of the given system) and is similar to a methodology (for a different problem) proposed in (Hadjicostis, 2010). The second approach uses a parallel construction that involves the given system and the K-detector introduced in (Hadjicostis and Seatzu, 2016); this approach has complexity $O(|X|^{|S|+2})$ and would be preferable in cases where the number $|S|$ of secret/critical states is relatively small. We leave to future work extensions that try to obtain trajectories that are also optimal under some appropriate criterion of interest and focus on finding feasible (current-state opacity-preserving) trajectories.

4.1 Parallel Composition of System and Observer

Given a DFA $G_d = (X, \Sigma, \delta, x_0)$ under an output mapping $\lambda : \Sigma \to Y \cup \{\epsilon\}$, we can construct its observer $G_{obs} = (X_{obs}, Y, \delta_{obs}, X_{0,obs})$ (assuming initial uncertainty $\hat{X}_0$ with $x_0 \in \hat{X}_0$ and taking $X_{0,obs} = UR(\hat{X}_0)$). We can simultaneously track the state of the system and the state of the observer via a parallel-like composition of G_d and G_{obs}, which is a DFA $G_{do} = AC(X_{do}, \Sigma, \delta_{do}, x_{0,do})$ constructed as follows:

(1) $X_{do} = X \times X_{obs}$
(2) $x_{0,do} = (x_0, X_{0,obs}))$
(3) The function δ_{do} is defined as follows:
(i) For $x \in X$ and $x_{obs} \in X_{obs}$, and all $\sigma \in \Sigma$ such that $\lambda(\sigma) \in Y$ (observable σ), we have

$$\delta_{do}((x, x_{obs}), \sigma) = (\delta(x, \sigma), \delta_{obs}(x_{obs}, \lambda(\sigma)))$$

(ii) For $x \in X$ and $x_{obs} \in X_{obs}$, and all $\sigma \in \Sigma$ such that $\lambda(\sigma) = \epsilon$ (unobservable), we have

$$\delta_{do}((x, x_{obs}), \sigma) = (\delta(x, \sigma), x_{obs})$$

(Note that $\delta_{do}((x, x_{obs}), \sigma)$ is undefined if $\delta(x, \sigma)$ is undefined.)

Effectively, G_{do} simultaneously tracks the state of the system (which is tightly associated with the control options at each state, e.g., the possible moves of the vehicle in the path planning example) and the state of the observer (knowledge of the intruder). In terms of the problem we are interested in, we would like to find a path in the transition digraph of G_{do} that starts from state $x_{0,do} = (x_0, X_{0,obs})$ and ends at a state of the form (x_f, x_{obs}) (for some $x_{obs} \in X_{obs}$), such that all states that are visited along this path (including starting and ending states) are of the form (x', x'_{obs}) where $x'_{obs} \cap (X \setminus S) \neq \emptyset$. If no such path can be found, then the problem of interest has no solution. The procedure is summarized as Algorithm 1 below.

Algorithm 1: Consider the problem formulation in Section 3, where we would like to find (if possible) a sequence of inputs $t \in \Sigma^*$, such that $\delta(x_0, t) = x_f$ and for each $s \in \overline{t}$, we have $\hat{X}(\lambda(s)) \cap (X \setminus S) \neq \emptyset$.

Step 1: We construct the observer of the given system $G_{obs} = (X_{obs}, Y, \delta_{obs}, X_{0,obs})$.
Step 2: We construct DFA $G_{do} = AC(X_{do}, \Sigma, \delta_{do}, x_{0,do})$.
Step 3: In the transition digraph of G_{do}, we eliminate nodes (states) of the form (x, x_{obs}) for which $x_{obs} \cap (X \setminus S) = \emptyset$.
Step 4: In what remains of the transition digraph of G_{do}, we (i) mark states (nodes) of the form (x_f, x_{obs}) (for some x_{obs}) as potential final states, and (ii) look for a path that starts from state $x_{0,do} = (x_0, UR(\hat{X}_0))$ and ends at a marked state. If a path can be found, then we have the desirable sequence of inputs; if no path can be found, then no solution exists.

4.2 Parallel Composition of System and K-Detector

The K-detector was introduced in (Hadjicostis and Seatzu, 2016) to solve the problem of strong K-detectability for

possible[2] initial states $\hat{X}_0$). We use $\hat{X}(\omega)$ to denote the set of states that the system might reside in, given that the sequence of outputs ω has been observed, and refer to it as the current-state estimate of the external observer. Notice that $\hat{X}(\omega) = R(\hat{X}_0, \omega)$.

Given the DFA $G_d = (X, \Sigma, \delta, x_0)$, under the output mapping λ with outputs Y, we can construct its current-state estimator (or observer), denoted by $G_{obs} = (X_{obs}, Y, \delta_{obs}, X_{0,obs})$ in a straightforward manner (see, for example, (Cassandras and Lafortune, 2009)). The important property of this construction is that each state of G_{obs} is associated with a unique subset of states of the original DFA G_d (so that $X_{obs} \subseteq 2^X$, i.e., there are at most $2^{|X|}$ states) and δ_{obs} is defined so that, when the sequence of inputs $t \in \Sigma^*$ is applied generating the sequence of outputs $\omega = \lambda(t)$, we have

$$\hat{X}(\omega) = \delta_{obs}(X_{0,obs}, \omega) \ .$$

Note that $X_{0,obs} = UR(X_0)$ (so that $\hat{X}(\epsilon) = X_{0,obs}$).

2.2 Current-State Opacity

Definition 2. (Current-State Opacity). Consider a DFA $G_d = (X, \Sigma, \delta, x_0)$, with initial state uncertainty $\hat{X}_0$ under an output mapping $\lambda : \Sigma \to Y \cup \{\epsilon\}$ (where Y is the set of outputs). Given a set of secret states $S \subseteq X$, automaton G_d is current-state opaque with respect to S if $\forall t \in \Sigma^*$, the following is true

$$\{\delta(x_0, t) \in S\} \Rightarrow$$
$$\{\exists s \in \Sigma^*, \exists x_0' \in \hat{X}_0 \{\lambda(s) = \lambda(t), \delta(x_0', s) \notin S\}\} \ .$$

More specifically, given DFA $G_d = (X, \Sigma, \delta, x_0)$ with output set Y and output mapping λ, we define the secret behavior of the system to be behavior that leads the system to a state within the secret set of states S, $S \subseteq X$. Note that the definition above is slightly different from the standard one, because in this paper we are dealing with a DFA with unknown initial state, and we are interested in ensuring opacity for all possible behavior from this initial state.

The above notion of opacity is sometimes referred to as *strong* current-state opacity. *Weak* current-state opacity is defined similarly except that it only requires the above property to hold for one (infinite) sequence of events. Here we define weak current-state opacity with respect to the DFA G_d assuming that G_d is live (i.e., from each state there is at least one transition that is defined).

Definition 3. (Weak Current-State Opacity). Consider a live DFA $G_d = (X, \Sigma, \delta, x_0)$, with initial state uncertainty $\hat{X}_0$ under an output mapping $\lambda : \Sigma \to Y \cup \{\epsilon\}$ (where Y is the set of outputs). Given a set of secret states $S \subseteq X$, automaton G_d is weak current-state opaque with respect to S if there exists an infinite $t \in L(G_d, x_0)$ such that for all $s \in \bar{t}$ we have

$$\{\delta(x_0, s) \in S\} \Rightarrow$$
$$\{\exists s' \in \Sigma^*, \exists x_0' \in \hat{X}_0 \{\lambda(s') = \lambda(s), \delta(x_0', s') \notin S\}\} \ .$$

Other state-based notions of opacity have also been developed, including initial-state opacity (Saboori and Hadjicostis, 2008), K-step opacity (Saboori and Hadjicostis, 2011a), and probabilistic opacity (Lakhnech and Mazaré, 2005; Saboori and Hadjicostis, 2014).

[2] We assume that $\hat{X}_0$ necessarily includes the true initial state x_0 (this can be easily ensured, e.g., by setting $\hat{X}_0 = X$).

3. MOTIVATION AND PROBLEM FORMULATION

As motivation for studying trajectory planning under current-state opacity constraints, we discuss below an example in the context of vehicle tracking (see also (Saboori and Hadjicostis, 2011b)). For a related example that captures privacy notions when offering location-based services, refer to (Wu et al., 2014).

Consider a vehicle that moves on a grid, such as the toy 2×2 grid on the left of Fig. 1, where each cell corresponds to a location. The moves that are allowed at each location depend on the particular terrain and the vehicle specifications/capabilities, and can be summarized by a kinematic model, i.e., a deterministic finite automaton whose states match the locations (cells) in the grid and whose transitions match the movements that the vehicle is allowed or is capable of making at each position (up, down, left, right, diagonal, etc.).

Suppose that an external observer/intruder is trying to locate the position of the vehicle in the grid based on certain observations that are generated by sensors that are available in the system. Note that more than one sensor may simultaneously emit a signal in case the vehicle is in a cell covered by multiple sensors. An observer with access to this sensory information could attempt to locate the vehicle (i.e., identify its current, past, or initial location) based on the sequence of observations it sees. To capture sensor information, the kinematic model can be enhanced with output (observation) symbols, denoted by $Y = \{y_1, y_2, ..., y_{|Y|}\}$ that are generated from the sensors that are available via some output mapping λ. This implies that the vehicle setting we described can be modeled via a DFA under some output mapping of the type we described in Section 2. In Fig. 1, a distinct event is associated with each transition of the kinematic model on the right; e.g., the transition σ_{21} is the event that takes us from state x_1 to state x_2 (not shown in the figure) and outputs $y_2 = \beta$.

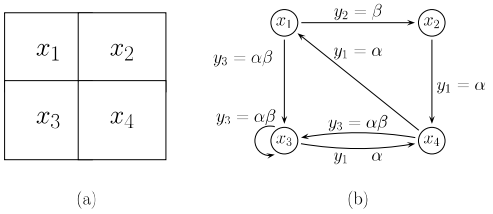


Fig. 1. Grid in which a vehicle can move (left) and kinematic model for a vehicle in the grid (right).

Since we are viewing each cell as the state (location) of the vehicle, we can think of a state sequence in the kinematic model as a vehicle trajectory (path). In fact, the starting position of the trajectory will correspond to the initial state of the finite automaton that captures the kinematic model, whereas the ending position of the vehicle will be the final (present or current) state of this finite automaton.

Consider the following challenge and its relationship to the motivating application above: given a DFA $G_d = (X, \Sigma, \delta, x_0)$ (vehicle) that starts at some initial state

to a final (target) state is relatively straightforward and possible as long as the target state is reachable from the initial state: any sequence of inputs that takes the deterministic system from the initial state to the final state would be an acceptable solution to the (unconstrained) trajectory (or path) planning problem. The main challenge for the control mechanism is the presence of an external (but passive) observer/intruder and the need to ensure that this observer is never in position to determine, with certainty, that the state of the system lies within a subset of states S that captures secret/critical states.

In terms of the opacity notions described above, the goal of the controller is to drive the system from the initial state to the final state by applying a sequence of inputs such that current-state opacity with respect to the set of secret states S is not violated at any point (from the perspective of the external observer that observes the sequence of outputs generated by the system while following the chosen trajectory). We develop two algorithms to find an appropriate sequence of inputs (if one exists). The first algorithm relies on the construction of an observer (Hadjicostis, 2010) and has exponential complexity in N (where N is the number of the states of the given system). The second algorithm, which relies on the use of a K-detector (where K is the number of secret states), has complexity polynomial in N and exponential in K.

2. BACKGROUND AND MOTIVATION

2.1 Preliminaries, Notation, and Observation Model

We use Σ to denote an alphabet of symbols and Σ^* to denote the set of all finite-length strings of elements of Σ, including the empty string ϵ. Given a string t, $t \in \Sigma^*$, we use $|t|$ to denote the length of t (with $|\epsilon| = 0$). Given strings $s, t \in \Sigma^*$, the string st denotes the concatenation of s and t, i.e., the sequence of events captured by s followed by the sequence of events captured by t. For a string s, $\overline{s}$ denotes the *prefix-closure* of s, and is defined as the set of strings $\overline{s} := \{t \in \Sigma^* \mid \exists t' \in \Sigma^* \{tt' = s\}\}$.

A deterministic finite automaton (DFA) is denoted by $G_d = (X, \Sigma, \delta, x_0)$, where[1] $X = \{x_1, x_2, \ldots, x_{|X|}\}$ is the (finite) set of states, $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_{|\Sigma|}\}$ is the (finite) set of inputs or events, $\delta : X \times \Sigma \to X$ is the deterministic, possibly partially defined, transition function, and $x_0 \in X$ is the initial state. The function δ can be extended from the domain $X \times \Sigma$ to the domain $X \times \Sigma^*$ in the routine recursive manner: for $x_j \in X$, we define $\delta(x_j, ts) := \delta(\delta(x_j, t), s)$ for $t \in \Sigma$ and $s \in \Sigma^*$, with $\delta(x_i, \epsilon) := x_i$ for all $x_i \in X$. Since δ is only partially defined, $\delta(x_j, ts)$ is assumed undefined if any of the transitions in the recursion turns out to be undefined. The behavior of DFA G_d is captured by its language $L(G_d) := \{s \in \Sigma^* \mid \delta(x_0, s) \text{ is defined}\}$.

A non-deterministic finite automaton (NFA) is denoted by $G_n = (X, \Sigma, \delta, X_0)$, where $X = \{x_1, x_2, \ldots, x_{|X|}\}$ is the set of states, $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_{|\Sigma|}\}$ is the set of inputs or events, $\delta : X \times \Sigma \to 2^X$ (where 2^X is the power set of X) is the non-deterministic transition function, and $X_0 \subseteq X$ is the set of possible initial states. The function δ can be extended from the domain $X \times \Sigma$ to the domain $X \times \Sigma^*$ in the routine recursive manner: for any state $x_j \in X$, we let $\delta(x_j, ts) := \bigcup_{x_i \in \delta(x_j, t)} \delta(x_i, s)$, for $t \in \Sigma$ and $s \in \Sigma^*$, with $\delta(x_i, \epsilon) := \{x_i\}$ for all $x_i \in X$. The behavior of NFA G_n is captured by its language $L(G_n) := \{s \in \Sigma^* \mid \exists x_0 \in X_0 \{\delta(x_0, s) \neq \emptyset\}\}$.

Given a DFA $G_d = (X, \Sigma, \delta, x_0)$ that models a system of interest, we will typically face constraints in terms of what is observed externally when activity takes place in the system. For simplicity, this paper adopts a model that closely resembles the traditional natural projection mapping that is used in much of the existing literature on fault diagnosis and opacity (Cassandras and Lafortune, 2009; Jacob et al., 2015). More specifically, we will assume that there is a set of output (observation) symbols $Y = \{y_1, y_2, ..., y_{|Y|}\}$ and a mapping λ that associates inputs (or events) in Σ to corresponding outputs (or observations) $\lambda : \Sigma \to Y \cup \{\epsilon\}$, where ϵ denotes the empty observation. For example, $\lambda(\sigma_1) = \lambda(\sigma_2) = y_1$ would imply that input events σ_1 and σ_2 emit the same output y_1 when they occur; thus, they will be indistinguishable by the external observer, unless other knowledge is taken into account. Similarly, $\lambda(\sigma_3) = \epsilon$ would indicate that σ_3 is unobservable (and might occur without emitting any output); thus, the occurrence of σ_3 will go undetected by the external observer, unless knowledge of the system model and subsequent observations are taken into account.

Given a sequence of events $t = \sigma_{i_1}\sigma_{i_2}...\sigma_{i_n}$ (where $\sigma_{i_j} \in \Sigma$ for $j = 1, 2, ..., n$) that occurs in the system, the corresponding sequence of outputs that is observed is captured by the concatenation of the corresponding outputs

$$\omega = \lambda(\sigma_{i_1})\lambda(\sigma_{i_2})...\lambda(\sigma_{i_n}) =: \lambda(t) \ .$$

Note that the above sequence could have length smaller than n if some of the inputs are unobservable. Also note that the difference of the above output mapping λ from the standard natural projection P is that P typically assigns a unique output to each observable symbol.

Definition 1. (Possible states following a sequence of observations ($R : 2^X \times Y^* \to 2^X$)) Suppose that DFA $G_d = (X, \Sigma, \delta, x_0)$ is known to be in a set of possible states T, $T \subseteq X$; the set of all possible states after subsequently observing $\omega \in Y^*$ is

$$R(T, \omega) = \{x \in X \mid \exists x' \in T, \exists t \in \Sigma^*, \text{ s.t. } \{\lambda(t) = \omega \wedge x = \delta(x', t)\}\} \ .$$

Note that, using the above definition, the unobservable reach $UR(T)$ (i.e., the set of states that can be reached from states in the set T via sequences of zero, one, or more events that are unobservable —see (Cassandras and Lafortune, 2009)) can be expressed as $UR(T) = R(T, \epsilon)$.

Consider a DFA $G_d = (X, \Sigma, \delta, x_0)$, with initial state uncertainty $\hat{X}_0$ under an output mapping $\lambda : \Sigma \to Y \cup \{\epsilon\}$ (where Y is the set of outputs). Upon observing the output string that is generated, namely $\omega = \lambda(t)$, the external observer aims to determine the possible states of the system, which might be uncertain due to the partial observation of events and/or due to the lack of knowledge of the initial state (e.g., the observer may not know precisely the initial state x_0, but only know a set of

[1] For any set X, we use $|X|$ to denote its cardinality.

Preprints of WODES 2018
May 30 - June 1, 2018
Sorrento Coast, Italy

Trajectory Planning under Current-State Opacity Constraints ⋆

Christoforos N. Hadjicostis

Department of Electrical and Computer Engineering
University of Cyprus, Nicosia, Cyprus
(email: chadjic@ucy.ac.cy)

Abstract: Privacy and security guarantee against curious observers or malicious actors has recently emerged as a critical aspect for maintaining, protecting, and securing complex automated systems that are implemented over shared (thus, non-secure) cyber-infrastructures, such as the Internet. In this paper, we discuss how current-state opacity formulations can be used to capture privacy properties of interest in automated systems that are modeled as controlled finite automata that need to be steered from one state (initial location) to another state (target location), while maintaining certain privacy guarantees. More specifically, given a deterministic finite automaton that is externally observed via some output mapping, along with a subset of states S that are considered critical/secret, we aim to drive it from a given initial state (starting location) to a given target state (final location) while ensuring that, in the process, the state (location) of the finite automaton at any given time is not exposed (i.e., the external observer cannot be certain that the state of the system belongs to the set of critical/secret states S). We develop two algorithms that can be used to solve this constrained trajectory planning problem and obtain an appropriate sequence of inputs (if one exists) or conclude that no such sequence exists (otherwise). The first algorithm has complexity $O(N2^N)$ and the second algorithm has complexity $O(N^{K+2})$, where N (K) is the number of states (secret states).

Keywords: Current-state opacity, weak current-state opacity, privacy, trajectory planning.

1. INTRODUCTION

The reliance of many emerging automation systems on shared (thus, non-dedicated and non-secure) cyber-infrastructures, such as public telecommunication systems and the Internet, has naturally led to increasing security and privacy concerns. As a result, various notions of security and privacy have received considerable attention from researchers. This paper continues this line of research in the context of automation systems that are captured by known controlled finite automata and are externally observed via known output mappings. Our goal is to devise a control strategy to drive these systems from a known initial state to some target final state while ensuring that the true state of the system maintains certain privacy properties. More specifically, we consider a trajectory (or path) planning problem under the requirement that the trajectory (path) that is followed —to get from the initial state (or starting location) to the final state (or target location)— maintains, at all times, current-state opacity requirements with respect to a subset of system states (called critical or secret states).

The observation capabilities of a curious observer (intruder) relate to existing notions that focus on characterizing the *information flow* from the system to the observer (Focardi and Gorrieri, 1994). Opacity falls in this category of information flow properties, and aims at determining whether a given system's *secret* behavior (i.e., a subset of the behavior of the system that is considered critical and is usually represented by a predicate) is kept opaque to outsiders (Bryans et al., 2005; Saboori and Hadjicostis, 2007). Earlier work (see, for example, (Saboori and Hadjicostis, 2007, 2008; Lin, 2011; Saboori and Hadjicostis, 2011a, 2013)) considered opacity with respect to predicates that are state-based. More specifically, given a non-deterministic finite automaton (NFA) with partial observation on its transitions (captured via a natural projection map), the secret behavior of the system is defined with respect to behavior under which the system's state evolves (at some point in time) to a subset of a given set of secret/critical states S. In other words, S (which is assumed to be a known subset of the system's states) is used to denote states that are secret/critical, in the sense that the observer should never be in position to determine that the system state lies exclusively within S. As common in these settings, we make the worst-case assumption that the intruder has full knowledge of the system model and is able to track activity in the system via some observation mapping.

In this paper, we consider systems that can be modeled as deterministic finite automata with inputs that can be chosen by a control mechanism that is fully aware of the state of the system. Thus, the task of driving the given deterministic finite automaton from the initial state

⋆ This material is based upon work supported in part by a grant by the University of Cyprus (UCY). Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of UCY.

for at least one unobservable event e_{uo_k}. This means that at least for one e_{uo_k} it should be either

$$\sum_{t\in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) - |w|_{e_{uo_k}} + 1 \leq 0\,, \tag{A.1}$$

or

$$-\sum_{t\in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) + |w|_{e_{uo_k}} + 1 \leq 0\,. \tag{A.2}$$

Given the two binary decision variables δ_{k1} and δ_{k2}, and given one unobservable event e_{uo_k}, the fulfilling of either (A.1) or (A.2) is equivalent to the satisfaction of the linear constraint

$$\delta_{k1} + \delta_{k2} = 1\,, \tag{A.3}$$

and of the two logical statements

$$\left[\sum_{t\in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) - |w|_{e_{uo_k}} + 1 \leq 0\right] \leftrightarrow [\delta_{k1} = 1]\,, \tag{A.4a}$$

$$\left[-\sum_{t\in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) + |w|_{e_{uo_k}} + 1 \leq 0\right] \leftrightarrow [\delta_{k2} = 1]\,, \tag{A.4b}$$

where " $\leftrightarrow$ " indicates the *if and only if* connective. According to the technique proposed in Bemporad and Morari (1999), each of the two logical statements (A.4) can be turned into a couple of linear inequalities. In particular, being B a sufficiently large integer, the predicate (A.4a) is equivalent to

$$\sum_{t\in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) - |w|_{e_{uo_k}} + 1 \leq B\cdot(1-\delta_{k1})\,,$$

$$\sum_{t\in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) - |w|_{e_{uo_k}} \geq -B\cdot\delta_{k1}\,,$$

while (A.4b) is equivalent to

$$-\sum_{t\in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) + |w|_{e_{uo_k}} + 1 \leq B\cdot(1-\delta_{k2})\,,$$

$$-\sum_{t\in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) + |w|_{e_{uo_k}} \geq -B\cdot\delta_{k2}\,.$$

Since (A.3), (A.5) and (A.6) should hold at least for one unobservable event e_{uo_k}, the equality (A.3) is turned into the constraints (4f) and (4g).

REFERENCES

Bemporad, A. and Morari, M. (1999). Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3), 407–427.

Cabasino, M., Giua, A., Lafortune, S., and Seatzu, C. (2012). A new approach for diagnosability analysis of Petri nets using verifier nets. *IEEE Transactions on Automatic Control*, 57(12), 3104–3117.

Cassandras, C. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*. Springer.

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

GLPK (2018). (GNU Linear Programming Kit). https://www.gnu.org/software/glpk/.

Jacob, R., Lesage, J., and Faure, J. (2016). Overview of discrete event systems opacity: Models, validation, and quantification. *Annual Reviews in Control*, 41, 135–146.

Lin, F. (2011). Opacity of discrete event systems and its applications. *Automatica*, 47(3), 496–503.

Löfberg, J. (2004). YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *Proc. CACSD Conference*. Taipei, Taiwan.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proc. of IEEE*, 77(4), 541–580.

Reiter, M. and Rubin, A. (1998). Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), 66–92.

Reveliotis, S. (2005). A linear characterization of the petri net reachability space corresponding to bounded-length fireable transition sequences and its implications for the structural analysis of process-resource nets with acyclic, quasi-live and strongly reversible process subnets. In *Proc. 44th IEEE Conference on Decision and Control and 2005 European Control Conference (CDC-ECC'05)*, 2113–2118. Seville, Spain.

Tong, Y., Li, Z., Seatzu, C., and Giua, A. (2017). Verification of state-based opacity using Petri nets. *IEEE Transactions on Automatic Control*, 62(6), 2823–2837.

Tong, Y., Ma, Z., Li, Z., Seatzu, C., and Giua, A. (2016). Verification of language-based opacity in Petri nets using verifier. In *American Control Conference (ACC), 2016*, 757–763. Boston, Massachusetts.

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

Yin, X. and Lafortune, S. (2017). A new approach for the verification of infinite-step and k-step opacity using two-way observers. *Automatica*, 80, 162–171.

which has only one secret word. Hence, Theorem 3 requires to check the feasibility problem (4) only for $w = abb$. As mentioned at the end of the previous section, the feasibility problem can be checked by solving a ILP problem. All the results presented in this section have been obtained by using the *GNU Linear Programming Kit* (GLPK (2018)) and the YALMIP parser (Löfberg, 2004) in the Matlab® environment.

Exploiting these tools it turns out that for the given labeled net system, the constraints (4) admit a solution, implying that the system is LBO when $\mathcal{M}_0 = \mathcal{M}'_0$. Indeed it can be easily verified that the word $w' = abab$ is enabled under the initial marking $\boldsymbol{m}'_0$ and $\Pr(w') = \Pr(abb) = bb$.

Let us now assume that, for the net shown in Fig. 1, the set of events is equal to $E'' = \{a\,,b\,,c\}$, with $E''_o = \{b\,,c\}$ and $E''_{uo} = \{a\}$. Let the set of initial markings be equal to

$$\begin{aligned}\mathcal{M}''_0 &= \left\{\boldsymbol{m}''_{0_1}\,,\boldsymbol{m}''_{0_2}\right\}\\ &= \Big\{(2\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)^T\,,\\ &\quad\ (2\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0)^T\Big\}\,.\end{aligned}$$

Furthermore, let us now assume that the new labeling function λ'' differs from the one specified in Fig. 1 only because $\lambda''(t_4) = c$.

The feasibility problem of Lemma 3 admits again a solution, since there is at least another word w'' which is enabled under $\boldsymbol{m}''_{0_2}$ whose projection is bb.

Let us now consider the labeled net system reported in Fig. 5, with $E = \{a\,,b\,,c\}$ and $E_{uo} = \{a\,,b\}$. If the secret language is $\mathcal{L}_s = \{abc\}$, then it is not possible to find a solution to the feasibility problem of Lemma 3 even if the net is LBO. Indeed the other word that has the same observable projection as the secret contains exactly the same unobservable events of the secret but in a different order, that is $w' = bac$. In this case, if we want to assess LBO we need to solve the feasibility problem of Lemma 2 which, indeed, admits a solution.

6. CONCLUSIVE REMARKS AND FUTURE DIRECTIONS

In this paper we have exploited the mathematical representation of labeled Petri nets to provide two conditions to check language-based opacity. In particular, first a necessary and sufficient condition is given, which exploits the possibility of fully describe the set of markings reachable by means of the firing of a sequence limited in length, by using a set of linear inequalities; hence without the need of explicitly computing that reachable set. In order to reduce the computational burden of the optimization problem associated to this necessary and sufficient condition, we have exploited the firing count vectors to reduce the number of integer variables involved in the optimization. This reduction of the computational effort comes at the price of conservatism; indeed the second condition given in this paper is just a sufficient one since we cannot take into account the order of the unobservable events in each unobservable subword of the secret.

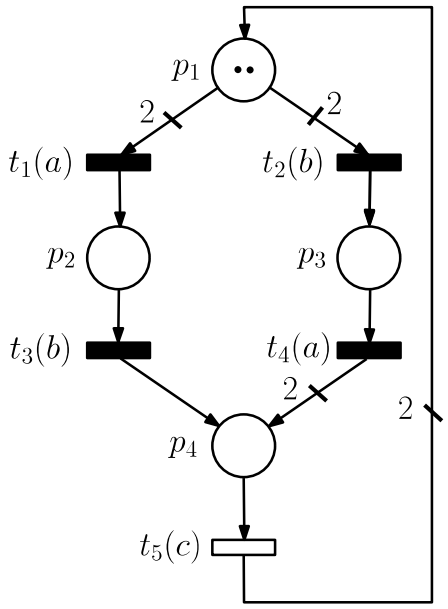


Fig. 5. Labeled net system for which it is not possible to assess LBO with respect to the secret language $\mathcal{L}_s = \{abc\}$ applying the sufficient condition stated in Theorem 3.

However, it is worth to note that both the conditions provided in this work do not require the computation of any kind of reachability graph, and can be applied also to unbounded labeled net systems. It follows that, since it does not require any offline computation, the proposed approach is particularly suitable when the plant is reconfigured online, for example in the case when the natural projection function changes during time.

Finally, there are several directions along which the proposed result can be extended; among the possibles we would like to mention:

- the possibility of considering the more general case of a non-secret language $\mathcal{L}_{NS} \subseteq \mathcal{L}(\mathcal{G}\,,\mathcal{M}_0)$;
- the possibility of considering the concept of K-step opacity (Yin and Lafortune, 2017);
- the possibility of formulate new conditions for diagnosability in labeled net systems; indeed diagnosability can be seen as a special case of opacity (see Lin (2011));
- the possibility of applying the proposed results to the *synthesis* problem, i.e. the enforcement of opacity in non-opaque systems.

Appendix A. APPENDIX A

In this appendix we briefly describe how to derive constraints (4b)–(4g) presented in Lemma 3 by exploiting the technique proposed in (Bemporad and Morari, 1999, Section 2).

In Lemma 3 there is the need of adding a set of linear constraints that, if fulfilled, assures that $\sum_{t\in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma} i_{|T_{uo}}(t)$ is different from $|w|_{e_{uo_k}}$

sufficiently large integer. If the set of constraints (2a)–(2d) and (4a)–(4g) admits a solution

$$\begin{aligned}
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \boldsymbol{\sigma}_{1_{|T_{uo}}} \geq \mathbf{0}\,,\\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \boldsymbol{\sigma}_{1_{|T_{uo}}} + \boldsymbol{c}^1 \geq \mathbf{0}\,,\\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{2} \boldsymbol{\sigma}_{i_{|T_{uo}}} + \boldsymbol{c}^1 \geq \mathbf{0}\,,\\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{2} \boldsymbol{\sigma}_{i_{|T_{uo}}} + \sum_{i=1}^{2} \boldsymbol{c}^i \geq \mathbf{0}\,,\\
&\dots \qquad (4a)\\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{\rho-1} \boldsymbol{\sigma}_{i_{|T_{uo}}} + \sum_{i=1}^{\rho-2} \boldsymbol{c}^i \geq \mathbf{0}\,,\\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{\rho-1} \boldsymbol{\sigma}_{i_{|T_{uo}}} + \sum_{i=1}^{\rho-1} \boldsymbol{c}^i \geq \mathbf{0}\,,\\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}} + \sum_{i=1}^{\rho-1} \boldsymbol{c}^i \geq \mathbf{0}\,,\\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}} + \sum_{i=1}^{\rho} \boldsymbol{c}^i \geq \mathbf{0}\,,\\
&\sum_{t \in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) - |w|_{e_{uo_k}} + 1 \leq B \cdot (1 - \delta_{k1})\,,\\
&\qquad\qquad \forall\ e_{uo_k} \in E_{uo}\,, \qquad (4b)\\
&\sum_{t \in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) - |w|_{e_{uo_k}} \geq -B \cdot \delta_{k1}\,,\\
&\qquad\qquad \forall\ e_{uo_k} \in E_{uo}\,, \qquad (4c)\\
&-\sum_{t \in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) + |w|_{e_{uo_k}} + 1 \leq B \cdot (1 - \delta_{k2})\,,\\
&\qquad\qquad \forall\ e_{uo_k} \in E_{uo},\,, \qquad (4d)\\
&-\sum_{t \in T^{e_{uo_k}}} \sum_{i=1}^{\rho} \boldsymbol{\sigma}_{i_{|T_{uo}}}(t) + |w|_{e_{uo_k}} \geq -B \cdot \delta_{k2}\,,\\
&\qquad\qquad \forall\ e_{uo_k} \in E_{uo}\,, \qquad (4e)\\
&\delta_{k1} + \delta_{k2} \leq 1\,, \qquad \forall\ k = 1\,, \dots\,, \mathrm{card}(E_{uo})\,, \qquad (4f)\\
&\sum_{k=1}^{\mathrm{card}(E_{uo})} (\delta_{k1} + \delta_{k2}) \geq 1\,. \qquad (4g)
\end{aligned}$$

Fig. 4. Constraints of the feasibility problem in Lemma 3.

- $\boldsymbol{\sigma}_1\,, \dots\,, \boldsymbol{\sigma}_\rho \in \mathbb{N}^n$
- $\mu_1\,, \dots\,, \mu_M \in \{0\,, 1\}$
- $\gamma_{ij} \in \{0\,, 1\}$ with $i = 1\,, \dots \rho$ and $j = 1\,, \dots\,, \mathrm{card}\left(T^{e_o^i}\right)$
- $\delta_{1i}\,, \delta_{2i} \in \{0\,, 1\}$ with $i = 1\,, \dots\,, \mathrm{card}(E_{uo})$

then there exists at least one $w' \in \mathcal{L}(\mathcal{G}\,, \mathcal{M}_0)$ such that $\Pr(w') = \Pr(w)$.

Proof. As in the proof of Lemma 2, the constraints (2a) and (2b) permit to *select* only one initial marking in $\mathcal{M}_0$, while (2c) and (2d) enable the firing of only one transition $t^j \in T^{e_o^i}$ for each observable event in the secret w.

Similarly to what has been done in the proof of Lemma 2, exploiting the acyclicity of the unobservable subnet and Theorem 1, it follows that if the integer vectors $\boldsymbol{\sigma}_1\,, \dots\,, \boldsymbol{\sigma}_\rho$ satisfy the inequalities (4a), then their sum $\hat{\boldsymbol{\sigma}} = \sum_{i=1}^{\rho} \boldsymbol{\sigma}_i$ represents the firing count vector of an explanation of w_o.

However, the constraints (4b)–(4g) imply that $\sum_{t \in T^{e_{uo_k}}} \hat{\boldsymbol{\sigma}}_{|T_{uo}}(t)$ is different from $|w|_{e_{uo_k}}$ for at least one unobservable event e_{uo_k} (more details on how to derive the constraints (4b)–(4g) are given in the Appendix). It follows that starting from one of the initial markings belonging to $\mathcal{M}_0$, there exists at least one enabled sequence σ' such that $\lambda(\sigma') \neq w$ and $\Pr(\lambda(\sigma')) = \Pr(w)$. This proofs the lemma.

The feasibility problem (4) in Lemma 3 requires

- $(n \cdot \rho)$ integer optimization variables
- $\left(\mathrm{card}(\mathcal{M}_0) + \sum_{i=1}^{\rho} \mathrm{card}\left(T^{e_o^i}\right) + 2 \cdot \mathrm{card}(E_{uo})\right)$ binary optimization variables

and involves a number of constraints equal to

$$\rho \cdot (2 \cdot m + 1) + 5 \cdot \mathrm{card}(E_{uo}) + 2\,.$$

It should be noticed that, in this case, neither the number of optimization variables, nor the constraints, depends on the number χ of unobservable events in the secret. Therefore, if $\rho \ll \chi$ then the sufficient condition is less demanding.

Similarly to what has been done in the case of the necessary and sufficient condition given by Theorem 2, the following sufficient condition to check LBO can be straightforwardly derived from Lemma 3.

Theorem 3. Let $\mathcal{G} = \langle N\,, \mathcal{M}_0\,, \lambda\rangle$ be a labeled net system and $\mathcal{L}_s \subseteq \mathcal{L}(\mathcal{G}\,, \mathcal{M}_0)$ a finite secret language. If for all $w \in \mathcal{L}_s$ the set of constraints (2a)–(2d) and (4) admits a solution, then $\mathcal{G}$ is LBO. ■

5. EXAMPLES

In this section we show the effectiveness of the results previously presented by means of two examples.

As we have already pointed out, the sufficient condition of Theorem 3 is generally less demanding from the computational point of view; hence, in the examples considered hereafter, we will always try to assess LBO first trying to solve the feasibility problem of Lemma 3 for each secret. If Theorem 3 will not hold, then we will try to solve the feasibility problem of Lemma 2.

Let us first consider the labeled net shown in Fig. 1 with $E_o = \{b\}$ and $E_{uo} = \{a\}$, and the set of initial markings

$$\mathcal{M}_0' = \{\boldsymbol{m}_0'\} = \left\{(2\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)^T\right\}.$$

Moreover, let us consider the secret language

$$\mathcal{L}_s = \{abb\}\,, \qquad (5)$$

$$\boldsymbol{\mu} = \boldsymbol{m}_{0_1} \circ (\mu_1 * \mathbf{1}) + \ldots + \boldsymbol{m}_{\mathbf{0_M}} \circ (\mu_{\mathbf{M}} * \mathbf{1}) \,, \tag{2a}$$

$$\sum_{i=1}^{M} \mu_i = 1 \,, \tag{2b}$$

$$\boldsymbol{c}^i = \sum_{t^j \in T^{c^i_o}} \boldsymbol{C}(\cdot\,, t^j) \circ (\gamma_{ij} * \mathbf{1}) \,, \qquad \forall\ i = 1 \,, \ldots \,, \rho \,, \tag{2c}$$

$$\sum_{j=1}^{\mathrm{card}\left(T^{c^i_o}\right)} \gamma_{ij} = 1 \,, \qquad \forall\ i = 1 \,, \ldots \,, \rho \,, \tag{2d}$$

$$\begin{aligned}
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \boldsymbol{\sigma}^1_{1_{|T_{uo}}} \geq \mathbf{0} \,, \\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{j=1}^{2} \boldsymbol{\sigma}^1_{j_{|T_{uo}}} \geq \mathbf{0} \,, \\
&\ldots \\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{j=1}^{J_1} \boldsymbol{\sigma}^1_{j_{|T_{uo}}} \geq \mathbf{0} \,, \\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{j=1}^{J_1} \boldsymbol{\sigma}^1_{j_{|T_{uo}}} + \boldsymbol{c}^1 \geq \mathbf{0} \,, \\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \left(\sum_{j=1}^{J_1} \boldsymbol{\sigma}^1_{j_{|T_{uo}}} + \boldsymbol{\sigma}^2_{1_{|T_{uo}}} \right) + \boldsymbol{c}^1 \geq \mathbf{0} \,, \\
&\ldots \\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{2} \sum_{j=1}^{J_i} \boldsymbol{\sigma}^i_{j_{|T_{uo}}} + \boldsymbol{c}^1 \geq \mathbf{0} \,, \\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{2} \sum_{j=1}^{J_i} \boldsymbol{\sigma}^i_{j_{|T_{uo}}} + \sum_{i=1}^{2} \boldsymbol{c}^i \geq \mathbf{0} \,, \\
&\ldots \\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \left(\sum_{i=1}^{\rho-1} \sum_{j=1}^{J_i} \boldsymbol{\sigma}^i_{j_{|T_{uo}}} + \sum_{j=1}^{J_\rho - 1} \boldsymbol{\sigma}^\rho_{j_{|T_{uo}}} \right) + \sum_{i=1}^{\rho-1} \boldsymbol{c}^i \geq \mathbf{0} \,, \\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{\rho} \sum_{j=1}^{J_i} \boldsymbol{\sigma}^i_{j_{|T_{uo}}} + \sum_{i=1}^{\rho-1} \boldsymbol{c}^i \geq \mathbf{0} \,, \\
&\boldsymbol{\mu} + \boldsymbol{C}_{uo} \cdot \sum_{i=1}^{\rho} \sum_{j=1}^{J_i} \boldsymbol{\sigma}^i_{j_{|T_{uo}}} + \sum_{i=1}^{\rho} \boldsymbol{c}^i \geq \mathbf{0} \,,
\end{aligned} \tag{2e}$$

$$\sum_{j=1}^{|w^i_{uo}|} \sum_{t \in T^{w^i_{uo}[j]}} \boldsymbol{\sigma}^i_{j_{|T_{uo}}}(t) \leq B \cdot (1 - \delta^i) \,, \qquad \forall\ i \in \{1 \,, \ldots \,, \rho\} \text{ such that } |w^i_{uo}| \neq 0 \,, \tag{2f}$$

$$\sum_{t \in T_{uo}} \boldsymbol{\sigma}^i_{1_{|T_{uo}}}(t) \leq B \cdot \delta^i \,, \qquad \forall\ i \in \{1 \,, \ldots \,, \rho\} \text{ such that } |w^i_{uo}| = 0 \,, \tag{2g}$$

$$\sum_{i=1}^{\rho} \delta^i \geq 1 \,, \tag{2h}$$

Fig. 3. Constraints of the feasibility problem in Lemma 2.

$$\boldsymbol{\sigma}^i_1 = \pi\left(\tilde{\sigma}^i_{uo}\right) \,, \tag{3a}$$

$$\boldsymbol{\sigma}^i_j = \mathbf{0} \,, \quad j = 2 \,, \ldots \,, |w^i_{uo}| \,, \tag{3b}$$

for $i = 1 \,, \ldots \,, \rho$. Note that the solution (3) hold also if $\tilde{\sigma}^i_{uo}$ is equal to the empty word for some values of i. This contradicts the initial assumption, and hence proves necessity.

The feasibility problem (2) can be solved by solving a ILP problem with a dummy objective function and with (2) as constraints. A possible choice for the objective function could be

$$\min \sum_{i=1}^{\rho} \sum_{j=1}^{J_i} \|\boldsymbol{\sigma}^i_j\|_1 \,,$$

where $\|\cdot\|_1$ denotes the 1-norm of a vector, that is the sum of the absolute values of the vector elements.

Moreover, given a secret word w whose lengths of the observable and unobservable part are $|w_o| = \rho$ and $|w_{uo}| = |w| - \rho = \chi$, respectively, the solution of the feasibility problem (2) requires

- $n \cdot (\chi + \xi)$ integer optimization variables
- $\left(\mathrm{card}(\mathcal{M}_0) + \sum_{i=1}^{\rho} \mathrm{card}\left(T^{e^i_o}\right) + \rho\right)$ binary optimization variables

and involves a number of constraints equal to

$$\rho \cdot (m + 2) + \chi \cdot m + \xi \cdot m + 2 \,,$$

where ξ is used to denote the number of *empty* unobservable subwords, that is ξ is the number of w^i_{uo} such that $|w^i_{uo}| = 0$. Note that the equality constraints (2a) and (2c) are redundant and have been added in (2) for the sake of readability.

Exploiting Lemma 2 it is now possible to state the following necessary and sufficient condition to check LBO in bounded labeled Petri net systems.

Theorem 2. Let $\mathcal{G} = \langle N \,, \mathcal{M}_0 \,, \lambda \rangle$ be a labeled net system and $\mathcal{L}_s \subseteq \mathcal{L}(\mathcal{G} \,, \mathcal{M}_0)$ a finite secret language. $\mathcal{G}$ is LBO *if and only if* for all $w \in \mathcal{L}_s$ the set of constraints (2) admits a solution. ■

Proof. The proof readily follows from Lemma 2 when Assumption 2 is taken into account.

The feasibility problem given in Lemma 2 requires to define, as optimization variables, one firing count vector for each of the unobservable events in the secret. In this way it is possible to explicitly take into account the order of the events when the solution of problem in Lemma 2 has the same length of w and each subword of unobservable events of the solution has the same length of w^i_{uo}. This is particularly relevant when the number of observable events in a secret is much less than the number of unobservable ones. In this case, the following lemma permits to reduce the number of optimization variable involved in the assessment of LBO at a price of more conservatism. Indeed, this lemma requires a number of firing count vectors equal to the observable events in the secret, and can be used to state a sufficient condition to check LBO in labeled net systems.

Lemma 3. Let $\mathcal{G} = \langle N \,, \mathcal{M}_0 \,, \lambda \rangle$ be a labeled net system, $w \in \mathcal{L}_S$ a secret word such that $|w_o| = \rho$, with $w_o = \Pr(w) = w_o = e_{o_1} \cdots e_{o_\rho}$, and B be a

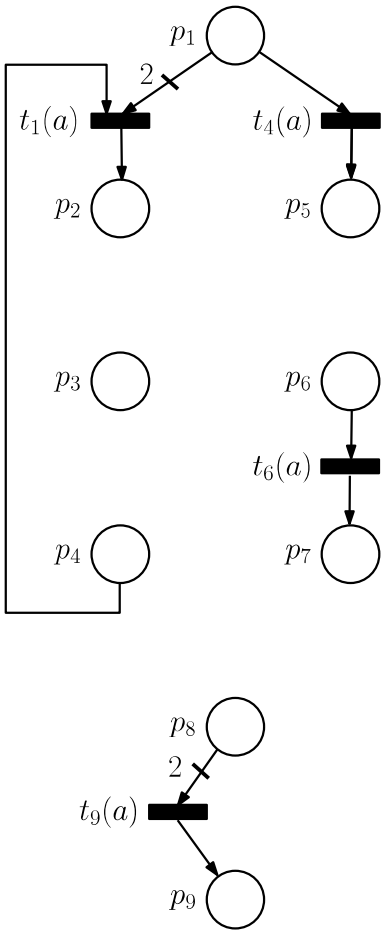


Fig. 2. Unobservable subnet induced on the net shown in Fig. 1 by the set of unobservable transitions $T_{uo} = \{t_1, t_4, t_6, t_9\}$.

Taking into account Lemma 1, in what follows we will consider only secret words that end with an observable event.

4. LBO ASSESSMENT IN LABELED PETRI NET SYSTEMS

Let us first introduce the assumptions exploited in this section to state the proposed conditions to check LBO for labeled Petri net systems.

Assumption 1. Given a labeled net system $\mathcal{G} = \langle N, \mathcal{M}_0, \lambda \rangle$, the correspondent unobservable subnet N_{uo} is assumed to be acyclic. ◇

Assumption 2. The secret language $\mathcal{L}_s$ has finite cardinality. ◇

Assumption 1 prevents the occurrence of arbitrarily long sequences of unobservable events, which would prevent an intruder to detect the occurrence of a secret for an arbitrarily long period. The same assumption is made also in Tong et al. (2016, 2017) to derive the result for bounded labeled net systems.

Assumption 2 implies that the intruder is interested to a finite number of secret sequences.

Before introducing the main result of this paper, let us consider the following lemma that holds for any secret word $w \in \mathcal{L}_S$, whose observable projection $w_o = \Pr(w)$ is such that $|w_o| = \rho$. In particular, in what follows, we will assume that the secret word has the following structure

$$w = w_{uo}^1 e_o^1 w_{uo}^2 e_o^2 \cdots w_{uo}^\rho e_o^\rho,$$

where $w_o = \Pr(w) = e_o^1 \cdots e_o^\rho$, and the unobservable subwords w_{uo}^i, with $i = 1, \dots \rho$, may also be empty.

Lemma 2. Let $\mathcal{G} = \langle N, \mathcal{M}_0, \lambda \rangle$ be a labeled net system, $w \in \mathcal{L}_S$ be a secret word such that $|w_o| = \rho$ and $w_o = \Pr(w) = e_o^1 \cdots e_o^\rho$. Let B be a sufficiently large integer, then there exists at least one $w' \in \mathcal{L}(\mathcal{G}, \mathcal{M}_0)$ such that $\Pr(w') = \Pr(w)$ *if and only if* the set of constraints reported in Fig. 3 admits a solution

- $\boldsymbol{\sigma}_j^i \in \mathbb{N}^n$, with $j = 1, \dots, J_i$. $i = 1, \dots, \rho$, and where
  - $J_i = |w_{uo}^i|$, if $|w_{uo}^i| \neq 0$
  - $J_i = 1$, otherwise
- $\mu_1, \dots, \mu_M \in \{0, 1\}$
- $\gamma_{ij} \in \{0, 1\}$, with $i = 1, \dots \rho$ and $j = 1, \dots, \text{card}\left(T^{e_o^i}\right)$
- $\delta^i \in \{0, 1\}$ with $i = 1, \dots, \rho$

Proof. First of all it should be noticed that, since μ_i are binary decision variables, the constraints (2a) and (2b) permit to *select* only one initial marking among the ones that belong to $\mathcal{M}_0$. Similarly, for each observable event e_o^i in the secret word w, the constraints (2c) and (2d) enable the firing of only one transition $t^j \in T^{e_o^i}$.

Let us first prove *sufficiency*. Exploiting Assumption 1, from Theorem 1 it follows that if the integer vectors $\boldsymbol{\sigma}_j^i$ satisfy the inequalities (2e), then their sum $\sum_{i=1}^{\rho} \sum_{j=1}^{|w_{uo}^i|} \boldsymbol{\sigma}_j^i$ represents the firing count vector of a sequence in the set $Pr^{-1}(w_o)$, called *explanation* of w_o, since its firing is required to "explain" the firing of w_o by means of a proper sequence of unobservable transitions.

Moreover, constraints (2f), (2g) and (2h) imply that either an unobservable explanation is made by the same unobservable events of the secret but with a different order, or that these unobservable events are different from the one of the secret. It follows that, starting from one of the initial markings belonging to $\mathcal{M}_0$, there exists at least one enabled sequence σ' such that $\lambda(\sigma') \neq w$ and $\Pr(\lambda(\sigma')) = \Pr(w)$.

In order to prove *necessity* let us assume, *ad absurdum*, that the set of constraints (2) does not admit a solution, while there exists a sequence $\tilde{\sigma}$ such that

$$\tilde{\sigma} = \tilde{\sigma}_{uo}^1 t_o^1 \tilde{\sigma}_{uo}^2 t_o^2 \cdots \tilde{\sigma}_{uo}^\rho t_o^\rho,$$

where $\lambda(\tilde{\sigma}) = \tilde{w} \in \mathcal{L}(\mathcal{G}, \mathcal{M}_0)$, with $\tilde{w} \neq w$ and $\Pr(\tilde{w}) = \Pr(w)$. If this is the case, exploiting again Assumption 1, it is possible to build a solution of (2) by letting (see also Reveliotis (2005))

The partition between observable and unobservable events in a labeled P/T net induces a similar partition on the set of transitions T. Therefore $T = T_o \cup T_{uo}$, where

$$T_o = \{t \in T \mid \lambda(t) \in E_o\} ,$$
$$T_{uo} = \{t \in T \mid \lambda(t) \in E_{uo}\} ,$$

and obviously $T_o \cap T_{uo} = \emptyset$. Given a firing count vector $\boldsymbol{\sigma} \in \mathbb{N}^n$, in this paper we are interested to distinguish among the firings of the observable and of the unobservable transitions. For this reason the following notation is adopted:

$$\boldsymbol{\sigma}_{|T_o} \in \mathbb{N}^n , \text{ with } \boldsymbol{\sigma}_{|T_o}(t) = \begin{cases} \boldsymbol{\sigma}(t) & \text{if } t \in T_o \\ 0 & \text{if } t \notin T_o \end{cases}$$

$$\boldsymbol{\sigma}_{|T_{uo}} \in \mathbb{N}^n , \text{ with } \boldsymbol{\sigma}_{|T_{uo}}(t) = \begin{cases} \boldsymbol{\sigma}(t) & \text{if } t \in T_{uo} \\ 0 & \text{if } t \notin T_{uo} \end{cases}$$

Hence, given a firing count vector $\boldsymbol{\sigma}$, it is

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{|T_o} + \boldsymbol{\sigma}_{|T_{uo}} .$$

Given a net N, we now introduce the definition of *unobservable subnet*, which is the subnet induced on N by the unobservable transition subset T_{uo} (see also Cabasino et al. (2012); Tong et al. (2016)).

Definition 2. (Unobservable subnet). Given the net $N = (P, T, \mathbf{Pre}, \mathbf{Post})$, the correspondent *unobservable subnet* N_L is the net induced on N by the set of transition T_{uo}, i.e.

$$N_{uo} = (P, T_{uo}, \mathbf{Pre}_{uo}, \mathbf{Post}_{uo}) ,$$

where $\mathbf{Pre}_{uo}$ and $\mathbf{Post}_{uo}$ are the restriction of $\mathbf{Pre}$ and $\mathbf{Post}$ to $P \times T_{uo}$; the incidence matrix of N_{uo} is given by

$$\boldsymbol{C}_{uo} = \mathbf{Post}_{uo} - \mathbf{Pre}_{uo} .$$

▲

In Fig. 1 a labeled net is shown (for each transition the correspondent event is also specified), where the set of events is $E = \{a, b\}$, with $E_o = \{b\}$ and $E_{uo} = \{a\}$. Given the two sets E_o and E_{uo}, the correspondent induced partition on T is

$$T_o = \{t_2, t_3, t_5, t_7, t_8, t_{10}, t_{11}\} ,$$
$$T_{uo} = \{t_1, t_4, t_6, t_9\} .$$

Moreover, the subnet induced by T_{uo} is the one reported in Fig. 2.

3. LANGUAGE-BASED OPACITY IN LABELED PETRI NET SYSTEMS

In this section we recall the definition of LBO (see also Reiter and Rubin (1998); Dubreil et al. (2010); Wu and Lafortune (2013); Tong et al. (2016)). We make specific reference to the definition given in Wu and Lafortune (2013), which was originally given in the context of finite state automata; in particular we recast that definition to the Petri net framework introduced in Section 2. Compared to the definition of LBO given in Tong et al. (2016), in the one proposed here the secret refers to a sequence of events rather than a sequence of transitions, coherently to what is usually proposed in the context of automata (see also Lin (2011)).

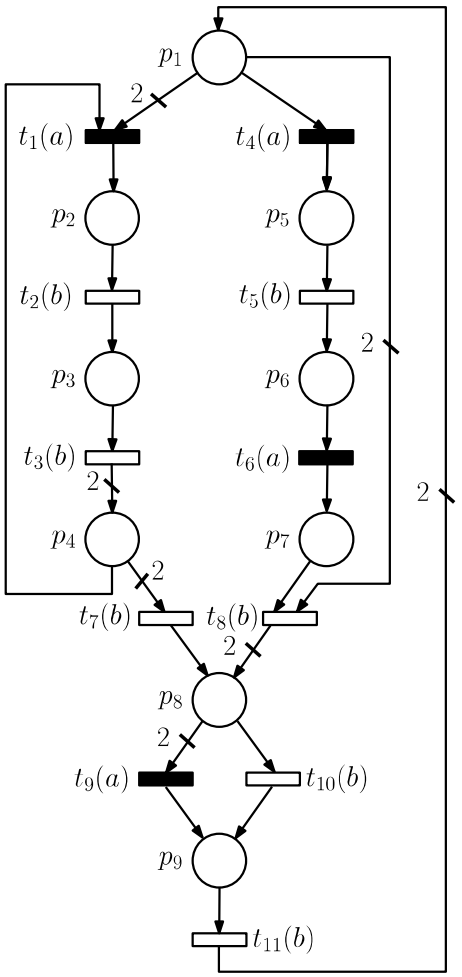


Fig. 1. Example of a labeled Petri net whose transitions are associated to both observable and unobservable events. For each transition the correspondent event is also specified.

Definition 3. (Language-based opacity). Given a labeled net system $\mathcal{G} = \langle N, \mathcal{M}_0, \lambda \rangle$, the correspondent natural projection function $\Pr(\cdot)$ and a *secret language* $\mathcal{L}_s \subseteq \mathcal{L}(\mathcal{G}, \mathcal{M}_0)$, $\mathcal{G}$ is *language-based opaque* (LBO) if for every word $w \in \mathcal{L}_s$, there exists another word $w' \in \mathcal{L}(\mathcal{G}, \mathcal{M}_0) \setminus \mathcal{L}_s$ such that $\Pr(w) = \Pr(w')$. Equivalently

$$\mathcal{L}_S \subseteq \Pr^{-1}\left[\Pr\left(\mathcal{L}(\mathcal{G}, \mathcal{M}_0) \setminus \mathcal{L}_s\right)\right] .$$

◇

Given the above definition the following lemma immediately follows, which gives a sufficient condition for LBO.

Lemma 1. Given a labeled net system $\mathcal{G} = \langle N, \mathcal{M}_0, \lambda \rangle$, the correspondent natural projection function $\Pr(\cdot)$ and a *secret language* $\mathcal{L}_s \subseteq \mathcal{L}$, if there exist at least one $\hat{w} \in \mathcal{L}_s$ such that $\hat{w} = \hat{w}' e_{uo}$, with $\hat{w}' \in E^*$ and $e_{uo} \in E_{uo}$, then $\mathcal{G}$ is LBO.

Proof. The proof readily follows from the Definition 3, since for the secret word $\hat{w}$ it is $\Pr(\hat{w}) = \Pr(\hat{w}')$.

2. NOTATION

In what follows different products will be considered. We will denote with "$\cdot$" the standard matrix multiplication, while "$*$" will denote the product between a scalar and a matrix, and "$\circ$" the Hadamard product.

In this paper we deal with labeled Petri net systems. Let first briefly introduce the concept of *Place/Transition* (P/T) net. A P/T net is a 4-tuple $N = (P, T, \mathbf{Pre}, \mathbf{Post})$, where P is a set of m places (represented by circles) and T is a set of n transitions (represented by empty boxes). $\mathbf{Pre} : P \times T \mapsto \mathbb{N}$ and $\mathbf{Post} : P \times T \mapsto \mathbb{N}$) are the *pre-* and *post- incidence* matrices, respectively. $\mathbf{Pre}(p,t) = w$ ($\mathbf{Post}(p,t) = w$) means that there is an arc with weight w from p to t (from t to p); $\boldsymbol{C} = \mathbf{Post} - \mathbf{Pre}$ is the incidence matrix.

A *marking* is a function $\boldsymbol{m} : P \mapsto \mathbb{N}$ that assigns to each place of a net a nonnegative integer number of tokens, drawn as black dots. The marking of a net is usually represented by a vector $\boldsymbol{m} \in \mathbb{N}^m$.

A *net system* $S = \langle N, \boldsymbol{m}_0 \rangle$ is a net N with an initial marking $\boldsymbol{m}_0$. A transition t is enabled at $\boldsymbol{m}$ if and only if $\boldsymbol{m} \geq \mathbf{Pre}(\cdot, t)$, and this is denoted as $\boldsymbol{m}[t\rangle$. An enabled transition t may fire, bringing the system to the marking

$$\boldsymbol{m}' = \boldsymbol{m} + \boldsymbol{C}(\cdot, t)\,,$$

and this is denoted as $\boldsymbol{m}[t\rangle\boldsymbol{m}'$.

A *firing sequence* enabled from $\boldsymbol{m}$ is a sequence of transitions $\sigma = t_1 t_2 \ldots t_k$ such that $\boldsymbol{m}[t_1\rangle\boldsymbol{m}_1[t_2\rangle\boldsymbol{m}_2 \ldots [t_k\rangle\boldsymbol{m}_k$, and this is denoted as $\boldsymbol{m}[\sigma\rangle\boldsymbol{m}_k$. The notations $\boldsymbol{m}[\sigma\rangle$ denotes an enabled sequence under a marking $\boldsymbol{m}$. Furthermore, $t_i \in \sigma$ denotes that the transition t_i belongs to the sequence σ, and the length of σ is denoted with $|\sigma|$.

A marking $\boldsymbol{m}'$ is said to be *reachable* from $\boldsymbol{m}_0$ if and only if there exists a sequence σ such that $\boldsymbol{m}_0[\sigma\rangle\boldsymbol{m}'$. The set $R(N, \boldsymbol{m}_0)$ contains all the reachable markings of the net system $\mathcal{S} = \langle N, \boldsymbol{m}_0 \rangle$.

The function $\boldsymbol{\sigma} : T \mapsto \mathbb{N}$, where $\boldsymbol{\sigma}(t)$ represents the number of occurrences of t in σ, is called *firing count vector* of the firing sequence σ. As it has been done for the marking of a net, the firing count vector is usually represented by a vector $\boldsymbol{\sigma} \in \mathbb{N}^n$. The notation $\boldsymbol{\sigma} = \pi(\sigma)$ is used to denote that $\boldsymbol{\sigma}$ is the firing count vector corresponding to σ.

If $\boldsymbol{m}_0[\sigma\rangle\boldsymbol{m}$, then it is well known that the so-called *state equation* of the net system holds

$$\boldsymbol{m} = \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{\sigma}\,. \tag{1}$$

Furthermore, for acyclic nets[1], the following result holds.

Theorem 1. ((Murata, 1989)). Consider a net system $\langle N, \boldsymbol{m}_0 \rangle$ and let N be an acyclic net. It results $\boldsymbol{m} \in R(N, \boldsymbol{m}_0)$ *if and only if* there exists a nonnegative integer solution $\boldsymbol{\sigma}$ satisfying (1). ■

[1] A net is acyclic if it does not include a direct circuit.

In Wu and Lafortune (2013) it is assumed that an external intruder aiming at discovering the occurrence of a secret sequence, although it knows the structure of the system – i.e., it knows the net topology – it does not have a precise information about the initial state. For this reason, we will consider net systems where there is an uncertainty on the initial marking, that is we assume that $\boldsymbol{m}_0$ belongs to any of the markings in the set $\mathcal{M}_0 \subseteq \mathbb{N}^n$. Hence, in the following we will consider net systems in the form $\mathcal{S} = \langle N, \mathcal{M}_0 \rangle$, where $\mathcal{M}_0 = \{\boldsymbol{m}_{0_1}, \ldots, \boldsymbol{m}_{0_M}\}$, with $M = \mathrm{card}(\mathcal{M}_0)$.

With the following notion of labeled P/T net, it is possible to associate events to the transitions; in particular, the same event can be associated to more than one transitions.

Definition 1. (Labeled P/T net system). A *labeled* P/T net system is the three-tuple $\mathcal{G} = \langle N, \mathcal{M}_0, \lambda \rangle$, where N is a standard P/T net, $\mathcal{M}_0$ is the initial marking set, and

$$\lambda : T \mapsto E\,,$$

is the *labeling function* which assigns to each transition $t \in T$ an event from the set E. ◇

In the following we will denote with

$$T^e = \{t \in T \mid \lambda(t) = e\,, \text{with } e \in E\}\,,$$

the set of transitions associated with the same event e. We denote with $\mathrm{card}(T^e)$ the cardinality of set T^e, and with w the word of events associated to a sequence σ, i.e. $w = \lambda(\sigma)$, assuming the usual extension of the labeling function to sequences of transitions and events[2], that is by considering $\lambda : T^* \mapsto E^*$. Given a word w we will denote with $|w|$ its length, and with $|w|_e$ the number of occurrences of the event e in w. Moreover, the notation $w[i]$ will denote the i-th event in the word w. For example, if $E = \{a, b, c\}$ and $w = bbcac$, then $|w| = 5$, $|w|_b = |w_c| = 2$ and $|w|_a = 1$, while $w[2] = b$ and $w[4] = a$.

The language of the labeled Petri net system $\mathcal{G}$ can be defined as

$$\mathcal{L}(\mathcal{G}, \mathcal{M}_0) = \{w \in E^* \mid w = \lambda(\sigma) \text{ with } \boldsymbol{m}_0[\sigma\rangle \text{ and } \boldsymbol{m}_0 \in \mathcal{M}_0\}\,.$$

When dealing with opacity, the set E is partitioned into the two disjoint sets of *observable* (whose correspondent transitions are represented by empty boxes) and *unobservable* events (whose correspondent transitions are represented by filled boxes), named respectively E_o and E_{uo}.

Given a word $w \in E^*$, its observation is the output of the *natural projection function* $\mathrm{Pr} : E^* \mapsto E_o^*$, which is recursively defined as

$$\mathrm{Pr}(we) = \mathrm{Pr}(w)\,\mathrm{Pr}(e)\,,$$

with $w \in E^*$ and $e \in E$; moreover, $\mathrm{Pr}(e) = e$ if $e \in E_o$, while $\mathrm{Pr}(e) = \varepsilon$ if $e \in E_{uo}$.

[2] The superscript S^* denotes the Kleene closure of the given set S (Cassandras and Lafortune, 1999, p. 55).

Preprints of WODES 2018
May 30 - June 1, 2018
Sorrento Coast, Italy

An algebraic characterization of language-based opacity in labeled Petri nets

F. Basile * **G. De Tommasi** **

* *Department of Electrical and Information Engineering and Applied Mathematics, University of Salerno, Fisciano 84084, Italy (email:* fbasile@unisa.it*)*
** *Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II, Via Claudio 21, 80125 Napoli, Italy (email:* detommas@unina.it*)*

Abstract: Opacity is a property of discrete event systems (DES) that is related to the possibility of hiding a secret to external observers (the *intruders*). The secret can be either a system state, or a sequence of events executed by the system itself. When the latter type of secrets is considered, the opacity property is referred to as *language-based opacity* (LBO). This paper deals with LBO when the DES is modeled by a labeled Petri net. One necessary and sufficient condition to check LBO by solving Integer Linear Programming problems is given; such a condition exploits the algebraic representation of Petri nets. A sufficient condition is then derived, which is less demanding from the computational point of view. The effectiveness of the proposed approach is shown by means of examples.

Keywords: Opacity, Petri nets, DES, ILP problems

1. INTRODUCTION

In the context of Petri nets, the authors of Tong et al. (2016, 2017) proposed approaches based on finite-time automata to check both language-based and current-state opacity for bounded labeled Petri net systems. In both cases, the check is performed on a graph derived form the Petri net system. In particular, language-based opacity (LBO) is checked by using a *verifier* automaton, while in order to assess current-state opacity an extended version of the reachability graph is proposed.

The main contribution of this work are two conditions that permit to verify if a labeled net system is LBO with respect to a given finite secret language, without requiring that the system is bounded. First a necessary and sufficient condition will be given, which is based on the solution of Integer Linear Programming (ILP) optimization problems. A sufficient condition is then derived by relaxing the original optimization problem. To the best of the authors' knowledge this is the first work where the algebraic representation of a Petri net is exploited to check LBO (for a review on opacity see also Jacob et al. (2016)). The algebraic representation enables the use of a standard tool, i.e. the mathematical programming, to check LBO avoiding the preliminary computation of graph structures, that may have a not negligible size, since the graph can be equal to the reachability graph.

The main technique exploited in the paper is derived from the exact characterization of the reachability space of a given Petri net through a finite set of linear inequalities when the net markings are reached by sequences whose length is not greater than a given value K. Moreover, the cardinality of this set is of polynomial size with respect to the size of the considered net, where the latter is defined by the number of its places and transitions, and K is bounded uniformly by a polynomial function of the net size (Reveliotis, 2005).

However, the key problem addressed in this paper requires to check if there exists a word having the same projection over the set of observable transitions with respect to a given one of finite length, that belongs to the secret language. Hence, it is necessary to characterize only the portion of the reachability space that enables the occurrence of word with finite length; it follows that the number of required linear inequalities results to be polynomial in size with respect to the length of the secret word.

The paper is structured as follows: in Section 2 the considered Petri nets framework is introduced. The definition of language-based opacity is given in Section 3. The main contributions are given in Section 4, that is the two conditions to check language-based opacity by solving ILP problems. In Section 5 some examples are presented in order to show the effectiveness of the proposed approach, while conclusive remarks and possible future directions are finally discussed in Section 6.

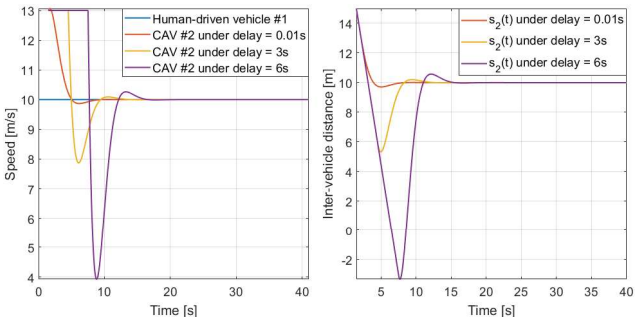


Fig. 12. The speed trajectories and the corresponding inter-vehicle distance under different communication delays.

framework are relatively independent, only the corresponding modules need modification.

5. CONCLUDING REMARKS AND FUTURE WORK

This paper proposed a discrete-event and hybrid simulation framework based on SimEvents for ITS analysis. SimEvents has become a valuable tool for discrete-event and hybrid simulations that fits the goal that the traffic model should be both time-driven and event-driven. The benefits of the simulation framework demonstrated in this paper include (1) abstraction of continuous-time components based on discrete event systems, (2) a modular architecture that allows different system configurations, (3) a framework that can be easily adapted to different traffic scenarios, (4) direct comparison among different simulations by introducing an event, (5) expandable functionality by incorporating functions from other MATLAB toolboxes, (6) scalability by simply adding more queues, servers, and storages. In addition, MATLAB provides users with full access to model details and flexibility to manipulate the model elements, and real-time state displays and performance reports as well.

Ongoing research includes the incorporation of the Dedicated Short Range Communication (DSRC) protocols as it is the key technology for V2V safety communications. Furthermore, more elaborate and diversified non-CAV models are required so as to study the interactions between CAVs and non-CAVs.

REFERENCES

Benekohal, R. and Treiterer, J. (1988). Carsim: Car-following model for simulation of traffic in normal and stop-and-go conditions. *Transportation Research Record*, (1194).

Bettisworth, C., Burt, M., Chachich, A., Harrington, R., Hassol, J., Kim, A., Lamoureux, K., LaFrance-Linden, D., Maloney, C., Perlman, D., Ritter, G., Sloan, S.M., and Wallischeck, E. (2015). Status of the Dedicated Short-Range Communications technology and applications: Report to Congress. Technical report.

Cameron, G.D. and Duncan, G.I. (1996). Paramics-parallel microscopic simulation of road traffic. *The Journal of Supercomputing*, 10(1), 25–53.

Cassandras, C.G. and Lafortune, S. (2009). *Introduction to Discrete Event Systems*. Springer Science & Business Media.

Clune, M.I., Mosterman, P.J., and Cassandras, C.G. (2006). Discrete event and hybrid system simulation with SimEvents. In *Proceedings of the 8th International Workshop on Discrete Event Systems*, 386–387.

Fellendorf, M. (1994). Vissim: A microscopic simulation tool to evaluate actuated signal control including bus priority. In *64th Institute of Transportation Engineers Annual Meeting*, 1–9. Springer.

Gilbert, E.G. (1976). Vehicle cruise: Improved fuel economy by periodic control. *Automatica*, 12(2), 159–166.

Hellström, E., Åslund, J., and Nielsen, L. (2010). Design of an efficient algorithm for fuel-optimal look-ahead control. *Control Engineering Practice*, 18(11), 1318–1327.

Hooker, J. (1988). Optimal driving for single-vehicle fuel economy. *Transportation Research Part A: General*, 22(3), 183–201.

Kato, S., Tsugawa, S., Tokuda, K., Matsui, T., and Fujii, H. (2002). Vehicle control algorithms for cooperative driving with automated vehicles and intervehicle communications. *IEEE Transactions on Intelligent Transportation Systems*, 3(3), 155–161.

Krajzewicz, D., Hertkorn, G., Rössel, C., and Wagner, P. (2002). Sumo (simulation of urban mobility)-an open-source traffic simulation. In *Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM20002)*, 183–187.

Lee, J. and Park, B. (2012). Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment. *IEEE Transactions on Intelligent Transportation Systems*, 13(1), 81–90.

Li, S.E., Peng, H., Li, K., and Wang, J. (2012). Minimum fuel control strategy in automated car-following scenarios. *IEEE Transactions on Vehicular Technology*, 61(3), 998–1007.

Li, W., Mani, R., and Mosterman, P.J. (2016). Extensible discrete-event simulation framework in SimEvents. In *Proceedings of the 2016 Winter Simulation Conference*, 943–954. IEEE Press.

Malikopoulos, A.A., Cassandras, C.G., and Zhang, Y. (2017). A decentralized energy-optimal control framework for connected automated vehicles at signal-free intersections. *Automatica*. (to appear).

Shladover, S.E., Desoer, C.A., Hedrick, J.K., Tomizuka, M., Walrand, J., Zhang, W.B., McMahon, D.H., Peng, H., Sheikholeslam, S., and McKeown, N. (1991). Automated vehicle control developments in the path program. *IEEE Transactions on Vehicular Technology*, 40(1), 114–130.

Versteegt, E., Klunder, G., and Arem, B.v. (2009). Modelling cooperative roadside and in-vehicle intelligent transport systems using the its modeller. In *12th World Congress on Intelligent Transport Systems 2005, 6 November 2005 through 10 November 2005, San Francisco, CA, 1558-1567*.

Zhang, Y. and Cassandras, C.G. (2018). The penetration effect of connected automated vehicles in urban traffic: an energy impact study. ArXiv: 1803.05577.

Zhang, Y., Cassandras, C.G., Li, W., and Mosterman, P.J. (2017a). A SimEvents model for hybrid traffic simulation. In *Proceedings of the 2017 Winter Simulation Conference*, 1455–1466. IEEE Press.

Zhang, Y., Cassandras, C.G., and Malikopoulos, A.A. (2017b). Optimal control of connected automated vehicles at urban traffic intersections: A feasibility enforcement analysis. In *Proceedings of the 2017 American Control Conference*, 3548–3553.

Zhang, Y., Malikopoulos, A.A., and Cassandras, C.G. (2016). Optimal control and coordination of connected and automated vehicles at urban traffic intersections. In *Proceedings of the 2016 American Control Conference*, 6227–6232.

Zhang, Y., Malikopoulos, A.A., and Cassandras, C.G. (2017c). Decentralized optimal control for connected automated vehicles at intersections including left and right turns. In *56th IEEE Conference on Decision and Control*, 4428–4433.

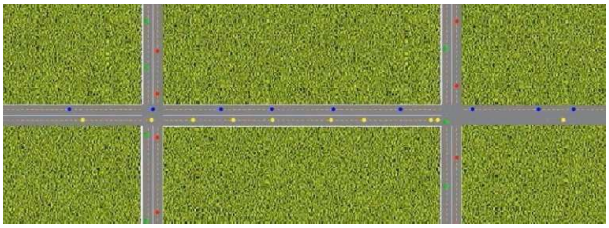


Fig. 9. Optimal control of CAVs crossing two adjacent urban intersections.

from *Events* blocks, for instance, a storm event. Note that the optimal control in Zhang et al. (2016) actually remains unchanged until an event occurs. As the storm may reduce the friction of the road surface, vehicles are forced to decelerate under this scenario and re-calculate the speed profiles. The speed trajectories of the first 3 CAVs are shown in Fig. 10, where the dashed lines and solid lines represent the projected speed trajectories before and after the event (storm) occurs, respectively. Observe that the re-calculation of the speed profiles is only triggered by the storm event. Otherwise, vehicles would proceed according to the original optimal control profiles (dashed lines).

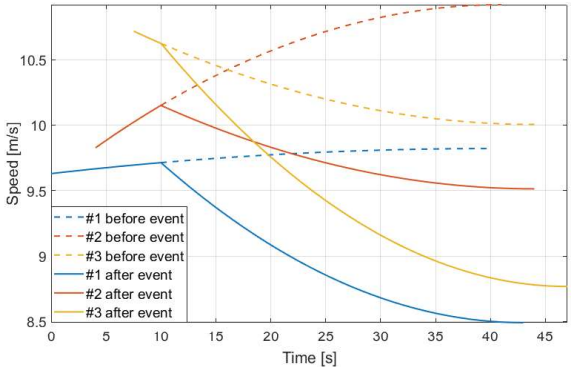


Fig. 10. The speed trajectories of the first 3 CAVs before and after a storm event occurs.

4.2 A Scenario with Only Non-CAVs

To demonstrate the efficiency of new control algorithms, comparisons are usually required. Therefore, it is of great importance that a traffic simulation framework can be adapted to different traffic scenarios without much effort, so that users can make comparisons among different approaches. Here, a baseline scenario is built and tested on the traffic simulation platform where the non-CAVs are controlled by the fixed-cycle traffic lights. In this paper, simple control policies are assumed for the non-CAVs, that is, a non-CAV (1) keeps cruising unless an event occurs that would affect its behavior; (2) decelerates when it approaches a red light; (3) accelerates to the desired speed when the red light turns green.

Demonstration examples A snapshot of the demonstration example for the non-CAVs crossing a signalized intersection under traffic light control is shown in Fig. 11, where both the green and red phases last for 30 seconds. In Fig. 11, queues can be observed that gradually formed in front of the red lights. It was shown in Zhang et al. (2017b) that compared with the baseline scenario, the optimal control of CAVs can achieve 42% improvement in reducing fuel consumption and 37% improvement in reducing travel delay.

Fig. 11. A snapshot of the scenario under traffic light control.

4.3 A Mixed-Traffic Scenario

As challenges remain before massive deployment of fully autonomous vehicles, the mixed-traffic scenario where both CAVs and non-CAVs travel on the roads must be considered. To model this scenario on the proposed traffic simulation platform, different control algorithms are implemented for CAVs and non-CAVs, respectively. For non-CAVs, simple control policies are assumed, that is, a non-CAV keeps cruising if no event occurs that would affect its driving behavior. For CAVs, a two-mode optimal control framework, introduced in Zhang and Cassandras (2018) is applied to minimize fuel consumption white adaptively maintaining a safe inter-vehicle distance if it is constrained by a preceding non-CAV.

An important feature of this framework is the potential to model and simulate communication protocols in order to study the effects of delay in V2V and V2I communications on safety. To cooperate with the control algorithms employed by the CAVs, low packet delay/loss/error is necessary for maintaining safety.

Demonstration examples The following example explores the influence of communication delay on inter-vehicle safety, where a non-CAV #1 is assumed to cruise at its initial speed and CAV #2 enters the CZ immediately after #1 on the same lane. If the inter-vehicle distance between vehicles #2 and #1, denoted as $s_2(t)$, falls below the minimum safe following distance $\delta = 10$m, CAV #2 simply forgoes the optimal control and adaptively follows the non-CAV #1, that is, maintaining the minimum safe following distance and the same speed as #1. Varying the service time of the server, which is used for simulating the communication delay between the CAVs and the coordinator, allows easy experimentation with the influence of large communication delay on the implementation of control algorithms and inter-vehicle safety.

The speed trajectories under different communication delays and the corresponding inter-vehicle distance between vehicles #2 and #1 are shown in Fig. 12. When the communication delay is low, that is, delay = 0.01s, CAV #2 is able to make adjustments in time so that the inter-vehicle distance $s_2(t)$ (red curve) does not fall too much below $\delta = 10$m; after a short period, CAV #2 starts to follow the non-CAV #1 while maintaining a distance of δ with the non-CAV #1. When the communication delay is high, that is, delay = 3s, the inter-vehicle distance $s_2(t)$ falls to 5.5m, which is undesired in terms of maintaining safety; while the communication delay increases to 6s, the inter-vehicle distance falls below 0, which indicates an accident. Using the simulation setting, the operation of the communication protocols can be easily examined and if a crash occurs, the protocols may need to be re-designed.

As shown in the previous demonstrations, the framework can be easily adapted to different scenarios. Since the modules in the

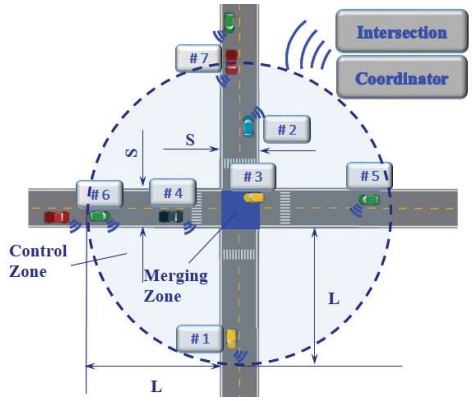


Fig. 5. Connected Automated Vehicles (CAVs) crossing an urban intersection.

a *Control Zone* (CZ) and a coordinator that can communicate with the CAVs traveling within the CZ. A M/M/1 queueing system (Cassandras and Lafortune (2009)) following a first-in-first-out order is assumed for CAVs that have entered the CZ.

The decentralized optimal control framework, introduced in Zhang et al. (2016), is used for optimally controlling CAVs crossing a signal-free intersection with the objective of minimizing energy consumption inside the CZ, subject to a throughput maximization requirement formulated in Malikopoulos et al. (2017). The rear-end safety can be ensured through an appropriately designed Feasibility Enforcement Zone (FEZ) that precedes the CZ, as discussed in Zhang et al. (2017b).

Demonstration examples **A group of CAVs crossing a single urban intersection** is considered, where the length of the CZ is $L = 400$m and the length of the MZ is $S = 30$m. For each direction, only one lane is considered. The minimum safe inter-vehicle distance is set to $\delta = 10$m. The vehicle arrivals are assumed to be given by a Poisson process and the initial speeds are uniformly distributed over $[8, 12]$m/s.

In the current framework, simple information exchange is assumed between vehicles and the coordinator. Every time a CAV enters the CZ, it sends information to the coordinator indicating its arrival. After a certain period of communication delay, which is modeled using servers, the coordinator sends relevant information back and based on that, the CAV can make decisions regarding the remaining trip.

For evaluation purposes, the position of the CAV must be continuously monitored and tracked. This represents the continuous time-driven component of the simulation framework. Combined with the discrete event-driven component, such as vehicle arrivals, the two together form the *hybrid* nature of the simulation framework.

A snapshot of the demonstration example is shown in Fig. 6, where the color represents the direction that the vehicle comes from. Note that the proposed simulation framework is capable of generating state displays, which can easily be achieved by incorporating functions from other MATLAB toolboxes. For instance, the optimal control output and the corresponding speed trajectories are shown in Fig. 7.

This scenario can be easily modified, for instance, by **including left the right turns** (Fig. 8). The solution to account for left and right turns under hard safety constraints is provided in Zhang et al. (2017c). Note that the proposed simulation framework is also equipped with performance reports. Fig. 8 shows how average fuel consumption and average travel time can be observed in real time.

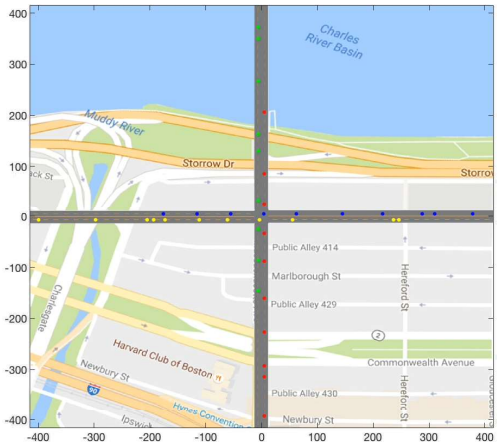


Fig. 6. Optimal control of CAVs crossing an urban intersection.

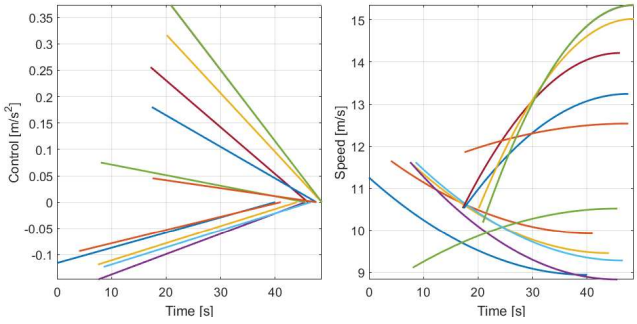


Fig. 7. Control and speed trajectories of the first 10 CAVs under decentralized optimal control framework.

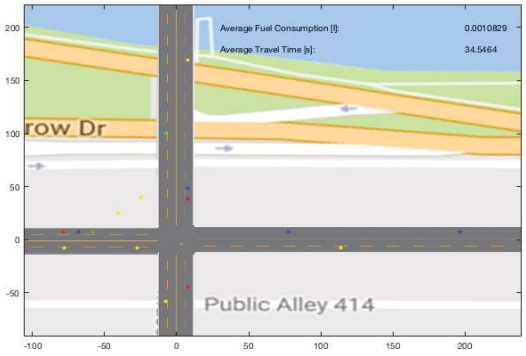


Fig. 8. Optimal control of CAVs including left and right turns.

The traffic simulation model is inherently scalable. For instance, the single intersection scenario can be easily extended to **a multi-intersection scenario** by adding more queues, servers, the MATLAB Discrete Event Systems or a combination of them. A snapshot of the demonstration example for CAVs crossing two adjacent urban intersections is shown in Fig. 9. Note that the two intersections can be coupled in different ways, which mainly depends on the distance between the two intersections D. In Fig. 9, the distance between the two intersections is set to $D = L = 400$m, which indicates that once the CAV exits the upstream MZ, it immediately enters the downstream CZ.

To explore the event-driven feature of the transportation systems, a scenario is created by **randomly generating an event**

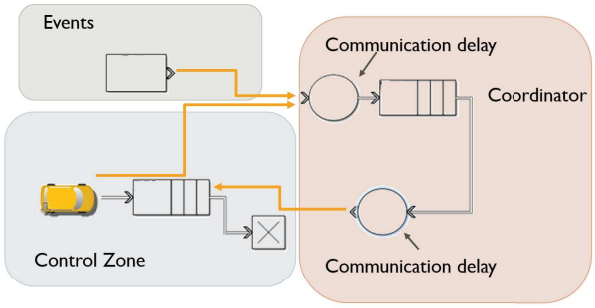


Fig. 1. Architecture of the hybrid traffic simulation framework.

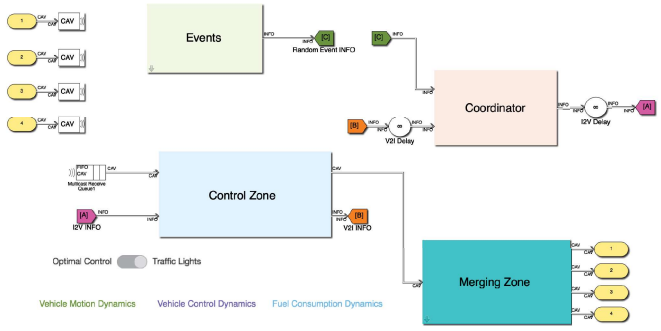


Fig. 2. Simulink® model of an intersection with four road segments (input/outputs).

The output of *Events* into the coordinator indicates that the coordinator is aware of all the event-based information by means of sensing and communication. Once a certain event occurs the coordinator will broadcast or send the information to the vehicles that might be affected, so that the vehicles can make appropriate decisions.

The *continuous* part in this framework includes vehicle motion dynamics, control dynamics, and fuel consumption dynamics that should be tracked continuously. For the *discrete-event* part, events are considered as they can affect vehicle behavior.

3. IMPLEMENTATION

The proposed hybrid traffic simulation framework is built based on MATLAB and Simulink that include various programming paradigms (Li et al. (2016)). The incorporation of SimEvents offers tools to work with discrete event components. The various programming options offer users a platform for rapid prototyping that is widely used in the automotive industry. The paradigms used in the proposed model include *Entity Flow*, *Graphical Programming*, and *Textual Programming*. The model structure of a single intersection is shown in Fig. 2.

3.1 Entity Flow

Entities are the discrete items of interest carrying a rich set of attributes, which can pass through a network of queues and servers during the discrete-event simulations. In the transportation modeling context, an entity can represent a vehicle, whose attributes may include ID, acceleration, speed, position, lane, destination, and so on (see the example shown in the yellow rectangle of Fig. 3).

3.2 Graphical Programming

The graphical programming paradigm enables users to work with discrete event components, whereby users can specify various functions associated with events such as entity entry and exit. These functions are event-driven actions, that is, they can only be triggered by a different class of events. For instance, in Fig. 3, a series of functions are defined in the red rectangle that can only be executed when the CAV is generated. These functions specify how the attributes, for instance, speed, of vehicles are initialized when they are generated.

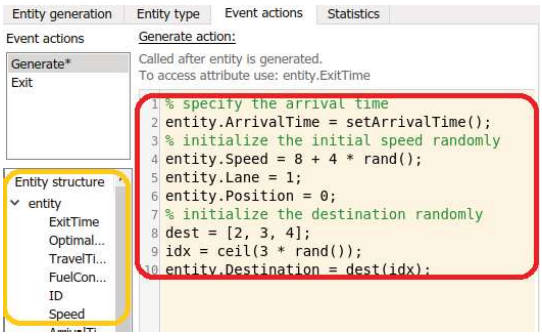


Fig. 3. Customized event actions of CAV generator block.

3.3 Textual Programming

The MATLAB Discrete Event System provides maximal flexibility as it offers users the capability to author an event-driven entity-flow system arbitrarily using object-oriented programming in MATLAB, whose functionality is expandable by incorporating functions from other MATLAB toolboxes.

```
% CAV arrives at the control zone
function [entity, events] = CAVEntryImpl(obj, storage, entity, ~)...
% CAV generates an info packet and sends it to the coordinator
function [entity, events] = INFOGenerateImpl(obj, ~, entity, tag)...
% CAV receives info from the coordinator and (re-)computes the
% control
function [entity, events] = INFOEntryImpl(obj, storage, entity, ~)...

% persistently monitor and track the status of CAVs by setting
% timers and iterators for evaluation purposes
function [entity, events] = CAVTimerImpl(obj, ~, entity, tag)...
function [entity, events, next] = CAVIterateImpl(obj, storage, ...
        entity, tag, ~)...
```

Fig. 4. The MATLAB® Discrete Event System for Control Zone (partial codes).

For instance, the control zone shown in Fig. 2 was designed using a MATLAB Discrete Event System in the proposed traffic model. The program specifies the properties of the control zone as well as the definition of different storages that contain user-defined entities, that is, CAVs and information packets (INFOs). In addition, different event actions are defined through methods as shown in Fig. 4.

4. DEMONSTRATIONS

The effectiveness of the proposed hybrid framework has been demonstrated through simulation under different traffic scenarios: 1) a scenario with only CAVs, 2) a scenario with only non-CAVs, 3) a mixed-traffic scenario where CAVs and non-CAVs co-exist. For each scenario, the control methodology is introduced first and then the demonstration examples are presented.

4.1 A Scenario with Only CAVs

In the first scenario, only CAVs are being considered. As shown in Fig. 5, the region at the center of each intersection, called *Merging Zone* (MZ) is the area of potential lateral CAV collision, which is taken to be a square. Each intersection has

SIM (Benekohal and Treiterer (1988)) and SUMO (Krajzewicz et al. (2002)), all of which offer a wide range of methods to design and evaluate traffic systems. As CAVs make use of more sophisticated and increasingly efficient control algorithms that heavily rely on sensing the transportation environment, we need platforms which are able to consider a large number of different traffic scenarios and encompass all different aspects of ITS operation. An example of such a platform is PreScan, which accommodates CAVs and Advanced Driver Assistance Systems (ADAS) based on sensor simulation and flexible scenario definition. The tool ITS Modeller proposed in Versteegt et al. (2009) complements PreScan in terms of evaluation at a traffic network level.

One common feature of the aforementioned traffic simulation platforms is the integration of MATLAB and Simulink via an interface that allows a user to design ICT methods and control algorithms. Examples can be found in Zhang et al. (2016), where a decentralized optimal control algorithm is implemented using MATLAB and applied to each vehicle, with the resulting vehicle behavior visualized and evaluated through VISSIM. This illustrates the powerful capabilities of MATLAB and Simulink as a test bed for ICT approaches and control algorithms. In some cases, a discrete-event simulation model cannot only capture event-driven behavior, but also abstract continuous-time components through event-driven components. As SimEvents provides users with various paradigms for building a discrete-event simulation model, we take advantage of the Discrete Event System (DES) simulation framework introduced in SimEvents®.

Earlier work in Zhang et al. (2017a) introduced a new traffic simulation framework based on SimEvents in conjunction with MATLAB and Simulink. This framework offers access to both physical components and cyber components which typically involve different ICT approaches and control strategies. Combined with the discrete-event/continuous-time hybrid simulation engine of the original SimEvents (Clune et al. (2006)), the simulation model includes both discrete-event components implemented by SimEvents, and continuous-time components implemented by Simulink. Thus, the overall traffic simulation framework is a hybrid dynamic model.

An important feature of the proposed traffic simulation framework is the capability for users to easily create different scenarios under which users can test various ICT methods and control algorithms. This paper describes and elaborates the capability of this hybrid simulation framework and includes demonstrations of how it can operate under various scenarios.

The paper is structured as follows. Section 2 reviews the hybrid traffic simulation framework introduced in Zhang et al. (2017a). Section 3 discusses in detail the implementation of the hybrid traffic simulation platform using SimEvents in conjunction with MATLAB and Simulink. Section 4 illustrates the effectiveness of the traffic simulation platform by demonstrating several different scenarios. Section 5 concludes with remarks and an outline of further research activities.

2. A DISCRETE-EVENT AND HYBRID TRAFFIC SIMULATION FRAMEWORK

To create a traffic simulation framework for vehicle behavior evaluation, a system consisting of physical elements (infrastructure and vehicles), cyber components (traffic control, communication, sensing technologies), and events is necessary.

Table 1. Infrastructure

Infrastructure	Property	Function
Road segment	length, number of lanes	sensing (sensors, cameras)
Merging zone	length/width, left/right turns	sensing (sensors, cameras)
Controller	control strategy, range	control, communication
Coordinator	range	communication

Table 2. Vehicles

Vehicle	Property & Dynamics
Properties	ID, acceleration, speed, position, lane, mpg, etc.
Motion dynamics	basic model, Kinematic model, Dynamic model, etc.
Control dynamics	optimal control, MPC, etc.
Fuel dynamics	gasoline engine, electric, hybrid, plug-in hybrid, etc.

Table 3. Events

Vehicle-to-Infrastructure Warning	Vehicle-to-Vehicle Warning
Red Light Violation	Emergency Electronic Brake Lights
Curve Speed	Forward Collision
Stop Sign Gap Assist	Intersection Movement Assist
Spot Weather Impact	Left Turn Assist
Reduced Speed/Work Zone	Blind Spot/ Lane Change
Pedestrian in Signalized Crosswalk	Do Not Pass
	Vehicle Turning Right in Front of Bus

The system is designed to comprise various elements of the SimEvents paradigms such as entities, queues, servers, terminators, and customized MATLAB Discrete Event Systems.

The model introduced in Zhang et al. (2017a) is briefly reviewed. There are three basic elements in the hybrid traffic simulation framework: infrastructure, vehicles, and events.

Infrastructure consists of roadside facilities that enable communication and carry out traffic management.

Vehicles can differ in motion dynamics, driver behavior models, fuel dynamics, and so on. For CAVs, they should also possess the ability to communicate with each other.

Events can be categorized into two classes in the hybrid systems: exogenous and endogenous events. Exogenous events include those originating from the outside world and force certain elements to change the behavior, for instance, an unexpected storm. Endogenous events occur when a time-driven state variable enters a particular set, for instance, the inter-vehicle distance falling below the minimum safe following distance, which may indicate a possible impending rear-end crash. Depending on whether they occur among vehicles or between vehicles and infrastructure (Bettisworth et al. (2015)), the events can also be categorized as listed in Table 3.

Figure 1 depicts the architecture of the traffic simulation framework and shows how different elements are connected. Certain elements should be capable of communicating with others. In this paper, it is assumed that only V2I communication is active, as V2V communication can be achieved through V2I2V communication. An important feature of the proposed framework is the inclusion of communication delay, as low packet delays are necessary for implementing the control algorithms employed by CAVs. In Fig. 1, the servers are used to model the communication delays.

Preprints of WODES 2018
May 30 - June 1, 2018
Sorrento Coast, Italy

A Discrete-Event and Hybrid Simulation Framework Based on SimEvents for Intelligent Transportation System Analysis ⋆

Yue Zhang * **Christos G. Cassandras** *,** **Wei Li** ***
Pieter J. Mosterman ***

* *Division of Systems Engineering*
** *Department of Electrical and Computer Engineering, Boston University, Brookline, MA 02446 USA (e-mail: joycez, cgc@bu.edu)*
*** *MathWorks, 1 Apple Hill Drive, Natick, MA 01760 USA (e-mail: wei.li, pieter.mosterman@mathworks.com)*

Abstract: Intelligent transportation systems combine physical elements with cyber components based on information and communication technologies and the use of control methodologies for Connected Automated Vehicles (CAVs). Intelligent transportation systems, therefore, contain event-driven dynamics along with time-driven dynamics. The hybrid nature of such systems motivates the development of new simulation platforms in order to test and evaluate their effectiveness. A discrete-event and hybrid simulation framework based on SimEvents is introduced within which these systems can be studied at the microscopic level. This framework enables users to apply different control strategies as well as communication protocols for CAVs and to carry out performance analysis of proposed algorithms by authoring customized discrete-event and hybrid systems that include various design paradigms such as entity flow, graphical programming, and object-oriented programming in MATLAB®. These paradigms provide users with the flexibility to select or combine modeling elements for achieving complex goals as the demonstrated scenarios in the paper illustrate. The framework spans multiple toolboxes including MATLAB, Simulink®, and SimEvents®.

Keywords: discrete event systems, hybrid systems, traffic control, intelligent transportation systems, smart cities

1. INTRODUCTION

An Intelligent Transportation System (ITS) combines time-driven dynamics governing its physical components with event-driven dynamics characterizing its cyber elements. The physical elements may include the infrastructure and vehicles, while the cyber components involve a variety of communication technologies, information processing, and control systems methodologies that aim at traffic flow optimization by exploiting vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication and make use of information fusion from multiple traffic sensing modalities.

The advent of CAVs provides the automotive industry with an unprecedented opportunity by enabling users to better monitor a transportation system's conditions, hence improving traffic flow in terms of reducing congestion as well as energy consumption and greenhouse gas emissions, while also improving safety. From a CAV's perspective, the physical domain is defined by vehicle mechanics, motion dynamics, etc., while the cyber domain involves the capability to sense the surroundings, communicate through V2V or V2I, and implement advanced control algorithms.

⋆ Supported in part by NSF under grants ECCS-1509084, CNS-1645681, and IIP-1430145, by AFOSR under grant FA9550-15-1-0471, by DOE under grant DOE-46100, by MathWorks and by Bosch.

Recent advances in CAVs focus on issues such as optimizing powertrain configurations, for instance, in plug-in hybrid electric vehicles (PHEV), as well as improving traffic conditions in terms of reducing travel delay and energy consumption. Examples include Lee and Park (2012) where the overlap between vehicle positions is minimized, and Gilbert (1976),Hooker (1988), Hellström et al. (2010), Li et al. (2012) where the focus is on improving energy economy. To evaluate the effectiveness of emerging proposed methodologies, a good way is to conduct field tests involving actual vehicles as discussed in Shladover et al. (1991) and Kato et al. (2002). Such tests take actual environmental factors into consideration, thus lending them credibility. However, field tests are often infeasible. In view of these factors, a suitable ITS simulation environment is needed.

The focus of this paper is on building microscopic transportation models allowing the evaluation of different information and communication technologies (ICT) and control algorithms. Microscopic models usually track individual transportation elements on a continuous-time basis; for instance, they must track the position of all vehicles. However, transportation systems must respond to events, some of which are random, such as vehicle arrivals or bad weather, while others are controllable, such as routing decisions or traffic light switches. Consequently, traffic models must be both event-driven and time-driven.

There are many traffic simulation platforms that can operate at the microscopic level, such as VISSIM (Fellendorf (1994)), PARAMICS (Cameron and Duncan (1996)), CAR-

Table 6. Main Functions in *PN_OPAC*

Name	Function
`BRG_Cur`	constructs the corresponding BRG for current-state opacity wrt a given secret
`showBRG_Cur`	shows the BRG in a readable text form
`CurOpac`	checks if the LPN system is current-state opaque wrt a given secret
`InitOpac`	checks if the LPN system is initial-state opaque wrt a given secret

```
>> [Pre, Post, M0] = getMatrix; E = {['a']}; L = {[1, 3]};
W = [-1 0 0 -1]; K = -2; S = { W; K};
>> [CSO, Obs, y] = CurOpac(Pre, Post, M0, L, E, S)
CSO =
     1
Obs =
    [4]    '1'    [4×3 double]    [0]    [ ]
y =
    [0]    [1×2 double]    [1×2 double]    [1×3 double]
```

Fig. 17. Check if the LPN system is current-state opaque.

for diagnosis is constructed using funcition `BRG_D`, based on which we construct the basis reachability diagnoser through function `BRD`. Then, the modified BRG is built using `MBRG`, and finally function `diagnosability` can be applied. The above steps and the obtained results are presented in Fig. 15, which shows that neither of f_1 and f_2 is diagnosable. Given an observation $w = a$, to diagnose if fault f_1 or f_2 has occurred we use function `diagnosis`, and the result is shown in Fig. 16. When nothing is observed (i.e., suffix equals *eps*) the diagnosis state is [0 0], which means that neither f_1 nor f_2 has occurred; when a is observed, the diagnosis state is [2 2], which means that f_1 and f_2 may have occurred and they are contained in one (but not in all) justification of w. ◇

4.4 State-Based Opacity Verification

Given an LPN system and a secret, opacity verification consists in checking if the system is opaque wrt the secret. PetriBaR can be used to verify current-state opacity (CSO) and initial state opacity (ISO). It is assumed that the secret is described by a set of GMECs, and the unobservable induced subnet is acyclic. For initial-state opacity verification, it is also assumed that none of the secret markings is weakly exposable, i.e., their unobservable reach is not strictly contained in the secret.

Function for opacity verification are in directory *PN_OPAC*, and are based on the results in (Tong et al., 2017). Some significant functions are listed in Table 6.

Example 7. Consider again the PN system in Fig. 8. Let $M_0 = [1\ 1\ 0\ 0]^T$, $T_u = \{t_2, t_4, t_5\}$, the labels assigned to t_1 and t_3 be a, and the secret $S = \{M | M(p_1) + M(p_4) \geq 2\}$. We use function `CurOpac` to check if the LPN is current-state opaque wrt S, and the result is presented in Fig. 17. The value of *CSO* is 1, which means that the LPN is current-state opaque; if the value is 0, the LPN system is not current-state opaque. ◇

5. CONCLUSIONS

The presented toolbox PetriBaR provides multiple functions for PN analysis, including the solutions to four problems: reachability, state estimation, fault diagnosis, and opacity verification. In all cases BRG-based techniques are implemented. Compared with other PN tools, PetriBaR has high flexibility, portability and compatibility. As a future work, we plan to integrate all functions with the graphical editor to form a fully functional software. Meanwhile, more functions will be continuously added to enrich the toolbox, such as functions for opacity enforcement.

REFERENCES

Balduzzi, F., Giua, A., and Menga, G. (2000). First-order hybrid Petri nets: a model for optimization and control. *IEEE transactions on robotics and automation*, 16(4), 382–399.

Cabasino, M.P. (2009). *Diagnosis and identification of discrete event systems using Petri nets*. Ph.D. thesis, Ph. D. Thesis.

Cabasino, M.P., Giua, A., Pocci, M., and Seatzu, C. (2011). Discrete event diagnosis using labeled Petri nets. an application to manufacturing systems. *Control Engineering Practice*, 19(9), 989–1001.

Cabasino, M.P., Giua, A., and Seatzu, C. (2010). Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, 46(9), 1531–1539.

Giua, A. (1992). Petri nets as discrete event models for supervisory control. *Rensselaer Polytechnic Institute, Troy, NY*.

INA (2003). Integrated net analyzer (version 2.2). `https://www2.informatik.hu-berlin.de/lehrstuehle/automaten/ina/`.

Ma, Z., Tong, Y., Li, Z., and Giua, A. (2017). Basis marking representation of Petri net reachability spaces and its application to the reachability problem. *IEEE Transactions on Automatic Control*, 62(3), 1078–1093.

Moody, J.O. and Antsaklis, P.J. (2000). Petri net supervisors for DES with uncontrollable and unobservable transitions. *IEEE Transactions on Automatic Control*, 45(3), 462–476.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Procedings of the IEEE*, 77(4), 541–580.

Pastravanu, O., Matcovschi, M.H., and Mahulea, C. (2004). Petri net toolbox teaching discrete event systems under MATLAB. In *Advances in Automatic Control*, 247–255. Springer.

PetriBaR (2017). PetriBaR (version 1.0). `https://www.researchgate.net/publication/321431345_MATLAB_Toolbox_for_Petri_Nets`.

Sessego, F., Giua, A., and Seatzu, C. (2008). Hypens: a matlab tool for timed discrete, continuous and hybrid Petri nets. *Lecture Notes in Computer Science*, 5062, 419–428.

TAPAAL (2017). TAPAAL (version 3.3.0). `http://www.tapaal.net/`.

Tong, Y., Ma, Z., Li, Z., Seatzu, C., and Giua, A. (2016). Verification of language-based opacity in Petri nets using verifier. In *American Control Conference (ACC), 2016*, 757–763. IEEE.

Tong, Y., Li, Z., Seatzu, C., and Giua, A. (2017). Verification of state-based opacity using Petri nets. *IEEE Transactions on Automatic Control*, 62(6), 2823–2837.

Table 4. Main Functions in *PN_ESTM*

Name	Function
`BRG_L`	builds the corresponding BRG of the LPN system
`showBRG_L`	shows the BRG in a readable text form
`obsRG`	computes the observer of the RG
`obsBRG`	computes the observer of the BRG
`estRG`	computes the initial state estimator of the RG
`estBRG`	computes the initial state estimator of the BRG
`CurEst_RG`	computes the set of markings consistent with the observation
`CurEst_BRG`	computes the set of basis markings consistent with the observation
`IniEst_RG`	computes the set of markings generating the observation
`IniEst_BRG`	computes the set of basis markings generating the observation

```
>> draw; [Pre, Post, M0] = getMatrix; E = {['a'];['b']}; L =
{[1];[3]};
w = 'aab';
>> [Cbw] = CurEst_BRG(Pre,Post,M0,L,E,w)
Cbw =
      [4×1 double]
```

Fig. 13. Computation of the set of basis markings consistent with aab.

```
>> B = BRG_L(Pre, Post, M0, L, E);
>> [ObsB, yb] = ObsBRG(B)
ObsB =
     [3]   '12'   [4×3 double]   [0]   [ ]
yb =
     [0]   [1]   [2]
```

Fig. 14. Observer of the BRG in Example 5.

where $\mathcal{C}_b(w) = \{[0\ 0\ 0\ 1]^T\}$. Analogously, for initial state estimation, we can use function `IniEst_BRG` to compute the set of basis markings generating aab, namely $\{[1\ 0\ 0\ 0]^T, [0\ 1\ 0\ 0]^T, [0\ 0\ 0\ 1]^T\}$.

Let us focus on the data structure of the observer obtained by function `obsBRG` (which could be helpful to readers interested in building their own functions). After constructing the BRG B through function `BRG_L`, we input `[ObsB,yb]=ObsBRG(B)` to compute the observer of the BRG (see Fig. 14). $ObsB$ is a 1×5 cell, and columns 1 to 5 represent the number of states [8] of the observer, the set of events [9], the transition function [10], the set of initial states, and the set of marked states, respectively. Each element of y_b corresponds to the set of basis markings of a state of the observer. We point out that the estimator of the BRG, and the observer/estimator of the RG obtained by functions in Table 4 have a similar structure of $ObsBRG$. Therefore, they are not discussed here. ◇

[8] It also denotes the set of states of the observer. Suppose the number of states is n. Then the set of states is $\{0, 1, \ldots, n-1\}$.

[9] Alphabetical symbols input to the LPN are orderly projected to numerical symbols in the observer. For instance, $E = \{['a']; ['b']\}$ of the LPN becomes $E =' 12'$.

[10] Suppose one row of the matrix is $[x_i\ j\ x_k]$. It means that from state x_i the occurrence of event k leads to state x_k.

Table 5. Main Functions in *PN_DIAG*

Name	Function
`BRG_D`	constructs the BRG for fault diagnosis wrt the class of faults
`showBRG_D`	shows the BRG in a readable text form
`MBRG`	constructs the modified BRG (MBRG) that is used to check whether the LPN system is diagnosable or not.
`showMBRG`	shows the BRG in a readable text form
`BRD`	constructs the basis reachability diagnoser, a diagnoser construted on the BRG.
`showBRD`	shows the BRG in a readable text form
`diagnosability`	tests if a bounded LPN system is diagnosable wrt each fault class
`diagnosis`	online diagnoses an LPN system for a given observation w

```
>> [Pre, Post, M0] = getMatrix; F = {[3]; [5]}; L = {[1]}; E =
{['a']};
BG = BRG_D(Pre, Pos, M0, F, L, E);
BD = BRD(BG, Pre,Post, M0, F, L, E);
T = MBRG(Pre, Pos, M0, F, L, E);
>> diagnosability(T, BD)
   All fault classes are not diagnosable since for each of them there
is an indeterminate loop.
Fault class 1:
            path = aa
            cycle = a
Fault class 2:
            path = aa
            cycle = a
```

Fig. 15. Diagnosability test in Example 6

```
>> w = 'a'; diagnosis(Pre, Post, M0, w, L ,F, E)
The following results summarizes the step of the on-line diagnosis
carried out by the observed word: - 'a'.
      SUFFIX        DIAGNOSIS STATE

        eps              0   0
         a               2   2
```

Fig. 16. Online diagnosis of observation $w = a$.

4.3 Fault Diagnosis

Given an LPN system and a set of fault transitions, which have been partitioned into several fault classes $F = \{f_1, f_2, \ldots, f_r\}$, two main problems are investigated in the literature: i) fault diagnosis, which consists in establishing, given an observation, if a fault in a given class has occurred, and ii) diagnosability analysis, which consists in establishing if the occurrence of a fault in a given class could be detected after the occurrence of a finite number of other events. Cabasino *et al.* have proposed efficient algorithms to solve the above problems using LPNs (see (Cabasino, 2009) for technical details). Such algorithms are based on the notion of BRG, and are implemented by the MATLAB functions in directory *PN_DIAG*. Table 5 lists some of the functions.

Example 6. Consider again the LPN in Fig. 8. Let t_3 and t_5 be fault transitions that belong to two different fault classes f_1 and f_2, respectively, t_2 and t_4 be regular unobservable transitions, and the label associated to t_1 be a. To check if f_1 and f_2 are diagnosable, first the BRG

```
>> showBRG(B)
    Basis Reachability Graph node's number n = 3
    # Marking M0=[1 1 0 0]'
    Observable transitions enabled to fire:
        (t1) -> M1: e=[0 0]
        (t4) -> M2: e=[1 0]
        (t5) -> M2: e=[1 1]
          ************
    # Marking M1=[0 2 0 0]'
    Observable transitions enabled to fire:
        (t4) -> M0: e=[1 0]
        (t5) -> M0: e=[1 1]
          ************
    # Marking M2=[2 0 0 0]'
    Observable transitions enabled to fire:
        (t1) -> M0: e=[0 0]
          ************
```

Fig. 10. Output of function `showBRG`.

```
>> Te=CompTe(Pre,Post)
Te =
     1
```

Fig. 11. Set of explicit transitions computed by function `CompTe`.

Example 4. Consider the LPN system in Fig. 8. Let $T_e = \{t_1, t_4, t_5\}$ be a feasible set of explicit transitions. First, we input the net system $(Pre, Post, M0)$ and the set $T_e = [1\ 4\ 5]$, and then compute its BRG through function `B=BRG(Pre,Post,M0,Te)` (see Fig. 9). Let us now briefly explain the data structure[6] of B. Entries in columns 1 to 6 are: the index associated with the current basis markings, the basis marking, explicit transitions whose minimal explanation vectors are not empty, the minimal explanation vector and the index of the basis marking reached, the tag recording if the basis marking has been analyzed, the explicit transition and the corresponding minimal explanations fired. Therefore, B contains all the information about the BRG, and in the future the reader can develop their own functions based on function `BRG` as Sections 4.2 to 4.4 show. The structure of the BRG can be presented in the form of text, as Fig. 10 shows, by using function `showBRG(B)`.

Given an arbitrary marking $M = [0\ 1\ 0\ 1]^T$, the result of running `r=IfReachTe(M,Pre,Post,M0,Te)` is $r = 1$, i.e., marking M is reachable from the given initial marking. On the contrary, if $M = [1\ 1\ 0\ 1]^T$, the output is $r = 0$, which means that the marking is not reachable.

Suppose the feasible set T_e of explicit transitions is not given. In this case, there are two ways to check if a marking is reachable. One method consists in first using `Te=CompTe(Pre,Post)` to obtain a minimal feasible set T_e (the result is $T_e = \{t_1\}$), then using function `IfReachTe` to check whether the given marking is reachable or not. Note that the reachability of a marking does not depend on the value of T_e. The other method consists in using function `IfReach` directly, as shown in Fig. 12. ◇

[6] We point out that the data structure of the BRG constructed later for LPNs, or fault diagnosis, or opacity verification is similar to the one presented here. For brevity, a detailed discussion on differences is omitted here and the reader is addressed to the comments inside the MATLAB functions.

```
>> M = [0 1 0 1]'; [r, Te] = IfReach(M, Pre, Post, M0)
r =
     1
Te =
     1
```

Fig. 12. Output of function `IfReach`.

4.2 State Estimation

In an LPN system $(Pre, Post, M_0, L, E)$ the labeling function L associates an output symbol with each transition. It could either be a symbol in a given alphabet E, or the empty word ε. Therefore, the transitions are partitioned into the set T_o of observable transitions, whose labels are symbols in E, and the set T_u of unobservable transitions, whose labels are the empty word.

The considered state estimation problem consists in determining the set of current markings consistent with the observation $w \in E^*$. Furthermore, the initial state estimation problem consists in determining the set of possible initial markings from which the observation $w \in E^*$ can be generated. Typically, the methods of solving the problems of current state estimation and initial state estimation are based on the construction of the observer and the initial state estimator, respectively, of the RG, whose complexity is known to be exponential wrt the number of reachable markings. However, if the unobservable transitions induced subnet is acyclic, the BRG-based technique can be applied (Cabasino et al., 2011). In such a case the complexity is still exponential but wrt the number of basis markings. To build the BRG for LPNs, observable transitions are taken as explicit transitions, while the unobservable transitions are the implicit transitions. Moreover, the label of an observable transition is also tagged on the BRG.

Let $\mathcal{C}(w)$ be the set of markings consistent with an observation w and $\mathcal{C}_b(w)$ the set of basis markings consistent with w. By Theorem 3.7 in (Tong et al., 2017), it holds that $\mathcal{C}(w) = \bigcup_{M_b \in \mathcal{C}_b(w)} \{M | M = M_b + C_u \cdot y, y \in \mathbb{N}^m\}$, where C_u is the incidence matrix of the unobservable transitions induced subnet. Namely, using the BRG-based technique, the set $\mathcal{C}(w)$ of consistent markings is represented by a set of linear constraints associated with basis markings. Therefore, computing the set $\mathcal{C}_b(w)$ of basis markings consistent with the observation, the set $\mathcal{C}(w)$ is also obtained.

Functions for state estimation are in directory *PN_ESTM* and are summarized in Table 4. Due to limited space, in Example 5 we mainly illustrate the methods based on the BRG.

Example 5. Consider again the PN system in Fig. 8. Let $M_0 = [1\ 0\ 0\ 0]^T$, $T_u = \{t_2, t_4, t_5\}$, the labels assigned to t_1 and t_3 be a and b, respectively, and the observation $w = aab$. For current state estimation, we can use function `CurEst_BRG` to compute $\mathcal{C}_b(w)$ directly. The procedure and the result is presented[7] in Fig. 13,

[7] Since the output *Cbw* is defined as a cell, the value of *Cbw* is not directly shown in the command window.

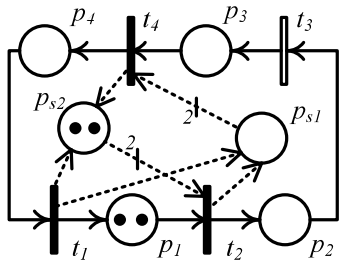


Fig. 7. Closed loop system in Example 2.

i.e., (w_1, k_1) is controllable while (w_2, k_2) is not controllable. Use function `[W2, K2] = controllablegmec(C, M0, w2, k2, u)` to compute a set of controllable GMECs (W_2, K_2) such that (W_2, K_2) is minimally restrictive and satisfies $\mathcal{M}(W_2, K_2) \subseteq \mathcal{M}(w_2, k_2)$, and the result is $W_2 = [0\ 2\ 2\ 1]^T$ and $K_2 = 2$. Finally, we use function `monitorplaces` to compute the incidence matrix CS of monitor places and their initial marking M_{S0}. The obtained results and the whole procedure are illustrated in Fig. 6. The corresponding closed loop system is illustrated in Fig. 7. Note that given a set of controllable/uncontrollable GMECs, function `monitordesign` returns the closed loop system such that its reachable markings are all legal but may not be maximally permissive [4]. ◇

4. FUNCTIONS USING BRG-BASED TECHNIQUES

In (Ma et al., 2017), a compact representation of the reachability graph (RG) is proposed. Partitioning the set of transitions into implicit and explicit transitions, it is shown that if the implicit transitions induced subnet is acyclic, only a subset of reachable markings, called *basis markings*, need to be enumerated, while other reachable markings are characterized by linear systems, one for each basis marking.

To help the reader in understanding the idea behind the programs, we briefly recall the notion of basis marking. Let $(P, T, Pre, Post, M_0)$ be the PN system, $T_e \subseteq T$ the set of *explicit* transitions, and $T_i = T \setminus T_e$ the set of *implicit* transitions. First, the initial marking M_0 is a basis marking. Then, given an explicit transition t_e if there is a *minimal* (in terms of its firing vector) firable sequence of implicit transitions $\sigma_i \in T_i^*$ enabling t_e, then the marking M reached by firing $\sigma_i t_e$ from M_0 is also a basis marking. Iteratively, the set $\mathcal{M}_b$ of basis markings is obtained. Obviously, the set of basis markings is a subset of the reachability set since basis markings are only markings reachable by firing the minimal implicit transition sequences and an explicit transition. Finally, we have the following important result.

Theorem 3. (Ma et al., 2017) Let $(P, T, Pre, Post, M_0)$ be a PN system, T_e a set of explicit transitions, and $\mathcal{M}_b$ a set of basis markings wrt T_e. Assume the T_i induced subnet [5] is acyclic. A marking $M \in \mathbb{N}^m$ is reachable in the PN system if and only if there exists a basis marking $M_b \in \mathcal{M}_b$ such that $M = M_b + C_I \cdot y$ has an integer solution $y \in \mathbb{N}^{n_I}$,

[4] If there is no solution, the function will output "This method can not be used".

[5] The T_i-induced subnet of $(P, T, Pre, Post)$ is the net obtained by removing from $(P, T, Pre, Post)$ all transitions not in T_i.

Table 3. Main Functions in *PN_REACH*

Name	Function
`BRG`	computes the corresponding BRG of the net system wrt a given feasible T_e
`showBRG`	shows the BRG in the form of text
`IfReachTe`	checks if a marking is reachable, given a feasible T_e
`CompTe`	computes a minimal feasible set T_e
`IfReach`	checks if a marking is reachable and outputs a minimal feasible set T_e

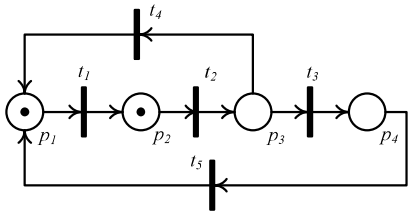


Fig. 8. The PN system in Example 4.

```
>> draw; [Pre, Post, M0] = getMatrix; Te = [1 4 5];
>> [B] = BRG(Pre,Post,M0,Te)
B =
    [1] [4×1 double] [1×3 double] {1×3 cell} [1] {1×3 cell}
    [2] [4×1 double] [1×3 double] {1×3 cell} [1] {1×3 cell}
    [3] [4×1 double] [1×3 double] {1×3 cell} [1] {1×3 cell}
```

Fig. 9. Construction of the BRG of the PN system in Fig. 8.

where m is the number of places, C_I is the incidence matrix of the implicit transitions induced subnet, and n_I is the number of implicit transitions.

In other words, after the set of basis markings are computed, the reachability problem reduces to the solution of the integer linear programming problem, and the union of convex sets associated with the different basis markings coincides with the set of reachable markings. A *basis reachability graph* (BRG) is a graph that describes the basis markings and their transition relations, and thus it compactly represents the RG. Based on the BRG representation, algorithms very efficient in practice have been proposed in (Cabasino et al., 2010; Ma et al., 2017; Tong et al., 2017; Cabasino et al., 2011) to solve the reachability problem, the fault diagnosis problem and the opacity verification problem. The efficiency of the BRG-based approaches with respect to other RG-based methods has been extensively shown in the aforementioned work.

Typically T_e is not a set of special transitions. As long as the set of transitions can be partitioned into T_e and T_i such that the T_i-induced subnet is acyclic (such a set T_e is called *feasible*), Theorem 3 applies. Thus, in some cases a feasible set T_e needs to be computed first so that BRG-based techniques can be applied to analyzing the system.

4.1 Reachability Analysis

Functions for reachability analysis are included in directory *PN_REACH* and listed in Table 3. All functions are coded using the results in (Ma et al., 2017), where the reader can also find the technical details.

Table 1. Main Functions in *PN_BASIC*

Name	Function	Name	Function
`graphPN`	builds the RG/CG of a PN system	`tree`	builds the RT/CT of a PN system
`show`	shows the RG/CG in a readable text form	`plottree`	draws the graph of a RT/CT
`play`	simulates the evolution of a PN system		
`bounded`	checks if a PN system and its places are bounded	`live`	checks if a PN system and its transitions are live
`dead`	checks deadlock markings in a PN system	`reachable`	checks if a marking is reachable from M_0
`firable`	checks if a sequence of transitions is enabled at M_0	`reversible`	checks if a PN system is reversible
`pinvar`	computes P-invariants (or P-semiflows)	`tinvar`	computes T-invariants (or T-semiflows)
`siphons`	computes siphons	`traps`	computes traps

```
>> draw; [Pre, Post, M0] = getMatrix;
>> b = bounded(Pre,Post,M0)
M =
           Inf     0     0
****The net is unbounded!****
b =
           0
```

Fig. 3. The PN system in Fig. 2 is bounded.

```
>> G = graphPN(Pre,Post,M0)
G =
        G11          G12   G13   G14   G15   G16
  1     0     0      0     -2    1  2  0  0  0  0
  0     1     0      1     -2    2  3  3  4  0  0
 Inf    0     0      2     -2    1  5  0  0  0  0
  0     0     2      2     -1    0  0  0  0  0  0
 Inf   Inf    0      3     -2    1  5  2  5  3  6
 Inf   Inf   Inf     5     -2    1  6  2  6  3  6
G21 3  3     1      0     0     0  0  0  0  0  0
```

Fig. 4. RG (in matrix form) of the PN system in Fig. 2.

transitions, and $G_{21}(3) = 1$ means the matrix represents the CG. ◇

The size of G depends on the number of markings in the RG/CG and the number of transitions. The functions using the RG/CG to analyze the behavioral properties are listed in Table 1. In the case of unbounded nets the CG gives necessary and sufficient conditions for boundness. However, for the absence of dead states, reversibility, liveness, reachability it only provides necessary conditions.

3. MONITOR DESIGN

A set of *generalized mutual exclusion constraints* (GMECs) (W, K), with $W = [w_1 \dots w_r]$ and $K = [k_1; \dots; k_r]$, defines a set legal markings

$$\begin{aligned}\mathcal{M}(W,K) &= \textstyle\bigcap_{i=1}^{r}\{M \in \mathbb{Z}^m | w_i^T \cdot M \le k_i\} \\ &= \{M \in \mathbb{Z}^m | W^T \cdot M \le K\},\end{aligned}$$

where $w_i \in \mathbb{Z}^m$ and $k_i \in \mathbb{Z}$ (with $i = 1, 2, \dots, r$). As shown in (Giua, 1992; Moody and Antsaklis, 2000), a set of r GMECs can be enforced adding r places, called monitors, with appropriate initial marking, to the given net system. In the presence of uncontrollable transitions, if the given GMECs (W, K) are uncontrollable[3], a set of *controllable* GMECs (W', K') may be computed such that the reachable marking set of the obtained system is contained in the set of legal markings $\mathcal{M}(W, K)$.

[3] A set of GMECs is uncontrollable if there is a pre arc from a monitor place to uncontrollable transitions.

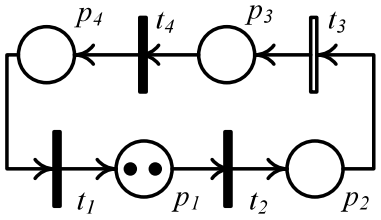


Fig. 5. The PN system in Exmaple 2, where t_3 is uncontrollable.

```
>> draw; [Pre, Post, M0] = getMatrix; C = Post-Pre;
w1 = [1 0 0 2]'; w2 = [0 0 2 1]';
k1 = 2; k2 = 2; u = [3];
W = [w1, w2]; K = [k1; k2];
[W2c, K2c] = controllablegmec(C, M0, w2, k2, u);
W2 = [w1,W2c']; K2 = [k1;K2c];
>> [CS, MS0] = monitorplaces(C, M0, W2, K2, u)
CS =
     1    1    0   -2
     1   -2    0    1
MS0 =
     0
     2
```

Fig. 6. Computation of monitor places enforcing a given set of GMECs.

Functions in directory *PN_MONIT* solve the monitor design problem. The most significant ones are listed in Table 2. The following example shows how to use them to compute monitor places enforcing a given set of GMECs.

Table 2. Main Functions in *PN_MONIT*

Name	Function
`checkgmecs`	tests if a set of GEMCs is controllable wrt a PN system
`controllablegmec`	finds all minimally restrictive controllable GMECs satisfying the given uncontrollable GMEC
`monitordesign`	designs a closed loop net system given a GMEC
`monitorplaces`	finds the row of the incidence matrix of the monitor place used to satisfy the set of controllable GMECs

Example 2. Consider the PN system in Fig. 5, where t_3 is uncontrollable. Let $(W = [w1, w2]$, $K = [k_1; k_2])$ be the set of GMECs, where $w_1 = [1\ 0\ 0\ 2]^T$, $k_1 = 2$, $w_2 = [0\ 0\ 2\ 1]^T$, $k_2 = 2$. Inputting the incidence matrix $C = [1\ -1\ 0\ 0;\ 0\ 1\ -1\ 0;\ 0\ 0\ 1\ -1;\ -1\ 0\ 0\ 1]$, the initial marking $M_0 = [2\ 0\ 0\ 0]^T$, the set of GMECs (W, K), and the uncontrollable transitions $u = [3]$ to function `[Wc,Kc,Wu,Ku]=checkgmecs(C,M0,W,K,u)`, we obtain that $W_c = w_1$, $K_c = k_1$, $W_u = w_2$ and $K_u = k_2$,

e.g., boundedness analysis, siphons computation, etc.; (2) monitor design for GMECs; (3) reachability analysis; (4) state estimation; (5) fault diagnosis, including diagnosability analysis; (6) state-based opacity verification. Functions in categories 3-6 exploit the approach based on BRG, therefore they are able to handle some relatively large-sized nets. PetriBaR can be freely downloaded from a web site (PetriBaR, 2017).

Summarizing, the main features of the tool.

• Besides structural and behavioral properties analysis, PetriBaR can also solve control problems and problems of reachability analysis, state estimation, fault diagnosis, and opacity verification taking advantage of the BRG approach.

• The MATLAB environment has been widely used to implement software tools for the control of continuous-time systems. In addition there exists also a few other Petri net tools based on MATLAB (e.g., Pastravanu et al. (2004)). A tool, called HYPENS (Sessego et al., 2008), has also been implemented in MATLAB by some of the authors of this paper to simulate and analyze First-Order Hybrid Petri nets (Balduzzi et al., 2000). The tool illustrated in this paper allows one to fill the gap of the MATLAB implementation of purely logic PNs and may lead to the MATLAB solutions to problems of state estimation, fault diagnosis, etc. in hybrid Petri nets.

• In most universities, MATLAB is taught in basic courses for engineering students. Thus, it is easy for students to get started. Moreover, PetriBaR is open source and built in a modular way. Once the user becomes familiar with the implemented function, he/she can easily modify and extend them depending on his/her own requirement. Finally, MATLAB runs on many platforms (Windows, Unix, MacOS).

2. BASIC PETRI NET ANALYSIS FUNCTIONS

A *Petri net system* (PN system) is a pair (N, M_0), where $N = (P, T, Pre, Post)$ is the net structure, $P = \{p_1, p_2, \ldots, p_m\}$ is the set of *places*, $T = \{t_1, t_2, \ldots, t_n\}$ is a set of *transitions*, $Pre : P \times T \to \mathbb{N}$ and $Post : P \times T \to \mathbb{N}$ are the *pre-* and *post-incidence functions* that specify the arcs directed from places to transitions and from transitions to places, respectively, $M_0 \in \mathbb{N}^m$ is the initial marking. The pre- and post-incidence functions are usually described by $m \times n$ dimensional pre- and post-incidence matrices, and clearly they uniquely determine the structure of the PN. Thus, to input a PN system to a function of the toolbox, users need to give the pre- and post-incidence matrices and a column vector that represents the initial marking. PetriBaR contains a function, `draw`, to assist the user in inputting the PN system and obtaining the corresponding matrices. Running function `draw`, a graphical interface appears (see Fig. 1), and the user can input the PN system by clicking "Place"/"Transition"/"Arc" and drawing them in the blank area. By double clicking a place (or an arc), the user can edit the number of tokens in the place (or the weight of the arc). Finally, choosing "Save Data", the value of the pre- and post-incidence matrices, and the initial marking are written in file `getMatrix`.

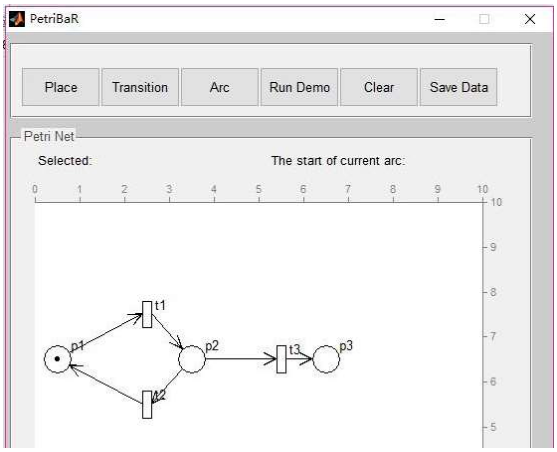


Fig. 1. GUI for inputting the PN system.

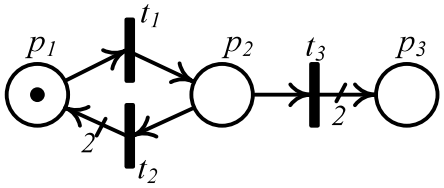


Fig. 2. The PN system in Exmaple 1.

A series of functions for basic PN structural and behavioral analysis, including the construction of the reachability/coverability graph/tree (RG/CG or RT/CT), are contained in the directory *PN_BASIC* of the toolbox. The most significant ones are listed in Table 1.

For sake of brevity, and since our focus here is on the notion of basis reachability graph, in the following example we restrict our attention on the function `graphPN` that allows us to construct the RG of a given bounded PN system.

Example 1. Input the PN system in Fig. 2 through function `draw`. Function `getMatrix` returns matrices *Pre*, *Post*, and $M0$. We check if the PN system is bounded using function `bounded`. The output of the function is $b = 0$: the net is unbounded (see Fig. 3). Then we input command `G=graphPN(Pre,Post,M0)` to construct its CG in the form of a matrix, as Fig. 4 shows. Let us now briefly explain the data structure of G.

Matrix G is partitioned into several submatrices:
• $G_{11} = G(1:6, 1:3)$: each row of G_{11} represents a marking (transposed) in the CG, and the index of each marking is their corresponding row number [2], i.e., $G_{11}(i, \cdot)^T = M_i$;
• $G_{12} = G(1:6, 4)$: the i-th element $G_{12}(i)$ of G_{12} denotes a father node of M_i;
• $G_{13} = G(1:6, 5)$: $G_{13}(i) = -2$ (resp., -1) means M_i is not a dead marking (resp., is a dead marking);
• G_{14}, G_{15} and G_{16} (every two columns after 5-th column and rows 1 to 6 compose a submatrix): each row of G_{14}, G_{15} and G_{16} describes a transition. For instance, the firing of transition $G_{14}(i, 1)$ at M_i leads to marking M_j, where $j = G_{14}(i, 2)$;
• $G_{21} = G(7, 1:3)$: the first element of G_{21} denotes the number of places, the second one $G_{21}(2)$ is the number of

[2] The initial marking is at the first row, i.e., corresponds to M_1.

Preprints of WODES 2018
May 30 - June 1, 2018
Sorrento Coast, Italy

PetriBaR: A MATLAB Toolbox for Petri Nets Implementing Basis Reachability Approaches

Siqi Liu *, **Yin Tong** *,[1], **Carla Seatzu** **, **Alessandro Giua** ***

* *School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China (e-mail: bk20122485@my.swjtu.edu.cn; yintong@swjtu.edu.cn).*
** *Department of Electrical and Electronic Engineering, University of Cagliari, 09123 Cagliari, Italy (e-mail: seatzu@diee.unica.it)*
*** *DIEE, University of Cagliari, Cagliari 09124, Italy, Aix-Marseille Univ, Université de Toulon, CNRS, ENSAM, LSIS, Marseille 13397, France (e-mail: giua@diee.unica.it)*

Abstract: This paper presents a MATLAB toolbox, called *PetriBaR*, for the analysis and control of Petri nets. PetriBaR is a package of functions devoted to basic Petri net analysis (including the computation of T-invariants, siphons, reachability graph, etc.), monitor design, reachability analysis, state estimation, fault diagnosis, and opacity verification. In particular, the functions for reachability analysis, state estimation, fault diagnosis, and opacity verification exploit the construction of the *Basis Reachability Graph* to avoid the exhaustive enumeration of the reachable set, thus leading to significant advantages in terms of computational complexity. All functions of PetriBaR are introduced in detail clarifying the syntax to be used to run them. Finally, they are illustrated via a series of numerical examples. PetriBaR is available online for public access.

Keywords: Petri nets, MATLAB toolbox.

1. INTRODUCTION

Petri nets (PNs) are a powerful discrete event model. The analysis of PNs basically concerns the study of some behavioral properties including reachability, boundedness, liveness, reversibility, repetitiveness, etc. and some structural properties related to the presence of P-invariants, T-invariants, siphons, traps, etc. (Murata, 1989).

Concerning the control of PNs under static specifications, one of the most popular approaches is based on the notion of *generalized mutual exclusion constraints* (GMECs). In such a case the set of legal markings is defined as a set of linear inequalities and this benefits from the main feature of PNs. In particular, Giua (Giua, 1992) demonstrated that, in the case of controllable and observable transitions, such constraints could be enforced simply adding some places called *monitors*, also guaranteeing maximal permissiveness. In (Moody and Antsaklis, 2000) such a theory has also been extended to the case where some transitions are uncontrollable and/or unobservable.

When dealing with problems of state estimation, fault diagnosis, opacity verification, etc. the system is typically modeled using *labeled Petri nets* (LPN) to describe the fact that certain transitions may produce no observation when they fire (silent transitions) or may produce the same observation of other transitions (indistinguishable transitions). To avoid exhaustively enumerating all reachable markings and firing vectors, the notions of basis marking and minimal explanation vector have been introduced in (Cabasino et al., 2010) to solve the problem of fault diagnosis. In (Cabasino et al., 2010) the authors also show that in the case of bounded nets, such notions allow one to describe the system's behaviour without enumerating all the reachable markings (as in the reachability graph) and introduce the *Basis Reachability Graph* (BRG). The BRG has also been efficiently used to solve a series of other problems, in particular, reachability analysis, state estimation and opacity verification (Ma et al., 2017; Tong et al., 2016, 2017).

The goal of this paper is that of illustrating a MATLAB tool implementing all the above mentioned approaches. We notice that there are many other tools for PN analysis and simulation, such as Integrated Net Analyzer (INA) (INA, 2003), TAPAAL (TAPAAL, 2017), PN Tool (Pastravanu et al., 2004), etc. While all these tools can be used to study basic properties and simulating the behavior of purely logical PNs, PNs with time, and also high level PNs, few of them can handle the problems of states estimation, fault diagnosis, and opacity verification. Furthermore, all the above tools offer a user-friendly graphical interface, which provides users an easy access but makes it hard to modify the codes and to extend them for other purposes. On the contrary, the PN toolbox *PetriBaR* we present consists of several MATLAB functions and is meant as support for research activities and classroom problem solving. Its functions can be classified into six categories:(1) basic analysis,

[1] Corresponding Author

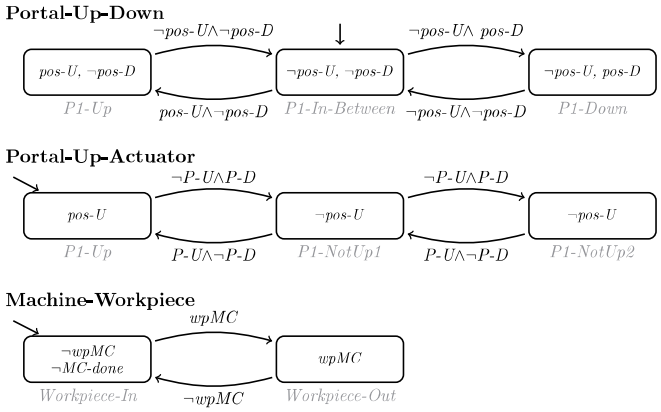


Fig. 7. Plant models for the nominal behavior of the vertical and horizontal portal movement and the machine on the compact line

generation of test sequences are all executed by the tool automatically. In addition, even some simple fragments of the nominal behavior of the system under test contribute to the reduction of test cases; of course, the more plant features are modeled, the higher reduction is obtained.

When the test generation is performed on the SCA from the original specification models solely with CCT approach, a test sequence of 9,647,120 steps is obtained. In the case study, ten plant models are used. As a final result, a test sequence with 448,752 steps is obtained, which leads to a reduction rate of 95% in comparison to the one obtained without plant features.

It can be stated that integrating knowledge about signal relations into the generation process drastically reduces the length of generated test sequences.

5. CONCLUSION

In this work, we presented the current version of our test case generator implementation. The objective of conformance testing is to determine the capability of a software product to adhere to standards, conventions and regulations. In this context, testing of nominal behavior by considering plant features can be a good supplement for complete testing or a replacement when complete testing is not feasible.

In future works, we would like to omit the monolithic composition, as this limits the complexity of the case studies that can be involved drastically. Although we achieved to improve our implementation, the computation is still primary memory intensive. Even though the resulting size is reduced due to the plant features, those extra models have to be considered during calculation, which leads to the question whether and how modular approaches can be applied and to what extent they reduce the computational effort.

Currently, only binary signals are taken into account for the control logic. This crucially restricts the applicability of the presented tool. Consequently, further investigation on extending the capability of handling integer signals is needed. Input equivalence class partitioning might be fruitfully applied. Furthermore, temporal and timing relations between signals can be formulated similar to mutual exclusion and premise.

REFERENCES

Bohlender, D., Simon, H., Friedrich, N., Kowalewski, S., and Hauck-Stattelmann, S. (2016). Concolic test generation for PLC programs using coverage metrics. In *Discrete Event Systems (WODES), 2016 13th International Workshop on. IEEE.*, 432–437.

Enoiu, E.P., Sundmark, D., and Pettersson, P. (2013). Model-based test suite generation for function block diagrams using the UPPAAL model checker. In *IEEE 6th Int. Conf. on Software Testing, Verification and Validation Workshops*, 158–167.

Guignard, A. and Faure, J.M. (2014). A Conformance Relation for Model-Based Testing of PLC. In L. Jean-Jacques (ed.), *12th Int. Workshop on Discrete Event Systems*, 412–419. Cachan.

Jordan, C., Ma, C., and Provost, J. (2017). An educational toolbox on supervisory control theory using matlab simulink stateflow – from theory to practice in one week. In *8th IEEE Global Engineering Education Conference (EDUCON 2017)*.

Kormann, B. and Vogel-Heuser, B. (2011). Automated test case generation approach for PLC control software exception handling using fault injection. In *37th Annual Conf. of the IEEE Ind. Elect. Soc.*, 365–372.

Lee, D., Member, S., and Yannakakis, M. (1996a). Principl Finite S. 84(8).

Lee, D., Sabnani, K.K., Kristol, D.M., and Paul, S. (1996b). Conformance testing of protocols specified as communicating finite state machines - A guided random walk based approach. *IEEE Transactions on Communications*, 44(5), 631–640.

Ma, C. and Provost, J. (2016). DTT-MAT: A software toolbox of design-to-test approach for testing programmable controllers. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, 878–884. Fort Worth, Texas, USA.

Ma, C. and Provost, J. (2017). A model-based testing framework with reduced set of test cases for programmable controllers. In *13th IEEE Conference on Automation Science and Engineering (CASE)*, 944–949.

Mani, P. and Prasanna, M. (2016). Automatic Test Case Generation for Programmable Logic Controller using Function Block Diagram. In *Int. Conf. on Information Communication and Embedded Systems (ICICES)*, 1–4.

Provost, J., Roussel, J.M., and Faure, J.M. (2011). Translating grafcet specifications into mealy machines for conformance test purposes. *Control Engineering Practice*, 19(9), 947–957. doi:10.1016/j.conengprac.2010.10.001.

Provost, J., Roussel, J.M., and Faure, J.M. (2014). Generation of single input change test sequences for conformance test of programmable logic controllers. *IEEE Transactions on Industrial Informatics*, 10(3), 1696–1704. doi:10.1109/TII.2014.2315972.

Input	Description
wpInBu	*True* when a workpiece is in the input buffer
loc-IL	*True* when the portal is at the drop location of indexed line
loc-In	*True* when the portal is at the input buffer location
loc-CL	*True* when the portal is at the drop location of compact line
pos-U	*True* when the portal is in its up end position
pos-D	*True* when the portal is in its down end position
p-2-IL	*True* when the current work piece should be brought to indexed line
p-2-CL	*True* when the current work piece should be brought to compact line
wpMC	*True* when the workpiece reaches the expected position in front of the machine
MC-done	*True* when the machine finishes its machining
wpCOut	*True* when the workpiece reaches the output position of compact line
wp-Output	*True* when the command of outputting the workpiece is received
wpCLBu	*True* when a workpiece is in the buffer of compact line
pos-E-PC	*True* when the pusher of compact line is in its extended position
pos-R-PC	*True* when the pusher of compact line is in its retracted position

Output	Description
P-G	activate the gripper
P-U	move the portal upwards
P-D	move the portal downwards
P-R	move the portal to the right (towards IL)
P-L	move the portal to the left (towards CL)
BC-P	run the belt of compact line in positive direction
MC	run the machine of compact line
PC-F	move the pusher of compact line forwards
PC-B	move the pusher of compact line backwards

Table 2. Table of inputs & outputs for the portal, belt and machine on the compact line

Initially, the maximum length of the shortest path between any couple of states is 5. Although this seems a reasonable value for controllability, the DTT approach can help to reach a better performance. After adding 5, 12 or 36 C-guards, the maximum length of the shortest path is reduced respectively to 4, 3 or 2 steps.

Now, with the modified specification models, full SIC-testability, full observability, and better controllability is achieved.

4.3 Applying test case reduction with plant features

According to the relations between inputs discussed in Sec. 3.3, for the case study, a set of exemplary plant models are presented in Fig. 7.

The first model in Fig. 7 presents a *mutual exclusion* relation between two input signals: in a nominal behavior, *pos-U* and *pos-D* should not be true at the same time, since a portal cannot be simultaneously in its *up* and *down* position. Similarly, a model for the horizontal movement can be found.

The second model presents a relation among input and output signals. The input *pos-U* remains *True* unless the output *P-D* is activated (and *P-U* is not active). At the same time it is stated that - reading the model from right to left - *pos-U* will not instantaneously but eventually be *True* when *P-U* is activated. Note that it is explicitly

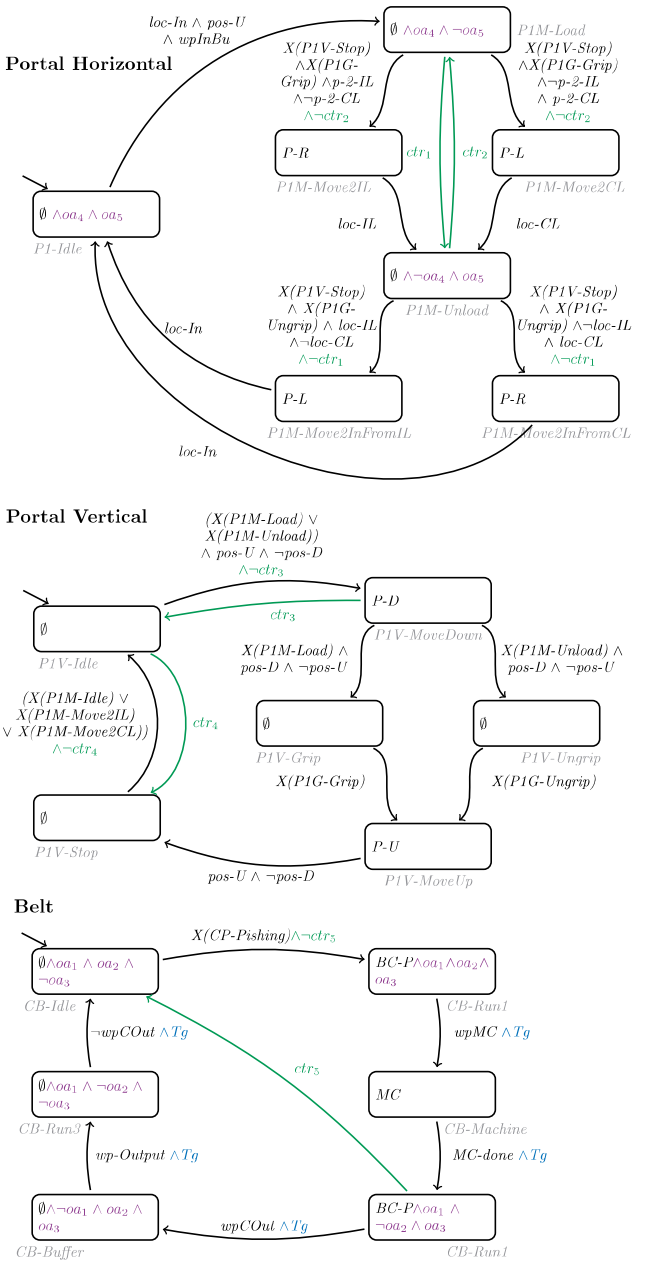


Fig. 6. Specification models for the horizontal portal movement, the vertical portal movement and the belt of the compact line

not stated that *pos-U* will be true directly after the portal movement has been activated, i.e. from location *P1-NotUp2*, *pos-U* will first remain *False* (as in location *P1-NotUp1*), and will eventually become *True* (as in location *P1-Up*). Analogously, a feature for the down movement can be found.

The third model presents a *premise* relation between two input signals. The input *MC-done* can only become *True* when the input *wpMC* is *True*, which means the machine does only operate when the workpiece is at the expected position in front of the machine.

It is worth mentioning that modeling of plant features is based on domain knowledge from engineers. Then, the rest process from composition of specification models to the

high	*False*	*False*	*True*	*True*
low	*False*	*True*	*False*	*True*

Table 1. Truth table for input combinations for the tank example. The non-nominal combinations (due to plant features) are marked in red.

nontrivial in automation systems. As presented in Fig. 4, sensors, actuators and all other physical elements (except the controller) are considered as plant. A controller is designed to control the plant, and implemented according to a set of specifications.

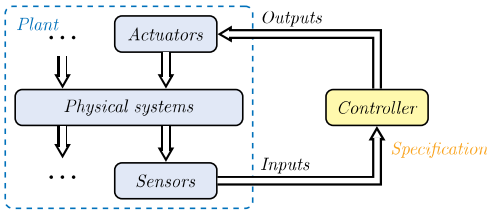


Fig. 4. Closed loop of plant and controller

By nature, the behavior of the plant is strongly influenced by the controller. However, as presented in Fig. 4, in a closed-loop system, the plant also restricts the reachable state space of the controller in normal operation. This leads to the idea of considering the closed-loop behavior also during test execution.

This hypothesis can be used to reduce the number of test cases by removing the cases that cannot occur in nominal behavior of a system, resulting in a reduced set of *meaningful* test cases. The undesired or unexpected behavior will not be tested during early test phases (or not at all if complete testing is not feasible). Given a tank with two level sensors: when the sensor indicating *Level High Reached* gives the value *True*, the sensor indicating *Level Low Reached* should normally not give the value *False*. A second example is a conveyor belt: if it does not run, the sensors for detecting the position of a workpiece should not change their value, since no workpiece has been moved and the sensors should not be triggered in a nominal situation. Such relations can be displayed in a truth table, indicating the possibility to reduce the number of input combinations as displayed in Tab. 1, where the combination of high and not low. Apparently, one out of four combinations can be neglected, leading to a reduction of 25% of test cases.

The aim of specifying relations between inputs as well as outputs and inputs is to reduce the set of states for which complete testing is performed. Effectively, this reduces the number of states and the input vectors. This results in fewer combinations of inputs that need to be imposed on the controller in open-loop testing. During composition, fewer states and evolutions need to be considered, which reduces the computational complexity.

4. CASE STUDY

In this paper, a logistics system is used as an illustrative example. The three subsystems are taken and adapted from the didactic platform presented in Jordan et al. (2017). The modules of interest in our case study are displayed in Fig. 5.

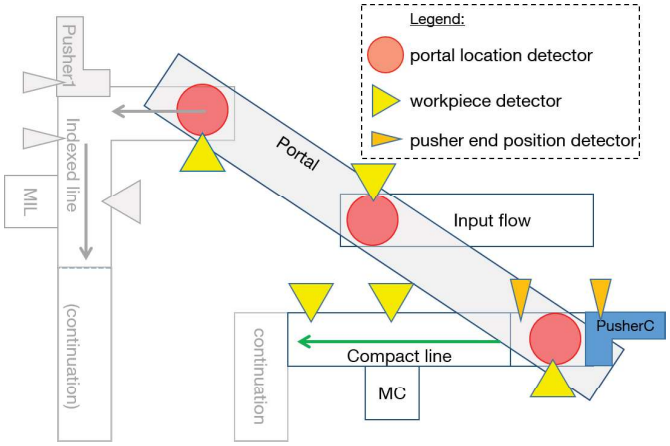


Fig. 5. Part of a logistics system containing a portal and two subsequent lines *(top view)*

4.1 System description

The *portal* transports workpieces from the *input buffer* to either the *compact line* or an indexed line. In this paper, the specification and plant behavior of the portal and compact line are analyzed and presented. The compact line contains a vertical buffer with a pusher, a conveyor belt and one machine station. Several location sensors are used to sense the position of the workpiece (yellow triangles), the pusher (orange longish triangles) and the portal (red circles).

Five FSM models have been used for the specification models of the system under consideration. In Fig. 6, three specification models for the portal and compact line are given as examples. [5] It is displayed, that the portal can move horizontally between three positions *In*, *IL* and *CL*. Only in those positions it can move up and down. Finally, only in the down end position it can activate the electromagnetic gripper to lift a workpiece or deactivate it (ungrip) in order to release a workpiece, respectively. On the compact line, a workpiece is brought to a machine via the belt; after the machining the workpiece is delivered to the output.

In total, 15 inputs and 9 outputs are considered, as listed in Tab. 2.

4.2 Applying DTT approach

For the sake of comparison, a monolithic composition of the system is performed resulting in an SCA with 221 states and 8524 evolutions.

After checking with DTT approach, all the 221 states contain non-SIC-testable parts; after adding T-guards to 8 transitions into the individual models, the system becomes fully SIC-testable.

In the SCA, 183 states suffer from observability issue. 5 O-actions are added to a set of locations in individual models, so that all states are directly distinguishable from each other.

[5] Initial specification models are drawn in black; T-guards are drawn in blue; O-actions are drawn in purple; the C-guard transitions are drawn in green, which permits to achieve 4 steps of controllability.

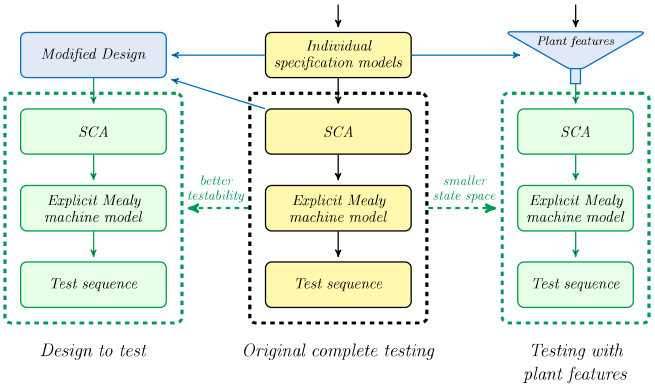


Fig. 2. Framework of complete testing, design to test, and testing with plant features methods

set. As a last step, a test sequence is generated by solving the Transition Tour problem of the set of minterms from all states and all input values.

3.2 Design to test approach

Several issues have been identified in the practice of complete testing regarding controllability, observability and SIC-testability. The design-to-test (DTT) approach aims to solve those issues by paying a limited effort on the modification of design, while keeping the nominal behavior during normal execution unchanged (Ma and Provost (2016)). SATE incorporates the algorithms of the 'DTT-MAT' toolbox.

A good design, which fulfills all functional requirements, is not always a good design with respect to testing. Two abstract Moore machine models (before and after the modification by DTT approach) are presented as examples in Fig. 3.

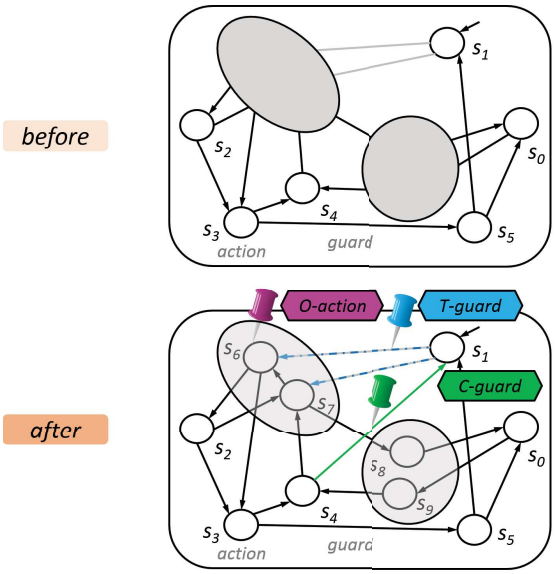


Fig. 3. Core idea of DTT approach: adding T-guards, O-actions and C-guards to modify the initial specification models

In the conformance testing of programmable controllers, controllability is a measure of whether and how fast the implementation can be brought from an (arbitrary) active state to another desired state. In Fig. 3, on the initial model, the shortest path from s_4 to s_1 is relatively long (with regard to the number of states in the example system), which represents a relatively bad controllability. The DTT approach solves this issue by adding a minimum number of extra controllable transitions, named as C-guard transitions, between some of the states (drawn with green color in the example), which can be used as shortcut transitions when set to *True*.

Observability concerns whether and how fast a state can be distinguished from other states. Apparently, in Fig. 3, the states in the two big gray circles cannot be distinguished directly by observation of system outputs in that moment, since they have the same outputs. The DTT approach solves this issue by adding extra observable actions, named as O-actions, to such states which suffer from the presented lack of observability.

SIC-testability issues occur in testing of controllers with cyclic execution mode, when several input signals are expected to change their values at the same time. Physically, multiple input changes (MIC) do not necessarily occur at the same time. Consequently, the changes might be read by the controller in different cycles. Then, the actual behavior may differ from the behavior under consideration that should be tested. Obviously, this would not be an issue in the case that the test sequence only contained single input changes between two successive steps, which is however hardly achievable in practical systems (Guignard and Faure (2014)). The DTT approach solves this issue by adding T-guards to the initial guards of transitions suffering from SIC-testability issues. All T-guards will be set to *False*, when multiple inputs should change at once, stopping the system at its current state such that there is enough time to stabilize the MIC. This enables to proceed as usual and make sure that the desired transition can be taken.

In summary, applying the design-to-test approach, the specification models are automatically analyzed by the tool; then, based on the need, a minimum number of C-guards, O-actions, and T-guards are automatically calculated and added to the models, so that the specification models fulfill the requirements of full SICtestability, full observability and better controllability.

3.3 Testing with plant features

With the DTT approach, complete testing can be done more effectively. However, for large scale systems, complete testing cannot always be achieved, because the number of test cases grows very rapidly to the complexity of a system, i.e. exponentially to the number of inputs and linearly to the number of states.

Therefore, a test generation method using plant features has been proposed for more efficient testing (Ma and Provost (2017)). The algorithms in SATE are based on the method in Ma and Provost (2017) but with improvements, i.e., applying plant features earlier in the generation of SCA rather than after the generation of explicit Mealy machine, which further reduces the state space of test generation. The concepts of plant and specification are

2. BACKGROUND

2.1 Communicating Moore machine with Boolean signals

In this paper, system specifications are modeled as communicating Moore finite state machines, adapted from Lee et al. (1996b).

Due to simplicity and a wide range of applications, Boolean signals are used as inputs and outputs in the illustration of the proposed method. However, the method can also be applicable to general digital signals with a few adaptations. An important thing to keep in mind is that, in contrast to event based models, where only one event can occur at a time, signal based models allow multiple changes of input values at once.

A communicating Moore machine extended with Boolean signals is defined by an 8-tuple $(L, l_{init}, I, C, O, G_\delta, \delta, \lambda)$ [1], where:

- L is a finite set of locations.
- l_{init} is the initial location, $l_{init} \in L$.
- I is a finite set of Boolean input signals.
- C is a finite set of internal Boolean communicating variables that are related to locations, a communicating variable is denoted as $X(l)$.
- O is a finite set of Boolean output signals.
- $G_\delta := expr(I, C)$ is a finite set of transition guards, which are Boolean expressions built up by input signals and communicating variables.
- $\delta : L \times G_\delta \to L$ is the transition function that maps the current location and transition guard to the next location; a transition is fired when its source location is active and its guard is evaluated as '1' (i.e. *True*).
- $\lambda : L \to 2^{O \cup C}$ is the output function that maps the locations to their corresponding output signals and communicating variables.

Moore machines are also represented in graphical form in this paper. A simple example is given in Fig. 1.

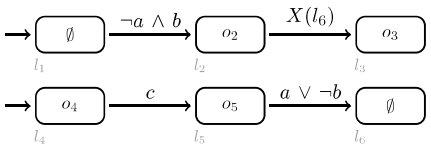


Fig. 1. A simple Moore machine example with Boolean signals

A location l is drawn as a rounded rectangle. A location can either have an externally observable action [2], e.g. o_2 in l_2; or no observable action, e.g. $\emptyset$ in l_1.

A transition δ is represented by a directed edge with its guard, e.g. $\neg a \wedge b$ for the transition from l_1 to l_2. The use of an internal communicating variable in transition guards is not complicated. For example, when the location l_6 is activated, $X(l_6)$ is then assigned the value '1'. If l_2 is active at that time, the transition from l_2 to l_3 can be fired.

[1] The subscript 'S' will be used to stand for *Specification*, the subscript 'P' for *Plant*: e.g. L_S and L_P mean the set of locations for specification and plant models

[2] For readability reasons, only active outputs are presented, i.e. in l_2, o_2 implicitly means $o_2 \wedge \neg o_3 \wedge \neg o_4 \wedge \neg o_5$.

2.2 Synchronous composition of individual models

Thanks to the use of internal communicating variables, interactions among individual parts of a system can be modeled conveniently. In order to validate the global behavior, individual models are first composed synchronously.

SATE's composition is based on the algorithms used by the tool 'Teloco' (Provost et al. (2011)), and the formalism introduced in Sec. 2.1 is extended with the following modifications:

- S is the set of states in a composed model. A state represents a combination of locations from the individual models.
- $G_e := expr(I)$ is a finite set of evolution guards [3], which are Boolean expressions built up by input signals.
- $e : S \times G_e \to S$ is the evolution function with stability search that maps the current state and evolution guard to the next state. A *transition* between states is named an *evolution*.

It is worth reminded that during the composition, a situation is stable if no transition in any of the Moore machines can be fired without changing the values of input signals; otherwise, it is transient. The stability search semantics implies that the firing of transitions continues until a stable situation is reached. The composed model contains only stable states, where only a change in the input values can trigger an evolution to another state. Therefore, the composed model is called *Stable Composed Automaton* (SCA) in this paper.

3. TEST CASE GENERATION METHODS

In this paper, the testing objective is to check whether an implemented programmable controller, seen as a black-box with inputs and outputs, behaves correctly with respect to its specifications. The execution of a testing process consists of three steps: feeding the input sequence to the controller, executing the program, comparing the observed output sequence to the expected one generated from specifications.

This paper focuses on the generation of test cases. As presented in Fig. 2, the process of *complete conformance testing* method is depicted in the center, while our *design to test* approach and methodology for *testing with plant features* are presented on the left and the right side, respectively.

3.1 Complete conformance testing

The basic complete conformance testing is structured and performed as follows. First, individual Moore machine models are modeled and composed into one global SCA. Afterwards, an equivalent Mealy machine is derived from the SCA, which explicitly represents all Boolean conditions of evolutions by a set of minterms [4] over the Boolean input

[3] A specific evolution guard is noted as $g_{e(I_S,s)}$, with regard to its source state and involved inputs

[4] A minterm is a basic element of an explicitly presented guard, e.g. if $g_{e(I,l)} = a \wedge \neg b$ and $I_S = \{a, b, c\}$, the corresponding minterms are $a \wedge \neg b \wedge c$ and $a \wedge \neg b \wedge \neg c$.

Preprints of WODES 2018
May 30 - June 1, 2018
Sorrento Coast, Italy

SATE: Model-Based Testing with Design-to-Test and Plant Features

Canlong Ma, Claudius Jordan, Julien Provost

Technical University of Munich, Munich, Germany
(e-mail: {canlong.ma, claudius.jordan, julien.provost}@tum.de).

Abstract: In this paper we present SATE, a tool aiming at increasing test efficiency of model-based testing of DES using two approaches: design-to-test and plant features. First, the design-to-test approach automatically modifies the design while maintaining the original system behavior to overcome controllability, observability and SIC-testability issues. Secondly, testing with plant features reduces the number of test cases taking into account restrictions on the input space of programmable logic controllers caused by the plant that is to be controlled.

Keywords: discrete event system, programmable logic controller, conformance testing, validation

1. INTRODUCTION

It is commonplace that industrial automation systems grow larger, and thus the programmable logic controller code grows in terms of complexity. This results in the demand for tools that enable testing such controllers efficiently. Many model-based techniques exist in literature to generate test cases for black-box testing; one of those is complete conformance testing (CCT). In order to show complete conformance of an implementation to its specification by means of testing, all possible combinations of inputs need to be evaluated for all states. Conformance testing of a programmable controller consists of three phases: test generation, test execution, and result verdict.

Here, we focus on the test generation phase in order to reduce the overall length of a test sequence, which is an artifact of this phase. It is a challenging endeavor to create a complete set of test cases for large scale systems as the number of test cases grows exponentially with the number of states and inputs. In addition to the before-mentioned state space explosion, in many cases CCT suffers from single-input-change-testability (SIC-testability) issues discussed in Provost et al. (2014).

During test sequences execution, the actual test case evaluation is performed after reaching a certain source state. Thus, additional computations have to be made to actually reach this desired state to perform the considered test step. Moreover, if the system evolution is not observable or the system state after the test step is not distinguishable from others, it has to be identified with state identification techniques, which demand further effort (Lee et al. (1996a)). These two actions, the homing and identification sequences, constitute a testing overhead, which can be remarkably big for larger systems. Consequently, reducing this testing overhead is an aim for our design-to-test (DTT) approach (Ma and Provost (2016)). To achieve this, the system design is modified automatically, introducing some design overhead with the goal of reducing the testing overhead discussed beforehand.

The second methodology addresses the reduction of the number of generated test cases in terms of reducing the controller input space in each system state, taking into account which outputs the physical plant can actually produce. The underlying hypothesis is that in the closed-loop, some inputs for the controller will never occur under nominal system behavior and thus can explicitly be neglected during testing.

Even limited knowledge about the system, i.e. a single plant feature, already leads to a reduction of test cases in comparison to the CCT approach. There are some common relations between inputs that can easily be identified such as *mutual exclusion* and *premise*. Similarly, relations between outputs and inputs can be found. Those features are presented and discussed in more detail in Ma and Provost (2017), where also templates are provided.

Other approaches to generate test cases automatically from models can be found in the literature. For example Enoiu et al. (2013) and Mani and Prasanna (2016) use a model checker to generate test suites based on Function Block Diagrams. In Bohlender et al. (2016) high coverage is realized more efficiently by symbolic execution. In contrast to the before-mentioned approaches, we consider full coverage taking into account the behavior of the system under consideration. Kormann and Vogel-Heuser (2011) create a reduced set of meaningful test cases to be executed in a simulated environment for hardware based component faults, whereas we focus on nominal system behavior.

In this paper, we present our tool, Stable Automaton-based TEsting (SATE), that implements the two approaches that aim for more effective and shorter test sequences. In the next section, necessary background on communicating Moore machines and their synchronous composition are recalled. In Sec. 3 the framework and actual implementation are discussed accompanied by a case study illustrating the effectiveness of the presented tool in Sec. 4. This work is concluded with some remarks on potential future work in the last section.

local controllers could also be used to call legacy code or automatic control functions.

Listing 2. Local controller implementation

```
IF LC_M1 THEN
        ton_M1 (IN:=TRUE, PT := T#1000MS);
        output.M1 := TRUE;
END_IF
ton_M1(IN:=timer_ls23.IN);
IF ton_M1.Q = TRUE THEN
        ton_M1(IN := FALSE);
        output.M1 := FALSE;
        VS_M1done := TRUE;
END_IF
```

Listing 2 depicts an example where the output `M1` should be active for one second. After this time, an uncontrollable virtual sensor event is emitted to notify the supervisor about the finished task.

Listing 3. Virtual sensor implementation

```
IF FE_ls THEN
        ton_vs(IN:=TRUE, PT := T#1000MS);
END_IF
ton_vs(IN:=ton_vs.IN);
IF ton_vs.Q = TRUE THEN
        ton_vs(IN := FALSE);
        virtualsensor.VS_ls := TRUE;
END_IF
```

Virtual sensors allow the user to emit uncontrollable events at predefined conditions. The conditions can be arbitrarily defined, a common implementation is depicted in Listing 3. In this case, the virtual sensor is emitted one second after the occurrence of `FE_ls`.

6. CONCLUSION

This paper has outlined a framework to control a physical plant using PLC code automatically generated from a symbolic, modular supervisor. It is used, coupled with a didactic plant, as a teaching platform to introduce students to model-based approaches and SCT. During this course the implementation proved to be reliable, and the integration of modeling and execution on the same platform enables fast and frequent modifications. In contrast to the previously-used implementation, larger supervisors and more complex systems can be considered.

There are still two major issues for the implementation of supervisory controllers on PLC:

- PLCs are signal-based, whereas the SCT is event-based. Events must be generated from signals and afterwards converted back to signals. This introduces a delay between plant and controller.
- The SCT assumes controllable events to be spontaneously generated. In practice, the controller has to implement a choice algorithm and guarantee fairness and/or determinism.

In the future, this framework will be extended and potentially adapted for an industrial-sized problem. An interesting extension would be to adopt a signal-interpreted approach, as introduced by Fouquet and Provost (2017).

The Python executable for the automatic code generation can be found on the chair website: `www.ses.mw.tum.de` .

REFERENCES

Balemi, S. (1992). *Control of discrete event systems: theory and application*. Ph.D. thesis, Automatic Control Laboratory, Swiss Federal Insitute of Technology.

Dietrich, P., Malik, R., Wonham, W.M., and Brandin, B.A. (2002). Implementation considerations in supervisory control. In *Synthesis and Control of Discrete Event Systems*, 185–201. Springer, Boston, MA.

Fabian, M. and Hellgren, A. (1998). PLC-based implementation of supervisory control for discrete event systems. In *Proceedings of the 37th IEEE Conference on Decision and Control*, volume 3, 3305–3310 vol.3.

Fouquet, K. and Provost, J. (2017). A Signal-Interpreted approach to the supervisory control theory problem. *IFAC-PapersOnLine*, 50(1), 12351–12358.

IEC (2014). Programmable controllers – part 3: Programming languages. Technical Report IEC 61131-3:2013.

Jordan, C., Ma, C., and Provost, J. (2017). An educational toolbox on supervisory control theory using MATLAB simulink stateflow: From theory to practice in one week. In *2017 IEEE Global Engineering Education Conference (EDUCON)*, 632–639.

Junior, M. and Leal, A.B. (2012). Modelling and implementation of supervisory control systems using state machines with outputs. *Manufacturing System*.

Leal, A.B., da Cruz, D., and Hounsell, M.S. (2012). PLC-based implementation of local modular supervisory control for manufacturing systems. *Manufacturing System*.

Li, Y. and Wonham, W.M. (1987). On supervisory control of Real-Time Discrete-Event systems. In *1987 American Control Conference*, 1715–1720. ieeexplore.ieee.org.

Malik, R., Åkesson, K., Flordal, H., and Fabian, M. (2017). Supremica–An efficient tool for Large-Scale discrete event systems. In *The 20th World Congress of the International Federation of Automatic Control, Toulouse, France, 9-14 July 2017*.

Miremadi, S., Akesson, K., and Lennartson, B. (2011). Symbolic computation of reduced guards in supervisory control. *IEEE Transactions on Automation Science and Engineering*, 8(4), 754–765.

Ramadge, P.J. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1), 206–230.

Vieira, A.D., Santos, E.A.P., de Queiroz, M.H., Leal, A.B., de Paula Neto, A.D., and Cury, J.E.R. (2017). A method for PLC implementation of supervisory control of discrete event systems. *IEEE Transactions on Control Systems Technology*, 25(1), 175–191.

Zaytoon, J. and Riera, B. (2017). Synthesis and implementation of logic controllers – a review. *Annual reviews in control*, 43(Supplement C), 152–168.

event is disabled in the active state set, i.e. evaluating a set of Boolean guards generated from the states restricting this event. For the controllable events, the synchronization determines whether this event will be taken or not. If an uncontrollable event is detected in a state where the synchronization disables it, the controller exits to the error mode where it will stay until the plant is restarted.

5. CASE STUDY

The framework presented in this paper is used to teach SCT to student groups. A didactic platform is separated into subsystems and each group is in charge of implementing the controller for their subsystem. This course is based on a previously held course described by Jordan et al. (2017). Since then the physical plant remained unchanged, but the control architecture has been renewed.

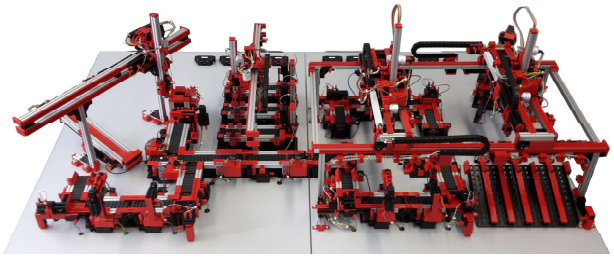


Fig. 4. Didactic Platform

5.1 Didactic Platform

The didactic platform (Figure 4) consists of two separable parts. The first part is a combination of assembly lines serviced by two 2-axis portal cranes. The portal cranes transport workpieces between different stations, while the assembly lines move the workpieces with conveyor belts and use a variety of tools on them.

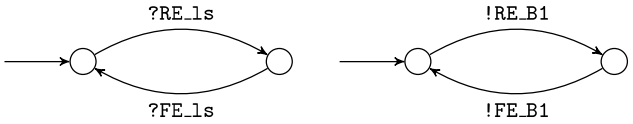
The second part revolves around two 3-axis portal cranes with overlapping domains, transporting workpieces between two assembly lines, a welding robot, and a storage.

Modularization of the platform The two parts are further divided into subsystems to simplify the modeling and to allow students to work independently. Each subsystem with 10 to 40 Boolean inputs and outputs is controlled by a remote input/output module, connected via EtherCAT to a Laptop running a Beckhoff TwinCAT Soft PLC.

Subsystem interaction The subsystems have interfaces where workpieces are transported from one subsystem to another. These interfaces require the cooperation of multiple student groups and thus cause the most difficulties. This high-level control is implemented by two global controllers, interacting with the subsystems through electrically connected inputs and outputs. The communication protocols usually require a subsystem to request clearance from the global controller.

5.2 Modeling Procedure

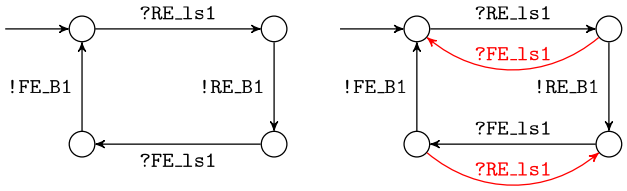
Each controller is synthesized based on a set of plant and specification models. They are implemented in Supremica (Malik et al. (2017)) and the symbolic, modular supervisor is generated with the built-in functions.



(a) Lightsensor (b) Conveyor Belt

Fig. 5. Example plant models

Plant models Plant models describe the behavior of the physical plant by restricting the language to what is physically possible. These models can be of varying detail. A common restriction for sensors and actuators is, for example, that rising and falling edges must occur alternatingly. Figure 5 displays a set of plant models for a light sensor (`ls`) and a conveyor belt (`B1`).



(a) (b)

Fig. 6. Possible conveyor belt specifications

Specification models The specification models restrict the capabilities of the plant to fit the requirements of the user. For example, a requirement might be that after an uncontrollable event occurs, an action is performed. Figure 6 displays two different specifications for a common cyclical execution: After a rising edge is detected, a belt is started. When the falling edge occurs, the belt is stopped. The plant models in Figure 5 can be used for this example.

When the students are modeling their first specifications, they usually model them similarly to the specification in Figure 6a and thus run into controllability issues because of disabled uncontrollable events. The specification in Figure 6b avoids these issues and recovers controllability. On the other hand, it might not be desirable to allow the removal of a piece before the belt is started. In this case, the plant model has to be modified to further restrict the occurrence of `RE_ls1` and `FE_ls1`.

Correctly modeling the plant behavior on the first try is hard. There has to be a balance between permissiveness, detail and solution independence. A very unrestrictive and solution-independent plant model will allow unrealistic event combinations and impede specification design. Plant models with too much detail will favor a specific solution and may conflict with the real behavior.

Local controllers and virtual sensors *Local controllers* are used to control outputs that should not be interrupted by the supervisor. When a combination of controllable actions should be executed in a predefined order and with strict timing constraints, the synthesized controller can not be trusted to take the right decisions. In this case, local controllers allow the implementation of arbitrary functions that are enabled as a controllable event. In addition,