A Signal-Interpreted Approach to the Supervisory Control Theory Problem

Kevin Fouquet, Julien Provost

Technical University of Munich, Safe Embedded Systems,
85748 Garching bei München, Germany
(e-mail: provost@ses.mw.tum.de)

Abstract: After highlighting the difficulties encountered when implementing a supervisor on a real controller and the limitations of existing solutions to handle them, this paper presents a *signal-interpreted approach* to the Supervisory Control Theory problem. Due to the differences between event- and signal-based approaches, new algorithms are introduced to apply a Supervisory Control Theory (SCT) approach on the basis of signal-interpreted Boolean Finite Automata extended with variables (EBFAs). The advantages of the proposed signal-interpreted approach are illustrated both on simple structures throughout the paper and on a case study at the end.

Keywords: Discrete Event Systems, Supervisory Control Theory, Automata, Finite State Machine, Programmable Logic Controllers

1. INTRODUCTION

Supervisory Control Theory (SCT) is a model-based approach that aims at automatically generating a supervisor, given a set of Discrete Event System (DES) models representing the uncontrolled plant behavior and the specifications to be fulfilled. Since the foundational work of Ramadge and Wonham (1987), SCT has been the subject of many theoretical results and it was expected to have an important impact in the industrial field. Yet, as earlier noticed by Fabian and Hellgren (1998) and Roussel and Giua (2005), and more recently by Vieira et al. (2017) and Zaytoon and Riera (2017) the use of SCT in industry is still not wide-spread. Some reasons for this are the problems encountered when implementing supervisors on industrial controllers such as Programmable Logic Controllers (PLCs).

As initially described in Fabian and Hellgren (1998), there is no actual rule on how to implement a supervisor. SCT relies on the automata theory and thus on an event-based formalism. Yet, when implementing it, most of the physical systems such as Programmable Logic Controllers (PLCs) are considered with Input/Output signals. Consequently, many problems can arise: The SCT framework handles events, while a PLC implementation handles I/O signals. Thus, to be consistent with the event-based formalism, the system has to be modeled using rising and falling edges of signals instead. Yet, events are asynchronous while a PLC runs cyclically: reading the input signals, executing the code, and updating the output signals. Therefore, several rising or falling edges can occur at the same time, which violates the fundamental definition of events introduced in the automata theory: *any two or more events cannot occur simultaneously.*

As detailed in Section 2.1, many previous works have proposed improvements to the SCT modeling framework or to its implementation on real controllers. Yet, to the best of our knowledge, all these works rely, at least partly, on event-based models.

The signal-interpreted approach proposed in this paper uses Boolean Finite Automata extended with variables (EBFAs), which handle exclusively I/O signals instead of events. This makes its implementation on signal-based controllers such as PLCs easier by avoiding the need to adapt an event-based approach to a signal-based environment.

The remainder of this paper is organized as follows. Related works and the main differences between signal-interpreted and event-based approaches are presented in section 2. The definition of an EBFA system and the generation of its Stable Location Automaton are introduced in section 3. After a presentation of the proposed methodology in section 4, the SCT algorithms developed for EBFA models are explained and illustrated in section 5. In section 6, a case study illustrates the advantages of the proposed signal-based approach over an event-based one. Finally, conclusion and future works are drawn in the last section.

2. BACKGROUND

2.1 Related works

One main drawback of event-based approaches is that events are assumed to occur asynchronously. This leads to the difficulty to represent conditions which depend not only on a single event but on several independent events. In practice, for large-scale systems with many concurrent processes –and for which the order of occurrence of some events is not relevant–, as well as for sampled systems, several rising or falling edges of signals can occur at the same time. With a basic event-based approach, such a system must be described by a model which represents all possible combinations of events or sequences of events. Consequently, this model will possess a large amount of "non-functional" states. With a signal-interpreted model, using Boolean conditions on signals permits to model the same system with fewer states. Fig. 1 gives an example of a condition where both a and b must be $True$. The signal-interpreted model requires only 2 states and 1 transition while the event-based model needs 4 states and 6 transitions. For complex conditions, depending on more inputs, the event-based model state space would grow exponentially.

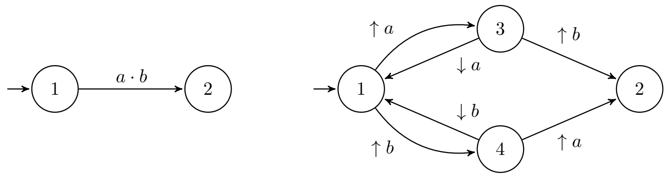


Fig. 1. A condition between two states 1 and 2 expressed using signals (left) and using events (right)

Regarding modeling formalisms, models extended with variables, such as Extended Finite Automata (EFA) and (EFSM), have been proposed by Skoldstam et al. (2007) and Chen and Lin (2000), respectively. These formalisms permit to define more compact models and, in turn, also to ease their readability. Yet, in these models, the transitions are still triggered by events while variables are used on top of an event either to define a guard and/or an action.

So far, symbolic SCT approaches have found limited applications. Le Gall et al. (2005) proposed a method for systems modeled by symbolic transition systems. However, this method does not guarantee maximal permissiveness. Later, Ouedraogo et al. (2010) proposed another method to generate a maximally-permissive supervisor from EFA models. Recently, Miremadi et al. (2012) proposed a method to efficiently represent EFA and compute a supervisor by direct manipulations of Binary Decision Diagrams (BDDs). Miremadi and Lennartson (2016) also investigated on-the-fly symbolic synthesis to reduce the state-space explosion problem during the reachability search. These methods generate a supervisor by computing more restrictive guard conditions to be applied to the transitions of the EFA plant models.

Regarding PLC implementations, as earlier presented by Fabian and Hellgren (1998) and Balemi and Brunner (1992), some properties, such as *delay insensitivity* and *interleave insensitivity*, have to be fulfilled by the system under control to avoid implementation problems. These properties ensure a consistent implementation on PLC, but also limit the variety of models that can be implemented. Several methods have been proposed from (Fabian and Hellgren (1998)) to Leal et al. (2012) and Vieira et al. (2017). However, even if recent methods permit to ease the PLC implementation of a supervisor they still do not verify automatically if the above can be fulfilled (Vieira et al. (2017)). Thus, the problem of using event-based models remains and PLC implementation of SCT supervisors is still an open problem.

2.2 Signal-interpreted and event-based models

In both the proposed EBFA and the commonly used event-based formalisms, states represent the same concept. This is also valid for the concepts of *marked* and *forbidden* states. The main difference between the two formalisms lies in the definition and the interpretation of the transitions.

Regarding these transitions, the difference between the two formalisms is that, in the event-based approach, events are listed in an *alphabet*, and each transition is given only one event. When this event occurs, the transition is immediately fired and the model switches from a location to another one.
With the signal-interpreted approach, a transition is not directly linked to one signal but to a Boolean function composed of several signals, called a *condition* or a *guard*. Also, according to the signal-interpreted semantics, self-loops are implicitly defined: for a given valuation of the signals, if no outgoing transition can be fired then the active state remains active.

Also, key feature of signal-interpreted models such as Signal-Interpreted Petri Nets (SIPN) or Grafcet is their *stability search* algorithm. This implies that after a change of the signals' valuations, several evolutions may occur in the model. These evolutions must be performed until a stable situation is reached. A formal definition of an EBFA system and its rules of evolutions with stability search are given in the next subsections.

For those who would prefer to consider a model without stability search, this is also possible as long as the flatten model satisfies the properties P5 to P8 of a Stable Location Automaton given in subsection 3.2.

3. SIGNAL-INTERPRETED BOOLEAN FINITE AUTOMATA'S DEFINITION

To suit the SCT framework, the definition of Boolean automata, introduced by Leiss (1981), is further defined here. An *EBFA system* is a set of connected deterministic *EBFAs* running in parallel, each of them described by a 6-tuple: $EBFA = \langle Q, q_0, S, \Delta, M, X \rangle$ with:

- Q a non-empty set of states,
- q_0 a non-empty set of initial states, $q_0 \subset Q$,
- S a non-empty set of Boolean signals,
- Δ a set of transitions, each transition being defined by $\delta = \langle q_{Src}, \delta_{Cond}, q_{Dest} \rangle$ with:
  - · q_{Src} a single state in Q,
  - · δ_{Cond} its firing condition, a Boolean function [1] in S,
  - · q_{Dest} a single state in Q.
- M a non-empty set of marked states, $M \subseteq Q$,
- X a set of forbidden states, $X \subset Q$, X may be empty.

Each EBFA also satisfies the following properties:

P1 For any transition $\delta \in \Delta$, $q_{Src}(\delta) \neq q_{Dest}(\delta)$.
Self-loops are implicitly defined: for a given valuation of the signals, if no outgoing transition can be fired then the active state remains active.

[1] "·", "+" and "$\overline{}$" correspond to the logical operators AND, OR, and NOT, respectively

P2 A forbidden state should not have any outgoing evolution.
This property is needed for EBFAs because an evolution between two stable locations may go through a transient location that contains a forbidden state.

The set of signals S can be subdivided into three categories. These categories are similar to the I/O events categories introduced in Balemi et al. (1993):

S_I: Input signals which represent signals the supervisor can only read, i.e. the supervisor cannot control their values. It typically corresponds to sensors' signals but could also be uncontrollable actuators commands.
S_O: Output signals which represent signals the supervisor can set or reset, i.e. the supervisor can control their values. It typically corresponds to actuators command's signals.
S_L: Internal signals are signals only the supervisor can control and read.

Also, S_I, S_O and S_L are defined such that: $S = S_I \cup S_O \cup S_L$, $S_I \cap S_O = \emptyset$, $S_O \cap S_L = \emptyset$ and $S_I \cap S_L = \emptyset$.

S may contain only Input or only Output signals. Yet, in general, the absence of output signals would lead to an empty supervisor as the supervisor would not be able to control any signals leading to a forbidden state.

For those who would prefer to consider an EBFA system where only one state is active at a time in each EBFA, the following properties have to be satisfied too:

P3 Each EBFA has only one initial state: $|q_0| = 1$.
P4 For any two transitions from the same source state, their conditions must be mutually exclusive:
$\forall(\delta_1, \delta_2) \in \Delta^2 : \delta_1 \neq \delta_2$,
$q_{Src}(\delta_1) = q_{Src}(\delta_2) \Rightarrow \delta_{cond}(\delta_1) \cdot \delta_{cond}(\delta_2) = 0$

3.1 Evolution rules

The evolution rules of an EBFA system are similar to those of SIPN and Grafcet (see Frey and Litz (1998) and Provost et al. (2011a) for more formal details):

R1 At the initialization, all the initial states are active. All the other states are inactive.
R2 A transition is enabled when its source state is active. A transition is fireable when it is enabled and when its transition condition is *True*. A fireable transition must be immediately fired.
R3 Firing a transition provokes simultaneously the deactivation of its source state and the activation of its destination state.
R4 When several transitions are simultaneously fireable, they are simultaneously fired.
R5 When a state shall be both activated and deactivated by applying the previous evolution rules, it is activated if it was inactive, or remains active if it was previously active.
R6 The previous rules are iterated until a stable location is reached (i.e. until no further transition can be fired given the current valuation of the signals).
Since the firing of a transition is supposed to take no time, iterated firing is interpreted as simultaneous.

According to the signal-interpreted semantics, several signals and conditions can be *True* at the same time, and for an unknown period. First, this implies that several transitions can be fired at once in each EBFA of an EBFA system (rules R2 and R4). Secondly, this also implies that several transitions can be sequentially fired without changes of the I/O signals until a stable location is reached (rule 6). The locations activated and deactivated during the search for a stable location are said *transient* and have no impact on the stable behavior. Thus, the use of a stability search algorithm avoids the presence of undesired avalanche (or cascade) effects in the implementation.

The generation of a Stable Location Automaton that represents the stable behavior of an EBFA system according to the above mentioned rules is given in the next subsection.

3.2 Stable Location Automaton generation

The Stable Location Automaton (SLA) formal definition and generation algorithms for an EBFA system are similar to those for a Grafcet (see Provost et al. (2011a) for more formal details). Thus, the TELOCO software (see Provost et al. (2011b)) has been slightly adapted for EBFAs.

Since the SLA model will be used as a basis for the proposed SCT algorithms, its main concepts are reminded here.

An SLA represents only the *reachable* and *stable* states of an EBFA system and the possible evolutions (i.e. sequences of simultaneous firings of transitions) between these states. If, during the generation of the SLA, a cycle is detected between two reachable locations, the EBFA system is considered as inconsistent and no SLA is generated.

A deterministic SLA can be described by a 6-tuple:
$SLA = \langle L, l_0, S, \Delta, L_M, L_X \rangle$ with:

- L a non-empty set of stable locations, each location being defined by $l = \langle Q_{Act}, L_{StabCond} \rangle$ with:
  · Q_{Act} a set of active states in the EBFA,
  · $L_{StabCond}$ its stability condition, a Boolean function in S.
- l_0 the initial unstable location, $l_0 \notin L$,
l_0 corresponds to the set of all initial states of the EBFA. This location is unstable because the stable location reached after the initialization will depend on the I/O signals valuations at that moment.
- S a non-empty set of I/O signals,
S is composed of I/O signals only ($S = S_O \cup S_I$).
EBFA's local signals' values are assigned and evaluated during the SLA generation.
- Δ a set of evolutions, each evolution being defined by $\delta = \langle l_{Src}, \delta_{Cond}, l_{Dest} \rangle \in (L \cup l_0) \times \mathbb{B} \times L$.
- L_M a set of marked locations, $L_M \subseteq L$,
- L_X a set of forbidden locations, $L_X \subset L$.

The SLA of an EBFA system is constructed iteratively, starting from the initial location, and adding reachable locations until a fixed-point is reached. By definition and construction of an SLA (and independently of whether the properties P3 and P4 have been used or not), the following properties are satisfied:

P5 Any location of an SLA is reachable from the initial location l_0.

P6 The stability condition of a location – its implicit self-loop – is defined as the Boolean negation of all the outgoing evolutions' conditions from this location.

P7 For any two outgoing evolutions from the same location, their conditions are mutually exclusive: $\forall(\delta_1, \delta_2) \in \Delta^2 : \delta_1 \neq \delta_2$,
$l_{Src}(\delta_1) = l_{Src}(\delta_2) \Rightarrow \delta_{cond}(\delta_1) \cdot \delta_{cond}(\delta_2) = 0$

P8 There can be at most one evolution between any two locations: $\forall(\delta_1, \delta_2) \in \Delta^2 : \delta_1 \neq \delta_2$,
$l_{Src}(\delta_1) = l_{Src}(\delta_2) \Rightarrow l_{Dest}(\delta_1) \neq l_{Dest}(\delta_2)$

Fig. 2 gives an instance of an EBFA system, composed of three EBFAs, and its SLA. As illustrated on the SLA, the initial location l_0 is unstable as at least one of its outgoing evolution would be *True* ($\overline{a} + a \cdot b + a \cdot \overline{b} = 1$). Thus, depending on the values of a and b at the initialization, the location reached after initialization will be $(12, 21, 31)$, $(11, 22, 31)$ or $(11, 23, 31)$. Also, according to the signal-interpreted semantics, if, for instance, the location $(12, 21, 31)$ is the active location and the output signal a is currently *False*, it is possible to stay in this location forever, irrespective of the value of the input signal b.

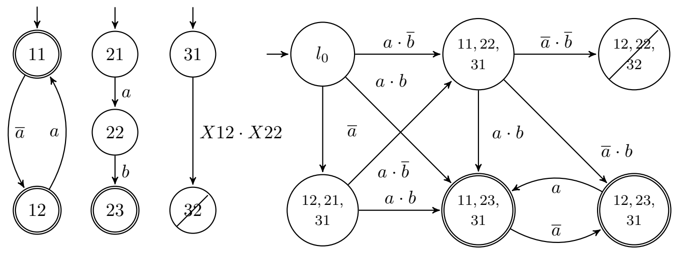


Fig. 2. An EBFA system composed of three EBFAs (on the left) and its SLA (on the right) [2]

4. METHODOLOGY

4.1 Banning occurrence of signals

In event-based models, the absence of a transition for a given event means that this event cannot occur (for a plant model) or should be forbidden (for a specification model). Thus, when needed, self-loops have to be explicitly defined. In signal-interpreted models, as previously mentioned, self-loops are implicitly defined. This implies that deleting a transition (resp. an evolution) or restricting its Boolean condition will not prevent a signal to change its value since this will only loosen its implicit self-loop Boolean condition. Thus, the banning of signals has to be explicitly defined using forbidden states (resp. locations). The condition on signals to be banned needs to be added to the guard of the transition (resp. evolution) leading to a forbidden state (resp. location).

From our point of view, the systematic use of forbidden states (in the plant and specification EBFA models) to ban a specific condition on signals eases their understanding. Forbidding a value of one signal or a condition on several signals is then explicitly defined, while in the event-based approach, the absence of a self-loop for a given event may either mean that this event should be forbidden from this state, or that given the other plant and specification models, this event cannot occur from this state and it is useless to represent it. Moreover, most of the time, only a small subset of combinations of signals needs to be explicitly forbidden from each state. It is consequently easier to write the combinations of signals which have to be forbidden in a transition guard rather than writing all the possible combinations which are permitted in self-loops, as described in Kumar et al. (1991).

4.2 Marked and forbidden locations of an SLA

In order to apply the SCT approach on the basis of plants and specifications EBFA models, the SLA generation algorithms have been adapted to take into account marked and forbidden locations. As a location of an SLA represents a combination of several active states of the EBFA system, the following rules are added to the SLA generation algorithms:

R7 A location is marked if and only if all of its active states are marked.

R8 A location is forbidden if and only if at least one of its active states is forbidden.

Thanks to the rule R8 and to local signals, it is also possible to define forbidden location in a modular way.
In the example given in Fig. 2, none of the states of the two first EBFAs are forbidden. If one wants to forbid the fact that the states 12 and 22 to be both active at the same time, one can add the third EBFA and use two local signals $X12$ and $X22$ (which are *True* when the states 12 and 22 are active, respectively, and *False* otherwise) to synchronize these EBFAs. This additional EBFA does not impact negatively the composed model state space since these two states are fully dependent on the current states of the two other EBFAs.

4.3 Evolution to and from forbidden location

As explained in subsection 4.1, forbidden states and locations are useful to specify which combinations of signals should be banned. Since a forbidden location is a location that should not be reached, there is no need to consider the possible outgoing evolutions from a forbidden location during the SLA generation. Thus, when a forbidden location is encountered during the SLA generation, the search algorithm stops the exploration of the current evolution and sets its stability condition to *True* (i.e. once in a forbidden location, whatever the signal valuations, this forbidden location will remain active). Also, since all forbidden locations of an SLA are equivalent (since they have the same transition function: an implicit self-loop) they could be merged into one. This leads to the following SLA property:

P9 There can be at most one forbidden location:
$|L_X| \leq 1$

These simplifications, also described in the Algorithm 1 (lines 4 to 19), are done symbolically on-the-fly using BDDs, which permits to save time and memory during the SLA generation.

[2] For legibility reason, the stability condition of each location is not represented in this figure. It can be calculated accordingly to **P6**

5. ALGORITHMS

In this section, the SCT algorithms developed for EBFA models are explained and illustrated. They lead to a non-blocking, controllable SLA, the supervisor. Futur works could consider improvement of the efficiency of these algorithms.

SCT algorithms are applied on the SLA obtained from the plants and specifications EBFAs. Thus, the term *evolution* refers to a sequence of simultaneous firings of transitions in the plants and specifications EBFAs but corresponds in the SLA to one transition from one stable location to another one.

5.1 Controllable and uncontrollable signals

Similarly to the event-based approach, distinction between controllable signals and uncontrollable ones is needed. In the remainder of the paper, the input signals S_I will be considered as uncontrollable while output signals S_O correspond to the controllable ones. As previously mentioned, an SLA does not contain any local signal: they have been assigned and evaluated during the SLA generation.

5.2 Controllability of an evolution

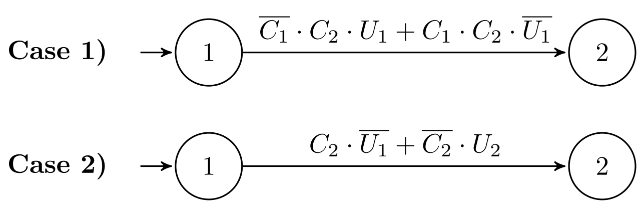
By definition, an evolution is said *controllable* if and only if it is possible to prevent its firing by acting only on controllable signals. Consequently, determining if an evolution is controllable or not, corresponds to determining if it exists at least one combination of controllable signals such that this evolution condition can be set to *False*.

However, in order to ensure the minimal-restrictiveness of the proposed SCT approach, determining if an evolution is *controllable* is not sufficient. It is also necessary to determine its *controllability condition*, defined as a Boolean condition representing all combinations of signals permitting to prevent its firing.

Determining the *controllability condition* of a Boolean evolution condition, which consists in determining all Boolean conditions making it *False*, can be efficiently solved using all-solutions SAT solvers (see Yu et al. (2014); Toda and Tsuda (2015) for more details). The negation of a Boolean evolution condition, denoted $\overline{\delta_{Cond}}$, is given in conjunctive normal form (`CNF( )`) as an input to an all-solutions SAT solver (`All-SAT( )`) which aims at finding all Boolean assignments of the controllable signals satisfying $\overline{\delta_{Cond}}$, independently of the values of the uncontrollable signals. These Boolean assignments are also represented symbolically as the Boolean *controllability condition* $ctrl_{Cond}$. If no assignment can be found, then the *controllability condition* is *False* and the considered evolution condition is said *uncontrollable*.
This can be expressed as follows:

$$ctrl_{Cond}(\delta_{Cond}) \leftarrow \texttt{All-SAT}\left(\texttt{CNF}\left(\overline{\delta_{Cond}}\right), S_{Ctrl}\right)$$

To better understand the goal of this algorithm, two examples are presented in Fig. 3. Both cases represent a single evolution from a location 1 to a location 2. C_1 and C_2 are controllable signals while U_1 and U_2 are uncontrollable signals.



Case 1)
1
$\overline{C_1} \cdot C_2 \cdot U_1 + C_1 \cdot C_2 \cdot \overline{U_1}$
2
Case 2)
1
$C_2 \cdot \overline{U_1} + \overline{C_2} \cdot U_2$
2

Fig. 3. Example of a controllable evolution (Case 1) and an uncontrollable evolution (Case 2)

Table 1 gives the evaluation of the evolution conditions for the cases 1 and 2 for the different values of their respective controllable signals. For the case 1, it can be observed that the evolution condition δ_{Cond} is *False* if and only if C_2 is set to *False*, while for the case 2, no matter the value of C_2, the value of the evolution condition δ_{Cond} will always depend on uncontrollable signals. Therefore, in the case 1, the evolution is controllable and its controllability condition is $\overline{C_2}$, while in the case 2, the evolution is not controllable and thus its controllability condition is *False*.

Table 1. Controllability evaluation applied to Case 1 (left) and Case 2 (right)

C_1	C_2	δ_{Cond}	$\overline{\delta_{Cond}}$
0	0	*False*	*True*
0	1	U_1	$\overline{U_1}$
1	0	*False*	*True*
1	1	$\overline{U_1}$	U_1

C_2	δ_{Cond}	$\overline{\delta_{Cond}}$
0	U_2	$\overline{U_2}$
1	$\overline{U_1}$	U_1

5.3 Applying SCT algorithms

Given an SLA, generated according to the rules defined in the subsections 4.2 and 4.3, solving the SCT problem consists in solving the controllability and the non-blocking problems.

A new method and algorithms are proposed in this section in order to generate a supervisor from a (signal-interpreted) SLA. This method is explained in the pseudo-code Algorithm 1 and will be illustrated through an example given in Fig. 4.
In the example given in Fig. 4, marked locations are represented with double circles and the forbidden location is represented with a crossed circle. The location 1 is reachable from the initial location, the locations 1, 3, 6 and 7 are marked (and thus co-reachable), the location L_X is the forbidden location. All locations are reachable only through the evolutions depicted in Fig. 4.

Considering the non-blocking problem, the first steps of the method consist in searching for non-co-reachable locations (see Algorithm 1, lines 1 to 3) and merging them into the forbidden location L_X (lines 4 to 19). Once these steps are performed, the SLA will contain only reachable and co-reachable locations and the forbidden location L_X. At this point, this SLA is non-blocking (or empty) but may still be non-controllable.
In the example given in Fig. 4 a), the locations 2 and 5 are not marked but are co-reachable: they should be kept; while the locations 4 and 8 are not co-reachable: they should be merged into the forbidden location L_X. After the execution of the Algorithm 1, the SLA presented in Fig. 4 b) is obtained.

Considering the controllability problem, since the SLA has only one forbidden location, this problem can be solved iteratively from the forbidden location L_X (see Algorithm 2, line 2). The first step of this iterative algorithm is to check the controllability condition of an evolution leading to the forbidden location L_X (line 5).
On the one hand, if an evolution leading to the forbidden location L_X is non-controllable, its source location is merged into the forbidden location L_X (lines 8 to 12). Then, locations that are reachable only from the source of this evolution are removed, as well as their incoming evolutions (lines 13 to 20). This operation also guarantees that the non-blocking property will be preserved. Finally, if two evolutions from a same state are leading to the forbidden location, one of them is removed and their guards are combined as a Boolean disjunction (lines 21 to 26).
On the other hand, i.e. an evolution leading to the forbidden location L_X is controllable, its guard is replaced by the negation of its controllability condition (line 28). Other evolutions leaving from the same source location see their guard combined as a Boolean conjunction with this controllability condition (line 31). As a consequence, some evolution guards may become *always False* (line 32). Such evolutions have to be removed and their destination check for reachability (line 35), similarly to the first case. This is repeated either until all evolutions leading to the forbidden location are controllable or until all locations have been merged into the forbidden location.

Algorithm 1: Initial non-blocking solving

```
Data: an SLA satisfying P9
Result: a non-blocking SLA satisfying P9

1  Co-reachability analysis:
2  $L_{Co} \leftarrow$ BackwardReach($L_M$)
3  ▷ All co-reachable locations have been identified
4  Forbidding all non-co-reachable locations:
5  foreach $l \in L \setminus (L_{Co} \cup L_X)$ do
6      foreach $\delta \in \Delta$ do
7          if $l_{Src}(\delta) = l$ then
8              $\Delta \leftarrow \Delta \setminus \{\delta\}$
9              break
10         else if $l_{Dest}(\delta) = l$ then
11             if $\exists \delta' \in \Delta : l_{Src}(\delta') = l_{Src}(\delta) \wedge l_{Dest}(\delta') = L_X$ then
12                 $\delta_{Cond}(\delta') = \delta_{Cond}(\delta') + \delta_{Cond}(\delta)$
13                 $\Delta \leftarrow \Delta \setminus \{\delta\}$
14             else
15                 $l_{Dest}(\delta) \leftarrow L_X$
16             break
17     $L \leftarrow L \setminus \{l\}$
18 ▷ All non-co-reachable locations have been merged into the forbidden location $L_X$
19 ▷ The SLA now contains only co-reachable locations and the forbidden location $L_X$
```

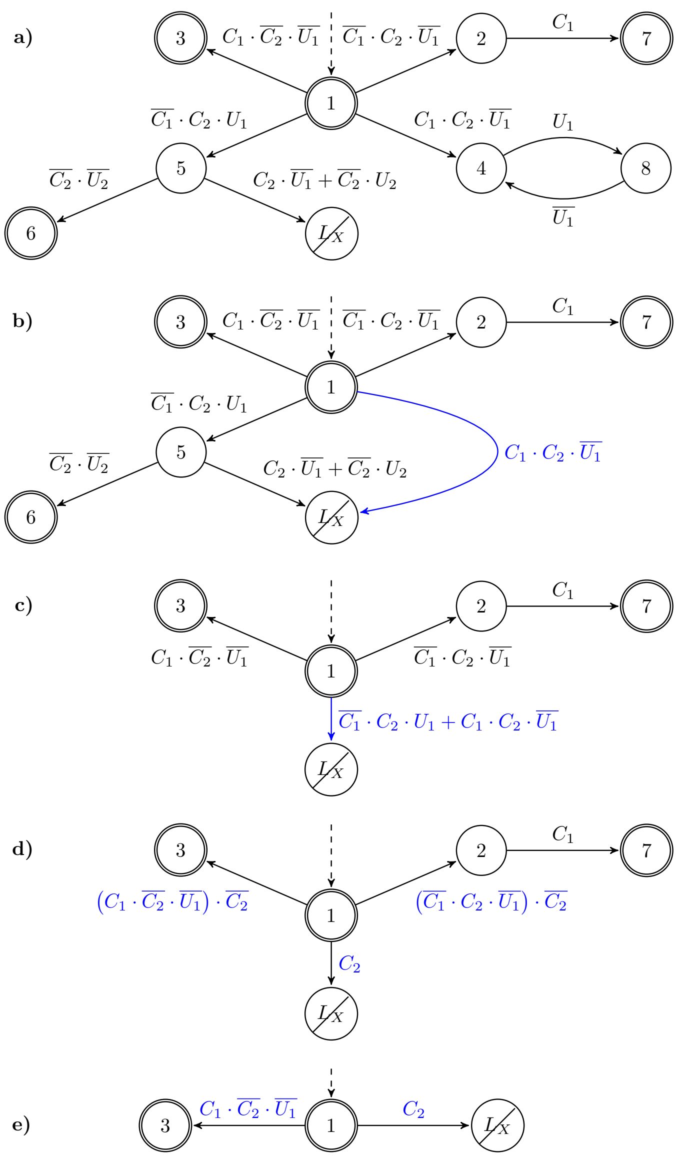
Algorithm 2: Non-blocking and controllability solving

```
Data: a non-blocking SLA satisfying P9
Result: a non-blocking controllable SLA satisfying P9, i.e. the supervisor

1  Controllability analysis:
2  $\Delta_{ToCheck} \leftarrow \{\delta \in \Delta : l_{Dest}(\delta) = L_X\}$
3  while $\Delta_{ToCheck} \neq \emptyset$ do
4      foreach $\delta \in \Delta_{ToCheck}$ do
5          $ctrl_{Cond} \leftarrow$ All-SAT$\left(\texttt{CNF}\left(\overline{\delta_{Cond}(\delta)}\right), S_O\right)$
6          if $ctrl_{Cond} = False$ then                    ▷ $\delta$ not controllable
7              $L_{Check} \leftarrow \emptyset$
8              foreach $\delta' \in \Delta : l_{Dest}(\delta') = l_{Src}(\delta)$ do
9                  $l_{Dest}(\delta') \leftarrow L_X$
10             foreach $\delta' \in \Delta : l_{Src}(\delta') = l_{Src}(\delta)$ do
11                 $L_{Check} \leftarrow L_{Check} \cup l_{Dest}(\delta')$
12                 $\Delta \leftarrow \Delta \setminus \{\delta'\}$
13             while $L_{Check} \setminus L_X \neq \emptyset$ do
14                 foreach $l \in L_{Check}$ do
15                     if $\nexists \delta' \in \Delta : l_{Dest}(\delta') = l$ then
16                         $L_{Check} \leftarrow L_{Check} \cup \{l' \in Ł : \exists \delta'' \in \Delta, l_{Src}(\delta'') = l \wedge l_{Dest}(\delta'') = l'\}$
17                         $L_{Check} \leftarrow L_{Check} \setminus \{l\}$
18                         $\Delta \leftarrow \Delta \setminus \{\delta \in \Delta : l_{Dest}(\delta) = l\}$
19                         $L \leftarrow L \setminus \{l\}$
20             ▷ All locations reachable only from $l_{Src}(\delta)$ and their incoming evolutions have been removed
21             if $\exists (\delta', \delta'') \in \Delta^2 : \delta' \neq \delta'' \wedge l_{Dest}(\delta') = l_{Dest}(\delta'') = L_X \wedge l_{Src}(\delta') = l_{Src}(\delta'')$ then
22                 $\delta_{Cond}(\delta') \leftarrow \delta_{Cond}(\delta') + \delta_{Cond}(\delta'')$
23                 $\Delta \leftarrow \Delta \setminus \{\delta''\}$
24                 $\Delta_{ToCheck} \leftarrow \Delta_{ToCheck} \setminus \{\delta''\}$
25                 $\Delta_{ToCheck} \leftarrow \Delta_{ToCheck} \cup \{\delta'\}$
26             ▷ Potential redundant evolutions leading to $L_X$ are combined as a disjunction; this new evolution needs to be checked
27         else                                              ▷ $\delta$ is controllable
28             $\delta_{Cond}(\delta) \leftarrow \overline{ctrl_{Cond}}$
29             foreach $\delta' \in \Delta : l_{Src}(\delta') = l_{Src}(\delta)$ do
30                 $L_{Check} \leftarrow \emptyset$
31                 $\delta_{Cond}(\delta') \leftarrow \delta_{Cond}(\delta') \cdot ctrl_{Cond}$
32                 if $\delta_{Cond}(\delta') = False$ then
33                     $L_{Check} \leftarrow L_{Check} \cup l_{Dest}(\delta')$
34                     $\Delta \leftarrow \Delta \setminus \{\delta'\}$
35                     repeat procedure lines 13 to 19
36         $\Delta_{ToCheck} \leftarrow \Delta_{ToCheck} \setminus \{\delta\}$
37 return
```

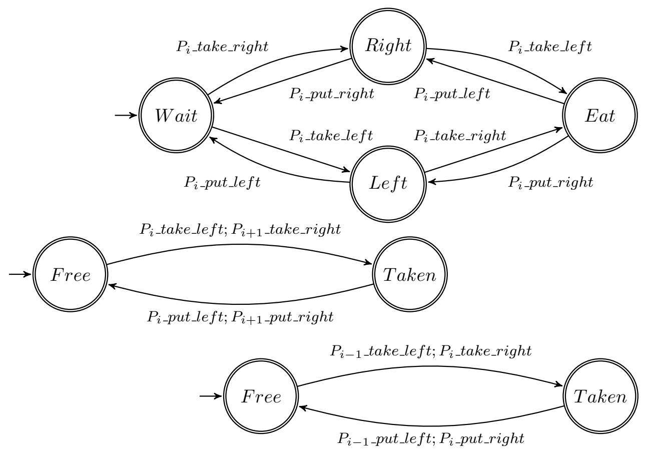
In the example given in Fig. 4 b), the locations 5 and 1 are the only locations directly leading to the forbidden location L_X. As illustrated in subsection 5.2, the evolution condition $C_2 \cdot \overline{U_1} + \overline{C_2} \cdot U_2$ from the location 5 to L_X is uncontrollable. After the execution of the Algorithm 2 until line 20 on this evolution, the location 5 is merged into L_X; the location 6 (as well as its incoming evolution) which was reachable only from the location 5 is also removed.
Since there are now two evolutions from the location 1 to L_X, with guards $\overline{C_1} \cdot C_2 \cdot U_1$ and $C_1 \cdot C_2 \cdot \overline{U_1}$, they are combined as a disjunction (lines 21 to 26); this gives the single evolution from the location 1 to L_X, with the guard $\overline{C_1} \cdot C_2 \cdot U_1 + C_1 \cdot C_2 \cdot \overline{U_1}$. At this point, the SLA presented in Fig. 4 c) is obtained.
Then, the Algorithm 2 is executed again on this new

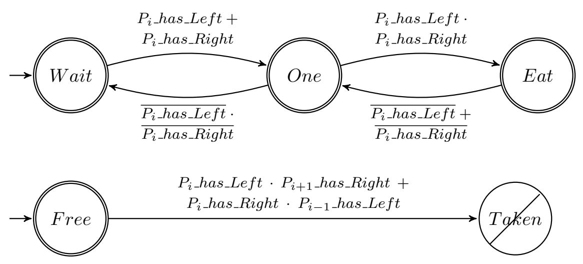
evolution from the location 1 to L_X As illustrated in subsection 5.2, this evolution condition is controllable. Consequently, this evolution condition is replaced by the negation of its controllability condition (line 28), i.e. C_2. Also, the evolutions from the location 1 to the locations 2 and 3 see their guard combined as conjunction with this controllability condition (line 31), which respectively gives $\left(\overline{C_1} \cdot C_2 \cdot \overline{U_1}\right) \cdot \overline{C_2}$ and $\left(C_1 \cdot \overline{C_2} \cdot \overline{U_1}\right) \cdot \overline{C_2}$. At this point, the SLA presented in Fig. 4 d) is obtained.
Since the new condition of the evolution from the location 1 to the location 2 is always *False* (line 32), the locations 2 and 7 as well as their incoming evolutions are removed (lines 32 to 35). No further evolution needs to be checked. The final version of the SLA is presented in Fig. 4 e). This SLA is now controllable and non-blocking.


a)
b)
c)
d)
e)
$C_1 \cdot \overline{C_2} \cdot \overline{U_1}$
$\overline{C_1} \cdot C_2 \cdot \overline{U_1}$
C_1
$\overline{C_1} \cdot C_2 \cdot U_1$
$C_1 \cdot C_2 \cdot \overline{U_1}$
U_1
$\overline{U_1}$
$\overline{C_2} \cdot \overline{U_2}$
$C_2 \cdot \overline{U_1} + \overline{C_2} \cdot U_2$
$\overline{C_1} \cdot C_2 \cdot U_1 + C_1 \cdot C_2 \cdot \overline{U_1}$
$(C_1 \cdot \overline{C_2} \cdot \overline{U_1}) \cdot \overline{C_2}$
$(\overline{C_1} \cdot C_2 \cdot \overline{U_1}) \cdot \overline{C_2}$
C_2
L_X
Fig. 4. Supervisor generation on a simple SLA

6. CASE STUDY

The *dining philosophers problem* is used to illustrate the advantages of using the proposed signal-based approach instead of an event-based approach. This problem consists in two (or more) philosophers seated at a round table with forks placed between each pair of adjacent philosophers. A philosopher can take a fork or put it back at any time, and he needs two forks to eat.


Wait
Right
Left
Eat
P_i_take_right
P_i_put_right
P_i_take_left
P_i_put_left
Free
Taken
P_i_take_left; P_{i+1}_take_right
P_i_put_left; P_{i+1}_put_right
P_{i-1}_take_left; P_i_take_right
P_{i-1}_put_left; P_i_put_right
Fig. 5. Event-based finite automaton models for one philosopher (above) and its two adjacent forks (below)


Wait
One
Eat
P_i_has_Left + P_i_has_Right
P_i_has_Left · P_i_has_Right
$\overline{P_i\text{\_has\_Left}} \cdot \overline{P_i\text{\_has\_Right}}$
$\overline{P_i\text{\_has\_Left}} + \overline{P_i\text{\_has\_Right}}$
Free
Taken
P_i_has_Left · P_{i+1}_has_Right + P_i_has_Right · P_{i-1}_has_Left
Fig. 6. EBFA models for one philosopher (above) and its adjacent forks' specification (below)

With an event-based approach, this problem can be modeled with the automata presented in Fig.5.
With the event-based approach, it is important, in the philosopher model, to make the distinction between the states where the philosopher has only picked his left fork or only picked the right one, as different events would be allowed/forbidden depending on the states (e.g. it is not possible to take twice the same fork).

With the signal-based approach, these two states can be merged into one thanks to Boolean conditions on transitions, as presented in the above model in Fig. 6.

Thus, the plant models describing the philosophers behavior have one state and four transitions less, and are thus more readable. Regarding the specification model, one forbidden state is needed to specify which combinations of signals are forbidden.

When considering more philosophers, as for the event-based models, the plant model of the philosopher will be duplicated. However, for the specification model, only the transition evolution needs to be modified, thus improving their readability.

Regarding the state-space explosion issue, for 5 philosophers, the event-based approach will generate a monolithic supervisor with 1971 states and 7985 transitions while the proposed signal-based approach generates a monolithic supervisor with only 124 states, but 15027 transitions.

The fact that this supervisor possesses more transitions, and more complex transition conditions, is due to the fact that the signal-based approach considers all possible simultaneous changes of signals. This drawback w.r.t. the complexity of the obtained supervisor is also an advantage when it comes to its implementation on a synchronous controller such as a PLC:

- First, when an asynchronous implementation will need many evolutions to process several changes (which could also leads to long processing queues), a synchronous implementation will get to the stable destination state in a single but slightly longer step.
- Most importantly, with the proposed signal-based approach, the fact that several philosophers may take one fork simultaneously does cause any delay insensitivity, interleave insensitivity, or inexact synchronization issue; all possible combinations of "simultaneous events" are calculated symbolically.

7. CONCLUSION

This paper proposed a signal-interpreted approach to the SCT problem, and presented adapted algorithms for the proposed EBFA models. The applicability of this approach has been illustrated on an academic case study, which highlights its advantages in terms of design expressibility and readability.

Future works will consider the efficiency of the proposed algorithm, and consider the size reduction of the supervisor:

- Boolean variables being already implemented through the use of internal signals (Fig. 2), a natural extension of this work would be the introduction of integer variables.
- Also, a modular approach should be considered, where the SCT algorithms should determine additional Boolean guards to be attached to the transitions of the initial plant EBFAs. To achieved it, the stability search algorithms used for the SLA generation should be modified to be used as a "reachability search" algorithm, and the SCT algorithms proposed in this paper should be executed in between two of its iterations. Thereby, blocking and non-controllable locations would be removed on-the-fly; thus, reducing the state-space to be explored.

REFERENCES

Balemi, S. and Brunner, U. (1992). Supervision of discrete event systems with communication delays. In *Proc. of 1992 American Control Conf.*, 2794–2798.

Balemi, S., Hoffmann, G.J., Gyugyi, P., Wong-Toi, H., and Franklin, G.F. (1993). Supervisory control of a rapid thermal multiprocessor. *IEEE Trans. on Automatic Control*, 38(7), 1040–1059.

Chen, Y.L. and Lin, F. (2000). Modeling of discrete event systems using finite state machines with parameters. In *Control Applications, 2000. Proceedings of the 2000 IEEE International Conference on*, 941–946. IEEE.

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

Frey, G. and Litz, L. (1998). Verification and validation of control algorithms by coupling of interpreted Petri nets. In *IEEE Int. Conf. on Systems Man and Cybernetics.*

Kumar, R., Garg, V., and Marcus, S.I. (1991). On controllability and normality of discrete event dynamical systems. *Systems & Control Letters*, 17(3), 157–168.

Le Gall, T., Jeannet, B., and Marchand, H. (2005). Supervisory control of infinite symbolic systems using abstract interpretation. In *44th IEEE Conf. on Decision and Control and European Control Conf.*, 31–35.

Leal, A.B., da Cruz, D.L., and Hounsell, M.d.S. (2012). *PLC-based implementation of local modular supervisory control for manufacturing systems.* INTECH Open Access Publisher.

Leiss, E. (1981). Succinct representation of regular languages by boolean automata. *Theoretical computer science*, 13(3), 323–330.

Miremadi, S. and Lennartson, B. (2016). Symbolic on-the-fly synthesis in supervisory control theory. *IEEE Trans. on Control Systems Technology*, 24(5), 1705–1716.

Miremadi, S., Lennartson, B., and Akesson, K. (2012). A BDD-based approach for modeling plant and supervisor by extended finite automata. *IEEE Trans. on Control Systems Technology*, 20(6), 1421–1435.

Ouedraogo, L., Kumar, R., Malik, R., and Åkesson, K. (2010). Symbolic approach to nonblocking and safe control of extended finite automata. In *6th IEEE Int. Conf. on Automation Science and Engineering*, 471–476.

Provost, J., Roussel, J.M., and Faure, J.M. (2011a). A formal semantics for grafcet specifications. In *7th IEEE Conf. on Automation Science and Engineering*, 488–494.

Provost, J., Roussel, J.M., and Faure, J.M. (2011b). Translating grafcet specifications into mealy machines for conformance test purposes. *Control Engineering Practice*, 19(9), 947–957.

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.

Roussel, J.M. and Giua, A. (2005). Designing dependable logic controllers using the supervisory control theory. *IFAC Proc. Volumes*, 38(1), 56–61.

Skoldstam, M., Akesson, K., and Fabian, M. (2007). Modeling of discrete event systems using finite automata with variables. In *Decision and Control, 2007 46th IEEE Conference on*, 3387–3392. IEEE.

Toda, T. and Tsuda, K. (2015). BDD construction for all solutions SAT and efficient caching mechanism. In *Proc. of the 30th Annual ACM Symposium on Applied Computing*, 1880–1886.

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

Yu, Y., Subramanyan, P., Tsiskaridze, N., and Malik, S. (2014). All-SAT using minimal blocking clauses. In *27th Int. Conf. on VLSI Design and 13th Int. Conf. on Embedded Systems*, 86–91.

Zaytoon, J. and Riera, B. (2017). Synthesis and implementation of logic controllers–a review. *Annual Reviews in Control.*