# A Signal-Interpreted Approach to the Supervisory Control Theory – Extension to Partially Controllable Signals

**Loïc Desgeorges, Julien Provost**

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

**Abstract:** After highlighting the difficulties encountered while implementing a supervisor on actual industrial controllers and different approaches already suggested to tackle them, this paper presents a signal-interpreted approach to the Supervisory Control Theory (SCT) framework. This work focuses on further developments of this approach for reactive systems. Due to fundamental differences between event- and signal-based approaches, new algorithms have to be implemented to apply an SCT approach on the basis of signal-interpreted Boolean finite automata extended with variables. Also, compared to a previous version of this approach, partially controllable signals are introduced.

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

## 1. INTRODUCTION

30 years after the publication of the foundational work on the Supervisory Control Theory (SCT) by Ramadge and Wonham (1987), this framework has been – and remains – investigated and extended by a large community of researchers in control engineering. However, despite its strong formal background, the adoption of SCT in industry still remains limited, as recently reported by Vieira et al. (2017) and Zaytoon and Riera (2017). Yet, some recent real-world applications, such as tap-changing transformers (Afzalian et al. (2010)), theme-park vehicles (Forschelen et al. (2012)) or MRI scanner (Theunissen et al. (2014)), show promising results.

As initially described in Fabian and Hellgren (1998), the main issue to the application of SCT, is that there is no actual rule on how to implement a supervisor. Indeed, the SCT is based on the *event*-based automata theory; in contrast, most industrial controllers, such as Programmable Logic Controllers (PLCs) consider input/output *signals*. Thus, many inconsistencies can appear. Events being asynchronous, they can occur, independently, at any instant, while signals connected to a PLC behave w.r.t. its cyclic execution: scanning the input signals, executing the code, and updating the output signals. Therefore, several rising or falling edges can occur during a PLC cycle, which corresponds to several events occurring at the same time. This is in contradiction with the fundamental definition of events in the automata theory: *any two or more events cannot occur simultaneously.*

As detailed in Section 2, many previous works have proposed improvements to the SCT modeling framework or to its implementation on actual controllers. To the best of our knowledge, all these works rely, at least partly, on event-based models.

The approach presented in this paper is an extension of a previous *signal-interpreted* approach to the SCT (Fouquet and Provost (2017)). In comparison to the latter, this paper considers the relation between input and output signals due to the scan cycle of a reactive system's controller. Thus, the notion of *partially controllable* output signal is introduced, which leads to a less restrictive behavior during the controllability checking. Also, to limit the state-space explosion, this paper proposes a step-by-step construction of a non-blocking and controllable Stable Composed Automata (SCA), by combining the forward reachability signal-interpreted composition, and the non-blocking and controllability analysis.

The remainder of this paper is organized as follows. First, related works are presented in section 2. Then, the definitions of Boolean Finite Automata extended with variables (EBFAs) and Stable Composed Automaton (SCA) are reminded and further developed in section 3. The methodology proposed in this paper is presented in section 4. Then, the algorithms proposed to apply SCT on signal-interpreted models are detailed in section 5. Finally, conclusions and future works are drawn in the last section.

## 2. RELATED WORKS

As mentioned in the introduction, the main difference between event-based and signal-based approaches lies in the fact that only one event can occur at a time while several signals can change synchronously (or at least be perceived as changing synchronously). Thus, when considering the implementation of event-based supervisors on industrial controllers, this leads to two types of issues:
First, the asynchronism between the signal-based physical environment and the event-based supervisor makes the implementation on industrial controllers such as PLCs difficult. As earlier presented by Fabian and Hellgren (1998)

and Balemi and Brunner (1992), some properties such as *interleave insensitivity* and *delay insensitivity* have to be fulfilled. However, even though some recent methods have improved the SCT implementation on PLCs, they do not guarantee a correct implementation of event-based models on a cyclically executed controller if the above-mentioned properties are not fulfilled (Leal et al. (2012), Vieira et al. (2017), Prenzel and Provost (2018)).
Secondly, with event-based models, the definition of *conditions* based on the occurrence of several events requires either the use of several non-functional states to represent the different possible paths to satisfy such a condition, or the enumeration as events of all possible combinations of simultaneous signal changes (e.g. $e_1 = \uparrow a \wedge \uparrow b \wedge \uparrow c, e_2 = \uparrow a \wedge \uparrow b \wedge \downarrow c, e_3 = \uparrow a \wedge \downarrow b \wedge \downarrow c, \ldots$). On the opposite, signal-based models allow to directly write conditions based on signals, and thus, simplify the models definitions.

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. Yet, these models still require events to trigger the firing of transitions. Variables are used on top of an event to define guards and actions.

Signal-based approaches have already been developed for the synthesis of a *controller* (e.g. Marchand et al. (2000)). The method proposed in the current paper considers the generation of a *supervisor*.

For more details, the interested reader can refer to the preliminary paper of this signal-based approach (Fouquet and Provost (2017) (section 2)).

## 3. SIGNAL-INTERPRETED BOOLEAN FINITE AUTOMATA'S DEFINITION

### *3.1 Boolean Finite Automata extended with variables*

To suit the SCT framework, the definition of Boolean automata introduced by Leiss (1981), was further defined by Fouquet and Provost (2017). This definition has been slightly adapted here:
An *EBFA system* is a set of communicating *EBFAs* running in parallel, each of them being described by a 6-tuple: $EBFA = \langle L, l_0, S, \Delta, L_M, L_X \rangle$ with:

- $L$ a non-empty set of locations;
- $l_0$ a non-empty set of initial locations, $l_0 \subset L$;
- $S$ a non-empty set of I/O and internal Boolean signals;
- $\Delta$ a set of transitions, each transition being defined by $\delta = \langle l_{Src}, \delta_{Cond}, l_{Dest} \rangle$ with:
  - $l_{Src}$ a single location in $L$;
  - $\delta_{Cond}$ its firing condition, a Boolean function [1] in $S$;
  - $l_{Dest}$ a single location in $L$;
- $L_M$ a non-empty set of marked locations, $L_M \subseteq L$;
- $L_X$ a set of forbidden locations, $L_X \subset L$, $L_X$ may be empty.

To be well-defined an EBFA has to observe the following properties:

- For any transition $\delta \in \Delta$: $l_{Src}(\delta) \neq l_{Dest}(\delta)$.
  Self-loops are implicitly defined: for a given valuation of the signals, if no outgoing transition can be fired then the active location remains active. Thus, there is no need to explicitly define them.
- A forbidden state should not have any outgoing evolution.

It is also worth mentioning that an EBFA can also be non-deterministic.

One key feature of this signal-interpreted model is the *stability search*: after a change of the input signals' values, several EBFAs' transitions can be fired simultaneously and/or sequentially during a single evolution step. This means that its evolution rules are defined similarly to those of Signal-Interpreted Petri Nets (SIPN) and Grafcet (see Frey and Litz (1998) and Provost et al. (2011)).

### *3.2 Stable Composed Automaton*

The Stable Composed Automaton (SCA) represents only the reachable and stable states of an EBFA system and the possible evolutions between these states. The SCA formal definition and generation algorithms for an EBFA system (i.e. a set of EBFA) are similar to those for a Grafcet (see Provost et al. (2011) for more formal details).

A deterministic SCA can be described by a 6-tuple: $SCA = \langle Q, q_0, S, \Delta, Q_M, q_x \rangle$ with:

- $Q$ a non-empty set of stable states, each state being defined by $q = \langle L_{Act}, S_{StabCond}, V_O \rangle$ with:
  - $L_{Act}$ a set of active locations in the EBFAs;
  - $S_{StabCond}$ its stability condition, a Boolean function in $S$;
  - $V_O \subseteq S_O$, if needed for temporal specifications, a subset of currently emitted output signals (see sections 4.1 and 4.3 for more details);
- $q_0$ the initial unstable state, $q_0 \notin Q$;
  $q_0$ corresponds to the set of all initial locations of the EBFAs;
- $S$ a non-empty set of I/O Boolean signals;
- $\Delta$ a set of evolutions, each evolution being defined by $\delta = \langle q_{Src}, \delta_{Cond}, q_{Dest} \rangle \in (Q \cup q_0) \times (\mathbb{B}^S) \times Q$;
- $Q_M$ a set of marked states, $Q_M \subseteq Q$;
- $q_x$ the forbidden state, $q_x \in Q$, $q_x \notin Q_M$.

## 4. METHODOLOGY

### *4.1 Signals Definition*

The set of signals $S$ can be divided into four disjoint categories. Three of them ($S_I$, $S_O$ and $S_L$) are similar to the I/O events categories introduced by Balemi et al. (1993) and were further defined by Fouquet and Provost (2017). The second and third ones ($S_O$ and $S'_O$) permit to capture the behavior of a reactive system. These four categories of signals are defined as follows:

$S_I$: Current input signals;
$S'_O$: Next output signals;
$S_O$: Current output signals. The values of the current output signals $S_O$ at a given cycle correspond to the values of the next output signals $S'_O$ at the previous cycle: $S_O[k] = S'_O[k-1]$;

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

$S_L$: Internal signals. These internal signals can be used within one EBFA or to communicate in between EBFAs, e.g. to define a modular specification models depending on the active state of other models.

EBFAs transitions' conditions can be defined using these four categories, while the resulting SCA evolutions' condition are only defined using $S_I$ and $S'_O$.

Since a PLC is executed cyclically, at each cycle the control code will calculate the value of the next output signals $S'_O$ given the current state $X$, the current value of the input signals $S_I$ and, if specified, the current value of some output signals $V_O \subseteq S_O$: $S'_O = f(X, S_I, V_O)$.

From an implementation perspective, it is not always necessary to store every value of current output signals $S_O$, but only the ones required to determine the next output signals' values, i.e. $V_O \subseteq S_O$. In the following sections, a specification which uses a subset of the current output signals' values will be termed a *temporal specification*; more details are given in section 4.3.

### *4.2 Partially controllable signals*

Similarly to events, a distinction has to be made between controllable and uncontrollable signals. By definition, the input signals $S_I$ and the current output signals $S_O$ are uncontrollable: $S_{\overline{Ctrl}} = S_I \cup S_O$. Then, regarding the next output signals $S'_O$, not all of them may be controllable.

In the SCT framework introduced by Ramadge and Wonham (1987) the supervisor can only enable or disable commands generated by a plant, it cannot force them to occur. To extend this framework, Golaszewski and Ramadge (1987) introduced the notion of *forced events*.

Recently, to model an event that has to be forced for safety reasons, Göbe et al. (2017) introduced the notion of *forcible events*. Applied to a signal-based framework, this method consists in enforcing the signal to its 0 logic value. Yet, the opposite could also occur in practice. In order to be able to handle these different scenario, the notion of *partially controllable signals* is introduced in this paper. In the case of Boolean signals, a controllable signal can be either: *controllable-at-0*, *controllable-at-1*, or *controllable-at-0-and-1*. Thus, the set of next output signals $S'_O$ is further defined as follows:

$$\begin{aligned} S'_O &= S'_{\text{Ctrl}} \cup S'_{\overline{\text{Ctrl}}} \\ &= S'_{\text{Ctrl0}} \cup S'_{\text{Ctrl1}} \cup S'_{\text{Ctrl01}} \cup S'_{\overline{\text{Ctrl}}} \end{aligned}$$

A controllable-at-0 signal ($s_1 \in S'_{\text{Ctrl0}}$) can have its value enforced at 0 by the supervisor; respectively at 1, and at 0 or at 1, for controllable-at-1 ($s_2 \in S'_{\text{Ctrl1}}$) and controllable-at-0-and-1 ($s_3 \in S'_{\text{Ctrl01}}$) signals.

### *4.3 Example of temporal specifications*

To illustrate the use of a temporal specification model and highlight its advantages and drawbacks, a toy example of an AGV with a single obstacle sensor is considered. The plant model is represented on the left part of Fig. 1. Semantically, a guard on the next output signals corresponds to an action to be performed on the corresponding output signals at the end of the cycle. The AGV output signals to commands its movement are $L$, $R$, $F$ and $B$, to turn *L*eft or *R*ight, and move *F*orward or *B*ackward, respectively. The specification model represented on the right part of Fig. 1 specifies that, if the AGV is turning left, it should stop turning left when in contact with an obstacle (associated to the input signal $c$); and similarly when turning right. According to these models, it will be possible to supervise the AGV controller and prevent it from reaching the forbidden state only if the next output signals $L'$ and $R'$ are controllable-at-0 (or controllable-at-0-and-1). This means that the supervisor is able to stop the AVG's movements but it is not aiming at commanding the start of the movements.

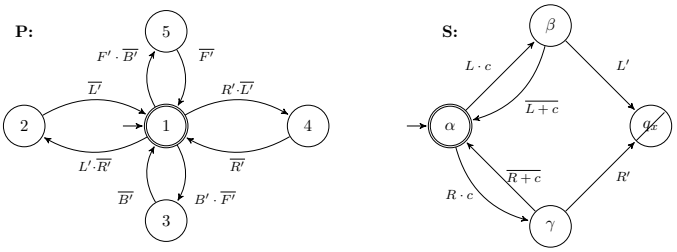


Fig. 1. AGV's plant (P) and specification (S) models

In this example, it is clear that not all current output signals' values have to be stored as $V_O$ in the SCA, but only the two ones concerned by the temporal specification: $L$ and $R$.

The classical specification, which does not consider the current outputs signals $S_O$, would require more locations to capture the actual state of the system.

From an implementation perspective, this solution requires fewer locations to be stored in memory, but it also requires to store the current outputs signals' values. The choice between a temporal or a classical specification models would depend on the structure of the specification and the number of current outputs signals' values to be stored.

### *4.4 Overview of the algorithms*

In the approach proposed in this paper, contrary to the previous approach proposed by Fouquet and Provost (2017), the state-space explosion is limited by combining, step-by-step, the signal-interpreted forward reachability and the removal of states due to the non-blocking and controllability analysis.

During the controllability analysis, it is an essential step to check the controllability of Boolean conditions. Considering reactive systems, this means checking whether, given current input and output signals, it is possible to allocate the values of the next output signals in order to prevent reaching the forbidden state.

These algorithms are detailed in the next section.

## 5. ALGORITHMS

Due to the stability search principle, the term *evolution* refers to a sequence of simultaneous firings of transitions in the plants and specifications EBFAs, but it corresponds to a single transition from one stable location to another one in the monolithic SCA.

### 5.1 Controllability of an evolution

By definition, an evolution is *controllable* if and only if it is possible to prevent its firing. Thus, in a signal-based framework, an evolution is controllable if and only if it is possible to find a combination of controllable signals that can set the evolution condition to *False*.

The calculation of the *controllability* of an evolution can be divided into different cases.
First, if there exists a combination of controllable signals that always makes the condition $False$, then the evolution is *certainly controllable*.
Secondly, if there exists a combination of uncontrollable signals that always makes the condition $True$, then the evolution is *certainly uncontrollable*.
Else, the evolution is neither *certainly controllable* nor *certainly uncontrollable*.

Determining if such a combination of controllable or uncontrollable signals exists can be efficiently solved using SAT-solvers.

Fig. 2 illustrates the cases of *certainly controllable* and *certainly uncontrollable* evolutions. In the first case, a combination of controllable signals $C_1 = 0 \wedge C_2 = 0$ makes the condition $False$, so this evolution is certainly controllable. In the second case, a combination of uncontrollable signals $u_1 = 1 \wedge u_2 = 1$ makes the condition $True$, so this evolution is certainly uncontrollable.

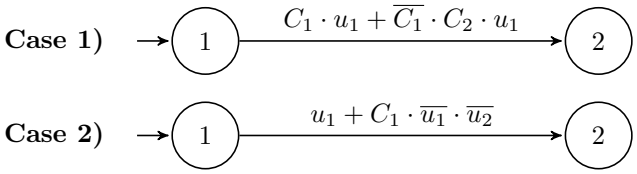


Fig. 2. A certainly controllable evolution (Case 1) and a certainly uncontrollable evolution (Case 2)

If an evolution is neither *certainly controllable* nor *certainly uncontrollable*, this evolution is controllable if and only if, for each value of the uncontrollable signals, it is possible to determine a value of the controllable signals making its condition $False$. Considering the notion of partially controllable signals, an evolution $\delta$, with condition $\delta_{cond}(\delta)$, is controllable if for each signal $c_i$:

- if $c_i$ is controllable-at-0 ($c_i \in S'_{\text{Ctrl0}}$):
  - $\forall v \in \mathbb{B}^{|S_{\overline{Ctrl}}|}\ |\delta_{cond}(\delta)_{|s,c_i=0} \neq True$
- if $c_i$ is controllable-at-1 ($c_i \in S'_{\text{Ctrl1}}$):
  - $\forall v \in \mathbb{B}^{|S_{\overline{Ctrl}}|}\ |\delta_{cond}(\delta)_{|s,c_i=1} \neq True$
- if $c_i$ is controllable-at-0-and-1 ($c_i \in S'_{\text{Ctrl01}}$):
  - $\forall v \in \mathbb{B}^{|S_{\overline{Ctrl}}|}\ |\delta_{cond}(\delta)_{|s,c_i=0} \cdot \delta_{cond}(\delta)|_{s,c_i=1} \neq True$

Fig. 3 illustrates the case of an evolution that is neither *certainly controllable* nor *certainly uncontrollable*.

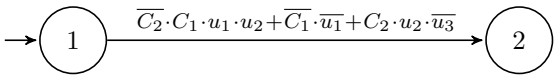


Fig. 3. Example of an evolution that is neither *certainly controllable* nor *certainly uncontrollable*.

Table 1 considers this evolution in the case of $C_1$ being controllable-at-1 and $C_2$ controllable-at-0. It can be observed that if the values of the uncontrollable signals $u_1$, $u_2$ and $u_3$ are 1, 1 and 0 respectively, it is not always possible to prevent the firing of the evolution. Thus, in this case, the condition is not controllable.

Table 1. Controllability with $C_1 \in S'_{\text{Ctrl1}}$

| $u_1$ | $u_2$ | $u_3$ | $\delta_{Cond}$ | $\delta_{Cond}\vert_{C_1=1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | $\overline{C_1}$ | $False$ |
| 1 | 0 | 0 | $False$ | $False$ |
| 0 | 1 | 0 | $\overline{C_1} + C_2$ | $C_2$ |
| 0 | 0 | 1 | $\overline{C_1}$ | $False$ |
| 1 | 1 | 0 | $\overline{C_2} \cdot C_1 + C_2$ | $True$ |
| 1 | 0 | 1 | $\overline{C_1}$ | $False$ |
| 0 | 1 | 1 | $False$ | $False$ |
| 1 | 1 | 1 | $\overline{C_2} \cdot C_1$ | $\overline{C_2}$ |

Table 1 considers this evolution in the case of $C_1$ being controllable-at-0-and-1 and $C_2$ controllable-at-0. It can now be observed that whatever the values of the uncontrollable signals $u_1$, $u_2$ and $u_3$, it is always possible to prevent the firing of the evolution. Thus, in this case, the condition is not controllable.

Table 2. Controllability with $C_1 \in S'_{\text{Ctrl01}}$ and $C_2 \in S'_{\text{Ctrl0}}$

| $u_1$ | $u_2$ | $u_3$ | $\delta_{Cond}$ | $\delta_{cond_{\vert C_1=0}} \cdot \delta_{cond_{\vert C_1=1}}$ | $\delta_{cond_{\vert C_1=0 \wedge C_2=0}} \cdot \delta_{cond_{\vert C_1=1 \wedge C_2=0}}$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | $\overline{C_1}$ | $False$ | $False$ |
| 1 | 0 | 0 | $False$ | $False$ | $False$ |
| 0 | 1 | 0 | $\overline{C_1} + C_2$ | $C_2$ | $False$ |
| 0 | 0 | 1 | $\overline{C_1}$ | $False$ | $False$ |
| 1 | 1 | 0 | $\overline{C_2} \cdot C_1 + C_2$ | $C_2$ | $False$ |
| 1 | 0 | 1 | $\overline{C_1}$ | $False$ | $False$ |
| 0 | 1 | 1 | $False$ | $False$ | $False$ |
| 1 | 1 | 1 | $\overline{C_2} \cdot C_1$ | $False$ | $False$ |

These methods are summarized in Algorithm 1.

### 5.2 Applying SCT algorithms

This subsection details the different algorithms used to generate an SCA representing the minimally-restrictive, reachable, non-blocking and controllable supervisor.

The Algorithm 2 describes the relations between the forward reachability, the non-blocking, and the controllability algorithms. For the sake of clarity, the non-blocking and the controllability algorithms are detailed in Algorithms 3 and 4, respectively. Details about the forward reachability algorithm can be found in Provost et al. (2011).

It is also worth mentioning that the notion of uncontrollable states is not treated here. The implicit definition of self-loops in signal-interpreted models makes this notion superfluous. However, if a Boolean condition composed only of uncontrollable signals is voluntarily associated to an evolution leading to the forbidden state, this case will be handled similarly to any other evolution's condition.

The non-blocking algorithm (Alg. 3), aims at finding blocking locations and at merging them with the forbidden location.

**Algorithm 1: IsCondCtrl** algorithm

**Data:** A condition $\delta_{cond}$, a set of signal $S$
**Result:** A boolean : Ctrl ← True, Unctrl ← False

**Controllability of one condition IsCondCtrl**$(\delta_{cond}(\delta), S)$
**if** $\{v \in \mathbb{B}^{|S'_{Ctrl}|} | (\overline{\delta_{Cond}(\delta)_{|s}}) = True\} \neq \emptyset$ **then**
  return True
**else if** $\{v \in \mathbb{B}^{|S_{\overline{Ctrl}}|} | \delta_{Cond}(\delta)_{|s} = True\} \neq \emptyset$ **then**
  return False
**else**
  **foreach** $C_i \in S'_{Ctrl}$ **do**
    **if** $C_i \in S'_{Ctrl0}$ **then**
      $\delta_{cond}(\delta) \leftarrow \delta_{cond}(\delta)_{|C_i=0}$
      **foreach** $v \in \mathbb{B}^{|S_{\overline{Ctrl}}|}$ **do**
        **if** $\delta_{cond}(\delta)_{|_v} = True$ **then**
          return False
    **else if** $C_i \in S'_{Ctrl1}$ **then**
      $\delta_{cond}(\delta) \leftarrow \delta_{cond}(\delta)_{|C_i=1}$
      **foreach** $v \in \mathbb{B}^{|S_{\overline{Ctrl}}|}$ **do**
        **if** $\delta_{cond}(\delta)_{|v} = True$ **then**
          return False
    **else** ▷ i.e. $C_i \in S'_{Ctrl01}$
      $\delta_{cond}(\delta) \leftarrow \delta_{cond}(\delta)_{|C_i=0} \cdot \delta_{cond}(\delta)_{|C_i=1}$
      **foreach** $v \in \mathbb{B}^{|S_{\overline{Ctrl}}|}$ **do**
        **if** $\delta_{cond}(\delta)_{|v} = True$ **then**
          return False
  ▷ If not returned False during the **foreach** loop:
  return True

**Algorithm 2:** Main algorithm

**Data:** A set of EBFA to compose $\{EBFA\}$
**Result:** a non-blocking and controllable SCA

**Main algorithm:**
$SCA' \leftarrow \langle \emptyset, q_0, S, \emptyset, \emptyset, q_X \rangle$
$k \leftarrow 1$
$Q[0] \leftarrow \emptyset$
**repeat**
  $SCA \leftarrow SCA'$
  $SCA' \leftarrow ForwardReach1Step(SCA, \{EBFA\})$
  ▷ $\Delta[k], Q[k]$ and $Q_M[k]$ are the set of new transitions, states and marked states
  ▷ $\Delta' = \Delta \cup \Delta[k]$
  ▷ $Q' = Q \cup Q[k]$
  ▷ $Q'_M = Q_M \cup Q_M[k]$
  $SCA' \leftarrow$ **NonBlocking**$(Q[k-1], Q[k], SCA')$
  ▷ This function correspond to the one in Algo. 3
  $SCA' \leftarrow$ **Controllability**$(Q[k], SCA')$
  ▷ This function correspond to the one in Algo. 4
  $SCA' \leftarrow$ **Trim**$(SCA')$
  $k \leftarrow k + 1$
**until** $SCA' = SCA$;
return $SCA$

Then, the controllability algorithm (Alg. 4) checks if each evolution leading the forbidden state is controllable. If not, it merges the incoming state with the forbidden state.

Finally, a **Trim** function is used to delete the locations which are not reachable anymore, as well as their outgoing transitions.

**Algorithm 3: NonBlocking** algorithm

**Data:** $Q[k-1]$,$Q[k]$,$SCA = \langle Q, q_0, S, \Delta, Q_M, q_X \rangle$
**Result:** A non-blocking SCA

**NonBlocking**($Q[k-1]$,$Q[k]$,$SCA$):
**foreach** $q \in Q[k-1]$ **do**
  $Q_{del} \leftarrow \emptyset$
  ▷ if from $q \in Q[k-1]$ it is not possible to reach $Q[k]$ and $q$ is not part of $Q_M$ then $q$ is blocking
  **if** $\big(\nexists n \in \mathbb{N}^+ | (\delta_1..\delta_n, q') \in \Delta^n \times Q[k] :$
    $q_{Src}(\delta_1) = q \wedge q_{Dest}(\delta_n) = q'$
    $\wedge (\forall i \in [1, n-1] : q_{Dest}(\delta_i) = q_{Src}(\delta_{i+1}))\big)$
    $\wedge q \notin Q_M$ **then**
    $Q_{del} \leftarrow q$
  ▷ Then, search backward for further blocking states
  **while** $Q_{del} \neq \emptyset$ **do**
    ▷ Remove the states leading to a blocking one, that cannot also reach a marked state or another state in $Q[k]$.
    **foreach** $q_B \in Q_{del}$ **do**
      $Q_{del} \leftarrow Q_{del} \cup \{q \mid$
        $\big(\exists \delta \in \Delta | q_{Src}(\delta) = q \wedge q_{Dest}(\delta) = q_B\big) \wedge$
        $\big(\nexists n \in \mathbb{N}^+ | (\delta_1..\delta_n, q'') \in \Delta^n \times \{Q[k] \cup Q_M\} :$
        $q_{Src}(\delta_1) = q \wedge q_{Dest}(\delta_n) = q''$
        $\wedge (\forall i \in [1, n-1] : q_{Dest}(\delta_i) = q_{Src}(\delta_{i+1}))\big)\}$
      $\Delta_B \leftarrow \{\delta \in \Delta \mid q_{Dest}(\delta) = q_B\}$
      **foreach** $\delta \in \Delta_B$ **do**
        $q_{Dest}(\delta) \leftarrow q_X$
      $\Delta \leftarrow \Delta \setminus \{\delta \in \Delta \mid q_{Src}(\delta) = q_B\}$
      $Q \leftarrow Q \setminus \{q_B\}$
$SCA \leftarrow \langle Q, q_0, S, \Delta, Q_M, q_X \rangle$
return $SCA$

These algorithms are repeated until a fixed-point is reached, i.e. until no more state is newly reached or deleted.

## 6. CONCLUSION

This paper proposed a signal interpreted approach to the SCT framework. The relation between input and output signals due to the scan cycle of a reactive system is developed. Then, the notion of partially controllable output signals is introduced in order to consider a supervisor point of view and determine the controllability of an evolution. Finally, the different algorithms used to generate a non-blocking and controllable SCA, which represents the supervisor, are presented.

Future works will consider the definition of a modular synthesis approach for signal-interpreted models. Given the current EBFA formalism, it is already possible to easily represent modular specifications; however, in its current version, the proposed approach generates a monolithic supervisor. The generation of additional Boolean guards (similarly to Miremadi et al. (2011)) to be attached to the transitions of the initial EBFAs models would fit well a signal-based approach.

**Algorithm 4: Controllability** algorithm

**Data:** $Q[k]$,$SCA = \langle Q, q_0, S, \Delta, Q_M, q_X \rangle$
**Result:** A non-forbidden $SCA$

**Controllability**($Q[k]$,$SCA$):
**foreach** $q \in Q[k]$ **do**
  **if** $q = Q_X$ **then**
    ▷ Search for evolutions leading to the forbidden state
    $\Delta_{ToCheck} = \{\delta \in \Delta \mid q_{Dest}(\delta) = q_x\}$
    **while** $\Delta_{ToCheck} \neq \emptyset$ **do**
      **foreach** $\delta \in \Delta_{ToCheck}$ **do**
        **if** *not* ***IsCondCtrl***$(\delta_{Cond}(\delta), S)$ **then**
        ▷ $\delta$ not controllable
          $Q_{Check} \leftarrow \emptyset$
          **foreach** $\delta' \in \Delta : q_{Dest}(\delta') = q_{Src}(\delta)$ **do**
            $q_{Dest}(\delta') \leftarrow q_x$
          **if** $\exists(\delta', \delta'') \in \Delta^2 : \delta' \neq \delta'' \wedge q_{Dest}(\delta') = q_{Dest}(\delta'') = q_x \wedge q_{Src}(\delta') = q_{Src}(\delta'')$ **then**
          ▷ if two transitions from the same source lead to $q_x$ then merge their conditions
            $\delta_{Cond}(\delta') \leftarrow \delta_{Cond}(\delta') + \delta_{Cond}(\delta'')$
            $\Delta \leftarrow \Delta \setminus \{\delta''\}$
            $\Delta_{ToCheck} \leftarrow \Delta_{ToCheck} \setminus \{\delta''\}$
            $\Delta_{ToCheck} \leftarrow \Delta_{ToCheck} \cup \{\delta'\}$
      $\Delta_{ToCheck} \leftarrow \Delta_{ToCheck} \setminus \{\delta\}$
return $SCA$

## REFERENCES


Afzalian, Ali A, N., SM, and Wonham, W. (2010). Discrete-event supervisory control for under-load tap-changing transformers ultc: from synthesis to plc implementation. *Discrete Event Simulations*, 285–310.

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 *Proc. of the 2000 IEEE International on Control Applications*, 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.

Forschelen, S.T.J., van de Mortel-Fronczak, J.M., Su, R., and Rooda, J.E. (2012). Application of supervisory control theory to theme park vehicles. *Discrete Event Dynamic Systems: Theory and Applications*, 22(4), 511–540.

Fouquet, K. and Provost, J. (2017). A signal-interpreted approach to the supervisory control theory problem. In *Proc. of the 20th World Congress of the International Federation of Automatic Control*. Elsevier.

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*.

Göbe, F., Aydin, S., and Kowalewski, S. (2017). Applicability of supervisory control theory for the supervision of PLC programs. In *Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on*. IEEE.

Golaszewski, C. and Ramadge, P.J. (1987). Control of discrete event processes with forced events. In *Decision and Control, 1987. 26th IEEE Conference on*, volume 26, 247–251. IEEE.

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.

Marchand, H., Bournai, P., Le Borgne, M., and Le Guernic, P. (2000). Synthesis of discrete-event controllers based on the signal environment. *Discrete Event Dynamic Systems*, 10(4), 325–346.

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

Prenzel, L. and Provost, J. (2018). PLC implementation of symbolic, modular supervisory controllers. In *14th Workshop on Discrete Event Systems (WODES 2018)*.

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

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.

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

Theunissen, R.J.M., Petreczky, M., Schiffelers, R.R.H., van Beek, D.A., and Rooda, J.E. (2014). Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner. *IEEE Trans. on Automation Science and Engineering*, 11(1), 20–32.

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.

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