A Process Algebra for Supervisory Coordination

Jos Baeten Bert van Beek Allan van Hulst Jasen Markovski*
Department of Mechanical Engineering, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands,
{j.c.m.baeten,d.a.v.beek,ahulst,j.markovski}@tue.nl

A supervisory controller controls and coordinates the behavior of different components of a complex machine by observing their discrete behaviour. Supervisory control theory studies automated synthesis of controller models, known as supervisors, based on formal models of the machine components and a formalization of the requirements. Subsequently, code generation can be used to implement this supervisor in software, on a PLC, or embedded microprocessor. In this article, we take a closer look at the control loop that couples the supervisory controller and the machine. We model both event-based and state-based observations using process algebra and bisimulation-based semantics. The main application area of supervisory control that we consider is coordination, referred to as supervisory coordination, and we give an academic and an industrial example, discussing the process-theoretic concepts employed.

1 Introduction

Control software development becomes an important issue due to the ever-increasing machine complexity and demands for better quality, performance, safety, and ease of use. Traditionally, the control requirements are formulated informally and, thereafter, manually translated into control software, followed by validation and rewriting of the code whenever necessary. The cycles of such a design-validate process are both error-prone and time-consuming due to frequent ambiguities in the informal specifications. This issue gave rise to supervisory control theory [22, 9, 17], where models of the supervisory controllers, referred to as *supervisors* are synthesized automatically based on formal models of the uncontrolled hardware, referred to as *plant*, and the model of the *control requirements*. Based on these models, the control software is generated automatically. The supervisory controller observes discrete machine behavior and sends back control signals about allowed activities. Assuming that the controller reacts sufficiently fast on machine input, this feedback loop, depicted in Figure 1a), was originally modeled as a pair of synchronizing processes [22, 9].

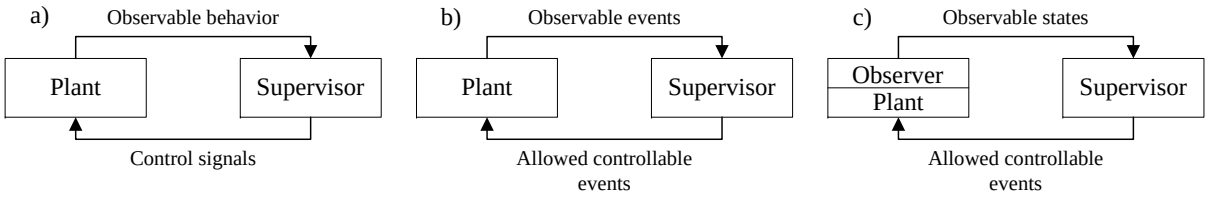

Figure 1: Control loop: a) general, b) with event-based, c) with state-based observations.

In this paper, we focus on the modeling of the control loop and the required process-theoretic concepts to capture the underlying behavior. The main motivation for the investigation is the oversimplification of the coupling between the plant and the supervisor in the original proposal of [22, 9], which still

*Research funded by C4C European project FP7-ICT-2007.3.7.c.

L. Aceto and M.R. Mousavi (Eds.): PACO 2011
EPTCS 60, 2011, pp. 36–55, doi:10.4204/EPTCS.60.3

prevails in modern state-of-the-art approaches, like [13, 11, 20, 26, 16] to name a few. Furthermore, we consider coordination as the main application area of supervisory control, where the coordinator(s) are implemented as supervisory controllers that ensure sequencing of events, or deadlock- and livelock-free behavior of the plant, according to the given set of (coordinating) control requirements.

Supervisory control loop To model different aspects of the plant and the control requirements, the discrete events that can occur are split into *controllable* and *uncontrollable* events. The former can be disabled by the supervisor and typically model actuator activities, e.g., starting or stopping a motor. The latter cannot be affected by the supervisor if enabled in the plant and standardly model sensor activities, e.g., the temperature has reached a given value. We distinguish two types of prominent supervisory control loops relying on event- and state-based observations, depicted in Figure 1b) and c), respectively.

The control loop in Figure 1b) depicts that the supervisor observes events that occur in the plant and sends back as feedback the set of controllable events that are allowed for execution. The most prominent operation that captures the coupling between the plant and the supervisor is automata-style synchronous parallel composition [22, 9]. This simple operation restricts the plant by omitting (controllable) events in the supervisor, thereby preventing synchronization and disabling the events. It was quickly realized that this synchronization produces large supervisors that actually memorize the complete supervised behavior as the supervisor keeps track of the state of the plant by keeping a complete history of observed events.

To mitigate the large size of the supervisor several synchronization operators were proposed that enable the plant to independently execute uncontrollable events, provided that this does not preclude the supervisor from correctly deducing the state of the plant [14, 13, 15, 10]. We note that there are other models of the control loop that employ the input/output transition paradigm that require an input (set of controllable/actuator events) from the supervisor to produce an output (uncontrollable/sensor event) from the plant [6, 7, 25]. Nonetheless, they have been shown to be equivalent to one of the above approaches with respect to the underlying notions [7].

What synchronous parallel composition or communication fails to model is the difference between the two flows of information, their role, and the different goals of the plant and the supervisor. To this end, we propose a send/receive communication to model the different flows of communication in Figure 1 and differentiate between the contributions of the plant and the supervisor. The event-based observation flow of Figure 1b) enables communication of all observable events, whereas the control signal flow transmits only controllable events. In addition, this setting also supports asynchronous communication between the plant and the supervisor, which affects almost every implementation of supervisory controllers [8].

As a solution to the problem of large supervisors, an alternative approach was proposed in [17], as depicted in Figure 1c). The plant (or the supervisor) is augmented with an observer or a tracker that deduces the state of the plant and submits this observed information to the supervisor. The supervisor, based on this state-based information, acts as a lookup table and feeds the plant with the allowed controllable events in the observed state. In such a way, the supervisor only incorporates necessary information in order to exercise control over the plant. Nonetheless, this feedback mechanism is not formalized in [17] and, here, we propose to model this variant of the control loop using a process-theoretic approach that employs root signal emission [1] to capture the state-based observations. Alternative modeling of such control loops is by means of shared variables and synchronization [19], but such approaches do not distinguish between the different flows of information depicted in Figure 1c).

Finally, when employing supervisory control for coordination of distributed systems, the supervisor communicates the control actions to several components that have different physical locations. To this end, we propose to model the feedback control signal communication from the supervisor by means of

broadcast communication [4]. To illustrate the proposed process theories that capture the behavior of the control loops of Figure 1b) and c) we revisit two cases, where we applied supervisory coordination: 1b) a simple case that introduces the main concepts and deals with coordination of an automated guided vehicle, involving event-based observations, and 1c) a part of an industrial study dealing with maintenance procedures inside complex high-tech printers, which employs state-based observations [18].

Process-theoretic approach The process-theoretic treatment of supervisory control theory is sustained by a behavioral relation that captures the notion of *controllability*, which states that supervisory control is possible only if the supervisor can achieve supervised behavior allowed by the control requirements without having to disable an uncontrollable event. Prior investigations to process-theoretic treatments of supervisory control resulted in a special prioritized synchronization operator [14, 13], while employing failure semantics. An alternative approach replaced this special operator with a refinement relation to characterize nondeterministic supervised behaviors [20]. In [26, 24] the refinement is given in terms of bisimulation and in terms of simulation in [16]. A coalgebraic approach introduced partial bisimulation as a behavioral relation suitable to define language-based controllability [23]. In essence, it states that controllable events should be simulated, whereas uncontrollable events should be bisimulated. This notion was lifted to a concurrency theory for supervisory control that succinctly captured the controllability for nondeterministic discrete-event systems [2]. Here, we extend this framework to elaborately model and formalize the behavior of the supervisory control loops depicted in Figure 1.

The rest of this paper is organized as follows. Section 2 revisits the process theory TCP* from [1] and establishes a link between partial bisimulation and supervisory control. Section 3 shows how to model supervisory control loop in the presented theory by applying supervisory coordination to an automated production line. Section 4 extends the process theory to incorporate guarded commands and root signal emission, which are employed in Section 5, where we revisit an industrial case study of coordination of maintenance procedures in a high-tech printer. We finish with a discussion of future challenges and the potential of applying process theory in supervisory control.

2 Process theory TCP*

In this section we revisit the process algebra TCP* (Theory of Communicating Processes with Iteration) [1] in which we introduce *generic communication actions* and we adopt *partial bisimulation* as a behavioral relation. This process algebra has a rich syntax, allowing us to express all key ingredients of concurrency theory, including termination, which enables a strong correspondence with automata theory.

Syntax We presuppose a finite *data alphabet* D and a finite set H of *channels*. We assume that $\mathsf{A} = \{c!_m?_n d \mid c \in \mathsf{H},\ m,n \in \mathbb{N},\ d \in \mathsf{D}\}$, where $c!_m?_n d$ is a generic communication action. If $m = n = 0$, then we treat the generic communication action $c!_0?_0 d$ as a basic event, possibly parameterized with data, notation $c(d)$. Otherwise, we handle it as an outcome from synchronization of m send and n receive actions. We employ the standard notation for handshaking communication [1], i.e., $c?d$ for $c!_0?_1 d$, $c!d$ for $c!_1?_0 d$, and $c!?d$ for $c!_1?_1 d$. Intuitively, these events denote that datum d is received, sent, or communicated along channel c, respectively.

The set of *process terms* T is generated by the following grammar:

$$T ::= 0 \mid 1 \mid a.T \mid T \cdot T \mid T + T \mid T \parallel T \mid T^* \mid \partial_E(T),$$

$$\mathbf{1}\ \frac{}{1\downarrow} \qquad \mathbf{2}\ \frac{}{a.p \xrightarrow{a} p} \qquad \mathbf{3}\ \frac{}{p^*\downarrow} \qquad \mathbf{4}\ \frac{p \xrightarrow{a} p'}{p^* \xrightarrow{a} p' \cdot p^*} \qquad \mathbf{5}\ \frac{p\downarrow,\ q\downarrow}{p\cdot q\downarrow} \qquad \mathbf{6}\ \frac{p\downarrow,\ q \xrightarrow{a} q'}{p\cdot q \xrightarrow{a} q'}$$

$$\mathbf{7}\ \frac{p \xrightarrow{a} p'}{p\cdot q \xrightarrow{a} p'\cdot q} \qquad \mathbf{8}\ \frac{p\downarrow}{(p+q)\downarrow} \qquad \mathbf{9}\ \frac{q\downarrow}{(p+q)\downarrow} \qquad \mathbf{10}\ \frac{p \xrightarrow{a} p'}{(p+q) \xrightarrow{a} p'} \qquad \mathbf{11}\ \frac{q \xrightarrow{a} q'}{(p+q) \xrightarrow{a} q'}$$

$$\mathbf{12}\ \frac{p\downarrow,\ q\downarrow}{p \parallel q\downarrow} \qquad \mathbf{13}\ \frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q} \qquad \mathbf{14}\ \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'} \qquad \mathbf{15}\ \frac{p\downarrow}{\partial_E(p)\downarrow}$$

$$\mathbf{16}\ \frac{p \xrightarrow{c!_\ell ?_k d} p',\ q \xrightarrow{c!_m ?_n d} q',\ \ell+k>0,\ m+n>0}{p \parallel q \xrightarrow{c!_{\ell+m} ?_{k+n} d} p' \parallel q'} \qquad \mathbf{17}\ \frac{p \xrightarrow{a} p',\ a \notin \{c!_m ?_n d \mid c!_m ?_n \in E,\ d \in \mathsf{D}\}}{\partial_E(p) \xrightarrow{a} \partial_E(p')}$$

Table 1: Operational rules for TCP*

where $a \in \mathsf{A}$ and $E \subseteq \{c!_m ?_n \mid c \in \mathsf{H},\ m,n \in \mathbb{N}\}$. Let us briefly comment on the operators in this syntax. The constant 0 denotes inaction or *deadlock*, whereas the constant 1 denotes *successful termination* [1]. For each action $a \in \mathsf{A}$ there is a unary operator $a.$ denoting *action prefix*; the process denoted by $a.p$ can do an a-transition to the process denoted by p. The binary operator $p \cdot q$ denotes *sequential composition* that behaves like p, followed by q only upon successful termination of p. The binary operator $p+q$ denotes *alternative composition* or choice on the first action transition of p and q. The binary operator $p \parallel q$ denotes *parallel composition (with generic channel communication actions)*; actions of both arguments can be interleaved or, alternatively, communication takes place that keeps track of how many send or receive actions are combined. The unary operator p^* is *iteration* or Kleene star that unfolds with respect to the sequential composition. The unary operator $\partial_E(p)$ *encapsulates* the process p in such a way that all (incomplete) communication actions, e.g., $c?d$ and $c!d$, are blocked for all data, so that the desired type of communication is enforced, e.g., if we were to enforce communication between k processes on channel c, then $E = \{c!_m ?_n \mid 0 < m+n,\ m+n \neq k\}$.

Semantics We give semantics to the process terms by a labeled transition relation $\longrightarrow \subseteq \mathsf{T} \times \mathsf{A} \times \mathsf{T}$ and a successful termination predicate $\downarrow \in \mathsf{T}$. We employ infix notation and write $s \xrightarrow{a} t$ if $(s,a,t) \in \longrightarrow$ and $s\downarrow$ if $s \in \downarrow$. We derive the transition relation and the successful termination predicate using structural operational semantics [21], given by the operational rules in Table 1. Alternatively, we depict them as a labeled transition system G, specified by the tuple $G = (\mathsf{T}, \mathsf{A}, \downarrow, \longrightarrow)$.

We briefly comment on the rules. The successful termination constant can successfully terminate, whereas the action prefix enables outgoing labeled transitions, as given by rules 1 and 2. Rule 3 states that iteration can always terminate successfully, which enables sequential composition of recursive processes. The unfolding of the iteration is with respect to the sequential composition, as given by rule 4. The sequential composition can terminate only if both processes can do so, as given by rules 5, whereas if only the first component can terminate successfully, it can continue behaving as the second. The outgoing transition of the first component is the same for the sequential composition as given by rule 7. Rules 8 and 9 state that alternative composition can terminate if one of the components can terminate, whereas the choice is made on the outgoing transitions, as stated by rules 10 and 11. The parallel composition can terminate only if both components can do so. Rules 13 and 14 enable interleaving of transitions. Rules 15 states that encapsulation does not prevent successful termination. Rule 16 defines synchronization which can occur on communication actions comprising at least one sending or receiving event. The communication actions are merged to accumulate the participating send and receive parties. Finally,

rule 17 states that all (incomplete) communication actions on a given channel comprising a predefined number of senders and receivers are blocked by the encapsulation operation.

We can easily extend the transition relation to traces of actions in A^*. For $p, p' \in \mathsf{T}$ and $t = a_1, \ldots, a_n \in \mathsf{A}^*$, we write $p \stackrel{t}{\twoheadrightarrow} p'$ if there exist $p_0, \ldots, p_n \in \mathsf{T}$ such that $p = p_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} p_n = p'$. By ε we denote the empty trace $a_1, \ldots, a_n$ for $n = 0$ and $p = p'$. Every finite automaton can be described up to isomorphism (and possibly by changing the communication operation) by a term in our setting, see [3].

Language-based supervision Now, we can translate the central notion of a supervisor [22, 9] in our setting. To this end, we partition the channel names into two disjoint sets of uncontrollable U and controllable C channels such that $\mathsf{H} = \mathsf{U} \cup \mathsf{C}$ and $\mathsf{U} \cap \mathsf{C} = \emptyset$. The uncontrollable and controllable channel names induce controllable and uncontrollable actions, respectively, given by $\mathsf{A}_\mathsf{C} \triangleq \{c!_m?_n d \mid c \in \mathsf{C},\ d \in \mathsf{D}\}$ and $\mathsf{A}_\mathsf{U} \triangleq \{u!_m?_n d \mid u \in \mathsf{U},\ d \in \mathsf{D}\}$. Next, we define the (prefix-closed) language recognized by the process term p or, alternatively, the automaton represented by p, as $\mathsf{L}(p) \triangleq \{t \in \mathsf{A}^* \mid \text{there exists } p' \in \mathsf{T} \text{ such that } p \stackrel{t}{\twoheadrightarrow} p'\}$. Note that traces do not need to end with successful termination. We denote by $LL' \triangleq \{tt' \mid t \in L,\ t' \in L'\}$ the concatenation of the languages L and L'.

Recall that the supervisor cannot achieve the control requirements by forbidding uncontrollable events, when synchronizing with the plant. Suppose that the plant, the control requirements, and the supervisor with respect to the former are determined by the languages recognized by the process terms $p, r, s \in \mathsf{T}$, respectively. If the operation modeling the control loop is denoted by p/s, then $\mathsf{L}(p/s) \subseteq \mathsf{L}(p)$ and $\mathsf{L}(p/s) \subseteq \mathsf{L}(r)$, where we refer to p/s as the *supervised plant*. We note that if strict equality holds, then the control requirements can be achieved completely. Often, this is not the case, so one attempts to synthesize a maximally-permissive supervisor, which makes $\mathsf{L}(p/s)$ as large as possible with respect to inclusion. For deterministic systems, this supervisor is unique, equal to the union of all possible supervisors [22, 9], whereas for nondeterministic systems, a unique maximally-permissive supervisor in general does not exist [2]. For standard supervisory control [22, 9], the operation that models the control loop p/s is the full synchronous parallel composition of automata [22, 9]. That s does not disable uncontrollable events is ensured by requesting that p/s is *controllable* with respect to p, expressed as $\mathsf{L}(p/s)\mathsf{U} \cap \mathsf{L}(p) \subseteq \mathsf{L}(p/s)$ [22, 9]. Controllability is interpreted as follows. If we observe a desired trace in the plant followed by an uncontrollable event, then the supervisor cannot request that this uncontrollable event should be disabled after allowing that trace. If r is controllable with respect to p, then one can guarantee the existence of a supervisor s, achieving the desired controlled behavior r by restricting the plant p by synchronization, i.e., $\mathsf{L}(p/s) = \mathsf{L}(r)$.

Nondeterminism and partial bisimulation The disadvantages of working in the language domain have been discussed on many occasions, e.g., see overviews in [13, 11, 2, 1]. Therefore, a proposal was made in [2] to lift controllability to support full nondeterminism in a process-theoretic setting. The underlying behavioral relation is partial bisimulation [23, 2], which is parameterized with a bisimulation actions set $B \subseteq \mathsf{A}$ that denotes which actions are to be bisimulated, whereas the other actions are simulated.

Definition 1 *A relation $R \subseteq \mathsf{T} \times \mathsf{T}$ is a partial bisimulation with respect to a bisimulation action set $B \subseteq \mathsf{A}$, if for all $(p,q) \in R$ it holds that:*

1. *if $p\downarrow$, then $q\downarrow$;*
2. *if $p \xrightarrow{a} p'$ for some $a \in \mathsf{A}$, then there exists $q' \in \mathsf{T}$ such that $q \xrightarrow{a} q'$ and $(p',q') \in R$;*

3. *if* $q \xrightarrow{b} q'$ *for some* $b \in B$, *then there exists* $p' \in \mathsf{T}$ *such that* $p \xrightarrow{b} p'$ *and* $(p', q') \in R$.

If $(p,q) \in R$, *we say that* p *is partially bisimilar to* q *with respect to* B *and we write* $p \leq_B q$. *If* $q \leq_B p$ *holds as well, we write* $p \leftrightarrow_B q$.

Note that $\leq_B$ is a preorder relation, making $\leftrightarrow_B$ an equivalence relation for all $B \subseteq \mathsf{A}$ [2]. If $B = \emptyset$, then $\leq_\emptyset$ coincides with strong similarity preorder and $\leftrightarrow_\emptyset$ coincides with strong similarity equivalence [12, 1]. When $B = \mathsf{A}$, $\leftrightarrow_\mathsf{A}$ turns into strong bisimilarity [12, 1]. Moreover, if $p \leq_B q$, then $p \leq_C q$ for every $C \subseteq B$. We also note that partial bisimilarity is a precongruence with respect to the operators of TCP* [2].

For given processes $p, r \in \mathsf{T}$, representing the plant and the control requirements, respectively, we ensure that $s \in \mathsf{T}$ is a valid supervisor that does not disable uncontrollable events by requiring that $p/s \leq_\emptyset r$ and $p/s \leq_{\mathsf{A_U}} p$, where $\mathsf{A_U} \subseteq \mathsf{A}$ is the set of uncontrollable events [2]. This setting covers both the existing deterministic and nondeterministic definition of controllability for discrete-event systems [2]. From the definition, it is also not difficult to observe, that one obtains the same supervised behavior for every $p' \leftrightarrow_{\mathsf{A_U}} p$. Thus, one direct benefit from our approach is a procedure for coarsest plant minimization that respects controllability, based on the partial bisimilarity equivalence.

Next, we model the supervisory control loop with event-based observations and we illustrate our approach by a use case involving coordination of an automated guided vehicle in a production line.

3 Control Loop with Event-Based Observations

We employ the process theory TCP* to formalize the behavior of the control loop with event-based observations, depicted in Figure 1b). According to the scheme, the plant cannot execute a controllable event without the permission of the supervisor, whereas the supervisor must not disable uncontrollable events. Nonetheless, the supervisor is able to observe execution of uncontrollable events in the plant, so that it can correctly determine the state of the plant and transmit correct control signals. Moreover, the supervisor should not execute uncontrollable events independently, as this does not contribute to his objective. In addition, the supervisor should not introduce deadlocks or livelocks explicitly, unless deadlock or livelock behavior is inherent to the plant. Finally, we assume that the supervisor is a (global) monolithic process, i.e., it is not comprised from multiple modular or distributed synchronizing supervisors [9]. Taking into account the above observations, we can specify the syntax of the plant processes P and the supervisor processes S as given by P and S, respectively:

$$P ::= 0 \mid 1 \mid c?d.P \mid u!_\ell?_k d.P \mid P \cdot P \mid P + P \mid P \parallel P \mid \partial_E(P) \mid P^*$$
$$S ::= 1 \mid c!d.S \mid u?d.S \mid S + S \mid S^*,$$

where $c \in \mathsf{C}$, $u \in \mathsf{U}$, $\ell, k \in \{0,1\}$, $d \in \mathsf{D}$, and $E \subseteq \{f!_m?_n \mid f \in \mathsf{H},\ m,n \in \mathbb{N}\}$. To implement broadcast communication in the case when the supervisor sends control signals to several distributed components, which do not have to receive the control signals simultaneously, one would also need to introduce action priorities, cf. [4]. Due to page restrictions, we will not employ broadcast in the general form in this paper and, instead, we enforce three-way communication by employing only the encapsulation operator.

Supervisory coordination of an automated production line To illustrate our approach to supervisory control and the model of the control loop, we discuss a simple example concerning coordination of an automated guided vehicle (AGV) in an automated production line, depicted in Figure 2. The AGV is responsible for transferring the preproduct made by Workstation M to Workstation N and transferring the finished product from Workstation N to the Delivery station. The workstations and the AGV

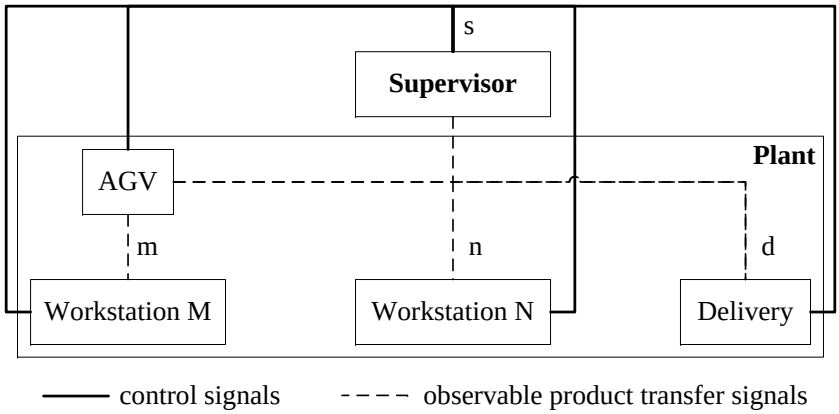

Figure 2: Automated production line

are coordinated by a supervisor, which sends the corresponding control signals. We can model the automated production system depicted in Figure 2 employing TCP*, where M, N, A, and S are process terms that model Workstation M, Workstation N, AGV, and the supervisor. We note that we abstract from the delivery station, depicted by a single event *deliver*, as it does not contribute to any interesting behavior. We retain the communication channel names as depicted in Figure 2, whereas the data elements are $\mathsf{D} = \{make, move2N, preproduct, product\}$. The uncontrollable channel names are $\mathsf{U} = \{m, n, produce, process, move, deliver\}$, whereas $\mathsf{C} = \{s\}$ is the set of controllable channel names.

$$M \triangleq (s?make.produce(preproduct).m!preproduct.1)^*$$
$$N \triangleq (n?preproduct.process(preproduct).n!product.1)^*$$
$$A \triangleq (m?preproduct.s?move2N.move(preproduct).n!preproduct.1 + n?product.deliver(product).1)^*$$
$$S \triangleq (s!make.s!move2N.1)^*.$$

Workstation M repeatedly waits for a command from the supervisor to make a preproduct, which is offered to the AGV once it is made. Workstation N waits for a preproduct from the AGV, which is thereafter processed and offered back to the AGV. The AGV can either pick up a preproduct at Workstation M, after which it asks for permission to move the preproduct to Workstation N, or pick up a finished product at Workstation N and deliver it. Now, the unsupervised plant is given by the process

$$U \triangleq \partial_F(M \parallel N \parallel A), \text{ where } F = \{m?, m!, n?, n!\}.$$

At this point, we note that we enforce meaningful communication of uncontrollable channels within the plant by encapsulation and this does not restrict the behavior of the unsupervised plant, but only ensures its meaningful behavior. Following the framework outlined above, it can be readily observed that the plant $U \in \mathsf{P}$ follows the outlined syntax.

In this first modeling instance, we assume that the AGV is responsible for delivering the final product and we propose a supervisor as given by the process S. Note that the supervisor $S \in \mathsf{S}$ follows the outlined syntax and it does not make use of any observed information. Supervisor S repeatedly gives orders to Workstation M for new products to be made, followed by orders to the AGV to transfer the preproduct to Workstation N. Thus, the automated production system is modeled as

$$U/S \triangleq \partial_E(S \parallel U), \text{ where } E = \{s?, s!\},$$

$$\xi(0) = 0 \qquad \xi(1) = 1 \qquad \xi(p^*) = \xi(p)^* \qquad \xi(p \diamond q) = \xi(p) \diamond \xi(q) \quad \text{for } \diamond \in \{+, \cdot, \|\}$$
$$\xi(c!_m?_n d.p) = \xi(c!_m?_n d).\xi(p) \text{ for } c \in \mathsf{H},\ d \in \mathsf{D},\ m,n \in \mathbb{N}$$

Table 2: Renaming function

which enforces communication of control signals and transfer of (pre)products. One can directly check that S is a valid supervisor by establishing that the supervised plant is partially bisimulated by the original plant with respect to the uncontrollable events. To this end, we must employ renaming of events, as the original plant has open communication actions that wait for synchronization with the supervisor. This renaming function ξ traverses the process terms and renames all open communication actions to succeeded communication actions. We note that we overload the name of the renaming function of the process terms and apply it to the communication action names as well. Also, we only specify the communication actions that are actually renamed. The definition of the renaming operation is given by structural induction in Table 2.

Now, in order to verify that the supervisor does not disable uncontrollable events, it is sufficient to verify that it holds that

$$U/S \leq_{\mathsf{A_U}} \xi(U), \text{ where } \xi : s?d \mapsto s!?d \text{ for } d \in \mathsf{D},$$

which can be directly checked. We note that there was no restriction imposed on the control requirements, which in this case coincide with the plant and are, therefore, trivially satisfied.

Nonblocking supervision Unfortunately, our automated production system has a deadlock. The main reason for the deadlock is that a second preproduct can come too early, before the first product is completely finished and delivered, which is set off by sending a $s!make$ command too early, i.e., before the processed product has left Workstation N. Then, the AGV picks up the preproduct from Workstation M, but it cannot deliver it to Workstation N, as the latter also waits for a finished product to be picked. A trace that leads to deadlock is

$$s!?make\ produce(preproduct)\ m!?preproduct\ s!?move2N\ n!?preproduct$$
$$s!?make\ produce(preproduct)\ m!?preproduct\ s!?move2N\ process(preproduct)\ 0.$$

Such form of blocking behavior appears often, so in many cases the supervisor is additionally required to prevent deadlock and/or livelock, or also known as blocking, behavior [22, 9]. To this end, special marked states are introduced to automata in supervisory control. We note that these states roughly correspond to successful termination in our setting. The correspondence is not strict, mainly due to the absence of sequential composition and the Kleene star operator in the supervisory control literature and the role of the successful termination in these contexts, confer Table 1. Note that the marked states do not contribute to the formation of the recognized language of an automaton, which is different from its marked language [22, 9].

So, besides the control requirements, we impose an additional deadlock-freedom requirement on the supervisor, stated formally as: there exists no trace $t \in \mathsf{A}^*$ such that $U/S \xrightarrow{t} \!\!\!\!\rightarrow 0$. To ensure this additional nonblocking requirement, we have to modify the supervisor to accept requests for making a new preproduct only after the finished product has been loaded on the AGV, to be transferred to the delivery station.

To this end, the supervisor should allow for a new product to be made only after the finished product has been loaded to the AGV at Workstation N, which can be achieved by observing this additional information on channel n.

To this end, we modify the supervisor to S' as follows:

$$S' \triangleq (s!make.s!move2N.n?product.1)^*.$$

At this point, we note that communication on the channel n now must occur between three parties, i.e., Workstation N that sends information and the AGV and the supervisor that receive it. In order to enforce this communication, we employ the generic communication actions, i.e., we encapsulate all (incomplete) communication actions on n, except for $n!_1?_2product$. The definition of the deadlock-free supervised plant now becomes:

$$U/S' \triangleq \partial_{E'}(S' \parallel U), \text{ where } E' = \{s?, s!, n?, n!?\}.$$

Again, one directly verifies that the supervisor is valid by establishing partial bisimilarity between the supervised and the original plant following an appropriate renaming of the incomplete communication actions, given by $\xi : s?d \mapsto s!?d,\ n!?d \mapsto n!_1?_2d$ for $d \in \mathsf{D}$.

Next, we extend the process theory TCP* to accommodate state-based observations as well.

4 Control Loop with State-Based Observations

We propose TCP$^*_\perp$, an extension of TCP*, with propositional signals [5] and guarded commands in order to support the modeling of a control loop with state-based observations. To this end, we employ the Boolean algebra

$$\mathbb{B} = (\mathsf{N}, \mathsf{F}, \mathsf{T}, \neg, \wedge, \vee, \Rightarrow),$$

where $\mathsf{N} = \{P_1, \ldots, P_n\}$ are the propositional symbols, the constants represent false and true, whereas the operators denote negation, conjunction, disjunction, and implication, respectively. We use B to denote the standard Boolean expressions of $\mathbb{B}$, which are evaluated with respect to a given valuation $v\colon \mathsf{B} \to \{\mathrm{F}, \mathrm{T}\}$. The set of valuations is denoted by V.

Process theory TCP$^*_\perp$ We enrich the syntax of TCP* and the set of process terms T with the *inaccessible process* constant, *guarded commands*, and *signal emission*. The inaccessible process, notation $\perp$, specifies the process in which there are inconsistencies between the valuation of the propositional variables and the emitted propositional signals. Such a state cannot be reached from any consistent state. The guarded command, notation $\phi :\to p$, specifies a guard $\phi \in \mathsf{B}$ that guards a process $p \in \mathsf{T}$. If the guard is successfully evaluated, the process continues behaving as $p \in \mathsf{T}$ or, else, it deadlocks. The root signal emission process $\phi \curvearrowright p$, emits the propositional signal $\phi \in \mathsf{B}$ until the process $p \in \mathsf{T}$ takes an outgoing transition, provided that the propositional signal is consistent with the valuation. To be able to evaluate the propositional expressions, we couple the process terms with valuations, notation $\langle p, v\rangle \in \mathsf{T} \times \mathsf{V}$. The dynamics of the valuations, with respect to outgoing labeled transitions, is captured by a predefined valuation effect function, given by $\mathit{effect}\colon \mathsf{A} \times \mathsf{V} \to 2^{\mathsf{V}}$. With respect to the valuation we have to extend the successful termination predicate to $\downarrow \in \mathsf{T} \times \mathsf{V}$ and the labeled transition relation to $\longrightarrow \in \mathsf{T} \times \mathsf{V} \times \mathsf{A} \times \mathsf{T} \times \mathsf{V}$. We introduce an additional consistency predicate $\searrow \in \mathsf{T} \times \mathsf{V}$ that checks

$$\textbf{18}\ \frac{}{\langle 0,v\rangle \searrow} \qquad \textbf{19}\ \frac{}{\langle 1,v\rangle \searrow} \qquad \textbf{20}\ \frac{}{\langle 1,v\rangle \downarrow} \qquad \textbf{21}\ \frac{}{\langle a.p,v\rangle \searrow} \qquad \textbf{22}\ \frac{\langle p,v'\rangle \searrow,\ v' \in \mathit{effect}(a,v)}{\langle a.p,v\rangle \xrightarrow{a} \langle p,v'\rangle}$$

$$\textbf{23}\ \frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle,\ \langle q,v\rangle \searrow}{\langle p+q,v\rangle \xrightarrow{a} \langle p',v'\rangle} \qquad \textbf{24}\ \frac{\langle p,v\rangle \searrow,\ \langle q,v\rangle \xrightarrow{a} \langle q',v'\rangle}{\langle p+q,v\rangle \xrightarrow{a} \langle q',v'\rangle} \qquad \textbf{25}\ \frac{\langle p,v\rangle \searrow,\ \langle q,v\rangle \downarrow}{\langle p+q,v\rangle \downarrow}$$

$$\textbf{26}\ \frac{\langle p,v\rangle \downarrow,\ \langle q,v\rangle \searrow}{\langle p+q,v\rangle \downarrow} \qquad \textbf{27}\ \frac{\langle p,v\rangle \searrow,\ \langle q,v\rangle \searrow}{\langle p+q,v\rangle \searrow} \qquad \textbf{28}\ \frac{\langle p,v\rangle \downarrow,\ \langle q,v\rangle \downarrow}{\langle p\cdot q,v\rangle \downarrow} \qquad \textbf{29}\ \frac{\langle p,v\rangle \downarrow,\ \langle q,v\rangle \xrightarrow{a} \langle q',v'\rangle}{\langle p\cdot q,v\rangle \xrightarrow{a} \langle q',v'\rangle}$$

$$\textbf{30}\ \frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle,\ \langle p'\cdot q,v'\rangle \searrow}{\langle p\cdot q,v\rangle \xrightarrow{a} \langle p'\cdot q,v'\rangle} \qquad \textbf{31}\ \frac{\langle p,v\rangle \downarrow,\ \langle q,v\rangle \searrow}{\langle p\cdot q,v\rangle \searrow} \qquad \textbf{32}\ \frac{\langle p,v\rangle \searrow,\ \langle p,v\rangle \not\downarrow}{\langle p\cdot q,v\rangle \searrow}$$

$$\textbf{33}\ \frac{\langle p,v\rangle \searrow}{\langle p^*,v\rangle \downarrow} \qquad \textbf{34}\ \frac{\langle p,v\rangle \searrow}{\langle p^*,v\rangle \searrow} \qquad \textbf{35}\ \frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle}{\langle p^*,v\rangle \xrightarrow{a} \langle p'\cdot p^*,v'\rangle} \qquad \textbf{36}\ \frac{\langle p,v\rangle \downarrow,\ \langle q,v\rangle \downarrow}{\langle p \parallel q,v\rangle \downarrow} \qquad \textbf{37}\ \frac{\langle p,v\rangle \searrow,\ \langle q,v\rangle \searrow}{\langle p \parallel q,v\rangle \searrow}$$

$$\textbf{38}\ \frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle,\ \langle q,v\rangle \searrow,\ \langle q,v'\rangle \searrow}{\langle p \parallel q,v\rangle \xrightarrow{a} \langle p' \parallel q,v'\rangle} \qquad \textbf{39}\ \frac{\langle p,v\rangle \searrow,\ \langle p,v'\rangle \searrow,\ \langle q,v\rangle \xrightarrow{a} \langle q',v'\rangle}{\langle p \parallel q,v\rangle \xrightarrow{a} \langle p \parallel q',v'\rangle}$$

$$\textbf{40}\ \frac{\langle p,v\rangle \xrightarrow{c!_\ell ?_k d} \langle p',v'\rangle,\ \langle q,v\rangle \xrightarrow{c!_m ?_n d} \langle q',v''\rangle,\ \langle p' \parallel q',v'''\rangle \searrow,\ v''' \in \mathit{effect}(c!_{\ell+m}?_{k+n}d,v),\ \ell+k>0,\ m+n>0}{\langle p \parallel q,v\rangle \xrightarrow{c!_{\ell+m}?_{k+n}d} \langle p' \parallel q',v'''\rangle}$$

$$\textbf{41}\ \frac{p\downarrow}{\partial_E(p)\downarrow} \qquad \textbf{42}\ \frac{p\searrow}{\partial_E(p)\searrow} \qquad \textbf{43}\ \frac{p \xrightarrow{a} p' \quad a \notin \{c!_m?_n d \mid c!_m?_n \in E,\ d \in \mathsf{D}\}}{\partial_E(p) \xrightarrow{a} \partial_E(p')}$$

$$\textbf{44}\ \frac{\langle p,v\rangle \downarrow,\ v(\phi)=\mathrm{T}}{\langle \phi :\to p,v\rangle \downarrow} \qquad \textbf{45}\ \frac{\langle p,v\rangle \searrow,\ v(\phi)=\mathrm{T}}{\langle \phi :\to p,v\rangle \searrow} \qquad \textbf{46}\ \frac{v(\phi)=\mathrm{F}}{\langle \phi :\to p,v\rangle \searrow} \qquad \textbf{47}\ \frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle,\ v(\phi)=\mathrm{T}}{\langle \phi :\to p,v\rangle \xrightarrow{a} \langle p',v'\rangle}$$

$$\textbf{48}\ \frac{\langle p,v\rangle \downarrow,\ v(\phi)=\mathrm{T}}{\langle \phi \mathbin{\triangle} p,v\rangle \downarrow} \qquad \textbf{49}\ \frac{\langle p,v\rangle \searrow,\ v(\phi)=\mathrm{T}}{\langle \phi \mathbin{\triangle} p,v\rangle \searrow} \qquad \textbf{50}\ \frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle,\ v(\phi)=\mathrm{T}}{\langle \phi \mathbin{\triangle} p,v\rangle \xrightarrow{a} \langle p',v'\rangle}$$

Table 3: Operational rules for $\mathrm{TCP}^*_\perp$

whether the state is consistent. The operational rules in Table 3 give the semantics of the new predicate and the transition relation with respect to the new operators. We note that the operational rules of Table 1 have to be enhanced by decorating the process terms with valuations and additional checks for consistency.

The rules ensure that when taking an action transition, the target state is always consistent. We comment the important rules that are not directly taken from Table 1 and adapted in a setting with valuations. The deadlock, successful termination, and action prefix are always consistent as stated by rules 18, 19, and 21, respectively. The target process must be consistent for the target valuation, which is determined by the effect function as given by rule 22. Rules 23-27 introduce valuations and consistency for the alternative composition, whereas rules 28-32 do the same for the sequential composition and rules 33-35 describe iteration. Rules 38 and 39 introduce interleaving in the new setting. Rule 40 shows how the effect function is impacted by synchronization. For the effect function to be well-defined with respect to the valuations by interleaving and synchronization [1], we require additionally that

$$\mathit{effect}(c!_{\ell+m}?_{k+n}d,v) \subseteq \mathit{effect}(c!_m?_n d,\mathit{effect}(c!_\ell?_k d,v)) \cap \mathit{effect}(c!_\ell?_k d,\mathit{effect}(c!_m?_n d,v))$$

for all $\ell,k,m,n \in \mathbb{N}$ with $\ell+k>0$ and $m+n>0$. Rules 41-43 introduce the encapsulation operator in the new setting. Rules 44 and 47 show that a guarded process does not deadlock only when the guard evaluates to true. We note, however, that the value of the guard does not affect the consistency of the term,

provided that the term that is guarded is consistent. This is in direct contrast with signal emission, see rule 49, where the consistency is preserved only if the emitting signal is consistent within the valuation. In that case, the process that emits the signal can continue with its normal execution.

Finally, we also have to adapt our behavioral relation in order to correctly handle the valuations. Here, we directly employ the approach of [5, 1], where this extension is shown for bisimulation. We consider a relation $R \subseteq \mathsf{T} \times \mathsf{T}$ to be a partial bisimulation with respect to a bisimulation action set $B \subseteq \mathsf{A}$, if for all $(p,q) \in R$ it holds that:

1. if $\langle p, v\rangle \downarrow$ for some $v \in \mathsf{V}$, then $\langle q, v\rangle \downarrow$;
2. if $\langle p, v\rangle \xrightarrow{a} \langle p', v'\rangle$ for some $v \in \mathsf{V}$ and $a \in \mathsf{A}$, then there exists $q' \in \mathsf{T}$ such that $\langle q, v\rangle \xrightarrow{a} \langle q', v'\rangle$ and $(p', q') \in R$;
3. if $\langle q, v\rangle \xrightarrow{b} \langle q', v'\rangle$ for some $v \in \mathsf{V}$ and $b \in B$, then there exists $p \in \mathsf{T}$ such that $\langle p, v\rangle \xrightarrow{b} \langle p', v'\rangle$ and $(p', q') \in R$.

Again, if $(p,q) \in R$, we say that p is partially bisimilar to q with respect to B and we write $p \leq_B q$. If $q \leq_B p$ holds as well, we write $p \leftrightarrow_B q$. Also, we consider a process $s \in \mathsf{T}$ to be a supervisor of the plant $p \in \mathsf{T}$ with respect to the control requirements $r \in \mathsf{T}$ if $p/s \leq_\emptyset r$ and $p/s \leq_{\mathsf{A_U}} p$.

Plant and supervisor syntax Now, we can model the control loop with state-based observations as depicted in Figure 1c). Intuitively, the plant emits a signal that identifies the observable states. Upon observing such a signal, the supervisor checks which controllable actions are allowed in the state identified by the signal. Allowance of actions is specified in the form of guarded prefixes in which a process term is bound to a propositional formula deduced from the control requirements. These new concepts introduce further asymmetry in the control loop, where the syntax of the plant and the supervisor is again given by P and S, respectively:

$$P ::= 0 \mid 1 \mid c?d.P \mid u!_\ell?_k d.P \mid P \cdot P \mid P + P \mid P \parallel P \mid \partial_E(P) \mid \phi :\to P \mid \phi \mathbin{\blacktriangleright} P \mid P^*$$
$$S ::= 1 \mid c!d.S \mid S + S \mid \phi :\to S \mid S^*,$$

for $c \in \mathsf{C}$, $u \in \mathsf{U}$, $\ell, k \in \{0,1\}$, $d \in \mathsf{D}$, $\phi \in \mathsf{B}$, and $E \subseteq \{f!_m?_n \mid f \in \mathsf{H}, m, n \in \mathbb{N}\}$.

We note that in the state-based setting, the control requirements can be stated directly in terms of states, i.e., signals that the state is emitting, and additionally, one can specify which events are allowed with respect to the emitted signals. The control requirements R have the following syntax given by R:

$$R ::= \phi \mid \xrightarrow{f!_m?_n d} \Rightarrow \phi \mid \phi \Rightarrow \xrightarrow{f!_m?_n d},$$

for $f \in \mathsf{H}$, $d \in \mathsf{D}$, $m, n \in \mathbb{N}$, and $\phi \in \mathsf{B}$. Given control requirements $r \in \mathsf{R}$ are satisfied with respect to process $p \in \mathsf{T}$ in the valuation $v \in \mathsf{V}$, notation $\langle p, v\rangle \models r$, according to the following operational rules:

$$\textbf{51}\ \frac{v(\phi) = \mathsf{T}}{\langle p, v\rangle \models \phi} \qquad \textbf{52}\ \frac{\langle p, v\rangle \models \neg\phi \Rightarrow \xrightarrow{f!_m?_n d}}{\langle p, v\rangle \models \xrightarrow{f!_m?_n d} \Rightarrow \phi} \qquad \textbf{53}\ \frac{v(\phi) = \mathsf{T},\ \langle p, v\rangle \xrightarrow{f!_m?_n d}}{\langle p, v\rangle \models \phi \Rightarrow \xrightarrow{f!_m?_n d}},$$

where $\langle p, v\rangle \xrightarrow{a}$ for $a \in \mathsf{A}$ holds if there does not exist $\langle p', v'\rangle$ such that $\langle p, v\rangle \xrightarrow{a} \langle p', v'\rangle$ with $v' = \mathit{effect}(a, v)$. We note that the second form of the control requirements is introduced since it corresponds better to modeling intuition and it is equivalent to the third, which is easily seen from the operational rule 52. Furthermore, for the propositional symbols, we employ the notation $\mathit{in}(\textbf{StateName})$, where $\mathit{in}(\textbf{StateName})$ is a signal emitted from the process, corresponding to a state in the labeled graph representation identified by **StateName**. For example, in the Current Power Mode process in Figure 5, the process modeling the state with associated name **Standby** emits the signal $\mathit{in}(\textbf{Standby})$.

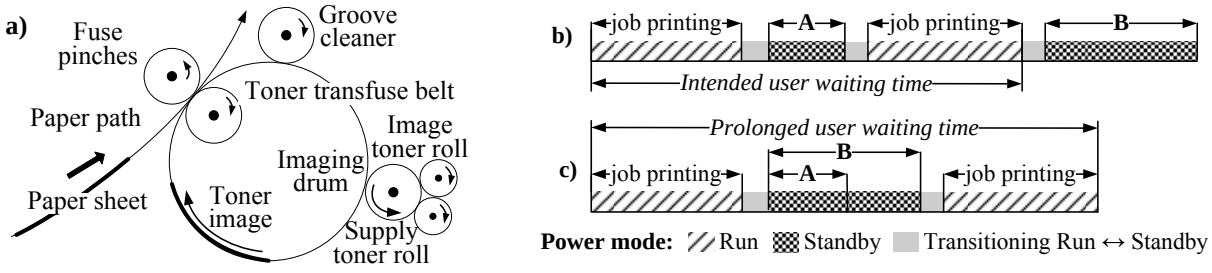

Figure 3: a) Printing process, b) maintenance operation, c) emergent behavior

5 Coordination Control of Maintenance Procedures

We employ the process theory $\mathrm{TCP}^*_\perp$ to model the coordination of maintenance procedures of a printing process of a high-end Océ printer [18]. The printing process consists of several distributed independent components as depicted in Figure 3a). The process applies the toner image onto the toner transfuse belt and fuses it onto the paper sheet. To maintain high printing quality, several maintenance operations have to be carried out, like: toner transfuse belt jittering, which displaces the transfuse belt to prolong its lifespan due to wearing by paper edges; black image operation, which removes paper dust by occasionally printing completely black pages; coarse toner particles removal operation; etc. Most maintenance operations are scheduled after a given number of prints, but must be carried out after a given strict threshold. To perform a maintenance operation, the printing process has to change its power mode, from Run mode, used for printing, to Standby mode, required for maintenance. However, this change can actually trigger pending maintenance operations, which may unnecessary prolong the user waiting time.

As an illustration, in Figure 3b) we depict the situation, where due to inevitable execution of maintenance operation **A**, the ongoing print job is suspended and the power mode of the printer is changed to Standby. However, an unwanted situation occurs, i.e., the power mode change triggers a longer, yet postponable maintenance operation **B** as depicted in Figure 3c). For instance, a black image operation (**A**) must be performed, which takes the time needed to print one page and is activated often, but the switching of the power mode triggers the much longer toner transfuse belt jittering (**B**), thus making the user wait unnecessarily.

The goal of the research performed for this use case was to eliminate undesired emergent behavior due to interactions of otherwise correctly-functioning distributed components, with primary focus at coordinating maintenance operations. Our approach was to synthesize a supervisory coordinator for the maintenance procedures [18], which here we model in the proposed process theory.

Informal description of the printing process An abstract view of the control architecture of a high-end printer is depicted in Figure 4. Print jobs are sent to the printer by means of the user interface. The printer controller communicates with the user and assigns print jobs to the embedded software, which actuates the hardware to realize print jobs. The embedded software is organized in a distributed way, per functional aspect, such as, paper path, printing process, etc. Several managers communicate with the printer controller and each other to assign tasks to functions, which take care of the functional aspects.

We depict a printing process function comprising one maintenance operation in Figure 4. We abstract from all timing behavior, which can be present in some control signals, e.g., execute a maintenance procedure after a given delay. Each function is hierarchically organized as follows: (1) controllers: Target Power Mode and Maintenance Scheduling, which receive control and scheduling tasks from the

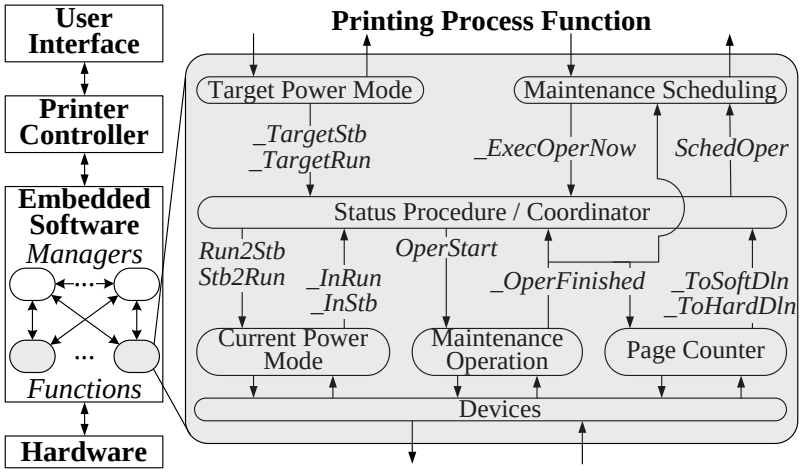

Figure 4: Printing process function.

managers; (2) procedures: Status Procedure, Current Power Mode, Maintenance Operation, and Page Counter, which handle specific tasks and actuate devices, and (3) devices as hardware interface.

The Status Procedure is responsible for coordinating the other procedures given the input from the controllers. It will be implemented as a supervisory coordinator with respect to the coordination rules given below. The Current Power Mode procedure sets the power mode to Run or Standby depending on the enabling signals from the Status Procedure *Stb2Run* and *Run2Stb*, respectively. The confirmation is sent back via the signals *_InRun* and *_InStb*, respectively. Maintenance Operation either carries out maintenance operation or it is idle. The triggering signal is *OperStart* and the confirmation is sent back by *_OperFinished*. The Page Counter procedure counts the printed pages since the last maintenance and sends signals when soft and hard deadlines are reached using *_ToSoftDln* and *_ToHardDln*, respectively. The counter is reset each time the maintenance is finished, by receiving the confirmation signal *_OperFinished* from Maintenance Operation. The controller Target Power Mode defines which mode is requested by the manager by sending the control signals *_TargetStb* and *_TargetRun* to the Status Procedure. Maintenance Scheduling receives a request for maintenance from Status Procedure via the signal *SchedOper*, which it forwards to a manager. The manager confirms the scheduling with the other functions and sends a response back to the Status Procedure via the control signal *_ExecOperNow*. It also receives feedback from Maintenance Operation that the maintenance is finished in order to reset the scheduling.

Plant modeling in $\mathrm{TCP}^*_\perp$ We model the procedures by means of processes. We retain the names of the control signals, turning them into communication actions where appropriate. The controllable communicating channels are the given by $\mathsf{C} = \{\mathit{Run2Stb}, \mathit{Stb2Run}, \mathit{SchedOper}, \mathit{OperStart}\}$, modeled as receive actions in the plant. We note that we abstract from data elements as communication should only enforce ordering of events. The other actions are uncontrollable, also prefixed by _, where only *_OperFinished* is modeled as a communication action, as the procedure Maintenance operation must send signals and reset Page Counter and Maintenance Scheduling. The signals emitted from the plant uniquely identify the state of the plant. For clarity, we also depict the processes in Figure 5, where the signal names are given next to the states that emit them. Page Counter is modeled by the process *C*, where *_OperFinished* is modeled as a receive action, to be synchronized with Maintenance Operation:

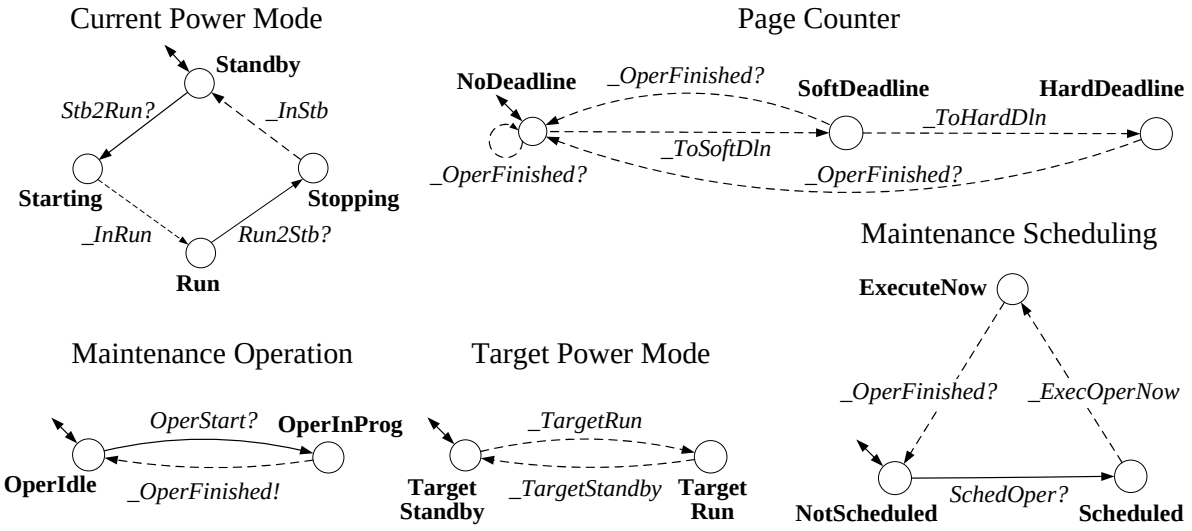

Figure 5: Plant modeling of the Printing Process Function.

$$C \triangleq \Big(in(\textbf{NoDeadline}) \mathbin{^\frown\!\blacktriangle} (\\ \quad \_OperFinished?.1 + \\ \quad \_ToSoftDln.\big(in(\textbf{SoftDeadline}) \mathbin{^\frown\!\blacktriangle} (\\ \quad\quad \_OperFinished?.1 + \_ToHardDln.in(\textbf{HardDeadline}) \mathbin{^\frown\!\blacktriangle} \_OperFinished?.1)))\Big)^*.$$

Maintenance Operation is specified by the process O, where $\_OperFinished$ broadcasts that the maintenance operation has finished:

$$O \triangleq \big(in(\textbf{OperIdle}) \mathbin{^\frown\!\blacktriangle} OperStart?.in(\textbf{OperInProg}) \mathbin{^\frown\!\blacktriangle} \_OperFinished!.1\big)^*.$$

Target Power Mode is modeled by T:

$$T \triangleq \big(in(\textbf{TargetStandby}) \mathbin{^\frown\!\blacktriangle} \_TargetRun.in(\textbf{TargetRun}) \mathbin{^\frown\!\blacktriangle} \_TargetStandby.1\big)^*,$$

whereas Current Power Mode is given by P:

$$P \triangleq \big(in(\textbf{Standby}) \mathbin{^\frown\!\blacktriangle} Stb2Run?.in(\textbf{Starting}) \mathbin{^\frown\!\blacktriangle} \_InRun. \\ in(\textbf{Run}) \mathbin{^\frown\!\blacktriangle} Run2Stb?.in(\textbf{Stopping}) \mathbin{^\frown\!\blacktriangle} \_InStb.1\big)^*.$$

Finally, Maintenance Scheduling is specified as M:

$$M \triangleq \big(in(\textbf{NotScheduled}) \mathbin{^\frown\!\blacktriangle} SchedOper?.in(\textbf{Scheduled}) \mathbin{^\frown\!\blacktriangle} \_ExecOperNow. \\ in(\textbf{ExecuteNow}) \mathbin{^\frown\!\blacktriangle} \_OperFinished?.1\big)^*.$$

Due to the generic valuation effect function, we need to impose additional restriction on the emitted signals. More precisely, we wish that the signals emitted in a process are not ambiguous, e.g., it cannot be that both $in(\textbf{Standby})$ and $in(\textbf{Run})$ are valid at the same time as these are two distinct states that belong to the same process. Note, however, that this situation is possible as one can easily construct a valuation

effect function that always assigns the same values to the above propositional symbols. However, such misconstrued valuations can actually lead to wrong supervised behavior as the supervisor bases its decision on the emitted signals, which are deduced from the valuations. At this point, we have two viable options. One is to make the signal emission complete and rewrite all signal emissions such that the effect function leads to inconsistencies unless it uniquely defines each state. For example, then we would have to rewrite T to T':

$$\begin{aligned} T' \triangleq\ & \big((in(\textbf{TargetStandby}) \wedge \neg in(\textbf{TargetRun}))\, \triangle\!\!\blacktriangleright \_TargetRun. \\ & (\neg in(\textbf{TargetStandby}) \wedge in(\textbf{TargetRun}))\, \triangle\!\!\blacktriangleright \_TargetStandby.1\big)^*, \end{aligned}$$

and adapt the rest of the processes analogously. The other option is to set an invariant process in parallel to the components that will ensure that only one state can be identified per process. To this end, we define the operation $\bigoplus_{P\in S} P \triangleq \bigvee_{P\in S}\left(P \wedge \bigwedge_{Q\in S\setminus\{P\}} \neg Q\right)$ for a set of propositional symbols $S \subseteq \mathsf{N}$, which ensures that only one propositional symbol, i.e., one signal, is exclusively emitted per state. Now, the invariant process I that enforces this restriction can be specified as:

$$I \triangleq \left(\left(\bigwedge_{i=1}^{5} \bigoplus_{P\in\{S_i\}} P\right) \triangle\!\!\blacktriangleright 0\right)^*,$$

where $S_i \subset \mathsf{N}$ for $i \in \{1,\ldots,5\}$ contain the signals emitted by the processes C, O, T, P, and M, respectively, i.e.,

$$\begin{aligned} S_1 &= \{in(\textbf{NoDeadline}), in(\textbf{SoftDeadline}), in(\textbf{HardDeadline})\}, \\ S_2 &= \{in(\textbf{OperIdle}), in(\textbf{OperInProg})\}, \\ S_3 &= \{in(\textbf{TargetStandby}), in(\textbf{TargetRun})\}, \\ S_4 &= \{in(\textbf{Standby}), in(\textbf{Starting}), in(\textbf{Stopping}), in(\textbf{Run})\}, \\ S_5 &= \{in(\textbf{NotScheduled}), in(\textbf{Scheduled}), in(\textbf{ExecuteNow})\}. \end{aligned}$$

Finally, the unsupervised plant can be specified as $U \in \mathsf{P}$ given by:

$$U \triangleq \partial_F(C \parallel O \parallel T \parallel P \parallel M) \parallel I,$$

where $F = \{\_OperFinished?, \_OperFinished!, \_OperFinished!_0?_2, \_OperFinished!?\}$ enforces a three-way communication between C, O, and M. We note that due to the stringent streamlining invariant, the role of the valuation effect function is now diminished and one can simply assume that $\mathit{effect}(a,v) = \mathsf{V}$ for every $a \in \mathsf{A}$ and $v \in \mathsf{V}$.

Coordination requirements We synthesized a coordinator that implements Status Procedure, see Figure 4, which coordinates the maintenance procedures with the rest of the printing process. The following coordination requests describe the behavior of the Status Procedure:

1. Maintenance operations can be performed only when the printing process is in standby;
2. Maintenance operations can be scheduled only if soft deadline has been reached and there are no print jobs in progress or a hard deadline is passed;
3. Maintenance operations can be started only after being scheduled;
4. The power mode of the printing process must follow the power mode dictated by the managers, unless overridden by a pending maintenance operation.

We formalize these control requirements as follows:

1. The maintenance procedure is performed if the process emits the signal $in(\textbf{OperInProg})$, while emitting the signal $in(\textbf{Standby})$ as well:

$$R_1 \triangleq in(\textbf{OperInProg}) \Rightarrow in(\textbf{Standby}).$$

2. For the control signal *SchedOper!* to be sent to Maintenance Scheduling, either one of the following must hold: (1) A soft deadline has been passed, identified by emission of the signal $in(\textbf{SoftDeadline})$, and there are no print jobs waiting, meaning that the target power mode is not in run, identified by the signal $in(\textbf{TargetRun})$; or (2) A hard deadline has been passed, indicated by the signal $in(\textbf{HardDeadline})$. This is captured by the following control requirement:

$$R_2 \triangleq \stackrel{SchedOper!}{\longrightarrow} \Rightarrow (in(\textbf{SoftDeadline}) \wedge \neg in(\textbf{TargetRun})) \vee in(\textbf{HardDeadline}).$$

3. The maintenance operation can be started by sending the control signal *OperStart!* only if it has been scheduled, prompted by the emission of the signal $in(\textbf{ExecuteNow})$:

$$R_3 \triangleq \stackrel{OperStart!}{\longrightarrow} \Rightarrow in(\textbf{ExecuteNow}).$$

4. If we want to switch from standby to run power mode, indicated by sending the control signal *Stb2Run!*, then this has been requested by the target power mode manager by emitting the signal $in(\textbf{TargetRun})$, provided that there are no maintenance operations scheduled, for which the signal $in(\textbf{ExecuteNow})$ should be checked:

$$R_{4,1} \triangleq \stackrel{Stb2Run}{\longrightarrow} \Rightarrow in(\textbf{TargetRun}) \wedge \neg in(\textbf{ExecuteNow}).$$

When switching from run to standby power mode, indicated by sending the control signal *Run2Stb!*, the target power mode should be in standby, given by emission of the signal $in(\textbf{TargetStandby})$. An exception is made when a maintenance operation is scheduled to be executed, given by emission of the signal $in(\textbf{ExecuteNow})$:

$$R_{4,2} \triangleq \stackrel{Run2Stb}{\longrightarrow} \Rightarrow in(\textbf{TargetStandby}) \vee in(\textbf{ExecuteNow}).$$

Supervisor synthesis With respect to the control requirements we synthesized a deadlock- and livelock-free maximally-permissive supervisor [18]. The supervisor sends the control signals upon observation of certain signal combinations, which are given in the form of guards. The indices of the guards correspond to the indices of the control requirements that concern the control signal:

$$g_2 \triangleq (in(\textbf{SoftDeadline}) \wedge in(\textbf{TargetStandby})) \vee in(\textbf{HardDeadline})$$
$$g_3 \triangleq in(\textbf{Standby}) \wedge in(\textbf{ExecuteNow})$$
$$g_{4,1} \triangleq \neg in(\textbf{ExecuteNow}) \wedge in(\textbf{TargetRun}) \wedge \neg in(\textbf{OperInProg})$$
$$g_{4,2} \triangleq (\neg in(\textbf{ExecuteNow}) \wedge in(\textbf{TargetStandby})) \vee in(\textbf{ExecuteNow}).$$

The supervisor is given by $S \in \mathsf{S}$:

$$S \triangleq \Big(g_2 :\rightarrow \mathit{SchedOper}!.1 + g_3 :\rightarrow \mathit{OperStart}!.1 + g_{4,1} :\rightarrow \mathit{Stb2Run}!.1 + g_{4,2} :\rightarrow \mathit{Run2Stb}!.1\Big)^*.$$

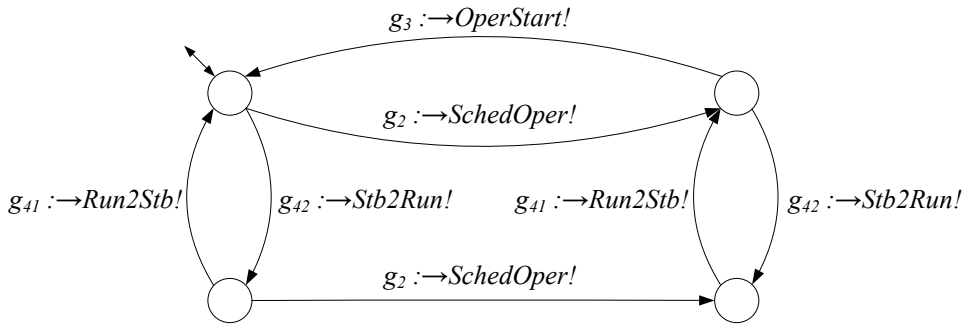

Figure 6: Alternative form of the supervisor

Now, the supervised plant U/S is given by:

$$U/S \triangleq \partial_E(S \parallel U), \text{ where } E = \{c!, c? \mid c \in \mathsf{C}\}.$$

Again, we can show that the supervised plant is partially bisimilar to the original plant with respect to the uncontrollable events by showing that

$$U/S \leq_{\mathsf{A_U}} \xi(U), \text{ where } \xi : c? \mapsto c!? \text{ for } c \in \mathsf{C}.$$

The above form of the supervisor does not provide much information regarding the choices made. It can be visualized as a single state transition system with four outgoing guarded transitions. However, it is not difficult to deduce that initially the event *Run2Stb* is not possible since the initial signal is $in(\textbf{Standby})$. Also, *StartOper* is initially unavailable as the signal $in(\textbf{ExecuteNow})$ is not emitted. In order to better understand the consequences of the control choices made by the supervisor and the thereafter enabled controllable events, we depict an alternative supervisor in Figure 6. We note that both variants of the supervisor produce equivalent supervised behavior (the guards remain the same), the difference being that the supervisor depicted in Figure 6 reveals the consequences of choosing a particular controllable action. We can now observe, that if the operation is scheduled while the printing process is in standby power mode, then it can be directly executed, returning the supervisor to the initial state. However, if the power mode is run, then the maintenance operation can still be scheduled, but the system has to switch to standby power mode before it can be executed.

6 Conclusions and Future Work

We modeled two prominent types of supervisory control loops, one employing event-based observations and the other employing state-based observations. To this end, we revisited the process theory TCP* of [1], where we introduced generic communication actions to model communication between multiple parties, and we applied the developed theory to model the control loop with event-based observations. We classified the processes modeling the unsupervised system and the controller to capture their specific goals. We illustrated our approach on an academic example of coordinating an automated guided vehicle in a production line. To model the control loop with state-based observations as well, we extended the process theory with guarded commands and root signal emission, leading to TCP$^*_\perp$. We reiterated on an industrial study dealing with coordination of maintenance procedures in a printing process of a high-tech printer. We demonstrated that our approach is capable of modeling the interaction in the control loop precisely by distinguishing between the information flows of the observations and the control signals.

Application of process theory in supervisory coordination The work presented in this paper is merely the third step in our investigations regarding application of process theory in supervisory control and coordination. Our prior work identified and employed partial bisimulation as a suitable behavioral relation to capture the central notion of controllability [2]. Based on this relation we developed an efficient minimization procedure for nondeterministic plants that respects controllability. Here, we modeled the most prominent variants of the supervisory control loop and further calibrating the process algebra with respect to the notions that are needed to correctly capture the central notions of supervisory control theory.

The issues are far from resolved. We intend to proceed in several directions of research, where we expect that a process-theoretic approach can advance the theory and/or define the notion more clearly and concisely. One issue that we partially treat in this paper is the notion of partial observability, which is an inherent property of plants in which due to unavailability of sensors certain information is unobservable to the supervisor [9]. There is a lot of work regarding partial observability of events, which can be treated as uncontrollable actions that are not communicated to the supervisor or as silent steps from which the supervisor has to abstract. The first option is already present in the current setting, whereas the second approach is more than familiar in the process-theoretic community. An unavoidable complication in supervisory control is that the supervisor must not make a wrong control choice, irrespective of not being able to observe the correct state of the plant, making partial observability a global property [2]. In the setting with state-based observations, one can easily abstract from state information by emitting slightly ambiguous signals, e.g., instead of uniquely identifying as being in states $\mathbf{S}$ or $\mathbf{T}$, one can emit the signal $in(\mathbf{S}) \vee in(\mathbf{T})$. We intend to further investigate the mechanics of state abstraction in supervisory control.

As expected, there are quantitative extensions of supervisory control theory employing real and stochastic timing, probabilities, and data. However, the supervisory control community seems to struggle with clear and acceptable definitions of controllability, as typically these follow the original approach of [22] and are, thus, given in trace semantics. There are other approaches that are instead based on games, but these often suffer from great computational complexities. We believe that here the community of process theory and verification can contribute a great deal, both in providing suitable definitions and algorithms for minimization and supervisor synthesis. Finally, the supervisor synthesis algorithms almost always have distributed, decentralized, modular, or hierarchical implementations. Concurrency is inherent to our work, and we believe that there are a lot of interesting problems, issues, and challenges that are hidden in this exciting field.

References

[1] J. C. M. Baeten, T. Basten & M. A. Reniers (2010): *Process Algebra: Equational Theories of Communicating Processes*. Cambridge Tracts in Theoretical Computer Science 50, Cambridge University Press.

[2] J. C. M. Baeten, D. A. van Beek, B. Luttik, J. Markovski & J. E. Rooda (2011): *A Process-Theoretic Approach to Supervisory Control Theory*. In: Proceedings of ACC 2011, IEEE. Available from: `http://se.wtb.tue.nl`.

[3] J. C. M. Baeten, B. Luttik, T. Muller & P. van Tilburg (2010): *Expressiveness modulo Bisimilarity of Regular Expressions with Parallel Composition (extended abstract)*. In: Proceedings of EXPRESS 2010, Electronic Proceedings of Theoretical Computer Science 41, pp. 1–15, doi:10.4204/EPTCS.41.1.

[4] J. C. M. Baeten & W. P. Weijland (1990): *Process algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, doi:10.1017/CBO9780511624193.

[5] J.C.M. Baeten & J.A. Bergstra (1997): *Process algebra with propositional signals*. Theoretical Computer Science 177, pp. 381–405, doi:10.1016/S0304-3975(96)00253-8.

[6] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi & G. F. Franklin (1993): *Supervisory control of a rapid thermal multiprocessor*. IEEE Transactions on Automatic Control 38(7), pp. 1040 –1059.

[7] G. Barrett & S. Lafortune (1998): *Bisimulation, the Supervisory Control Problem and Strong Model Matching for Finite State Machines*. Discrete Event Dynamic Systems 8(4), pp. 377–429, doi:10.1023/A:1008301317459.

[8] H. Beohar & P.J.L. Cuijpers (2010): *A theory of desynchronisable closed loops system*. In: Proceedings of ICE 2010, Electronic Proceedings in Theoretical Computer Science 38, Open Publishing Association, pp. 99–114, doi:10.4204/EPTCS.38.10.

[9] C. Cassandras & S. Lafortune (2004): *Introduction to discrete event systems*. Kluwer Academic Publishers.

[10] V. Chandra, Z. Huang, W. Qiu & R. Kumar (2004): *Prioritized Composition With Exclusion and Generation for the Interaction and Control of Discrete Event Systems*. Mathematical and Computer Modelling of Dynamical Systems 9(3), pp. 255 – 280.

[11] M. Fabian & B. Lennartson (1996): *On non-deterministic supervisory control*. Proceedings of the 35th IEEE Decision and Control 2, pp. 2213–2218.

[12] R. J. van Glabbeek (2001): *The linear time–branching time spectrum I*. Handbook of Process Algebra , pp. 3–99.

[13] M. Heymann & F. Lin (1998): *Discrete-Event Control of Nondeterministic Systems*. IEEE Transactions on Automatic Control 43(1), pp. 3–17, doi:10.1109/9.654883.

[14] M. Heymann & G. Meyer (1991): *Algebra of discrete event processes*. Technical Report NASA 102848, NASA Ames Research Center.

[15] R. Kumar & M. A. Shayman (1996): *Nonblocking Supervisory Control of Nondeterministic Systems via Prioritized Synchronization*. IEEE Transactions on Automatic Control 41(8), pp. 1160–1175, doi:10.1109/9.533677.

[16] R. Kumar & C. Zhou (2007): *Control of Nondeterministic Discrete Event Systems for Simulation Equivalence*. IEEE Transactions on Automation Science and Engineering 4(3), pp. 340–349, doi:10.1109/TASE.2006.891474.

[17] C. Ma & W. M. Wonham (2005): *Nonblocking Supervisory Control of State Tree Structures*. Lecture Notes in Control and Information Sciences 317, Springer.

[18] J. Markovski, K. G. M. Jacobs, D. A. van Beek, L. J. A. M. Somers & J. E. Rooda (2010): *Coordination of Resources using Generalized State-Based Requirements*. In: Proceedings of WODES 2010, IFAC, pp. 300–305.

[19] S. Miremadi, K. Akesson & B. Lennartson (2008): *Extraction and representation of a supervisor using guards in extended finite automata*. In: Proceedings of WODES 2008, IEEE, pp. 193 –199.

[20] A. Overkamp (1997): *Supervisory Control Using Failure Semantics and Partial Specifications*. IEEE Transactions on Automatic Control 42(4), pp. 498–510, doi:10.1109/9.566659.

[21] G. D. Plotkin (2004): *A structural approach to operational semantics*. The Journal of Logic and Algebraic Programming 60-61, pp. 17–139, doi:10.1016/j.jlap.2004.05.001.

[22] P. J. Ramadge & W. M. Wonham (1987): *Supervisory Control of a Class of Discrete Event Processes*. SIAM Journal on Control and Optimization 25(1), pp. 206–230, doi:10.1137/0325013.

[23] J. J. M. M. Rutten (1999): *Coalgebra, concurrency, and control*. SEN Report R-9921, Center for Mathematics and Computer Science, Amsterdam, The Netherlands.

[24] P. Tabuada (2008): *Controller synthesis for bisimulation equivalence*. Systems and Control Letters 57(6), pp. 443–452, doi:10.1016/j.sysconle.2007.11.005.

[25] S. Xu & R. Kumar (2008): *Asynchronous implementation of synchronous discrete event control*. In: Proceedings of WODES 2008, IEEE, pp. 181 –186.

[26] C. Zhou, R. Kumar & S. Jiang (2006): *Control of nondeterministic discrete-event systems for bisimulation equivalence*. IEEE Transactions on Automatic Control 51(5), pp. 754–765, doi:10.1109/TAC.2006.875036.