A Supervisor Synthesis Tool for Finite Nondeterministic Automata with Data

Aleksandar Kirilov[1], Darko Martinovikj[1], Kristijan Mishevski[1], Marija Petkovska[1], Zlatka Trajcheska[1], and Jasen Markovski[1,2(✉)]

[1] University Ss. Cyril and Methodius, PB 393, 1000 Skopje, Republic of Macedonia
[2] Eindhoven University of Technology, PB 513, 5600 MB Eindhoven, The Netherlands
j.markovski@tue.nl

Abstract. Supervisory control theory deals with automated synthesis of models of supervisory controllers based on the models of the unsupervised systems and the control requirements. The models of the supervisory controllers are referred to as supervisors. We present a supervisor synthesis tool for finite nondeterministic automata with data-based control requirements. The tool implements a process-theoretic approach to supervisory control theory, which employs the behavioral preorder partial bisimulation to characterize the notion of a supervisor. To illustrate the tool, we remodel an industrial case study dealing with coordination of maintenance procedures of a printing process of a high-tech printer.

1 Introduction

Development of control software with high quality has become a major bottleneck in design and production of high-tech systems [9]. Traditional techniques that employ (re)coding-testing loops struggle to satisfactorily cope with this challenge due to ever-increasing system complexity and frequent design changes in the (informal) control requirements, which results in a large amount of expensive iterations. These issues gave rise to supervisory control theory of discrete-event systems [4,16], which studies automated synthesis of models of supervisory control software that ensure safe and nonblocking coordination of discrete-event behavior of the concurrent components of the system.

Supervisory controllers observe the discrete-event behavior of the system, typically given by sensory information, as depicted in Fig. 1(a). Based upon the made observations, the controllers decide upon activities that are allowed to be carried out safely, avoiding potentially dangerous or otherwise undesired situations, and send back control signals to the hardware actuators. We work under the standard assumption that the supervisory controller timely reacts on system input and we model this *supervisory control feedback loop* as a pair of synchronizing processes [4, 16]. We refer to the model of the uncontrolled system as *plant*, whereas the model of the supervisory controller is referred to as *supervisor*. The supervisory control

Supported by Dutch NWO project ProThOS, no. 600.065.120.11N124.

S. Counsell and M. Núñez (Eds.): SEFM 2013 Collocated Workshops, LNCS 8368, pp. 101–112, 2014.
DOI: 10.1007/978-3-319-05032-4_8,

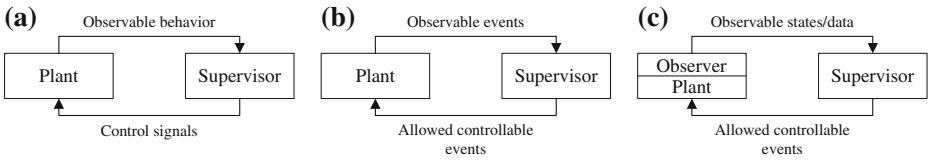

Fig. 1. (a) Generic supervisory control loop; (b) Loop with event-based observations; (c) Loop with an observer and state- or data-based observations

loop in which the system is coupled with the controller is modeled by the synchronization of the plant and the supervisor, resulting in the *supervised plant*, which specifies the behavior of the controlled system.

The activities of the system are traditionally modeled by means of discrete events. The supervisor is synthesized as a process that synchronizes with the plant, employing the synchronization to enable or disable available events in the plant [4]. Traditionally, the events are split into *controllable* and *uncontrollable events*, where the former usually model interaction with the actuators and the latter model user and environment interaction or observation of sensory information. Consequently, the supervisor is allowed to disable controllable events, but it cannot disable any available uncontrollable events, which is an important structural restriction [16]. In addition, the supervised plant must satisfy a set of *control requirements* that model the allowed behavior of the system by restrictions, typically given as safety properties.

Based on the type of observations, we distinguish between control loops with event-based control requirements or state- or data-based control requirements, which depend on the type of available observations from the system. The former situation is depicted in Fig. 1(b), where the allowed behavior is typically specified in terms of allowed languages. Some synthesis tools that allow this type of specifications are TCT [6], UMDES [4], or Supremica [1]. In the latter situation, the supervision relies on state- or data-based observations, that are usually supplied by an auxiliary process to the plant, known as an observer, which provides the supervisor with observation information of interest. An example of a state-based tool is NBC [10], whereas Supremica [1] also admits data-based control requirements in a restricted structural form. There also exist synthesis tools that admit temporal logic specifications, like extensions of the model checker NuSMV [19], but they usually suffer from high computational complexity.

Our contribution is a supervisor synthesis tool with data-based requirements for finite nondeterministic automata with data. Admittedly, the synthesis tool Supremica supports this model, but the control requirements must be specified as automata as well, which leads to certain structural restrictions. We specify the control requirements with respect to the data, independent of the structure of the plant. In addition, we employ a different type of controllability condition: Supremica relies on state-controllability [15], whereas we employ the behavioral relation termed partial bisimulation. It has been shown that partial bisimulation is a coarser notion of controllability that exhibits desirable algebraic properties,

unlike state controllability that is not a preorder relation [12]. Extensions of supervisory control theory with data have a two-fold gain. They allow for a more concise specification due to parametrization of the systems [5,15] and they provide for a greater expressiveness and modeling convenience [7,18].

In the remainder of this paper, we define the model and the corresponding notion of controllability that relies on the partial bisimulation preorder. Then, we discuss the synthesis algorithm and the extraction of a supervisory controller. We illustrate the synthesis tool by revisiting an industrial case study that deals with coordination of maintenance procedures of a printing process of an Océ prototype printer [14]. Due to confidentiality issues, we can only present an obfuscated part of the case study. The goal of the case study is to synthesize a supervisory coordinator that ensures that quality of printing is uncompromised by timely performing maintenance procedures, while interrupting ongoing print jobs as little as possible.

2 Finite Automata with Variables

We model the unsupervised system by means of finite nondeterministic automata with data. For a complete process-theoretic treatment to supervisory control theory, we refer to [2,3,11] for event-, state-, and data-based supervision, respectively. We introduce some preliminary notation.

The set of finite data variables is denoted by V, where given a variable $X \in \mathsf{V}$, its finite domain is denoted by $\mathrm{dom}(X)$. We keep track of the data assignments by employing a function $\delta \in \Delta(V)$, where $\Delta(V) = V \to \mathrm{dom}(V)$. Standard arithmetical expressions, like addition $+$ or subtraction $-$, over a set of variables $V \subset \mathsf{V}$ are denoted by $\mathsf{E}(V)$ and they can be evaluated by a function $\mathrm{evar}_\delta \colon \mathsf{E}(V) \to \mathrm{dom}(V)$. For the sake of clarity and compactness, we do consider invalid expressions that evaluate outside the variable domain. We note that such inconsistent processes can be treated by a straightforward extension of the approach of [3]. We denote Boolean expressions over the set of variables $V \subseteq \mathsf{V}$ by $\mathsf{B}(V)$. The atomic propositions are formed by comparison predicates over variables induced by $\{<, =, >\}$ together with the logical constants false F and true T. To form the Boolean expression, we employ the standard set of logical operators $\{\neg, \wedge, \vee, \Rightarrow\}$ denoting logical negation, conjunction, disjunction, and implication. The Boolean expressions are evaluated with respect to a valuation function $\mathrm{evbl}_\delta \colon \mathsf{B}(V) \to \{\mathrm{F}, \mathrm{T}\}$ that depends on the current data assignments. The set of actions is denoted by A.

We define a finite nondeterministic automaton with data as a tuple $G = (S, A, V, \longmapsto, \gamma, \upsilon, (s_0, \delta_0))$, where

- S is a finite set of states;
- $A \subseteq \mathsf{A}$ is a finite set of event labels;
- $V \subseteq \mathsf{V}$ is a finite set of variables;
- $\longmapsto \subset S \times A \times S$ is a labeled transition relation;

– $\gamma \colon \longmapsto \to \mathsf{B}(V)$ are transition guards;
– $\upsilon \colon (\longmapsto \times V) \rightharpoonup \mathsf{E}(V)$ is a partial variable updating function; and
– (s_0, δ_0) is the initial state $s_0 \in S$ and initial data assignment $\delta_0 \in \Delta$.

We employ infix notation and we write $s \overset{a}{\longmapsto} s'$ for $(s, a, s') \in \longmapsto$.

The dynamics of the finite automaton with variables G is induced by the instantiated labeled transition system $\longrightarrow \subseteq S \times \Delta(V) \times A \times S \times \Delta(V)$ that depends on the valuation of the transition guards with respect to the current data assignments. Its semantics is given by the instantiated labeled transition system $\mathrm{T}(G)$, which is defined by the tuple $\mathrm{T}(G) = (S \times \Delta(V), A, \longrightarrow, (s_0, \delta_0))$, where the set of states is coupled with the valuation of the data variables, the initial state is induced by the initial state s_0 of the automaton with the initial data assignment δ_0, and the dynamics of the instantiated labeled transition relation is captured by operational rule (1), where (s, δ) denotes the state $s \in S$ in the data assignment environment given by δ:

$$\frac{s \overset{a}{\longmapsto} s',\ \mathrm{evbl}_\delta(\gamma(s,a,s')) = \mathrm{T}, \qquad \text{for all } X \in V \colon \delta'(X) = \begin{cases} \mathrm{evar}_\delta(\upsilon((s,a,s'),X)), & \text{if } ((s,a,s'),X) \in \mathrm{dom}(\upsilon) \\ \delta(X), & \text{otherwise} \end{cases}}{(s,\delta) \xrightarrow{a} (s',\delta')}. \tag{1}$$

Rule (1) states that labeled transitions are instantiated when such transition is defined in the automaton, the guard of that transition evaluates to true, whereas the variables are updated according to the variable updating function. We note that if the set of variables V of the automaton G is empty, i.e., $V = \emptyset$, then $\longmapsto$ and $\longrightarrow$ coincide, provided that the (then trivial) transition guards are set to be true, and G reduces to a standard automaton. By $\xrightarrow{t}{}^*$, we denote the multistep labeled transition relation for $t \in A^*$. We define it inductively as $(s, \delta) \xrightarrow{\varepsilon}{}^* (s, \delta)$ for the empty trace ε, and $(s, \delta) \xrightarrow{ta}{}^* (s', \delta')$ if there exists $(s'', \delta'') \in S \times \Delta(V)$ such that $(s, \delta) \xrightarrow{t}{}^* (s'', \delta'')$ with $t \in \mathsf{A}^*$ and $(s'', \delta'') \xrightarrow{a} (s', \delta')$ with $a \in \mathsf{A}$.

To model the behavior of the supervised system, we need to define a synchronous composition of two finite nondeterministic automata with variables. In general, this composition cannot be consistently defined due to conflicts induced by the partial assignment functions υ. For example, if two automata synchronize on transitions that update the same variable to two different values, then this synchronization leads to a conflict as the data assignment cannot be consistently executed [18]. Again, for the sake of clarity, we do not consider these conflicting situations, which are easily detectable as none of the conditions for the synchronization from below apply. Moreover, the synchronization of the plant and the supervisor is always well-defined as the supervisor does not update any shared variables with the plant, so these conflicting situations are not of importance in the setting of this paper.

By $f|_D$ we denote the restriction of the function f to the domain $\mathrm{dom}(f) \cap D$. By $f[g] = f|_{\mathrm{dom}(f) \setminus \mathrm{dom}(g)} \cup g$ we denote the replacement of the function f by g on

their common domain. Given two automata $G_1 = (S_1, A_1, V_1, \longmapsto_1, \gamma_1, \upsilon_1, (s_{01}, \delta_{01}))$ and $G_2 = (S_2, A_2, V_2, \longmapsto_2, \gamma_2, \upsilon_2, (s_{02}, \delta_{02}))$ such that $\delta_{01}|_{\mathrm{dom}(\delta_{02})} = \delta_{02}|_{\mathrm{dom}(\delta_{01})}$, we define their synchronous composition as $G_1 \parallel G_2 = (S_1 \times S_2, A_1 \cup A_2, V_1 \cup V_2, \longmapsto, \gamma, \upsilon, ((s_{01}, s_{02}), \delta_0))$, where $\delta_0 = \delta_{01}[\delta_{02}] = \delta_{02}[\delta_{01}]$, $\longmapsto$, γ, and υ are defined by Eqs. (2)–(4) as follows.

$$(s_1, s_2) \overset{a}{\longmapsto} \begin{cases} (s_1', s_2), & \text{if } s_1 \overset{a}{\longmapsto}_1 s_1', a \in A_1 \setminus A_2 \\ (s_1, s_2'), & \text{if } s_2 \overset{a}{\longmapsto}_2 s_2', a \in A_2 \setminus A_1 \\ (s_1', s_2'), & \text{if } s_1 \overset{a}{\longmapsto}_1 s_1', s_2 \overset{a}{\longmapsto}_2 s_2', a \in A_1 \cap A_2 \end{cases} \tag{2}$$

$$\gamma((s_1, s_2), a, (s_1', s_2')) = \begin{cases} \gamma_1(s_1, a, s_1'), & \text{if } s_1 \overset{a}{\longmapsto}_1 s_1', a \in A_1 \setminus A_2 \\ \gamma_2(s_2, a, s_2'), & \text{if } s_2 \overset{a}{\longmapsto}_2 s_2', a \in A_2 \setminus A_1 \\ \gamma_1(s_1, a, s_1') \wedge \gamma_2(s_2, a, s_2') & , \text{if } s_1 \overset{a}{\longmapsto}_1 s_1', s_2 \overset{a}{\longmapsto}_2 s_2', a \in A_1 \cap A_2 \end{cases} \tag{3}$$

$$\upsilon(((s_1, s_2), a, (s_1', s_2')), X) = \begin{cases} \upsilon_1((s_1, a, s_1'), X), & \text{if } ((s_1, a, s_1'), X) \in \mathrm{dom}(\upsilon_1), ((s_2, a, s_2'), X) \notin \mathrm{dom}(\upsilon_2) \\ \upsilon_2((s_2, a, s_2'), X), & \text{if } ((s_2, a, s_2'), X) \in \mathrm{dom}(\upsilon_2), ((s_1, a, s_1'), X) \notin \mathrm{dom}(\upsilon_1) \\ \upsilon_1((s_1, a, s_1'), X), & \text{if } \begin{array}{l} ((s_1, a, s_1'), X) \in \mathrm{dom}(\upsilon_1), ((s_2, a, s_2'), X) \in \mathrm{dom}(\upsilon_2), \\ \upsilon_1((s_1, a, s_1'), X) = \upsilon_2((s_2, a, s_2'), X) \end{array} \end{cases} \tag{4}$$

Unlike [18] that defines the synchronous parallel composition in terms of the instantiated labeled transition systems, we define the synchronous parallel composition directly in terms of automata. We note that both definition are compatible [12], i.e., they induce the same labeled transition systems.

To capture the notion of controllability, we employ the behavioral relation termed partial bisimulation, originally proposed in [17] as a suitable relation to capture controllability of deterministic discrete-event systems. The notion was lifted in [2] to a process theory for supervisory control of nondeterministic discrete-event systems. Here, we provide a variant for finite nondeterministic automata with data.

Partial bisimulation is parameterized by a so-called bisimulation action set $B \subseteq \mathsf{A}$. Intuitively, this relation states that all transitions of the first automaton should be simulated by the second automaton, whereas the transitions with labels in the bisimulation action set should be bisimulated in the sense of [8]. In the supervisory control setting, the bisimulation action set comprises the uncontrollable actions that must always be enabled both in the original and the supervised plant, whereas controllable events are only simulated as they are possibly restricted by the supervisor.

Let $\mathrm{T}_1 = (Q_1, A_1, \longrightarrow_1, s_{01})$ and $\mathrm{T}_2 = (Q_2, A_2, \longrightarrow_2, s_{02})$ be two transition systems. We say that a relation $R \subseteq Q_1 \times Q_2$ is a partial bisimulation with respect to a bisimulation action set $B \subseteq A_2$, if for all $(q_1, q_2) \in R$, it holds that:

1. if $q_1 \xrightarrow{a} q_1'$ for $a \in A_1$ and $q_1' \in Q_1$, then there exists $q_2' \in Q_2$ such that $q_2 \xrightarrow{a} q_2'$ and $(q_1', q_2') \in R$;
2. if $q_2 \xrightarrow{b} q_2'$ for $b \in B$ and $q_2' \in Q_2$, then there exists $q_1' \in Q_1$ such that $q_1 \xrightarrow{a} q_1'$ and $(q_1', q_2') \in R$;

If R is a partial bisimulation relation such that $(q_{01}, q_{02}) \in R$, then T_1 is partially bisimilar to T_2 with respect to B and we write $\mathrm{T}_1 \leq_B \mathrm{T}_2$. If $\mathrm{T}_2 \leq_B \mathrm{T}_1$ holds as well, we write $\mathrm{T}_1 =_B \mathrm{T}_2$.

We note that due to the first condition, it must hold that $A_1 \subseteq A_2$, whereas due to the second condition, it holds that $B \subseteq A_1$ as well. It can be shown that partial bisimilarity is a preorder [2]. Moreover, following the guidelines of [17], it can be shown that $\leq_B$ is a partial bisimulation relation with respect to B. Thus, we obtain standard results for the partial bisimulation preorder and equivalence, similarly as for simulation preorder and equivalence [8]. Moreover, the partial bisimulation preorder is a precongruence with respect to the most prominent process operations [2]. Finally, we note that $\mathrm{T}_1 =_{A_1 \cup A_2} \mathrm{T}_2$ amounts to bisimulation, whereas $\mathrm{T}_1 \leq_\emptyset \mathrm{T}_2$ reduces to simulation preorder and $\mathrm{T}_1 =_\emptyset \mathrm{T}_2$ reduces to simulation equivalence [2].

3 Supervisor Synthesis

As discussed above, we split the action set A to set of controllable C and uncontrollable U actions such that $\mathsf{C} \cap \mathsf{U} = \emptyset$ and $\mathsf{C} \cup \mathsf{U} = \mathsf{A}$. The plant is typically modeled by a set of synchronizing components, ultimately resulting in automaton $P = (S_P, A_P, V_P, \longmapsto_P, \gamma_P, \upsilon_P, (s_{0P}, \delta_0))$. We note that we assume that the parallel composition of the components is well-defined and that there are no restrictions regarding nondeterministic behavior inside the plant.

We require, however, that the supervisor is a deterministic process that sends unambiguous feedback to the plant. Moreover, the supervisor cannot alter the internal state of the plant as it only observes its discrete-event behavior, i.e., it does not comprise any variable assignments [11]. In the setting of this paper, the supervisor relies on data observations from the plant to make supervision decisions in the vein of [11,15]. Its behavior is given as an deterministic automaton $S = (S_S, A_S, V_S, \longmapsto_S, \gamma_S, \emptyset, (s_{0S}, \delta_0))$, where $V_S \subseteq V_P$, and the labeled transition function $\longmapsto_S$ is such that if $s \xmapsto{a}_S s'$ and $s \xmapsto{a}_S s''$, then $s' = s''$ for every $s, s', s'' \in S_S$ and $a \in A_S$. The supervisor does not necessarily synchronize on all events from the plant, i.e., in general $A_S \subseteq A_P$, implying that the events in the set $A_P \setminus A_S$ are unconditionally enabled. As the supervisor does not update any variables, i.e., $\upsilon_S = \emptyset$, there arise no conflicts in Eq. (4) for the update function of the synchronization, and the synchronous composition $P \parallel S$ is always well-defined.

The composition $P \parallel S$ models the supervised plant, i.e., the behavior of the controlled system as given by the supervisory feedback loop of Fig. 1(c). To state that the supervisor has no control over the uncontrollable actions, i.e., all available uncontrollable actions in the reachable states should be enabled,

we employ the partial bisimulation preorder. We express this controllability condition by requesting that the transition system of the supervised plant is partially bisimulated by the transition system of the original plant with respect to the uncontrollable events, i.e.,

$$\mathrm{T}(P \parallel S) \leq_U \mathrm{T}(P). \tag{5}$$

It can be shown that for deterministic processes, relation (5) reduces to the original notion of controllability of [12,16,17].

The set of control requirements R comprises control requirements with the following form R:

$$R ::= \phi \mid \xrightarrow{a} \Rightarrow \phi,$$

where $\phi \in \mathsf{B}$ and $a \in \mathsf{A}$. A given instantiated state (s, δ) satisfies a requirement $R \in \mathsf{R}$, notation $(s, \delta) \models R$, if the following is satisfied:

- $(s, \delta) \models \phi$ if and only if $\mathrm{evbl}_\delta(\phi) = \mathrm{T}$; and
- $(s, \delta) \models \xrightarrow{a} \Rightarrow \phi$ if and only if for all $(s, \delta) \in S \times \Delta$ such that $(s, \delta) \xrightarrow{a}$ it holds that $(s, \delta) \models \phi$.

The first form of control requirements enforces an invariant on the data assignments that must hold for all states of the instantiated transition system $\mathrm{T}(P \parallel S)$, whereas the second form restricts the possible occurrences of events, i.e., outgoing events are conditioned by the data assignments.

In addition to conforming to the control requirements, we also require that the supervisor is nonblocking, i.e., it prevents deadlock and livelock behavior in the system. Deadlock behavior occurs in states where no outgoing transitions are possible, whereas livelocks occur when the system remains in a set of states in which it cannot successfully execute its tasks, nor leave this set of states. We model successful termination by marking certain states as final, referred to as marked states in the literature [4,16], denoted by $M \subseteq S$ for a given state set S. The supervisor must assure that a marked state is reachable from all reachable states in the supervised plant.

The synthesis algorithm is an adaptation of the synthesis algorithms of [4,15,16], which employ backtracking from the marked states in order to ensure nonblocking behavior, whereas controllability is ensured by eliminating all blocking states and their predecessors that are reachable by (inverse) uncontrollable transitions. To this end, we define the notion of an inverse uncontrollable reach. Given an instantiated state (s, δ), we inductively define its reverse uncontrollable reach $\mathrm{UR}(s, \delta)$ as follows. Initially, $\mathrm{UR}(s, \delta) = \{(s, \delta)\}$. For every state (s', δ') such that $(s', \delta') \xrightarrow{a} (s, \delta)$ for some $a \in \mathsf{U}$, we put $\mathrm{UR}(s, \delta) = \mathrm{UR}(s, \delta) \cup \mathrm{UR}(s', \delta') \cup \{(s', \delta')\}$. Note that given a state (s, δ), all incoming transitions to its uncontrollable reach $\mathrm{UR}(s, \delta)$ from states outside $\mathrm{UR}(s, \delta)$ are labeled by controllable actions.

We summarize the synthesis algorithm of the maximal supervised behavior in Alg. 1. Line 1 instantiates the labeled transition system. Lines 2–10 eliminate the states or transitions that do not conform to the control requirements. When

Alg. 1: An algorithm for computing a maximal supervised behavior for a given finite automaton with data $G = (S, A, V, \longmapsto, \gamma, \upsilon, (s_0, \delta_0))$, a set of final states $M \subseteq S$, and a set of control requirements R

1 Compute the instantiated labeled transition system $\mathrm{T}(G) = (Q, A, \longrightarrow, (s_0, \delta_0)$, $Q = S \times \Delta(V)$;
2 **for** $\phi \in R$ *and* $(s, \delta) \in (S \times \Delta(V))$ **do**
3 **if** $(s, \delta) \not\models \phi$ **then**
4 Eliminate $\mathrm{UR}(s, \delta)$ from $\mathrm{T}(G)$, $Q = Q \setminus \mathrm{UR}(s, \delta)$;
5 **for** $\xrightarrow{a} \Rightarrow \phi \in R$ *and* $(s, \delta) \in (S \times \Delta(V))$ **do**
6 **if** $(s, \delta) \not\models \xrightarrow{a} \Rightarrow \phi$ **then**
7 **if** $a \in \mathsf{U}$ **then**
8 Eliminate $\mathrm{UR}(s, \delta)$ from $\mathrm{T}(G)$, $Q = Q \setminus \mathrm{UR}(s, \delta)$;
9 **else**
10 Eliminate all transitions $(s, \delta) \xrightarrow{a}$;
11 **repeat**
12 $B = \emptyset$;
13 **for** $(s, \delta) \in Q$ **do**
14 **if** $\nexists\, t \in A^*$ *and* $(s', \delta') \in Q$ *such that* $s' \in M$ *and* $(s, \delta) \xrightarrow{t}{}^{*} (s', \delta')$ **then**
15 $B = B \cup \{(s, \delta)\}$;
16 **for** $(s, \delta) \in B$ **do**
17 Eliminate $\mathrm{UR}(s, \delta)$ from $\mathrm{T}(G)$, $Q = Q \setminus \mathrm{UR}(s, \delta)$;
18 **until** $B = \emptyset$;

a state is eliminated, then its complete inverse uncontrollable reach must be eliminated from the labeled transition system. The elimination of these states, actually requires that controllable transitions are disabled by the supervisor. Lines 2–4 only consider the data-based invariants, whereas lines 5–10 take care of restrictions of labeled transitions. We note that if the transition is controllable, then it can be safely disabled, whereas if it is uncontrollable, then the whole state with its inverse uncontrollable reach must be eliminated.

Once the control requirements are applied, we iteratively ensure nonblockingness by eliminating states that cannot reach marked states in lines 13–17 and checking if by eliminating their inverse uncontrollable reach, we have made some other states blocking, given as the end condition of the repeat loop $B = \emptyset$, where B is the set that holds the blocking states. The end result of Alg. 1 is the maximal restriction of the instantiated labeled transition system of the plant that conforms to the control requirements. By comparison with the original system, we compute the supervisor as a function $\mathrm{sup}\colon \delta \to 2^{(A \cap \mathsf{C})}$ in the vein of [15]. The correctness of the algorithm with respect to our notion of controllability is by construction as the uncontrollable reach is preserved for every state in the supervised plant. This directly implies partial bisimulation with respect to the uncontrollable events on the controllable restriction of the plant. As an adaptation of the

synthesis algorithms of [4,15,16], our algorithm has a comparable polynomial worst-case complexity in the number of transitions of the system.

We note that the internal representation of the instantiated labeled transition system can be optimized by employing binary decision diagrams see, e.g., [15]. We decided to keep the state space explicit as a preparation for future work, where we intend to employ parallel algorithms for supervisor synthesis that will harness the computing power of modern multicore processors. To implement the synthesis tool, which is available from [13], we employed Java and the supporting software package JFLAP, see http://www.jflap.org/, that enables libraries for manipulation of formal languages and automata.

4 Supervisory Coordination of Maintenance Procedures

We illustrate the modeling process on a case study involving coordination of maintenance procedures of a printing process of a high-end Océ printer of [14]. We abstractly depict a printing process function in Fig. 2, where the control architecture of the printer is given to the left. Once a user initiates a print job, the job is forwarded to the printer controller that coordinates different parts of the printer. Here, we coordinate the function responsible for the printing process, which applies the toner image onto the toner transfuse belt and fuses it onto the paper sheet. This function coordinates the power mode of the printer with the maintenance procedures. Namely, the printer executes print jobs in run mode of operation. However, to maintain high printing quality, several maintenance operations have to be carried out, e.g., coarse toner particles removal operation that ensures high quality prints. However, to perform a maintenance operation, the printing process needs to switch to standby mode of operation. Moreover, maintenance operations are scheduled based on the amount of prints since the last performed maintenance. There are two types of deadlines: soft deadlines, which denote that a maintenance operation can be scheduled, and hard deadlines, which denote that the maintenance must be scheduled. Maintenance procedures with expired soft deadlines can be postponed if there is an ongoing print job, but hard deadlines must be respected not to compromise print quality.

A printing process function comprising one maintenance operation in depicted in Fig. 2. The supervisory control problem is to synthesis a model of the Status Procedure, which is responsible for coordinating the other procedures given input from the controllers. The plant that models the printing process function is given in Fig. 2. Uncontrollable events are underscored, whereas variable updates are placed below transitions labels. Initial states have incoming arrows, whereas marked states are gray. The plant is formed by the synchronization of the automata in Fig. 2. Current Power Mode sets the power mode to run or standby using *Stb2Run* and *Run2Stb*, respectively, and sends back feedback by employing *_InRun* and *_InStb*, respectively. Maintenance Operation either carries out a maintenance operation, started by *OpStart* or it is idle. The confirmation is sent back by the event *_OpFin*, which synchronizes with Maintenance Scheduling and Page Counter. Page Counter announces when soft or hard deadlines are reached using *_SoftDln* and *_HardDln*, respectively. The page counter

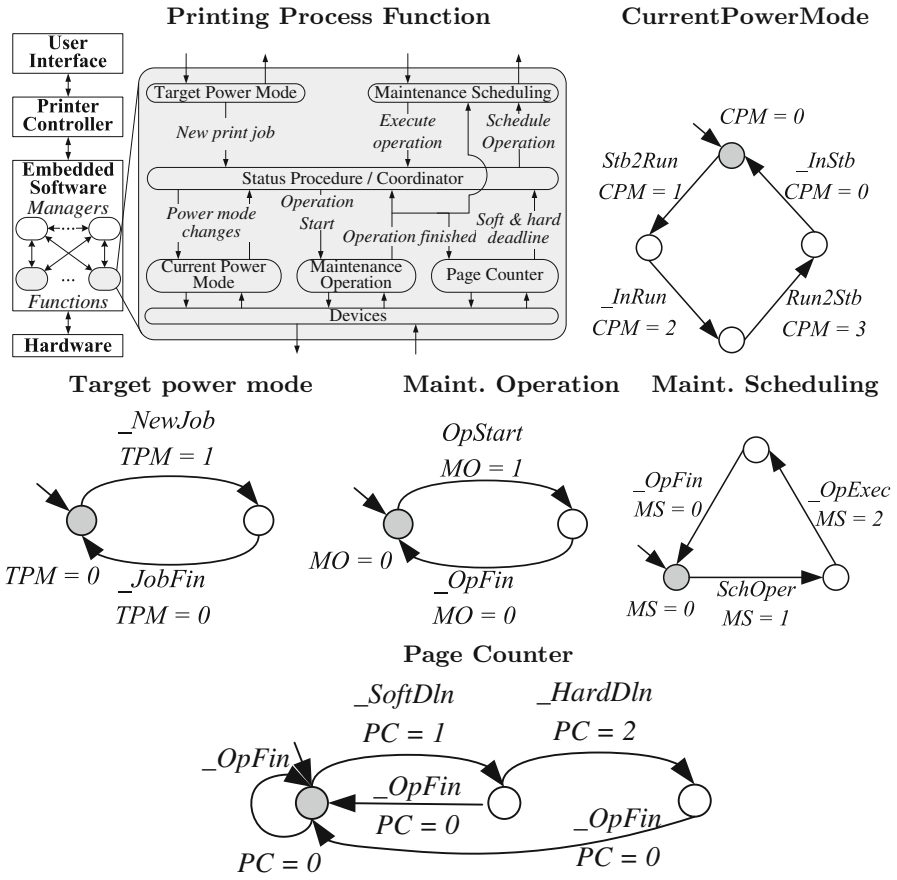

Fig. 2. Printing process function and plant

is reset, triggered by the synchronization on *_OpFin*, each time the maintenance is finished. The controller Target Power Mode sends signals regarding incoming print jobs to Status Procedure by *_NewJob*, which should set the printing process to run mode for printing and standby mode for maintenance and power saving. When the print job is finished, the signal *_NoJob* is sent. Maintenance Scheduling receives a request for maintenance with respect to expiration of Page Counter from Status Procedure, by the signal *SchOper* and forwards it to the manager. The manager confirms the scheduling with the other functions and sends a response back to the Status Procedure, using *_ExOper*. It also receives feedback from Maintenance Operation that the maintenance is finished in order to reset the scheduling, again triggered by *_OpFin*.

The coordination is performed according to the following requirements:

1. Maintenance operations can be performed only when Printing Process Function 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. Only scheduled maintenance operations can be started;
4. The power mode of the printing process must follow the power mode dictated by the managers, unless overridden by a pending maintenance operation.

For a detailed account of the model-based systems engineering process and specification and formalization of the control requirements, we refer to [14].

1. To model this requirement, we consider the states from Current Power Mode and Maintenance Operation, identified by $CPM = 1$ and $MO = 2$, respectively. We require that it must always hold

$$MO = 2 \Rightarrow CPM = 1. \tag{6}$$

2. The states identified by $PC = 1$ and $PC = 2$ indicate when soft and hard deadline is reached, respectively. State with $TPM = 1$ of Target Power Mode states that there is a print job in progress. The event *SchOper* is responsible for scheduling maintenance procedures. We specify the requirement as follows:

$$\stackrel{SchOper}{\longrightarrow} \Rightarrow (PC = 2 \wedge \neg TPM = 2) \vee PC = 3. \tag{7}$$

3. The maintenance operation can be started when the maintenance scheduling is completed, which is modeled as:

$$\stackrel{OpStart}{\longrightarrow} \Rightarrow MS = 3. \tag{8}$$

4. The last condition is modeled by two separate requirements for switching from Run to Standby mode, and vice versa. We can change from run to standby mode if this is required by the manager, i.e., identified by $TPM = 2$, and there is no need to start a maintenance operation, identified by $MS \neq 3$. The transitions labeled by *Stb2Run* are enabled as follows:

$$\stackrel{Stb2Run}{\longrightarrow} \Rightarrow TPM = 2 \wedge \neg MS = 3. \tag{9}$$

In the other direction, we have:

$$\stackrel{Run2Stb}{\longrightarrow} \Rightarrow TPM = 1 \vee MS = 3. \tag{10}$$

Employing the control requirements of Eqs. (6)–(10), we synthesize a supervisor equivalent to the one of [14].

5 Concluding Remarks

We presented a tool for supervisor synthesis based on a process-theoretic approach to supervisory control for finite nondeterministic automata with data. The approach relies on the partial bisimulation preorder to capture controllability of nondeterministic discrete-event systems. To illustrate the modeling process, we revisited an industrial case study dealing with supervisory coordination maintenance procedures of a high-tech printer.

References

1. Akesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems. In: Proceedings of WODES 2006. pp. 384–385. IEEE (2006)
2. Baeten, J.C.M., van Beek, D.A., Luttik, B., Markovski, J., Rooda, J.E.: A process-theoretic approach to supervisory control theory. In: Proceedings of ACC 2011, pp. 4496–4501. IEEE (2011)
3. Baeten, J., van Beek, D., van Hulst, A., Markovski, J.: A process algebra for supervisory coordination. In: Proceedings of PACO 2011. EPTCS, vol. 60, pp. 36–55. Open Publishing Association (2011)
4. Cassandras, C., Lafortune, S.: Introduction to discrete event systems. Kluwer Academic, Dordrecht (2004)
5. Chen, Y.L., Lin, F.: Modeling of discrete event systems using finite state machines with parameters. In: Proceedings of CCA 2000, pp. 941–946 (2000)
6. Feng, L., Wonham, W.M.: TCT: a computation tool for supervisory control synthesis. In: Proceedings of WODES 2006, pp. 388–389. IEEE (2006)
7. Gaudin, B., Deussen, P.: Supervisory control on concurrent discrete event systems with variables. In: Proceedings of ACC 2007, pp. 4274–4279 (2007)
8. van Glabbeek, R.J.: The linear time – branching time spectrum I. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 3–99. Elsevier, Amsterdam (2001)
9. Leveson, N.: The challenge of building process-control software. IEEE Softw. **7**(6), 55–62 (1990)
10. Ma, C., Wonham, W.M.: Nonblocking Supervisory Control of State Tree Structures. LNCIS, vol. 317. Springer, Heidelberg (2005)
11. Markovski, J.: Communicating processes with data for supervisory coordination. In: Proceedings of FOCLASA 2012. EPTCS, vol. 91, pp. 97–111. Open Publishing Association (2012)
12. Markovski, J.: Controllability for nondeterministic finite automata with variables. In: Proceedings of ICSOFT 2013. CCIS, Springer (2013) (To appear)
13. Markovski, J.: Supervisor synthesis tool and demo models. http://sites.google.com/site/jasenmarkovski (2013)
14. Markovski, J., Jacobs, K.G.M., van Beek, D.A., Somers, L.J.A.M., Rooda, J.E.: Coordination of resources using generalized state-based requirements. In: Proceedings of WODES 2010. pp. 300–305. IFAC (2010)
15. Miremadi, S., Akesson, K., Lennartson, B.: Extraction and representation of a supervisor using guards in extended finite automata. In: Proceedings of WODES 2008, pp. 193–199. IEEE (2008)
16. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete-event processes. SIAM J. Control Opt. **25**(1), 206–230 (1987)
17. Rutten, J.J.M.M.: Coalgebra, concurrency, and control. In: Boel, R., Stremersch, G. (eds.) Proceedings of WODES 2000, pp. 31–38. Kluwer, Dotretch (2000)
18. Skoldstam, M., Akesson, K., Fabian, M.: Modeling of discrete event systems using finite automata with variables. In: Proceedings of CDC 2007, pp. 3387–3392. IEEE (2007)
19. Ziller, R., Schneider, K.: Combining supervisor synthesis and model checking. ACM Trans. Embed. Comput. Syst. **4**(2), 331–362 (2005)