Synthesis of controlled behavior with modules of signal nets

Gabriel Juhás, Robert Lorenz, Christian Neumair

Angaben zur Veröffentlichung / Publication details:

Juhás, Gabriel, Robert Lorenz, and Christian Neumair. 2004. "Synthesis of controlled behavior with modules of signal nets." In *Applications and theory of petri nets 2004: 25th International Conference, ICATPN 2004, Bologna, Italy, June 21–25, 2004*, edited by Jordi Cortadella and Wolfgang Reisig, 3099:238–57. Berlin [u.a.]: Springer. https://doi.org/10.1007/978-3-540-27793-4_14.

Synthesis of Controlled Behavior with Modules of Signal Nets

Gabriel Juhás, Robert Lorenz, and Christian Neumair*

Lehrstuhl für Angewandte Informatik
Katholische Universität Eichstätt, 85071 Eichstätt, Germany
{gabriel.juhas,robert.lorenz,christian.neumair}@ku-eichstaett.de

Abstract. In this paper we present a methodology for synthesis of controlled behavior for systems modelled by modules of signal sets. Modules of signal nets are modules, which are based on Petri nets enriched by two kinds of signals and an signal input/output structure. They are also known as net condition/event systems (or modules) [5, 4, 6]. Given an uncontrolled system (a plant) modelled by a module of a signal net, and a control specification given as a regular language representing the desired signal output behavior of this system, we show how to synthesize the maximal permissive and nonblocking behavior of the plant respecting the control specification. Such a behavior serves as an input for an algorithm (presented in [11]), which computes a controller realized as a module of a signal net which in combination with the plant module ensures this behavior.

1 Introduction

In classical control theory it is given a system which can interfere with environment via inputs and outputs. The aim of its control is to ensure desired behavior by giving the system right inputs in order to get the right outputs. The central idea in control theory is, that system and control build a so called *closed loop* (or *feedback loop*), which means, roughly speaking, that the control gives inputs to the system based on the system outputs which are observed by the control. In this paper, we are interested in control of discrete event systems, where the dynamic behavior of a system is described by occurrence of discrete events changing the states of the system. The crucial question to be answered when choosing a formalism for modelling control systems is how to formalize "giving inputs and observing outputs".

An event of a system can have two kinds of inputs: Actuators, which try to force the event, or sensors, which can prohibit the event. Events associated to inputs are called *controllable*. Of course there can be uncontrollable events in the system. Regarding for example a printer, a "paper jam" event can occur without any influence from the control. The following two kinds of outputs can be observed: Either the occurrence of an event (via actuators) or the fact that a state is reached (via sensors). Events resp. states associated to outputs are called *observable*. Of course there can be unobservable events resp. states. It would be natural to model control of a discrete event system by

* Supported by DFG: Project "SPECIMEN".

influencing its behavior by actuators and sensors in order to observe desired outputs as described above.

However, the solution in the discrete event control community, which is now quite accepted, is to use only the sensors. More exactly, in supervisory control [2, 14] the events of the system to be controlled are divided as above into controllable and uncontrollable. But the controllable events can only be enabled/prohibited by a supervisor. Thus, in supervisory control actuators can only be modelled indirectly using the "sensor principle" by prohibiting all controllable events, except the event which is actuated ([1], pp. 185 - 202). There arises the natural question, why not directly model actuators? In our paper, we adapt the framework of supervisory control providing a methodology for control of discrete event systems using **both** concepts, namely actuators and sensors. Such a methodology with the slogan "forcing and prohibiting instead of only prohibiting" would be more appropriate for the class of discrete event systems, where actuators and commands are used in practice.

As a modelling formalism, we use modules communicating by means of the above described signals. This formalism was developed in the series of papers [5, 4, 6] under the name net condition/event systems. In this paper we are using the name signal nets. One reason is that the name condition/event nets is used in the Petri net context for a well known basic net class. A signal net is a Petri net enriched by *event signals*, which force the occurrence of (enabled) events (typically switches), and *condition signals* which enable/prohibit the occurring of events (typically sensors). Adding input and output signals to a signal net, one gets a *module of a signal net*. Modules of signal nets can be composed by connecting their respective input and output signals. There are several related works employing modules of signal nets in control of discrete event systems. In [5, 4, 6] effective solutions for particular classes of specifications, such as forbidden states, or simple desired and undesired sequences of events, are described. Recently, an approach for control specifications given by cycles of observable events was presented in [13]. However, in [13] the actuators are used only to observe events of the controlled system, but surprisingly, for control actions only condition signals for prohibiting events are taken. In our paper, we consider a general class of control specification in form of a language over steps of event outputs (steps of observable events). We have steps (i.e. sets) of outputs, rather then simple outputs, because some outputs can be simultaneously synchronized by an event of the system. We allow also steps containing an input with some outputs. Such a situation describes that an input signal is trying to synchronize a controllable event of the system, which is also observable. So the controller can immediately (i.e. in the same step) observe whether the input signal has forced the event to occur or not. However, since the control is assumed to send inputs based on observed outputs (as stated in the beginning), we do not allow the symmetric situation: observable events can not synchronize inputs in the same step.

In our framework we identify which input signals have to be sent to the module of the plant in order to observe only such sequences of (steps of) output signals, which are prefixes of the control specification, and every sequence of (steps of) output signals can be completed to a sequence of output signals belonging to the control specification (i.e. the behavior is nonblocking). We construct a language over steps of input and output signals of the module of the plant, which represents the maximally permissive nonblocking behavior and fulfills the control specification.

In [11], moreover the construction of a control module (of a signal net), which will in composition with the plant module realize this maximally permissive nonblocking behavior, is shown. It is proven that such a control module always exists, if there is such a behavior of the plant fulfilling the control specification.

The paper is organized as follows: After introducing some preliminary mathematical notations in Section 2, in Section 3 we present *modules of signal nets* with definition of step semantics, composition rules and input/output behavior. In Section 4 we outline our control framework implementing the "forcing and prohibiting"-paradigm by means of modules of signal nets. It is compared in detail to classical supervisory control. The main result is the synthesis of the maximally permissive nonblocking behavior of the module of a signal net (representing the plant) respecting a given regular specification language. The Section splits into two parts. In Subsection 4.1 the special case of a prefix closed specification language is considered. In Subsection 4.2 the general situation is addressed.

2 Mathematical Preliminaries

We need the following language theoretic notations ([8]). For a finite set A we denote $2^A = \{B \mid B \subseteq A\}$ the set of all subsets of A and $A^* = \{a_1 \dots a_n \mid n \in \mathbb{N}_0, a_1, \dots, a_n \in A\}$ the set of all finite words over the alphabet A. Let $L \subseteq A^*$ be a language over a finite alphabet A. The empty word is denoted by ϵ, $\overline{L} = \{v \in A^* \mid \exists x \in A^* : vx \in L\}$ is the prefix closure of L, and $post(L) = \{v \in A^* \mid \exists w \in L, \exists x \in A^* : v = wx\}$ is the language of all possible extensions of words of L. Observe that for a regular language L also $post(L)$ is regular. We will use another operation on languages preserving regularity: For two languages $L_1, L_2 \subseteq A^*$: $L_1/L_2 = \{w \in A^* \mid \exists v \in L_2 : wv \in L_1\}$ is the *quotient of* L_1 *and* L_2. We will consider languages over alphabets $A = 2^X$ for finite sets X. We need an extension of the projection operator to such languages for subsets $Y \subseteq X$. Define the *hiding operator* λ_Y *w.r.t.* Y by:
For a character $\xi \in A$: $\lambda_Y(\xi) = \xi \setminus Y$ if $\xi \setminus Y \neq \emptyset$, and $\lambda_Y(\xi) = \epsilon$ otherwise.
For a word $w \in A^*$: $\lambda_Y(w) = \lambda_Y(\xi_1) \dots \lambda_Y(\xi_n)$ if $w = \xi_1 \dots \xi_n$, and $\lambda_Y(w) = \epsilon$ if $w = \epsilon$. For a language $L \subseteq A^*$: $\lambda_Y(L) = \{\lambda_Y(w) \mid w \in L\}$.
The hiding operator defines equivalence classes over A^* in the following way. For a $w \in A^*$ denote $[w]_Y = \{v \in A^* \mid \lambda_Y(w) = \lambda_Y(v)\}$. Regular languages are represented by regular expressions or finite automata. Remember that states of a deterministic finite automata DFA can be denoted as equivalence classes over A^*:
$[w]_{DFA} = \{v \in A^* \mid \text{Execution of } v \text{ and } w \text{ lead to the same state}\}$.

3 Modules of Signal Nets

We use an extension of elementary Petri nets (1-safe Petri nets) equipped with the so called *first consume, then produce* semantics (since we want to allow loops, e.g. [10]). The first step in the extension is to add two kinds of signals, namely active signals, which force the occurrence of (enabled) events (typically switches or actuators), and passive signals which enable/prohibit the occurrence of events (typically sensors).

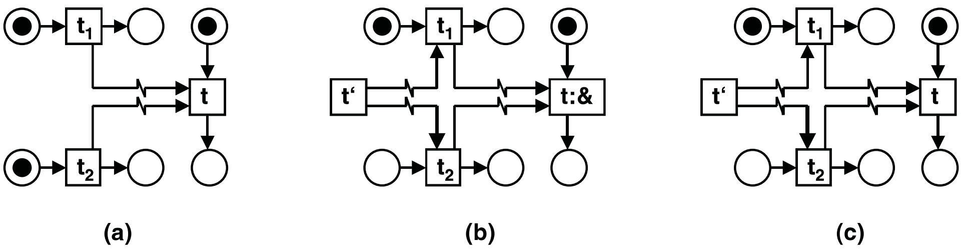

Fig. 1. In Figure (a) the enabled steps are $\{t_1, t\}$ and $\{t_2, t\}$. Figure (b) shows a signal net in AND-semantics: here the only enabled step is $\{t', t_1\}$, i.e. t is not synchronized. In Figure (c) the same net is shown in OR-semantics: here we have the enabled step $\{t', t_1, t\}$, i.e. t is synchronized.

These signals are expressed using two kind of arcs. A Petri net extended with such signals is simply called a *signal net*. Active signals are represented using arcs connecting transitions and can be interpreted in the following way: an active signal arc, also called *event arc*, leading from a transition t_1 to a transition t_2 specifies that if transition t_1 occurs and transition t_2 is enabled to occur then the occurrence of t_2 is forced (synchronized) by the occurrence of t_1, i.e. transitions t_1 and t_2 occur in one (synchronized) step. If t_2 is not enabled, t_1 occurs without t_2, while an occurrence of t_2 without t_1 is not allowed. Event arcs are not allowed to build cycles. In general (synchronized) steps of transitions are build inductively in the above way. Every step starts at a unique transition, which is not synchronized itself. Consider a transition t which is synchronized by several transitions $t_1, \ldots, t_n$, $n \geqslant 2$. Then two situations can be distinguished. For simplicity consider the case $n = 2$. If the transitions t_1, t_2 do not build a synchronized step themselves, either t_1 or t_2 can synchronize transition t in the above sense, but never transitions t_1, t_2 can occur in one synchronized step. As an example you can think of several switches to turn a light on (see Figure 1, part (a)). If the transitions t_1, t_2 build a synchronized step themselves, then there are two dialects in literature to interpret such a situation: In the first one ([5, 4, 6]) both transitions t_1, t_2 have to agree to synchronize t. Thus the only possible step of transitions involving t has to include transitions t_1, t_2, too. We call this dialect AND-semantics (see Figure 1, part (b)). In the second one ([3]) the occurrence of at least one of the transitions t_1 and t_2 synchronizes transition t, if t is enabled. It is also possible, that t_1, t_2 and t occur in one synchronized step. We call this dialect OR-semantics (see Figure 1, part (c)). In general the relation given by event arcs builds a forest of arbitrary depth. In this paper we introduce the most general interpretation, where both semantics are possible and are interpreted locally backward. That means we distinguish between OR- and AND-synchronized transitions. An OR-synchronized transition demands to be synchronized by at least one of its synchronizing transitions, whereas an AND-synchronized transition demands to be synchronized by all of its synchronizing transitions.

Since we allow loops w.r.t. single transitions, we also allow loops w.r.t. steps of transitions (see Figure 2, part (a)).

Passive signals are expressed by so called *condition arcs* (also called read arcs or test arcs in the literature) connecting places and transitions. A condition arc leading from a place to a transition models the situation that the transition can only occur if

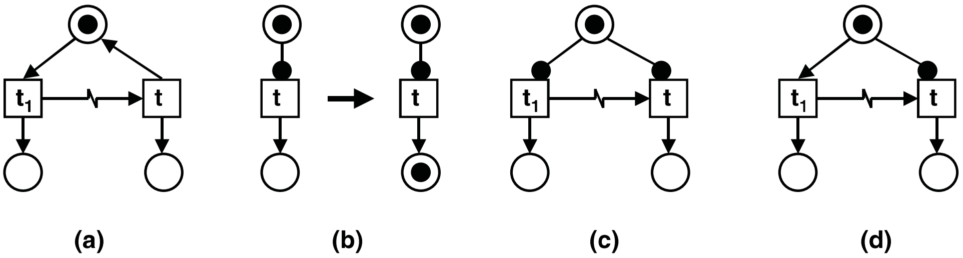

Fig. 2. Figure (a) shows an enabled step $\{t_1, t\}$. The left part of Figure (b) shows an enabled transition t, which tests a place to be marked. The occurrence of t leads to the marking shown in the right part of Figure (b). Figures (c) and (d) again present situations of an enabled step $\{t_1, t\}$.

the place is in a certain state but this state remains unchanged by the transition's occurrence (read operation) (see Figure 2, part (b)). Of course several transitions belonging to a synchronized step can test a place to be in a certain state via passive signals simultaneously, since the state of this place is not changed by their occurrence (see Figure 2, part (c)). We also allow that a transition belongs to a synchronized step of transitions testing a place to be in a certain state via a passive signal, whereas the state of this place is changed by the occurrence of another transition in this step. That means we use the so called *a priori* semantics ([9]) for the occurrence of steps of transitions, where testing of states precedes changing of states by occurrence of steps of transitions (see Figure 2, part (d)).

Definition 1 (Signal nets). *A signal net is a six-tuple $N = (P, T, F, CN, EN, m_0)$ where P denotes the finite set of* places*, $T = T_{AND} \dot{\cup} T_{OR}$ the distinct union of the finite sets of AND-*synchronized transitions *T_{AND} and OR-*synchronized transitions *T_{OR} ($P \cap T = \emptyset$), $F \subseteq (P \times T) \cap (T \times P)$ the* flow relation*, $CN \subseteq (P \times T)$ the set of* condition arcs *($CN \cap (F \cup F^{-1}) = \emptyset$), $EN \subseteq (T \times T)$ the acyclic set of* event arcs *($EN^+ \cap id_T = \emptyset$), and $m_0 \subseteq P$ the* initial marking.

Places, transitions and the flow relation are drawn as usual using circles, boxes and arrows. To distinguish between AND- and OR-synchronized transitions, AND-synchronized transitions are additionally labelled by the symbol "&". Event arcs and condition arcs are visualized using arcs of a special form given in Figure 1 and Figure 2.

For a place or a transition x we denote ${}^\bullet x = \{y \mid (y, x) \in F\}$ the *preset* of x and $x^\bullet = \{y \mid (x, y) \in F\}$ the *postset* of x. For a transition t we denote ${}^+t = \{p \mid (p, t) \in CN\}$ the *positive context* of t, ${}^{\rightsquigarrow}t = \{t' \mid (t', t) \in EN\}$ the *synchronization set* of t, $t^{\rightsquigarrow} = \{t' \mid (t, t') \in EN\}$ the *synchronized set* of t. Given a set $\xi \subseteq T$ of transitions, we extend the above notations to: ${}^\bullet\xi = \bigcup_{t \in \xi} {}^\bullet t$ and $\xi^\bullet = \bigcup_{t \in \xi} t^\bullet$, ${}^{\rightsquigarrow}\xi = \bigcup_{t \in \xi} {}^{\rightsquigarrow}t$, $\xi^{\rightsquigarrow} = \bigcup_{t \in \xi} t^{\rightsquigarrow}$.

Definition 2 (Enabling of transitions). *A transition $t \in T$ is* enabled *at a marking $m \subseteq P$, if ${}^\bullet t \cup {}^+t \subseteq m$ and $(t^\bullet \setminus {}^\bullet t) \cap m = \emptyset$.*

The following definition introduces a notion of steps of transitions which is different to the usual one used in Petri nets. A step denotes a set of transitions connected by event arcs, which occur synchronously. A transition, which is not synchronized by another transition, is a step. Such transitions are called *spontanuous*. In general, steps are sets of transitions such that for every non-spontaneous OR-synchronized transition in this step at least one of it's synchronizing transitions belongs also to this step, and for every AND-synchronized transition in this step all of it's synchronizing transitions belong also to this step.

Definition 3 (Steps). *Given a signal net N, steps are sets of transitions ξ defined inductively by*
– If $t \in T$ with ${}^{\rightsquigarrow}t = \emptyset$ (t is spontaneous), then $\xi = \{t\}$ is a step.
– If ξ is a step, and $t \in T \setminus \xi$ is a transition, then $\xi \cup \{t\}$ is a step, if either $t \in T_{OR}$ and ${}^{\rightsquigarrow}t \cap \xi \neq \emptyset$, or $t \in T_{AND}$ and $\emptyset \neq {}^{\rightsquigarrow}t \subseteq \xi$.
The set of all steps of N is denoted by Σ_N.

Now we introduce how a step is enabled to occur. A step ξ is said to be potentially enabled at a marking m if every transition $t \in \xi$ is enabled at m and no transitions $t_1, t_2 \in \xi$ are in conflict, except for possible loops $p \in {}^\bullet\xi \cap \xi^\bullet$ w.r.t. ξ, where $p \in m$ is required. From all steps potentially enabled at a marking only those are enabled which are maximal with this property.

Definition 4 (Potential enabling/enabling of steps). *A step ξ is* potentially enabled *in a marking m if*
– $\forall t \in \xi$: ${}^\bullet t \cup {}^+t \subseteq m$ and $(t^\bullet \setminus {}^\bullet\xi) \cap m = \emptyset$ and
– $\forall t, t' \in \xi$, $t \neq t'$: ${}^\bullet t \cap {}^\bullet t' = t^\bullet \cap (t')^\bullet = \emptyset$ (t, t' are not in conflict).
The step ξ is enabled, *if ξ is potentially enabled, and there is not a potentially enabled step $\eta \supsetneq \xi$ (ξ is maximal).*

Definition 5 (Occurrence of steps and follower markings). *The occurrence of an enabled step ξ yields the follower marking $m' = (m \setminus {}^\bullet\xi) \cup \xi^\bullet$. In this case we write $m[\xi\rangle m'$.*

Definition 6 (Reachable markings, occurrence sequences). *A marking m is called reachable from the initial marking m_0 if there is a sequence of markings $m_1, \ldots, m_k = m$ and a sequence of steps $\xi_1, \ldots, \xi_k$, such that $m_0[\xi_1\rangle m_1, \ldots, m_{k-1}[\xi_k\rangle m_k$. Such a sequence of steps is called an occurrence sequence. The set of all reachable markings is denoted by $[m_0\rangle$.*

Adding some inputs and outputs to signal nets, i.e. adding condition and event arcs coming from or going to an environment, we get modules of signal nets with input and output structure.

Definition 7 (Modules of signal nets). *A module of a signal net is a triple $M = (N, \Psi, c_0)$, where $N = (P, T, F, CN, EN, m_0)$ is a signal net, and $\Psi = (\Psi^{sig}, \Psi^{arc})$ is the input/output structure, where $\Psi^{sig} = C^{in} \cup E^{in} \cup C^{out} \cup E^{out}$ is a set of input/output signals, and $\Psi^{arc} = CI^{arc} \cup EI^{arc} \cup CO^{arc} \cup EO^{arc}$ is a set of arcs connecting input/output signals with the elements of the net N. Namely, C^{in} denotes a finite set of condition inputs, E^{in} a finite set of event inputs, C^{out} a finite set of condition outputs, E^{out} a finite set of event outputs (all these sets are pair-wise disjoint), $CI^{arc} \subseteq C^{in} \times T$ a set of condition input arcs, $EI^{arc} \subseteq E^{in} \times T$ a set of event input arcs, $CO^{arc} \subseteq P \times C^{out}$ a set condition output arcs and $EO^{arc} \subseteq T \times E^{out}$ a set of event output arcs. $c_0 \subseteq C^{in}$ is the initial state of the condition inputs.*

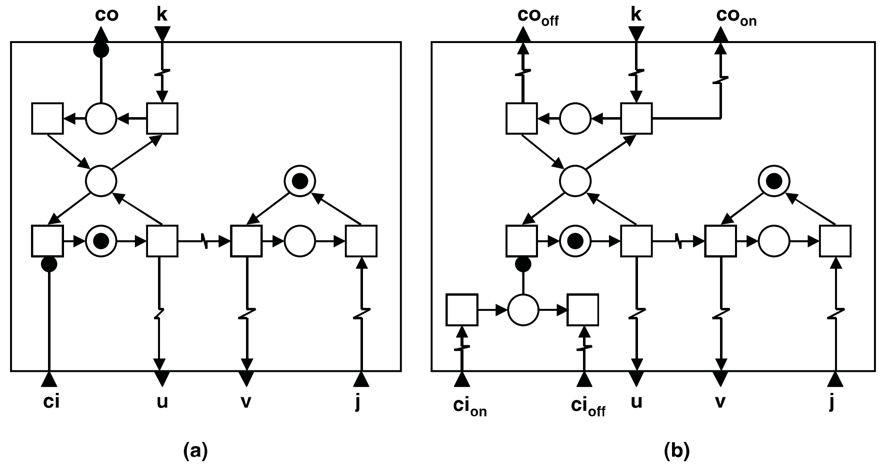

Fig. 3. Figure (a) shows a module of a signal net. Figure (b) shows the same module, where condition inputs and outputs are replaced by equivalent structures involving event inputs and outputs.

We extend the notions of preset, postset, positive context, synchronization set and synchronized set to the elements of Ψ^{sig} in the obvious way. An example of a module of a signal net, with $C^{in} = \{ci\}$, $E^{in} = \{j, k\}$, $C^{out} = \{co\}$ and $E^{out} = \{u, v\}$ is shown in the Figure 3, part (a).

Two modules can be composed by identifying some inputs of the one module M_1 with appropriate outputs of the other module M_2 and replacing the connections of the nets to the involved identified inputs and outputs by direct signal arcs respecting the identification (see Figure 4).

Definition 8 (Composition of modules of signal nets). *Let* $M_1 = (N_1, \Psi_1, c_{01})$, $M_2 = (N_2, \Psi_2, c_{02})$ *be modules of signal nets with input/output structures* $\Psi_i = (\Psi_i^{sig}, \Psi_i^{arc})$, $i = 1, 2$ *and initial markings* m_{01}, m_{02}. *Let further* $Q \subseteq \Psi_1^{sig}$ *and* $\Omega : Q \to \Psi_2^{sig}$ *be an injective mapping, such that the initial markings are compatible with the initial states of the condition inputs in the sense:* $(p, co) \in CO_1^{arc} \wedge \Omega(co) \in c_{02} \Rightarrow p \in m_{01}$ *and* $(p, co) \in CO_2^{arc} \wedge \Omega^{-1}(co) \in c_{01} \Rightarrow p \in m_{02}$. *Moreover* Ω *has to satisfy* $\Omega(E_1^{in} \cap Q) \subseteq E_2^{out}$, $\Omega(E_1^{out} \cap Q) \subseteq E_2^{in}$, $\Omega(C_1^{in} \cap Q) \subseteq C_2^{out}$, *and* $\Omega(C_1^{out} \cap Q) \subseteq C_2^{in}$, *such that no cycles of event arcs are generated.*

Then the composition $M = M_1 *_\Omega M_2$ of M_1 and M_2 w.r.t. Ω *is the module* $M = (N, \Psi, c_0)$ *with* $N = (P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2, CN, EN, m_{01} \cup m_{02})$ *and* $\Psi = (\Psi^{sig}, \Psi^{arc})$, *where involved inputs, outputs and corresponding signal arcs are deleted, i.e.*

$$\begin{aligned}
\Psi^{sig} &= (\Psi_1^{sig} \setminus Q) \cup (\Psi_2^{sig} \setminus \Omega(Q)), \\
\Psi^{arc} &= (\Psi_1^{arc} \setminus (({}^\bullet Q \times Q) \cup (Q \times Q^\bullet))) \cup \\
&\quad\ (\Psi_2^{sig} \setminus (({}^\bullet \Omega(Q) \times \Omega(Q)) \cup (\Omega(Q) \times \Omega(Q)^\bullet))), \\
c_0 &= (c_{01} \setminus Q) \cup (c_{02} \setminus \Omega(Q)),
\end{aligned}$$

and new signal arcs are added according to Ω *in the following way:*

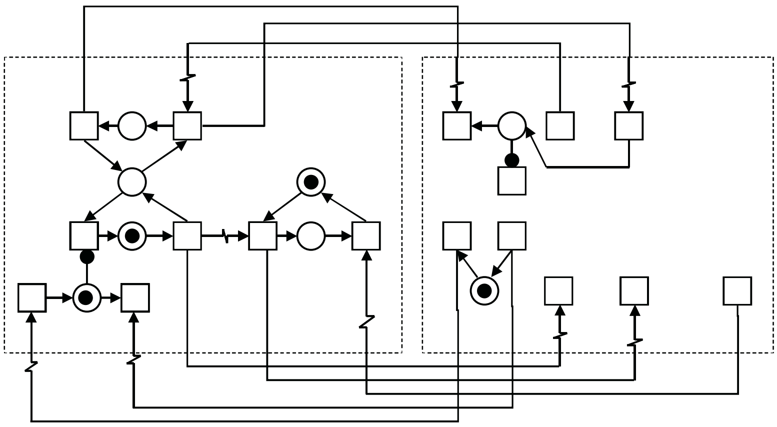

Fig. 4. The standalone of the module of a signal net in Figure 3 (a). The composed modified modules are indicated by dashed boxes. The input/output behavior of the module is given by the set of all occurrence sequences of this standalone, where the transitions in the left box are hidden. The control module with respect to a given specification can be synthesized by adding appropriate net structure to the maximally environment module represented by the right box.

$$
\begin{aligned}
CN = {} & CN_1 \cup CN_2 \cup \\
& \{(p,t) \mid \exists co \in C_1^{out} : (p,co) \in CO_1^{arc} \wedge (\mathit{\Omega}(co),t) \in CI_2^{arc}\} \cup \\
& \{(p,t) \mid \exists ci \in C_1^{in} : (ci,t) \in CI_1^{arc} \wedge (p,\mathit{\Omega}(ci)) \in CO_2^{arc}\}, \\
EN = {} & EN_1 \cup EN_2 \cup \\
& \{(t,t') \mid \exists eo \in E_1^{out} : (t,eo) \in EO_1^{arc} \wedge (\mathit{\Omega}(eo),t') \in EI_2^{arc}\} \cup \\
& \{(t,t') \mid \exists ei \in E_1^{in} : (ei,t') \in EI_1^{arc} \wedge (t,\mathit{\Omega}(ei)) \in CO_2^{arc}\}
\end{aligned}
$$

In the following, we define the input/output behavior of Petri modules as the set of all possible sequences of input signals sent to the module and output signals sent from the module. Since condition signals are signals with duration, we replace them for this purpose by equivalent structures of event signals in the following way (see Figure 3):

Definition 9. *Let* $M = (N, \mathit{\Psi}, c_0)$ *be a module with* $\mathit{\Psi} = (\mathit{\Psi}^{sig}, \mathit{\Psi}^{arc})$*. Define* $M_m = (N_m, \mathit{\Psi}_m, c_{0m})$*,* $\mathit{\Psi}_m = (\mathit{\Psi}_m^{sig}, \mathit{\Psi}_m^{arc})$ *by deleting condition inputs and outputs via*

$$C_m^{in} = C_m^{out} = CI_m^{arc} = CO_m^{arc} = \emptyset,$$

and adding new structures involving event inputs and outputs in the following way:

$$
\begin{aligned}
\text{For } c \in C^{out} : \quad & E_m^{out} = \{c.on,\ c.\mathit{off} \mid c \in C^{out}\} \\
& EO_m^{arc} = \{(t, c.on),\ (t', c.\mathit{off}) \mid c \in C^{out},\ t \in {}^{\bullet}({}^{+}c),\ t' \in ({}^{+}c)^{\bullet}\} \\
\text{For } c \in C^{in} : \quad & P_m = P \cup \{p_{c.on} \mid c \in C^{in}\} \\
& T_m = T \cup \{t_{c.on}^{net},\ t_{c.\mathit{off}}^{net} \mid c \in C^{in}\} \\
& F_m = F \cup \{(t_{c.on}^{net}, p_{c.on}),\ (p_{c.on}, t_{c.\mathit{off}}^{net}) \mid c \in C^{in}\} \\
& E_m^{in} = \{c.on,\ c.\mathit{off} \mid c \in C^{in}\} \\
& EI_m^{arc} = \{(c.on, t_{c.on}^{net}),\ (c.\mathit{off}, t_{c.\mathit{off}}^{net}) \mid c \in C^{in}\}
\end{aligned}
$$

Finally every $p_{c.on}$ *is marked if* $c \in c_0$*.* M_m *is called modified module (of* M*).*

For modified modules, composing a condition output c_1^{out} of one module with a condition input c_2^{in} of another module translates to composing $c_1^{out}.on$ with $c_2^{in}.on$ and $c_1^{out}.\mathit{off}$ with $c_2^{in}.\mathit{off}$ (Observe that the initial marking of $p_{c_2^{in}.on}$ is then chosen according to the initial marking of the place in ${}^{+}c_1^{out}$). Then every occurrence sequence of the composition of the original modules corresponds to the same occurrence sequence of the composition of the modified modules in which every step of transitions involving a transition in ${}^{\bullet}({}^{+}c_1^{out})$ resp. $({}^{+}c_1^{out})^{\bullet}$ additionally includes the transition $t^{net}_{c_2^{in}.on}$ resp. $t^{net}_{c_2^{in}.\mathit{off}}$. In order to define formally the input/output behavior of a module M as the set of all possible sequences of input and output event signals we first compose M with another module E representing the maximally permissive environment of M. Then we can represent sequences of input and output event signals of M by occurrence sequences of the composed module restricted to the transition set of E (see Figure 4). As the mentioned maximally permissive environment we recognize a module E, which

- at any moment can send event inputs to M: so each event signal of M is modelled in E by a corresponding always enabled transition;
- at any moment can enable and disable condition inputs of M: so each condition input of M is modelled in E by a corresponding place, which can be marked and unmarked by associated transitions;
- can observe outputs of M: every output of M is modelled in E by a corresponding transition, which is synchronized in the case of an event output, and enabled in the case of an condition output;
- does not allow synchronization between its transitions: in particular, inputs should not be sent in steps from E to M, and outputs M should only be observed by E and not synchronize inputs of M via E.

Definition 10 (Maximally permissive environment). *Let* $M = (N, \Psi, c_0)$ *be a module with* $\Psi = (\Psi^{sig}, \Psi^{arc})$. *Define the maximally permissive environment module* $E_M = (N_E, \Psi_E, c_{0E})$, $\Psi_E = (\Psi_E^{sig}, \Psi_E^{arc})$, *w.r.t.* M *by* $EN_E = CN_E = \emptyset$ *and*

$$\begin{aligned}
\mathit{For}\ c \in C^{out} :\ & T_E = \{t_c \mid c \in C^{out}\},\\
& C_E^{in} = \{ci_c \mid c \in C^{out}\}, CI_E^{arc} = \{(ci_c, t_c) \mid c \in C^{out}\},\\
\mathit{For}\ c \in C^{in} :\ & T_E = T_E \cup \{t_{ci.on},\ t_{ci.\mathit{off}} \mid ci \in C^{in}\},\ P_E = \{p_{ci.on} \mid ci \in C^{in}\}\\
& F_E = \{(t_{ci.on}, p_{ci.on}),\ (p_{ci.on}, t_{ci.\mathit{off}}) \mid ci \in C^{in}\}\\
& m_{0E} = \{p_{ci.on} \mid ci \in C^{in} \cup c_0\}\\
& C_E^{out} = \{co_c \mid c \in C^{in}\},\ CO_E^{arc} = \{(p_c, co_c) \mid c \in C^{in}\},\\
\mathit{For}\ e \in E^{out} :\ & T_E = T_E \cup \{t_e \mid e \in E^{out}\},\\
& E_E^{in} = \{ei_e \mid e \in E^{out}\}, EI_E^{arc} = \{(ei_e, t_e) \mid e \in E^{out}\}\\
\mathit{For}\ e \in E^{in} :\ & T_E = T_E \cup \{t_e \mid e \in E^{in}\},\\
& E_E^{out} = \{eo_e \mid e \in E^{in}\}, EO_E^{arc} = \{(t_e, eo_e) \mid e \in E^{in}\}.
\end{aligned}$$

We call the composition of the modified module of M with the modified module of its maximally permissive environment the *standalone of M* (observe that this composition has empty input/output structure) (as an example see Figure 4). The restriction of the occurrence sequences of the standalone to the transition set of the environment is then formalized by using the hiding operator defined in section 2, which is used to make the inner transitions of a module invisible.

Definition 11. (Standalones) *Let M be a module of a signal net, E be the maximally permissive environment module of M and $M_m = (N, \Psi)$, $E_m = (N_E, \Psi_E)$ be their modified modules. The* standalone *of M is the composition module $M_S = (N_S, \Psi_S) = N_m *_\Omega E_m$ w.r.t. the following composition mapping $\Omega : \Psi^{sig} \to \Psi_E^{sig}$:*

$$\Omega(e) = ei_e \textit{ for } e \in E_E^{out}$$
$$\Omega(e) = eo_e \textit{ for } e \in E_E^{in}$$

The set $I = \{t \in T_S \cap T_E \mid t^{\rightsquigarrow} \neq \emptyset\}$ is called the set of input transitions of M_S, the set $O = \{t \in T_S \cap T_E \mid {}^{\rightsquigarrow}t \neq \emptyset\}$ is called the set of output transitions of M_S.

Definition 12. (Input/output behavior of modules of signal nets) *Let M be a module of a signal net with the set of transitions T. Let L_M be the set of all finite occurrence sequences of the standalone M_S of M. Then the language $\lambda_T(L_M)$ is called input/output behavior of the module M.*

L_M represents the set of all possible sequences of steps of input and output signals under the assumptions: Output signals of M can not synchronize input signals of M via the maximally permissive environment module. Input signals of M are not sent in steps from the maximally permissive environment module.

4 Controller Synthesis

Throughout this section we consider a module of a signal net $\mathcal{P}$ as a model of an uncontrolled plant. As in the previous section T denotes the set of transitions of $\mathcal{P}$, and I resp. O denote the sets of input resp. output transitions of the standalone of $\mathcal{P}$.

In our modelling formalism we consider the inside of $\mathcal{P}$ as a black box: We only can send input signals to $\mathcal{P}$ and meanwhile observe sequences of output signals. In particular, the behavior of the DES (represented by $\mathcal{P}$) is forced, not only restricted from outside. Of course this approach leads to formal and technical differences to the classical supervisory control approach:

Mainly, all events of $\mathcal{P}$ are assumed to be uncontrollable and unobservable. Controllable are only the input signals, modelled by the set of transitions I of the maximally permissive environment of $\mathcal{P}$, and observable are, beside the input signals, exactly the output signals, modelled by the set of transitions O of the maximally permissive environment of $\mathcal{P}$.

We specify a desired behavior of $\mathcal{P}$ by a set of desired sequences only of output signals. Observe moreover that, since the event arc relation produces a step semantics, we observe sequences of steps of output signals. Therefore we suppose a specification to be given as a language $L_c \subseteq \left(2^O\right)^*$. Since sets of occurrence sequences of signal nets are regular languages[1] over an alphabet of steps, we assume L_c also to be regular.

[1] Observe that we use elementary nets, which have a finite reachability graph.

The question is, whether it is possible to force the behavior of $\mathcal{P}$ via input signals to respect the given specification of output signals in a *maximally permissive way*. This can be formalized by asking, whether there is a module of a signal net $\mathcal{C}$ modelling the control and a composition $\mathcal{P} *_{\Omega} \mathcal{C}$, such that the set of occurrence sequences of $\mathcal{P} *_{\Omega} \mathcal{C}$ respects L_c.

The answer of the question above splits into two parts. One part is to decide whether a control exists for the given specification and in the positive case to compute the behavior of the desired control module resp. the resulting composed module, also called the *behavior of the controlled plant*. The other part is, to implement the given controlled behavior via synthesizing the control module $\mathcal{C}$ from the behavior of the controlled plant and composing this module with the plant module $\mathcal{P}$, such that the resulting composition has exactly this controlled behavior. The scope of this paper is the first part. We will finally give the behavior of the controlled plant by a deterministic finite automaton, which is intended to represent the marking graph of the composed module, where only input and output transitions are visible.

The second part is presented in [11]. The main idea is to synthesize $\mathcal{C}$ from the automaton representing the behavior of the controlled plant, by adding new net structure to the modified maximally permissive environment module $\mathcal{E}$ of $\mathcal{P}$ (see Figure 4). That means the control module $\mathcal{C}$ is constructed from $\mathcal{E}$, and composed with $\mathcal{P}$ via the given connections between $\mathcal{P}$ and $\mathcal{E}$.

For a detailed running example of the computation of the behavior of the controlled plant and the implementation of this behavior via synthesizing a control module we recommend the paper [12].

We will formulate our approach language theoretically similarly as it is done in classical supervisory control. We will see, that despite the mentioned differences, some algorithms of classical supervisory control can at least be adapted to our framework. While omitting therefore most details of these algorithms, out paper remains self contained, i.e. can be understood without previous knowledge of supervisory control.

Without knowing any control module $\mathcal{C}$ we search for a sublanguage K (of occurrence sequences) of the language $L_{\mathcal{P}}$ (of all occurrence sequences of the standalone of $\mathcal{P}$) which represents the behavior of a composition of the plant module $\mathcal{P}$ and a control module $\mathcal{C}$ (i.e. the behavior of the controlled plant):

- If an occurrence sequence in K can be extended by a step of output transitions or invisible transitions to an occurrence sequence in $L_{\mathcal{P}}$, then also this extended occurrence sequence should be in K. This follows the paradigm: "*what cannot be prevented, should be legal*".
- According to unobservability of some events, some occurrence sequences in $L_{\mathcal{P}}$ cannot be distinguished by the control. As a consequence, following the paradigm "*what cannot be distinguished, cannot call for different control actions*", if an input is sent to the plant after a sequence w of steps has occurred, then the same input has to be sent after occurrence of any other sequence, which is undistinguishable to w.

Observe that the first condition corresponds to the classical one supervisory control. The second one is due to our step semantics, where an input can synchronize different invisible and output transitions depending on the state of $\mathcal{P}$, in combination with the notion of *observability* in supervisory control. Such a sublanguage K is called controllable w.r.t. $L_{\mathcal{P}}$, I and O:

Definition 13 (Controllable Language). *Given three finite, disjoint sets T, I and O and a prefix closed regular language L over the alphabet $2^{T \cup I \cup O}$, then a prefix closed sublanguage K of L is said to be* controllable w.r.t. L, I and O *(or simply controllable, if the sets are clear), if*

- $\forall w \in K, \forall o \in 2^{O \cup T} : \ wo \in L \Rightarrow wo \in K,$
- $\forall vj \in K,\ j \in 2^{I \cup O \cup T},\ j \cap I \neq \emptyset,\ \forall j' \in 2^{I \cup O \cup T}, \lambda_{O \cup T}(j) = \lambda_{O \cup T}(j'),\ \forall v' \in K,\ \lambda_T(v) = \lambda_T(v') : \ v'j' \in L \Rightarrow v'j' \in K.$

Of course we are searching for such a K, which additionally respects L_c and is maximal with this property.

Definition 14 (Maximally Permissive Controllable Language). *Let T, I and O be finite, disjoint sets, L be a prefix closed regular language over the alphabet $2^{T \cup I \cup O}$ and L_c be a regular language over the alphabet 2^O.*

Let $K \subseteq L$ be controllable w.r.t L, I and O satisfying

$$\lambda_{T \cup I}(K) \subseteq \overline{L_c}.$$

We say that K is maximally permissive controllable *w.r.t. L_c, L, I and O (or simply maximally permissive controllable, if the sets are clear), if there exists no language K' satisfying $K \subsetneq K' \subseteq L$, which is controllable w.r.t. L, I and O and fulfills $\lambda_{T \cup I}(K') \subseteq \overline{L_c}$.*

It is possible to get the result $K = \{\epsilon\}$ as maximally permissive controllable language, what means that the maximal behavior respecting the specification is empty, but there happens nothing wrong without inputs from outside. If even without any input the specification can be violated, we call L_c unsatisfiable w.r.t. L, I and O.

Definition 15. *L_c is said to be* unsatisfiable w.r.t. L, I and O *(or simply unsatisfiable, if the sets are clear), if*

$$\exists w \in (2^{O \cup T})^* : \ w \in L \wedge \lambda_T(w) \notin \overline{L_c}. \tag{1}$$

We call this condition unsatisfiability condition.

Consider a maximally permissive controllable language K: by definition every occurrence sequence in K is a prefix of an occurrence sequence respecting L_c. But it can happen there are such occurrence sequences that cannot be extended within K to an occurrence sequence respecting L_c, i.e. the desired behavior is blocked. We require additionally K to be nonblocking:

Definition 16 (Nonblocking Language). *Let T, I and O be finite , disjoint sets, L be a prefix closed regular language over the alphabet $2^{T \cup I \cup O}$ and L_c be a regular language over the alphabet 2^O.*

Let $K \subseteq L$ be maximally permissive controllable w.r.t L_c, L, I and O.

Let $M \subseteq K$ be controllable w.r.t. L, I and O satisfying

$$\forall r \in M : \exists x \in (2^{O \cup I \cup T})^* \textit{ with } rx \in M,\ \lambda_{I \cap T}(rx) \in L_c. \tag{2}$$

We say that M is nonblocking controllable *w.r.t. L_c, L, I and O (or simply nonblocking controllable, if the sets are clear). If it is maximal with this property, M is called* maximally permissive nonblocking controllable language.

In the next two paragraphs we synthesize the maximally permissive nonblocking controllable language M, if it exists. First we examine the case, when L_c is prefix closed. In this case the maximally permissive controllable sublanguage of $L_\mathcal{P}$ is already nonblocking. In particular safety properties can be formalized via a prefix closed specification L_c.

4.1 Safety Properties

Safety properties specify undesired behavior, that should not happen (for example forbidden states of the system). If some undesired behavior is realized by an occurrence sequence, the whole possible future of this occurrence sequence is undesired too.

The searched (control) language M as defined in the last paragraph is computed in several steps. First we define the (potentially safe) language L_{psafe} as the set of all occurrence sequences of $L_\mathcal{P}$ respecting L_c.

Definition 17. *We define $L_{psafe} = \{w \in L_\mathcal{P} \mid \lambda_{I \cup T}(w) \in L_c\}$ and $L_{unsafe} = L_\mathcal{P} \setminus L_{psafe}$.*

Observe that L_{psafe} is only a first approximation to M, since it is in general not controllable. In particular it may contain occurrence sequences which are not closed under extensions by outputs (first condition in definition 13). Such occurrence sequences must be cut at the last possible input (the last possibility of control), if there is one. The prefixes ending with these inputs are collected in the language L_{danger}. Deleting the futures of occurrence sequences in L_{danger} from L_{psafe} gives the language L_{safe}, which we will prove below to be the searched language M.

Definition 18. *We define*

$$\begin{aligned} L_{danger} = \{vj \in L_{psafe} \mid j \in 2^{T \cup I \cup O},\ j \cap I \neq \emptyset,\ &(\exists v' \in [v]_T) \\ &\wedge (\exists j' \in 2^{T \cup I \cup O} : \lambda_{T \cup O}(j) = \lambda_{T \cup O}(j')) \\ &\wedge (\exists y \in (2^{T \cup O})^* : v'j'y \in L_{unsafe})\}. \end{aligned}$$

$$L_{safe} = L_{psafe} \setminus post(L_{danger}).$$

It is obvious from the definitions of L_{psafe}, L_{unsafe}, L_{danger} and L_{safe} that every equivalence class $[w]_T \cap L_\mathcal{P}$ is either subset of or disjoint to these languages.

The main result of this subsection is the following theorem:

Theorem 1. *L_{safe} is maximally permissive nonblocking controllable, if L_c is not unsatisfiable.*

Before proving this theorem we give an algorithm to compute L_{safe}: It is essentially shown, that L_{safe} can be constructed by appropriate operations on regular languages. We want to remark here that for computing the maximally permissive controllable language also the more involved framework presented in [2] could be adapted (since our different notion of controllability is still compatible with the union operation $\cup$). There can also be found some hints to the complexity of the computation.

Let us first see that the language L_{psafe} is regular. We use the following general construction to pump regular languages by new characters:

Definition 19. *Let A, A' be two finite, disjoint sets and α be a regular expression over the alphabet 2^A. We construct a regular expression $\alpha_{ext(A')}$ over the alphabet $2^{A\cup A'}$ by replacing every $x \in 2^A$ in α by*

$$ext_{A'}(x) = \left(\sum_{y\in 2^{A'}} y\right)^* \left(\sum_{y\in 2^{A'}} x\cup y\right)\left(\sum_{y\in 2^{A'}} y\right)^* .$$

By this construction we want to generate a regular expression whose corresponding language $L(\alpha_{ext})$ contains exactly all those words, which belong to $L(\alpha)$ when all characters of the alphabet A' are hidden. Later on, this construction will be used to pump L_c by $I \cup T$. Then L_{psafe} can be computed as the intersection of such pumped L_c and $L_{\mathcal{P}}$. Pumping a regular language by A' in the above way computes the preimage $\lambda_{A'}^{-1}()$ of this language w.r.t. the appropriate hiding operator:

Lemma 1. *Let A, A' be finite disjoint sets and α be a regular expression over the alphabet 2^A. Each $w \in \left(2^{A\cup A'}\right)^*$ satisfies*

$$\lambda_{A'}(w) \in L(\alpha) \Leftrightarrow w \in L(\alpha_{ext(A')}).$$

Proof. We will show both directions of the above Lemma.
'$\Rightarrow$': we show by structural induction over the construction rules of regular expressions that the above property holds true:

Let $\alpha = x \in 2^A \cup \{\epsilon\}$ (these are the constants of a regular expression over 2^A), and $w \in (2^{A\cup A'})^*$ satisfy $\lambda_{A'}(w) = x$. Then w is of the form $w = e_1 \dots e_n$, $e_i \in 2^{A\cup A'}$, such that there exists an index i_0 satisfying $\lambda_{A'}(e_{i_0}) = x$ and $\lambda_{A'}(e_j) = \epsilon$ for $j \neq i_0$. It follows immediately from the construction above, that $w \in L(ext_{A'}(x)) = L(\alpha_{ext(A')})$.
Let α^1 and α^2 be two regular expressions over the alphabet 2^A satisfying the induction hypothesis, α^1_{ext} and α^2_{ext} be the corresponding extensions according to the above construction, and $w \in (2^{A\cup A'})^*$.

(i) Let $\alpha = \alpha^1 + \alpha^2$ and $\lambda_{A'}(w) \in L(\alpha)$. Then $\lambda_{A'}(w) \in L(\alpha^i)$, $i = 1$ or $i = 2$. By induction hypothesis $w = w_i$, $w_i \in L(\alpha^i_{ext})$, $i = 1$ or $i = 2$. So $w \in L(\alpha^1_{ext} + \alpha^2_{ext}) = L(\alpha_{ext})$, what follows immediately by the above construction.

(ii) Let $\alpha = \alpha^1\alpha^2$ and $\lambda_{A'}(w) \in L(\alpha)$. Then $w = w_1w_2$, such that $\lambda_{A'}(w_i) \in L(\alpha^i)$, $i = 1, 2$. By induction hypothesis $w = w_1w_2 \in L(\alpha^1_{ext})L(\alpha^2_{ext})$. So $w \in L(\alpha^1_{ext}\alpha^2_{ext}) = L(\alpha_{ext})$.

(iii) Let $\alpha = (\alpha^1)*$ and $\lambda_{A'}(w) \in L(\alpha)$. Then $w = w_1 \ldots w_n, \lambda_{A'}(w_i) \in L(\alpha^1), n \in \mathbb{N}$. By induction hypothesis $w = w_1 \ldots w_n \in L(\alpha^1_{ext})^*$. So $w \in L((\alpha^1_{ext})^*) = L(\alpha_{ext})$.

Observe that obviously $\alpha^1_{ext} + \alpha^2_{ext} = (\alpha^1 + \alpha^2)_{ext}$, $\alpha^1_{ext}\alpha^2_{ext} = (\alpha^1\alpha^2)_{ext}$ and $(\alpha^1_{ext})^* = ((\alpha^1)^*)_{ext}$.
'$\Leftarrow$': let be $w \in L(\alpha_{ext})$. It follows immediately by the construction of α_{ext} that $\lambda_{A'}(w) \in L(\alpha)$, because each constant $x \in 2^A$ of α is pumped up only by characters of $2^{A'}$. □

Corollary 1. *L_{psafe} is regular.*

Proof. Let α be a regular expression with $L(\alpha) = L_c$. Then according to Lemma 1

$$L_{psafe} = \{w \in L_{\mathcal{P}} \mid \lambda_{I\cup T}(w) \in L_c\} = L_{\mathcal{P}} \cap L(\alpha_{ext(I\cup T)}),$$

i.e. L_{psafe}, as the intersection of regular languages, is regular. □

Since $post(L)$ is obviously a regular language for regular languages L, it remains to show L_{danger} to be regular in order to show L_{safe} to be regular. Since with L_{psafe} also the language L_{unsafe} is regular, we can give L_{danger} as a simple formula on the regular languages $(2^{O\cup T})^*$ and L_{unsafe}. First observe that the regular language

$$L^{real}_{danger} = (L_{unsafe}/(2^{O\cup T})^*) \cap L_{\mathcal{P}},$$

where the symbol "/" denotes the quotient operation on languages, is the set of those words $vj \in L_{danger}$, which themselves can be extended by an $y \in (2^{O\cup T})^*$ to a word in L_{unsafe}. The remaining words in L_{danger} are of the form $v'j'$ with $v' \in [v]_T$ and $j' \cap I = j \cap I$ for a word $vj \in L^{real}_{danger}$. We get these words by means of a special defined hiding operator $\overline{\lambda} : (2^{O\cup I\cup T})^* \to (2^{O\cup I\cup T})^*$, defined by

$$\text{For } w = vx,\ v \in (2^{O\cup I\cup T})^*,\ x \in 2^{O\cup I\cup T} :\ \overline{\lambda}(w) = \lambda_T(v)\lambda_{T\cup O}(x),$$

and extended in the obvious way to languages. Obviously the operators $\overline{\lambda}$ and $\overline{\lambda}^{-1}$ preserve regularity of languages, since this is the case for the hiding operator λ as argued in lemma 1. We get

Lemma 2. $L_{danger} = \overline{\lambda}^{-1}[\overline{\lambda}(L^{real}_{unsafe})] \cap L_{psafe}$.

From now on we assume L_c to be satisfiable. The main theorem 1 now is shown in two steps by the following lemmata.

Lemma 3. *L_{safe} is controllable.*

Proof. First we have to show, that

$$\forall w \in L_{safe}, \forall o \in 2^{O\cup T} : wo \in L_{\mathcal{P}} \Rightarrow wo \in L_{safe}.$$

Assume $w \in L_{safe}$ and $o \in 2^{O\cup T}$ satisfying $wo \in L_{\mathcal{P}}$, but $wo \notin L_{safe}$. There are two cases:

- $wo \in L_{psafe}$: Then $wo \in post(L_{danger})$. This implies obviously $w \in post$ (L_{danger}), what contradicts $w \in L_{safe}$.
- $wo \notin L_{psafe}$: Then by definition $wo \in L_{unsafe}$. Since L_c is satisfiable, w has a prefix in L_{danger}. This again contradicts $w \in L_{safe}$.

It remains to show, that

$$\begin{aligned}&\forall vj \in L_{safe},\ j \in 2^{I \cup O \cup T},\ j \cap I \neq \emptyset,\ \forall j' \in 2^{I \cup O \cup T}, \lambda_{O \cup T}(j) = \lambda_{O \cup T}(j'),\\ &\forall v' \in L_{safe},\ v' \in [v]_T : \qquad v'j' \in L_{\mathcal{P}} \Rightarrow v'j' \in L_{safe}.\end{aligned}$$

For vj and $v'j'$ as above we have according to the definition of L_{danger}:
$vj \in post(L_{danger}) \Leftrightarrow v'j' \in post(L_{danger})$. □

Lemma 4. *There is no language $K \subseteq L_{\mathcal{P}}$ satisfying $L_{safe} \subsetneq K$, which is controllable, and which fulfills $\lambda_{I \cup T}(K) \subseteq \overline{L_c}$.*

Proof. We choose a $w \in K \setminus L_{safe}$ and construct from w a $w' \in K$ satisfying $\lambda_{I \cup T}(w') \notin \overline{L_c}$. As $w \in L_{\mathcal{P}}$, there are two cases:

- $w \notin L_{psafe}$: Then $w \in L_{unsafe}$ and thus $\lambda_{I \cup T}(w) \notin L_c$.
- $w \in L_{psafe}$: Then $w \in post(L_{danger})$, i.e. w has a prefix $vj \in L_{danger}$. That means, there are $v' \in [v]_T$, $j' \in 2^{I \cup O \cup T}$ with $j \cap I = j' \cap I$ and $y \in 2^{O \cup T}$, such that $v'j'y \in L_{unsafe}$, i.e. $\lambda_{I \cup T}(v'j'y) \notin L_c$. Since K is controllable, $v'j'$ also belongs to K (second property), and consequently $v'j'y \in K$ (first property). □

It follows immediately from the above proof, that L_{safe} is the unique maximally permissive (nonblocking) language, analogously to related results in supervisory control.

4.2 Nonblocking Control

More general properties as for example the full execution of certain tasks cannot be formalized by a regular language L_c which is prefix closed. Of course a maximally permissive controllable language K w.r.t. a not prefix closed L_c, $L_{\mathcal{P}}$, I and O should contain occurrence sequences of the standalone of $L_{\mathcal{P}}$ which represent prefixes of words in L_c, but only such ones, which can be extended to a word in L_c within K, i.e. which are nonblocking. In other words the set of blocking words of K

$$K_{blocking} = \{r \in K \mid \nexists x \in (2^{O \cup I \cup T})^* : \ rx \in K \wedge \lambda_{I \cup T}(rx) \in L_c\}.$$

should be empty.

For this purpose replace L_c by $\overline{L_c}$ in the definitions of L_{psafe} and L_{unsafe} (definition 17). We now search for a sublanguage L_{nbsafe} of L_{safe}, which is controllable, nonblocking and maximal with these two properties according to definition 16. Since in our framework the special property, namely that every controllable event is also observable, is fulfilled, we are able to adapt a result in supervisory control ([2], subsection 3.7.5), which states under the assumption of this property: If there is at least one controllable language respecting L_c which is nonblocking, then there is a unique maximal one.

We compute L_{nbsafe}, if it exists, in two steps:
First we represent L_{safe} by a finite automaton A which
– separates by its states words respecting L_c from words not respecting L_c:
$\forall w \in (2^{I \cup OUT})^*$, $\forall v \in [w]_A : \lambda_{I \cup T}(v) \in L_c \Leftrightarrow \lambda_{I \cup T}(w) \in L_c$, and
– separates by its states words, for which undistinguishable words with different futures exist.
Second we recursively delete input edges in A, which can lead to blocking words.

Such an automaton always exists. We omit the construction of A due to space limitations. For a detailed investigation see [11].

According to the first property of A we can define the set of states $S_c = \{[w]_A \mid \lambda_{I \cup T}(w) \in L_c\}$ representing L_c. So finding blocking words translates to finding blocking states, from which there is no continuation in A to a state in S_c. The following lemma directly follows:

Lemma 5. *Let $K = L_{safe}$ and let $r \in K$.*

- *$r \in K_{blocking}$, iff every $x \in (2^{I \cup OUT})^*$ with $rx \in L_{safe}$ fulfills $[rx]_A \notin S_c$.*
- *If $r \in K_{blocking}$ and $[r]_A = [r']_A$ for a word $r' \in K$, then $r' \in K_{blocking}$.*
- *If $r \in K_{blocking}$ and $r \in M \subseteq K$, then $r \in M_{blocking}$.*

That means we can construct L_{nbsafe} from L_{safe} by deleting edges in A representing inputs, which lead to blocking states in the sense of the above lemma in a maximally permissive way. Of course such edges must be deleted in all undistinguishable paths in A (see step 4 of the algorithm later on). The second property of A ensures that two different words ending with the same such edge in A, are both undistinguishable from words with a blocking future (i.e. by deleting such edges no futures of words, which have no blocking future, are cut). Since the deletion of an edge can produce new blocking states, the procedure is iterative. As for controllability (condition (1)), there is a condition saying when we cannot find a controllable nonblocking sublanguage: Every controllable sublanguage of L_{safe} contains all words in L_{safe} of the form $w \in (2^{OUT})^*$. Therefore:

Lemma 6. *Let $M \subseteq L_{safe}$. If there is a $w \in M_{blocking} \cap (2^{OUT})^*$, then there is no controllable sublanguage of M, which is nonblocking w.r.t. L_c.*

We call this condition *blocking condition*. It is only sufficient for nonexistence of a controllable nonblocking sublanguage, but not necessary. We are now prepared to state the algorithm:

Input: Automaton $A^0 = A$, Integer $k = 0$
Output: Automaton A_{nbsafe}, if L_{nbsafe} exists
Step 1: **If** $L(A^k)$ fulfills the blocking condition, **then return** "*L_{nbsafe} does not exist*"
Step 2: **If** $(L(A^k))_{blocking} = \emptyset$, **then return** A^k
Else Choose $w \in (L(A^k))_{blocking}$
Step 3: Compute a prefix vj of w with $j \cap I \neq \emptyset$, such that $w = vjy$ for $y \in (2^{OUT})^*$
Step 4: Compute the set of states $S^k_{delarc} = \{[u]_{A^k} \mid u \in [v]_T\}$
Step 5: Delete every edge starting in any state $[u]_{A^k} \in S^k_{delarc}$ with a label i fulfilling $i \cap I = j \cap I$

Step 6: Set $k = k + 1$
Set A^k to be the new constructed automaton
Goto Step 1

Let us state the main theorem of this subsection:

Theorem 2. *There exists a maximally permissive nonblocking controllable sublanguage of L_{safe}, if and only if the previous algorithm outputs an automaton A_{nbsafe}. In this case $L(A_{nbsafe})$ is this searched sublanguage.*

Proof. Let $A_{safe} = A^0, \dots, A^{N_0}$ be the sequence of automata the algorithm has computed until it has stopped.

We first show the "only if"-part:
Assume the previous algorithm outputs "L_{nbsafe} *does not exist*". We have to show, that there is no maximally permissive nonblocking controllable sublanguage of L_{safe}. For this it is enough to prove, that every controllable sublanguage K of L_{safe} must contain a blocking word w.r.t. K.

According to lemma 5 it is enough to find a word in K which is blocking w.r.t. L_{safe}. By **Step 1** $L(A^{N_0})$ fulfills the blocking condition, i.e.

$$\exists v_0 \in (2^{OUT}) : \; v_0 \in L^{N_0}_{blocking}.$$

Since K is assumed to be controllable, K contains all words in $L_{safe} \cap [v_0]_T$. If one of these words is blocking w.r.t. L_{safe}, we are done.

So assume all $u \in L_{safe} \cap [v_0]_T$ not to be blocking w.r.t. L_{safe}. That means every such u can be extended by an $y \neq \epsilon$ (remark that $\lambda_{T \cup I}(v_0) \notin L_c$!) such that $[uy]_{A^0} \in S_c$, in particular v_0. Observe that

- If none of all such possible extensions y of v_0 is in K, we are done (this would imply v_0 to be blocking w.r.t. K). So assume that there is at least one such extension $v_0 y_0 \in K$ with $[v_0 y_0]_{A^0} \in S_c$.
- We have $v_0 y_0 \in L(A^0) = L_{safe}$ but $v_0 y_0 \notin L(A^{N_0})$ (since v_0 blocking w.r.t. L^{N_0}).

Therefore there must be an index $N_1 < N_0$ and a prefix xi of y_0 (with $x \in (2^{I \cup OUT})^*$, $i \in 2^{I \cup OUT}$, $i \cap I \neq \emptyset$), such that $[v_0 x]_{A^{N_1}} \in S^{N_1}_{delarc}$ and the edge starting in state $[v_0 x]_{A^{N_1}}$ with label i was deleted from A^{N_1} in **Step 5**. This deletion was caused by the existence of a $v_1 \in L^{N_1}_{blocking}$ of the form $v_1 = w_1 j w_2$, where $j \in 2^{I \cup OUT}$, $w_1 \in (2^{I \cup OUT})^*$, and

$$(a)\; w_1 \in [v_0 x]_T \cap L_{safe}, \quad (b)\; j \cap I = i \cap I \quad \text{and} \quad (c)\; w_2 \in (2^{OUT})^*.$$

Remember now that all prefixes of $v_0 y_0$, in particular $v_0 x$ and $v_0 xi$, belong to K. Since K is assumed to be controllable, K contains all words in $L_{safe} \cap [v_0 x]_T$, in particular w_1 (property (a)). From the second condition of controllability and (b) we get further $w_1 j \in K$, and therefore $v_1 = w_1 j w_2 \in K$ (first condition of controllability and (c)).

By repeating this construction we get an strictly decreasing sequence of natural numbers $N_0 > N_1 > \ldots$ and associated words $v_0, v_1, \ldots \in K$, such that $v_i \in L^{N_i}_{blocking}$, $i = 0, 1, \ldots$. Finally $N_k = 0$ for some k, which implies $v_k \in K$ blocking w.r.t. K.

Next we consider the "if"-part:
By construction $L(A^{N_0}) = L(A_{nbsafe})$ is controllable and nonblocking. It remains to show that it is maximally permissive with these two properties. Assume another language K to be maximally permissive nonblocking controllable w.r.t. L_c, $L_{\mathcal{P}}$, I and O satisfying $L(A_{nbsafe}) \subsetneq K \subseteq L_{safe}$. We will construct inductively a blocking word in K.

There is a $x \in K \setminus L(A_{nbsafe})$. As $L(A_{nbsafe})$ and K are prefix closed we can assume (without loss of generality) $x = wj$, $w \in L(A_{nbsafe})$, $j \in 2^{I \cup OUT}$, $j \cap I \neq \emptyset$. Because K is controllable, we have

$$\forall y \in (2^{OUT})^*, \forall w' \in [w]_T, \forall j' \in 2^{I \cup OUT}, j' \cap I = j \cap I :\\ w'j'y \in L_{safe} \Rightarrow w'j'y \in K.$$

Since in some step $N_1 < N_0$ the edge starting from the state $[w]_{A^0}$ with label j was deleted, one of these words $w'j'y$, call it v_0, must be blocking w.r.t. $L(A^{N_1})$. Moreover, analogously to the "only if"-part, either one of these words is blocking w.r.t. L_{safe} (and we are done), or all of them must have an extension within K to a word respecting L_c, in particular v_0. Let v_0y_0 be this extension of v_0. Proceed now as in the "only if"-part.
□

5 Conclusion

In this paper we have presented a methodology for synthesis of the controlled behavior of discrete event systems employing actuators which try to force events and sensors which can prohibit event occurrences. As a modelling formalism, we have used modules of signal nets. The signal nets offer a direct way to model typical actuators behavior. Another advantage of such modules consists in supporting input/output structuring, modularity and compositionality in an intuitive graphical way.

In the paper we were not focusing on complexity issues. It is known that the complexity of the supervisory control problem is in general PSPACE-hard, and sometimes even undecidable ([16], pp. 15 - 36). To get efficient algorithms one has to restrict the setting in some way, for example by considering only special kinds of specifications.

References

1. B. Caillaud, P. Darondeau, L. Lavagno and X. Xie (Eds.). Synthesis and Control of Discrete Event Systems Kluwer Academic Press, 2002
2. C. G. Cassandras and S. Lafortune. Introduction to Discrete Event Systems. Kluwer, 1999.
3. J. Desel, G. Juhás and R. Lorenz. Input/Output Equivalence of Petri Modules. In *Proc. of IDPT 2002*, Pasadena, USA, 2002.

4. H-M. Hanisch and A. Lüder. A Signal Extension for Petri nets and its Use in Controller Design. *Fundamenta Informaticae*, 41(4) 2000, 415–431.
5. H.-M. Hanisch, A. Lüder, M. Rausch: Controller Synthesis for Net Condition/Event Systems with Incomplete State Observation, European Journal of Control, Nr. 3, 1997, S. 292-303.
6. H.-M. Hanisch, J. Thieme and A. Lüder. Towards a Synthesis Method for Distributed Safety controllers Based on Net Condition/Event Systems. *Journal of Intelligent Manufacturing*, 5 ,1997, 8, 357-368.
7. L.E. Holloway, B.H. Krogh and A. Giua. A Survey of Petri Net Methods for Controlled Discrete Event Systems. *Discrete Event Dynamic Systems: Theory and Applications*, 7, 1997), 151–190.
8. J.E. Hopcroft, R. Motwani and J.D.Ullman. Introduction to Automata Theory, Languages, and Computation. Addison Wesley, 2001.
9. R. Janicki and M. Koutny. Semantics of Inhibitor Nets. *Information and Computations*, 123, pp. 1–16, 1995.
10. G. Juhás. On semantics of Petri nets over partial algebra. In J. Pavelka, G. Tel and M. Bartosek (Eds.) *Proc. of 26th Seminar on Current Trends in Theory and Practice of Informatics SOFSEM'99*, Springer, LNCS 1725, pp. 408-415, 1999.
11. G. Juhás, R. Lorenz and C. Neumair. Modelling and Control with Modules of Signal Nets. To appear in Advanced in Petri Nets, LNCS, Springer 2004.
12. J. Desel, H. -M- Hanisch, G. Juhás, R. Lorenz and C. Neumair. A Guide to Modelling and Control with Modules of Signal Nets. To appear in LNCS, Springer 2004.
13. L.E. Pinzon, M.A. Jafari, H.-M. Hanisch and P. Zhao Modelling admissible behavior using event signals submitted
14. P.J. Ramadge, W.M. Wonham: The Control of Discrete Event Systems. Proceedings of the IEEE, 77 (1989) 1, S. 81-98.
15. G. Rozenberg, and J. Engelfriet. Elementary Net Systems. In W. Reisig and G. Rozenberg (Eds.) *Lectures on Petri Nets I: Basic Models*, Springer, LNCS 1491, pp. 12-121, 1998.
16. P. Darondeau and S. Kumagai (Eds.). Proceedings of the *Workshop on Discrete Event System Control*. Satellite Workshop of ATPN 2003.
17. M.C. Zhou und F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems.* Kluwer Adacemic Publishers, Boston, MA, 1993.