Modelling and Control 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. We present a modular formalism and methodology for modelling and control of discrete event systems, such as flexible manufacturing systems. The formalism is based on Petri net modules which communicate via signals. Two kinds of signals are employed, namely active signals, which force occurrence of (enabled) events (typically switches), and passive signals which enable/prohibit occurring of events (typically sensors). Modelling with such modules appears to be very natural from engineering perspective, enables hierarchical structuring, and support locality principle.
Further, we discuss the role of both kinds of signals in control tasks and we focus on the control aspects in general. We present a methodology for synthesis of controlled behavior for systems modelled by modules of signal sets. 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 non-blocking behavior of the plant respecting the control specification. Finally, we show how to synthesize the controller (as a module of a signal net) forcing the plant to realize the controlled behavior.

1 Introduction

Petri Nets are already widely used for modelling and control of Discrete event systems [10, 22], because of their modelling power, graphical expression, strong theoretical background, very developed analytical methods, tools, and many other features. However, there are still some features which are not directly supported by Petri Nets (at least in their basic version), but are, on the other hand, quite natural for engineers working with real applications. For example, to cover control tasks, Petri nets were extended by adding external conditions, which are necessary for enabling occurrence of transitions [10]. In the following paragraphs we are trying to identify some of features which are important for applications and are not directly supported by Petri nets. Based on this discussion we are presenting an extension of Petri nets, which can still benefit from all strong advantages that Petri nets bring, but also enables to deal with the discussed unsupported features in an effective way.

Petri nets are in principle distributed, however they do not support modularity. Modularity is quite natural and important in engineering. In complex application, models are usually built in several steps and are described on several levels of abstraction. Almost

* Supported by DFG: Project "SPECIMEN".

each system is a part of a bigger system, such as a robot is a part of a manufacturing cell, as well as almost each system itself is composed from subsystems. This fact gives an importance to principle of compositionality. Thinking on one level of abstraction one does not need to reason about all details of subsystems which were taken into consideration in a sublevel. It is usually sufficient to consider just those parts of subsystems, which are in contact with environment, i.e. "input/output" parts and to consider the "inside" of the subsystems being a "black box". Such approach supports local changes in the whole system, it enables a replacement of one module by another with the same "input/output" functionality. A typical example of a modular approach in control applications are block diagrams. It would be very nice to have such a modular approach based on Petri nets. There are already developed many compositional frameworks for Petri nets, mostly based on gluing common places and/or transitions. However, because the subject of engineering are mostly complex systems, it is desirable that the composition of modules preserve the structure of modules.

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 behaviour 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 behaviour 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". In order to answer the question, let us discuss a very simple example. Consider a switch which can turn on a light. Turning on the switch forces the bulb to light, however, only if the bulb is not damaged. And of course the bulb can not start to light, if no switch is turned on. In other words, in the previous situation the switch is the *actuator* of the bulb. In this example the switch plays the role of the control, while the light (bulb with cables etc.) plays the role of the system to be controlled. Thus, inputs to the system can be actuators representing conditional assymmetric synchronization - events of the control are trying to force events in the system. This is a typical situation in control of discrete event systems: a product line will not start without pressing a control button, or a mobile phone will not call a number without pressing appropriate buttons, but a printer is not printing without paper even if the "print" button was pressed. The other typical interaction between control and discrete event systems are sensors readings: An event in the system can occur only if a sensor in the control is in a certain state and vice versa. Thus events can be enabled/prohibited via states of sensors.

Thus, 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). Event resp. states associated to outputs are called *observable*. Of course there can be unobservable events resp. states. As mentioned, it would be natural to model control of a discrete events systems by influencing its behaviour by actuators and sensors in order to observe desired outputs as decribed 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, 16] 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). For example, modelling a switch and a light, one needs to prohibit all controllable events except the event "a bulb starting to light" to model the situation when the switch turns on. In fact, in case of supervisory control, the control means to restrict the behaviour of the system to fulfil the control specification. As mentioned in [1], pp. 185 - 202, "sometimes it is desirable to have a controller which not only disables controllable events but also chooses one among the enabled ones. This event can be interpreted as a command given to the plant." The solution to such cases is given by a construction of "an inplementation", which is a special supervisor, enabling at most one controllable event at a time. There arises the natural question, why not directly model actuators?

We would like to have an extension of Petri nets, which support input/output structuring using actuator and sensors, modularity and compositionality in an intuitive graphical way.

So, as a modelling formalism, we use modules communicating by means of the above described signals. This formalism is based on the work [18], where automata were used to describe the internal behavior of a system, and the paper [19], where Petri nets were used for this purpose. We call these models, i.e. Petri nets equipped with two kinds of signals, *signal nets*. Nowadays, this concept is successfully used in modelling and control of discrete event systems by a growing community. There are several dialects of these nets and several different names, such as net condition/event systems [8, 7, 9] or signal nets [20]. 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 [8, 7, 9] 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 [15]. However, in [15] 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 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. In addition, we consider a general class of control specification in form of a language over steps of event outputs (steps of observable events). We consider 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.

As it was already mentioned, in case of supervisory control, the behavior of the DES can not be forced by the supervisor: control means to restrict the behavior of the system to fulfill the control specification. Formally (see e.g. [2]), there is given a regular prefix closed language over the set of system events. This language represents the uncontrolled behavior of the system. Control specification is given in form of a regular subset of this language and is representing the desired behavior. Moreover, some states in an automaton representing the uncontrolled behavior are marked. The sequences (words) of events leading to these states describe completed tasks. Remember also, that the events of the system are divided into controllable events, which can be enabled/prohibited by the control, and uncontrollable events. The basic aim of supervisory control is to find a supervisor, which will prohibit the controllable events in such a way, that the behavior of the system is restricted to its maximal regular sublanguage, which still respects the control specification, and is moreover non-blocking (every sequence of this language can be completed to a marked state, i.e. no dead- or livelocks occur in unmarked states). Such a supervisor is called minimally restrictive nonblocking supervisor.

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. The presented solution is maximal in the sense, that we match all sequences of (steps of) outputs which can be achieved by sending appropriate inputs without being in danger to observe a sequence of (steps of) outputs which is not a prefix of a sequence in the control specification, or a sequence of (steps of) outputs which can not be completed to a sequence in the control specification (i.e. which is blocking). The maximality is achieved under the paradigm, that no output signal of the plant can synchronize an input signal of the plant (as already stated above). In other words, 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. Finally, we show that for such a behavior there exists a control module (of a signal net), which will in composition with the plant module realize this behavior. As the main result we will construct such a control module.

The paper is organized as follows: In Section 2 we present *modules of signal nets* with definition of step semantics, composition rules and input/output behavior. In Section 3 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 Section splits into two parts. In Subsection 4.1 we synthesize the maximally permissive nonblocking behavior of a module of a signal net (representing the plant) respecting a given regular specification language. Finally, in Subsection 4.2 we present the construction of the controller as a module of a signal net.

2 Modules of Signal Nets

As mentioned in the introduction we present an extension of Petri nets which allows to model actuators, sensors and modularity, and still has all the benefits that Petri nets bring. We assume the underlying Petri nets to be 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. [13]). 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). 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. Taking an example, an event turning on a switch would be modelled via the transition t_1, while the event lighting the bulb would be modelled via transition t_2.

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. Notice that this implies that event arcs build no cycles. 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 and 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 and t_2 can occur in one synchronized step. As an example you can think of several switches to turn on a light on (see Figure 1, part (a)).

If the transitions t_1 and t_2 build a synchronized step themselves, then there are two dialects in literature to interpret such a situation: In the first one ([8, 7, 9]) 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 and t_2, too. We call this dialect AND-semantics (see Figure 1, part (b)).

In the second one ([4]) 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 pos-

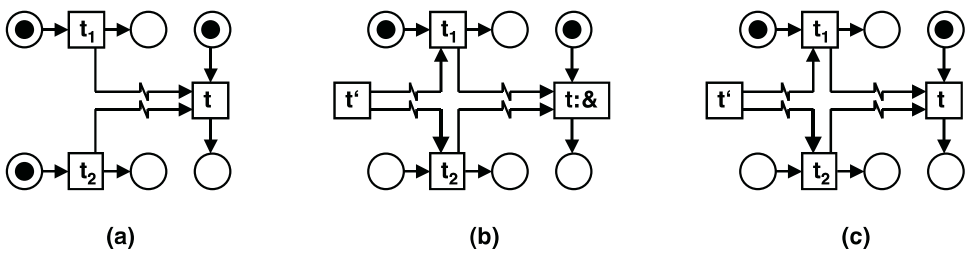


Fig. 1. In Figure (a) the enabled steps are $\{t_1, t\}$ and $\{t_2, t\}$. Figure (b) shows a signal net in AND-semantics: here the only enabled step is $\{t', t_1\}$, i.e. t is not synchronized. In Figure (c) the same net is shown in OR-semantics: here we have the enabled step $\{t', t_1, t\}$, i.e.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

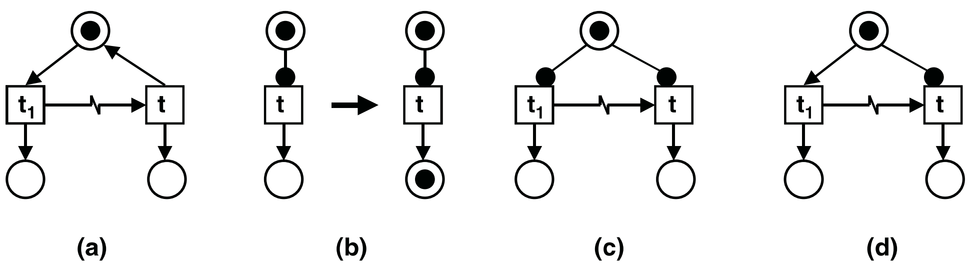


Fig. 2. Figure (a) shows an enabled step $\{t_1, t\}$. The left part of Figure (b) shows an enabled transition t, which tests a place to be marked. The occurrence of t leads to the marking shown in the right part of Figure (b). Figures (c) and (d) again present situations of an enabled step $\{t_1, t\}$.

sible 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 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 ([12]) 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} *(*$P \cap T = \emptyset$*),*
$F \subseteq (P \times T) \cup (T \times P)$ *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,
$x^{\bullet} = \{y \mid (x, y) \in F\}$ 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 notions 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* ${}^{\rightsquigarrow}t \subseteq \xi$*.*

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 ((Potentially) enabling of steps). *A step* ξ *is* potentially enabled *in a marking* m *if*

- *All $t \in \xi$ are enabled: ${}^{\bullet}t \cup {}^{+}t \subseteq m$ and $(t^{\bullet} \setminus {}^{\bullet}\xi) \cap m = \emptyset$ and*
- *No pair of transitions $t, t' \in \xi$ is in conflict: ${}^{\bullet}t \cap {}^{\bullet}t' = t^{\bullet} \cap (t')^{\bullet} = \emptyset$.*

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.*

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 finite set of input/output signals, and
$\Psi^{arc} = CI^{arc} \cup EI^{arc} \cup CO^{arc} \cup EO^{arc}$ is a finite set of arcs connecting input/output signals with the elements of the net N. Namely,
C^{in} resp. E^{in} denotes the set of condition resp. event inputs,
C^{out} resp. E^{out} the set of condition resp event outputs,
$CI^{arc} \subseteq C^{in} \times T$ resp. $EI^{arc} \subseteq E^{in} \times T$ the set of condition resp event input arcs,
$CO^{arc} \subseteq P \times C^{out}$ resp. $EO^{arc} \subseteq T \times E^{out}$ the set condition resp event output arcs,
$c_0 \subseteq C^{in}$ the initial state of the condition inputs.

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.

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 vice versa with a composition mapping Ω. The connections of the nets to the involved identified inputs and outputs are replaced by direct signal arcs respecting the identification (see Figure 7), such that

- the initial markings are compatible with the initial states of the condition inputs, and
- no cycles of event arcs are generated.

The composition of M_1 and M_2 w.r.t. Ω is denoted by $M_1 *_{\Omega} M_2$.

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 $\Psi_i = (\Psi_i^{sig}, \Psi_i^{arc})$ and initial markings m_{0i} ($i = 1, 2$).*

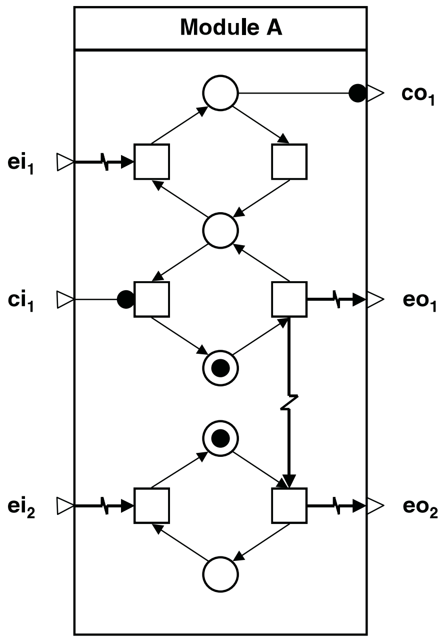


Fig. 3. A module of a signal net with condition inputs $C^{in} = \{ci\}$, event inputs $E^{in} = \{j, k\}$, condition outputs $C^{out} = \{co\}$ and event outputs $E^{out} = \{u, v\}$.

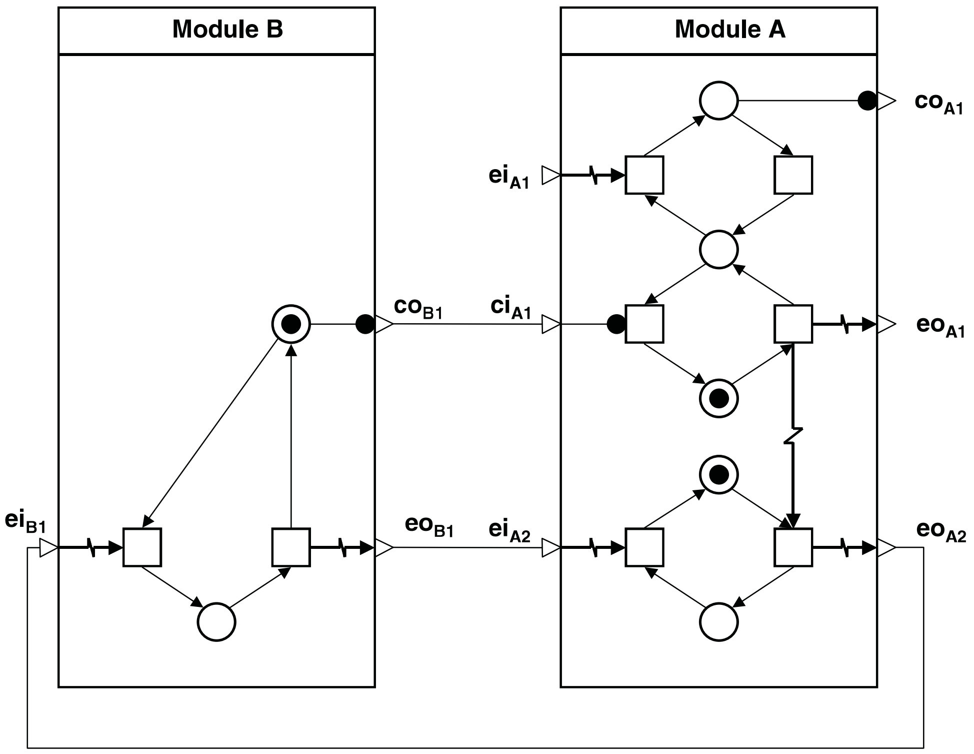


Fig. 4. The composition of two modules.

Let $Q \subseteq \Psi_1^{sig}$ and $\Omega : Q \rightarrow \Psi_2^{sig}$ be an injective mapping, such that the initial markings are compatible with the initial states of the condition inputs:
$(p, co) \in CO_1^{arc} \wedge \Omega(co) \in c_{02} \Rightarrow p \in m_{01}$ and
$(p, co) \in CO_2^{arc} \wedge \Omega^{-1}(co) \in c_{01} \Rightarrow p \in m_{02}$.

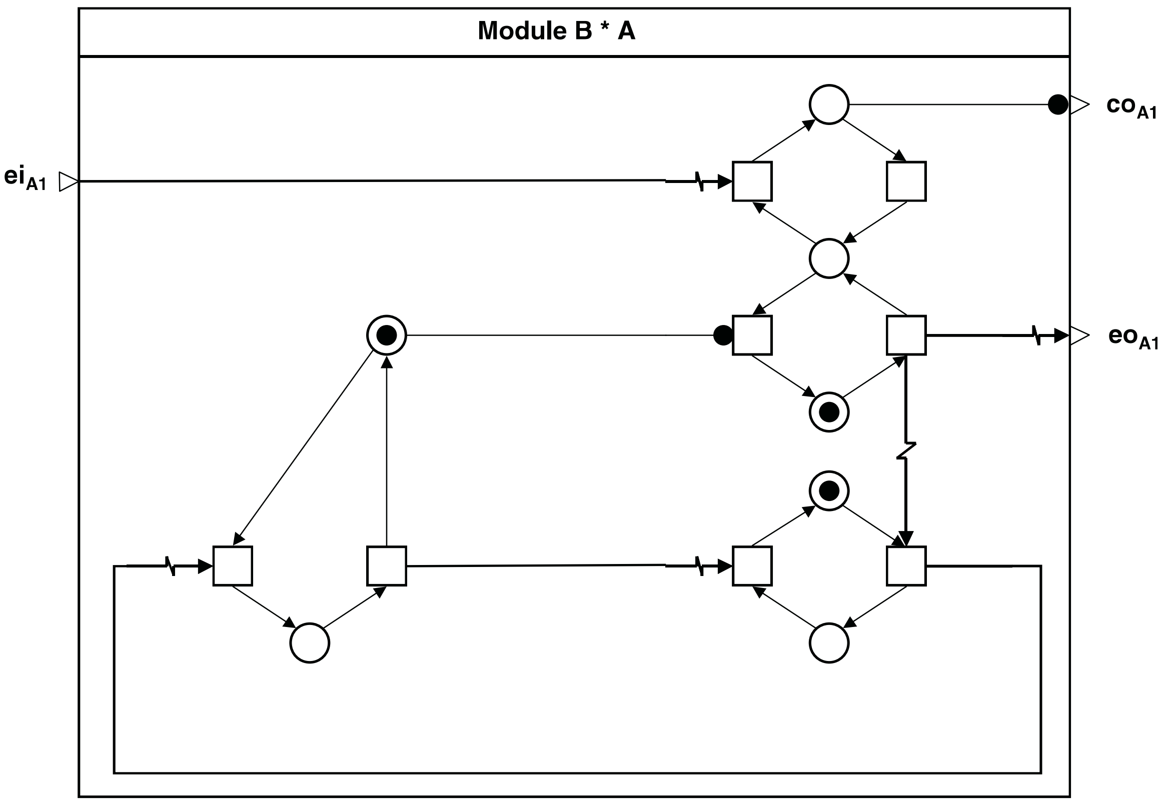


Fig. 5. The result of the composition of the modules from Figure 4.

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* $\Omega(C_1^{out} \cap Q) \subseteq C_2^{in}$.
Finally, no cycles of event arcs should be 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:

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

Remark 1. For each new arc (t, t') in a composed module $M_1 *_\Omega M_2$ w.r.t. an event output $eo \in E_1^{out}$ with $(t, eo) \in EO_1^{arc}$ and $(\Omega(eo), t') \in EI_2^{arc}$ we say that t' *replaces* eo and t *replaces* $\Omega(eo)$. A similar notion is used also for new arcs w.r.t. event inputs and condition inputs and outputs.

In order to define the behavior of a module, observe: transitions connected by an event input to the environment are not able to occur spontaneously, but need to be synchronized by the event input in order to occur. Similar a transition connected by an condition input to the environment is only able to occur, if the condition input is activated. Therefore we are interested in the behavior of the module w.r.t. a given environment. In the most general case this environment is assumed to be maximal permissive in the sense, that there is no causal restriction in sending event inputs and activating condition inputs. We will model such an environment also as a module $\mathcal{E}$ of a signal net and then compose the environment module appropriately with the original module M. $\mathcal{E}$ realizes a maximally permissive environment in the following sense:

- at any moment $\mathcal{E}$ can send event inputs to M: so each event signal of M is modelled in $\mathcal{E}$ by a corresponding always enabled transition;
- at any moment $\mathcal{E}$ can enable and disable condition inputs of M: so each condition input of M is modelled in $\mathcal{E}$ by a corresponding place, which can be marked and unmarked by associated transitions;
- $\mathcal{E}$ can observe outputs of M: every output of M is modelled in $\mathcal{E}$ by a corresponding transition, which synchronized in the case of an event output, and enabled in the case of an condition output;
- in $\mathcal{E}$ no synchronization between its transitions is allowed: in particular, inputs should not be sent in steps from $\mathcal{E}$ to M, and outputs M should only be observed by $\mathcal{E}$ and not synchronize inputs of M via $\mathcal{E}$.

Definition 9 (Maximally permissive environment). *Let $M = (N, \Psi, c_0)$ be a module with $\Psi = (\Psi^{sig}, \Psi^{arc})$. Define the maximally permissive environment module $\mathcal{E} = (N_\mathcal{E}, \Psi_\mathcal{E}, c_{0\mathcal{E}})$, $\Psi_\mathcal{E} = (\Psi_\mathcal{E}^{sig}, \Psi_\mathcal{E}^{arc})$, w.r.t. M by $EN_\mathcal{E} = CN_\mathcal{E} = \emptyset$ and*

$$\begin{aligned}
P_\mathcal{E} &= \{p_{ci.on} \mid ci \in C^{in}\},\\
T_\mathcal{E} &= \{t_c \mid c \in C^{out}\} \cup\\
&\quad \{t_{ci.on} \mid ci \in C^{in}\} \cup \{t_{ci.off} \mid ci \in C^{in}\} \cup\\
&\quad \{t_e \mid e \in E^{out} \cup E^{in}\},\\
F_\mathcal{E} &= \{(t_{ci.on}, p_{ci.on}) \mid ci \in C^{in}\} \cup \{(p_{ci.on}, t_{ci.off}) \mid ci \in C^{in}\},\\
m_{0\mathcal{E}} &= \{p_{ci.on} \mid ci \in C^{in} \cap c_0\},\\
C_\mathcal{E}^{in} &= \{ci_c \mid c \in C^{out}\},\\
C_\mathcal{E}^{out} &= \{co_c \mid c \in C^{in}\},\\
E_\mathcal{E}^{in} &= \{ei_e \mid e \in E^{out}\},\\
E_\mathcal{E}^{out} &= \{eo_e \mid e \in E^{in}\},\\
CI_\mathcal{E}^{arc} &= \{(ci_c, t_c) \mid c \in C^{out}\},\\
CO_\mathcal{E}^{arc} &= \{(p_c, co_c) \mid c \in C^{in}\},
\end{aligned}$$

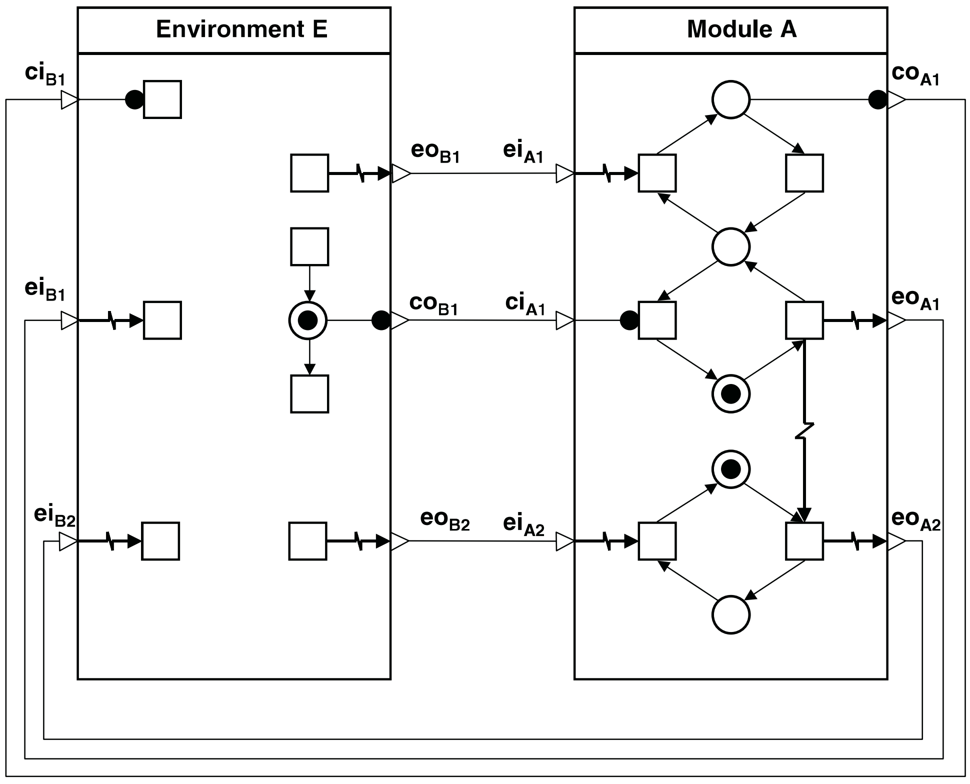


Fig. 6. The composition of the module in Figure 3 with its maximally permissive environment module.

$$EI^{arc}_{\mathcal{E}} = \{(ei_e, t_e) \mid e \in E^{out}\},$$
$$EO^{arc}_{\mathcal{E}} = \{(t_e, eo_e) \mid e \in E^{in}\}.$$

The composition of M with its maximally permissive environment $\mathcal{E}$ is called the standalone of M (observe that this composition has empty input/output structure) (as an example see Figure 7).

Definition 10 (Standalones). *Let M be a module of a signal net and $\mathcal{E}$ be the maximally permissive environment module of M. The* standalone *of M is the composition module $M_S = (N_S, \Psi_S) = M *_{\Omega} \mathcal{E}$ w.r.t. the following composition mapping $\Omega : \Psi^{sig} \to \Psi^{sig}_{\mathcal{E}}$:*

$$\Omega(e) = ei_e \, for \; e \in E^{out},$$
$$\Omega(e) = eo_e \, for \; e \in E^{in},$$
$$\Omega(c) = ci_c \, for \; c \in C^{out},$$
$$\Omega(c) = co_c \, for \; c \in C^{in}.$$

Definition 11 (Behavior of modules of signal nets). *Let M be a module of a signal net and let $M_S = (N_S, \Psi_S)$ be the standalone of M. The set L_M of all occurrence sequences of N_S is called the behavior of the module M.*

L_M represents the set of all possible sequences of steps of input signals, output signals and inner transitions of M under the assumptions: Output signals of M can

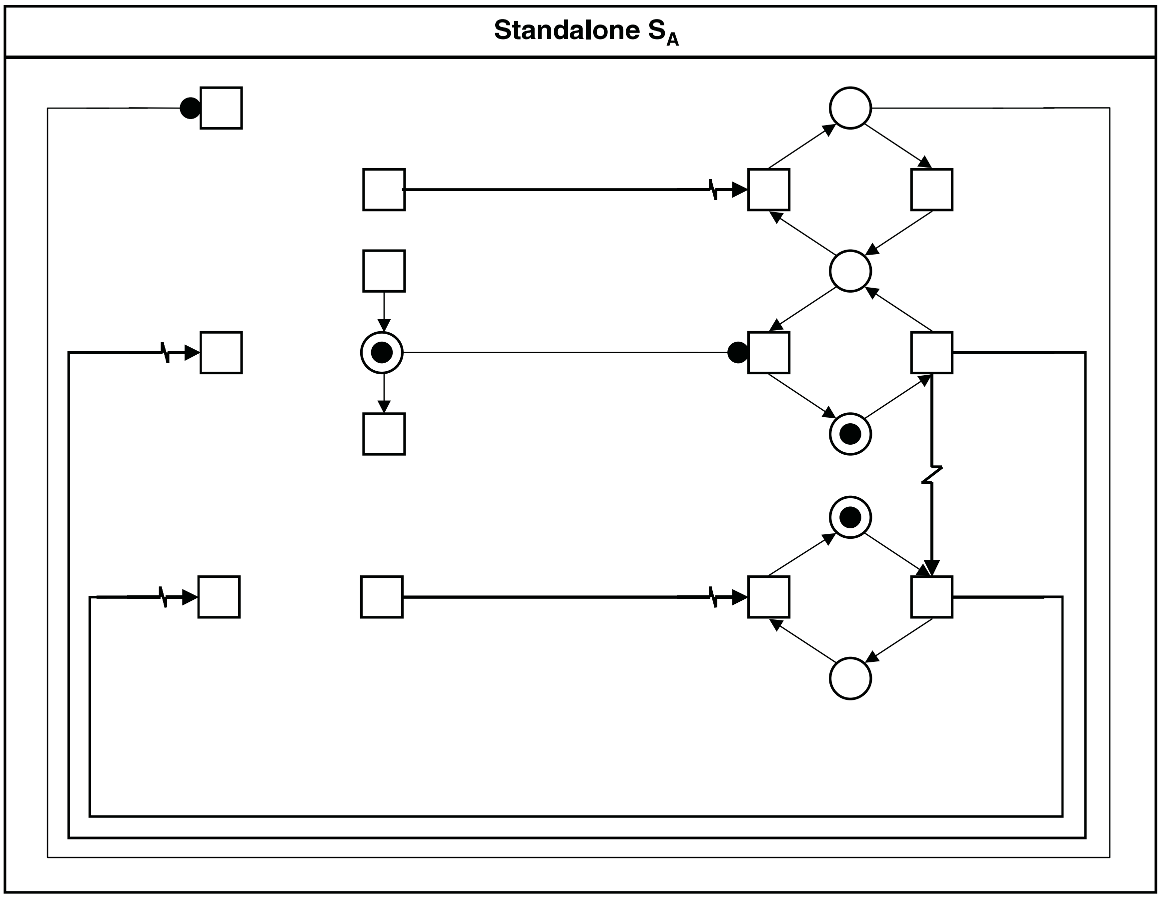


Fig. 7. The standalone of the module of a signal net in Figure 3.

not synchronize input signals of M via the maximally permissive environment module. Several input signals of M can not be sent in steps from the maximally permissive environment module.

Thus, modules of signal nets are a Petri net extension supporting input/output structuring, modularity and compositionality in an intuitive graphical way. They are used in many applications in the area of design, modelling and control of discrete event systems, such as flexible manufacturing systems and control of traffic systems for more than ten years, see e.g. [8, 7, 9, 19]. This fact gives a motivation for a more detailed theoretical investigation of this extension of Petri nets. In this section we have provided a proper formal foundation for this modelling framework, including definitions of input/output structure and composition of modules. In [14], we have concentrated on a definition of an equivalence w.r.t. input/output behaviour, which is preserved by the composition of modules. It is a crucial concept for hierarchical modelling, which enables to replace a module with a more abstract/concrete module with the same "input/output" functionality.

In the following sections we discuss the role of both kinds of signals in control tasks and we focus on the control aspects in general.

3 Controller Synthesis

As mentioned in the introduction, in classical control the aim is to influence the behavior of a system by a control via sensors and actuators in order to get a specified desired behavior. In principle there are two possibilities to express a desired behavior (see [2] for an actual survey, and [1, 21] for recent developments):

- the event based approach used in the seminal work of Ramadge and Wonham on supervisory control of discrete event systems (DES) [16]. In this framework the desired behavior is given in the form of legal sequences of events.
- the state based approach ([10]), where the desired behavior is derived from a set of legal resp. forbidden states.

Considering discrete event systems (DES) in both approaches the main problem is that the considered modelling formalisms (languages, automata, Petri nets) do not provide a mechanism for asymmetric synchronization intended by actuators.

For example in classical supervisory control this problem is solved by modelling actuators via prohibiting all other possible events ([5]). As a consequence, the behavior of the DES cannot be forced by the control, now called supervisor, but only be restricted. Formally there is given a regular prefix closed language over a fixed set of events representing the uncontrolled behavior of the DES and a regular subset of this language representing the restricted desired behavior. In the most general case one distinguishes between controllable events (which can be prohibited by the supervisor) and uncontrollable events, and between observable events (which can be observed by the supervisor) and unobservable events. The question is, which controllable events should be prohibited by the supervisor after observing a certain sequence of observable events in order to disable all undesired behavior in a *minimal restrictive way*.

We present an alternative to the existing approaches to control of DES with *direct modelling of actuators*. Our formalism is suitable for both kind of specifying the desired behavior. Because in literature the event based approach is more developed than the state based approach in the sense that it allows more general specifications ([23]), we concentrate in this paper on a event based specification of the desired behavior.

In particular, we specify the desired behavior by sequences of event output signals. Therefore we consider modules, modelling the plant, without condition output signals (which correspond to states). Notice that a condition output signal c in a behavior specification could be replaced by two event output signals $c.on$ and $c.off$ synchronized by transitions marking and emptying the place in ^+c, respectively.

Our framework could be easily adapted to behavior specifications which include input signals: In this case one could additionally consider specifications of the form *"After sending input i, we want to observe a sequence of outputs w"* or *"input i always synchronizes output o"*. The restriction to sequences of output signals is only for sake of simplicity.

Throughout this section we consider a module $\mathcal{P}$ of a signal net as a model of an uncontrolled plant and its maximally permissive environment $\mathcal{E}$. As in the previous section T denotes the set of transitions of $\mathcal{P}$. We additionally fix the set I of transitions of $\mathcal{E}$ corresponding to event input signals together with transitions switching condition input signals on/off, and the set O of transitions of $\mathcal{E}$ corresponding to event output signals:

$$\begin{aligned} I &= \{t_e \mid e \in E^{in}\} \cup \{t_{ci.on} \mid ci \in C^{in}\} \cup \{t_{ci.off} \mid ci \in C^{in}\}, \\ O &= \{t_e \mid e \in E^{out}\}. \end{aligned}$$

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 $\mathcal{E}$, and observable are, beside the input signals, exactly the output signals, modelled by the set of transitions O of $\mathcal{E}$.

Remember that we specify a desired behavior of $\mathcal{P}$ by a set of desired sequences only of output signals (in difference to supervisory control, where the specification is over all events, observable and unobservable, controllable and uncontrollable ones). Observe that, since the event arc relation produces a step semantics, we observe sequences of steps of output signals.

The aim of control synthesis is to find a control module $\mathcal{C}$ which appropriately composed (by a composition mapping Ω) with $\mathcal{P}$ fulfills: Each occurrence sequence of the underlying signal net of $\mathcal{C} *_\Omega \mathcal{P}$ respects the desired behavior in the sense that the projection of this occurrence sequence onto the set of transitions of $\mathcal{C}$ which replaces output signals of $\mathcal{P}$ (see remark 1) belongs to the desired behavior.

We synthesize such a control module $\mathcal{C}$ in two steps. First we define conditions of *controllability* of a subbehavior of the behavior $L_\mathcal{P}$ of the plant module $\mathcal{P}$ (analogously to [16]) and show how to compute the maximal controllable subbehavior of $L_\mathcal{P}$ respecting the desired behavior (if it exists), see subsection 4.1. Second we show that for every controllable subbehavior of $L_\mathcal{P}$ there is a control module $\mathcal{C}$, which in composition with the plant module $\mathcal{P}$ realizes this controllable subbehavior. As the main result of the second step we will construct such a control module by adding new net structure to the maximally permissive environment module $\mathcal{E}$ in $\mathcal{E} *_\Omega \mathcal{P}$ (see Figure 7), see subsection 4.2.

Since sets of occurrence sequences of signal nets are regular languages[1] over an alphabet of steps, we assume the desired behavior to be a regular language. In the following section we provide a short introduction to the theory of regular languages, which will be used in the subsections 4.1 and 4.2.

4 Regular Languages

We need the following language theoretic notations ([11] and [2]). For a finite set A we denote

- $2^A = \{B \mid B \subseteq A\}$ the set of all subsets of A,
- ϵ the empty word,
- $A^* = \{a_1 \dots a_n \mid n \in \mathbb{N}_0,\, a_1, \dots, a_n \in A\} \cup \{\epsilon\}$ the set of all finite words over the alphabet A.

In paragraph 3 we are solely concerned with regular languages $L \subseteq A^*$ with $A = 2^X$ for finite sets X. In the following we will briefly introduce some representations of regular languages and some operations on regular languages.

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

Finite Automata. A regular language L can be represented as the language $L(G)$ of a (deterministic) finite automaton $G = (S, A, \delta, F, s_0)$, where

- S is the set of states.
- $\delta : S \times A \to S$ is the transition function: $\delta(s, a) = s'$ means that the automaton reaches state s' when reading a in state s.
- s_0 is the initial state.
- $F \subseteq S$ is the set of accepting states.

The transition function is extended in the obvious way to $\delta : S \times A^* \to S$: $\delta(s, w) = s'$ means that the automaton reaches state s' when reading w in state s. A word $w \in A^*$ belongs to $L(G)$ if and only if $\delta(s_0, w) \in F$. The states of G can be denoted as equivalence classes over A^*:

$$[w]_G = \{v \in A^* \mid \delta(s_0, v) = \delta(s_0, w)\}.$$

A finite automaton can be viewed as a labelled transition system (S, Σ, A), where S is the set of states, $\Sigma = \{s \xrightarrow{a} s' \mid \delta(s, a) = s'\}$ the set of transitions and A the set of (event) labels.

Regular Expressions. A regular language L can be represented as the language $L(\alpha)$ of a regular expression α over A. Regular expressions are build inductively from the elementary regular expressions $\alpha = x$ with $x \in A \cup \{\epsilon, \emptyset\}$, where $L(\alpha) = L(x) = \{x\}$ for $x \neq \emptyset$ and $L(\emptyset) = \emptyset$, by:

- *union*:
  α, β regular expressions $\Rightarrow \alpha + \beta$ regular expression with $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$.
- *concatenation*:
  α, β regular expressions $\Rightarrow \alpha\beta$ regular expression with $L(\alpha\beta) = L(\alpha)L(\beta) = \{uv \mid u \in L(\alpha),\ v \in L(\beta)\}$.
- *iteration*:
  α regular expressions $\Rightarrow \alpha^*$ regular expression with $L(\alpha^*) = (L(\alpha))^* = \{u_1 \ldots u_n \mid u_i \in L(\alpha),\ n \in \mathbb{N}\}$.

Operations on Regular Languages. There is the so called *prefix relation* $\leqslant \subset A^* \times A^*$:

$$u \leqslant v \Leftrightarrow \exists x \in A^* : \ ux = v.$$

In this case u is called a *prefix* of v. For $x \neq \epsilon$ we call u a *proper prefix* of v. Beside the set operations $\cap$, $\cup$ and $\setminus$ there are the following operations on languages:

- cocatenation L_1L_2 (already mentioned).
- iteration L^* (already mentioned).
- prefix closure $\overline{L} = \{v \in A^* \mid v \text{ is prefix of a word in } L\}$.
- postfix closure $post(L) = \{v \in A^* \mid \text{ a word in } L \text{ is prefix of } v\} = LA^*$.
- minimal words $min(L) = \{v \in L \mid \text{ no proper prefix of } v \text{ is in } L\}$.
- quotient $L_1/L_2 = \{v \in A^* \mid \exists w \in L_2 : \ vw \in L_1\}$.

- projection $P_B(L) = \{P_B(w) = P_B(x_1) \dots P_B(x_n) \mid w = x_1 \dots x_n \in L\}$, where $P_B(\epsilon) = \epsilon$ and
$$P_B(x) = \begin{cases} x \text{ for } x \in B, \\ \epsilon \text{ otherwise.} \end{cases}$$
- pumping $P_B^{-1}(L) = \{w \in (A \cup B)^* \mid P_B(w) \in L\}$.

The set of regular languages is closed under all these operations.

The Hiding Operator. For the alphabets of the form $A = 2^X$ as we consider, we need a more sophisticated projection operator, called *hiding* operator, to hide characters from subsets $Y \subseteq X$. We define the *hiding operator* λ_Y *w.r.t.* Y by:

- For a character $a \in A$:
  $\lambda_Y(a) = a \setminus Y$ if $a \setminus Y \neq \emptyset$, and $\lambda_Y(a) = \epsilon$ otherwise.
- For a word $w \in A^*$:
  $\lambda_Y(w) = \lambda_Y(a_1) \dots \lambda_Y(a_n)$ if $w = a_1 \dots a_n$, 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
$$[w]_Y = \{v \in A^* \mid \lambda_Y(w) = \lambda_Y(v)\}.$$

The set of regular languages is closed under the hiding and corresponding pumping operations. This can be seen by constructing from a given regular expression α two regular expressions $\lambda_Y(\alpha)$ and $ext_Y(\alpha)$ such that $L(\lambda_Y(\alpha)) = \lambda_Y(L(\alpha))$ and $L(ext_Y(\alpha)) = \lambda_Y^{-1}(L(\alpha))$:

Definition 12. *Let X, Y be two finite sets and α be a regular expression over the alphabet $A = 2^X$.*

(a) We construct a regular expression $\lambda_Y(\alpha)$ over the alphabet 2^Y by replacing every character $a \in 2^X$ in α by $\lambda_Y(a)$.
(b) We construct a regular expression $ext_Y(\alpha)$ over the alphabet $2^{X \cup Y}$ by replacing every character $a \in 2^X$ in α by
$$ext_Y(a) = \left(\sum_{b \in 2^Y} b\right)^* \left(\sum_{b \in 2^Y} a \cup b\right) \left(\sum_{b \in 2^Y} b\right)^*,$$
and by replacing the character ϵ by
$$ext_Y(\epsilon) = \left(\sum_{b \in 2^Y} b\right)^*.$$

Observe that by construction

$$\begin{aligned} ext_Y(\alpha_1+\alpha_2) &= ext_Y(\alpha_1)+ext_Y(\alpha_2),\\ \lambda_Y(\alpha_1+\alpha_2) &= \lambda_Y(\alpha_1)+\lambda_Y(\alpha_2),\\ ext_Y(\alpha_1\alpha_2) &= ext_Y(\alpha_1)ext_Y(\alpha_2),\\ \lambda_Y(\alpha_1\alpha_2) &= \lambda_Y(\alpha_1)\lambda_Y(\alpha_2),\\ ext_Y(\alpha^*) &= (ext_Y(\alpha))^*,\\ \lambda_Y(\alpha^*) &= (\lambda_Y(\alpha))^*. \end{aligned}$$

Lemma 1. *Let X, Y be finite sets and α be a regular expression over the alphabet $A = 2^X$.*

(a) Each $w \in \left(2^Y\right)^*$ satisfies

$$w \in \lambda_Y(L(\alpha)) \Leftrightarrow w \in L(\lambda_Y(\alpha)).$$

(b) Each $w \in \left(2^{X\cup Y}\right)^*$ satisfies

$$\lambda_Y(w) \in L(\alpha) \Leftrightarrow w \in L(ext_Y(\alpha)).$$

Proof. We will only show '(b):$\Rightarrow$'. The proofs of the other cases are similar argumentations using again structural induction over the construction rules of regular expressions: Let $\alpha = x \in 2^X \cup \{\epsilon\}$ (these are the constants of a regular expression over 2^X), and let $w \in (2^{X\cup Y})^*$ satisfying $\lambda_Y(w) = x$. Then w is of the form $w = x_1 \dots x_n$ ($x_i \in 2^{X\cup Y}$), such that there exists an index k satisfying $\lambda_Y(x_k) = x$ and $\lambda_Y(x_i) = \epsilon$ for $i \neq k$. It follows immediately from the construction above, that $w \in L(ext_Y(\alpha))$.

Let α_1 and α_2 be regular expressions over the alphabet 2^X satisfying the induction hypothesis, and let $\beta_1 = ext_Y(\alpha_1)$ and $\beta_2 = ext_Y(\alpha_2)$ be the corresponding extensions according to the above construction. We get for $w \in (2^{X\cup Y})^*$:

(i) Assume $\lambda_Y(w) \in L(\alpha)$ for $\alpha = \alpha_1 + \alpha_2$:
In this case $\lambda_Y(w) \in L(\alpha_1)$ or $\lambda_Y(w) \in L(\alpha_2)$. By induction hypothesis $w \in L(\beta_1)$ or $w \in L(\beta_2)$. This implies $w \in L(\beta_1 + \beta_2) = L(ext_Y(\alpha))$.

(ii) Assume $\lambda_Y(w) \in L(\alpha)$ for $\alpha = \alpha_1\alpha_2$:
There are $w_1, w_2 \in (2^{X\cup Y})^*$ satisfying $w = w_1w_2$ and $\lambda_Y(w_i) \in L(\alpha_i)$ ($i = 1, 2$). By induction hypothesis
$w = w_1w_2 \in L(\beta_1)L(\beta_2) = L(\beta_1\beta_2) = L(ext_Y(\alpha))$.

(iii) Assume $\lambda_Y(w) \in L(\alpha)$ for $\alpha = (\alpha^1)*$:
There are $w_1 \dots w_n \in (2^{X\cup Y})^*$ satisfying $w = w_1 \dots w_n$ and $\lambda_Y(w_i) \in L(\alpha_1)$ ($i = 1, \dots, n$). By induction hypothesis $w = w_1 \dots w_n \in (L(\beta_1))^* = L(\beta_1^*) = L(ext_Y(\alpha))$.

$\square$

4.1 The Behavior of the Controlled Plant

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 selfcontained, i.e. can be understood without previous knowledge of supervisory control.

We search for a sublanguage K (of occurrence sequences) of the language $L_{\mathcal{P}}$ representing the behavior of the controlled plant, which can be realized by a composition of the plant module $\mathcal{P}$ and a control module $\mathcal{C}$. This implies the following requirements on K:

- If an occurrence sequence in K can be extended by a step of output transitions or unobservable 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 the 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 send after occurrence of any other sequence, which is undistinguishable to w

Observe that the first condition correspond to the classical one in supervisory control. The second one is due to our step semantics, where an input can synchronize different unobservable 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 (figure 8):

Definition 13 (Controllable Language). *A prefix closed, regular sublanguage K of $L_{\mathcal{P}}$ is said to be* controllable w.r.t. $L_{\mathcal{P}}$, I and O*, if*

$$\forall w \in K, \forall o \in 2^{O \cup T} : \; wo \in L_{\mathcal{P}} \Rightarrow wo \in K, \tag{1}$$

$$\begin{aligned} \forall vj \in K,\, j \cap I \neq \emptyset, \\ \forall j',\, j \cap I = j' \cap I, \\ \forall v' \in K,\, \lambda_T(v) = \lambda_T(v') : \;\; v'j' \in L_{\mathcal{P}} \Rightarrow v'j' \in K. \end{aligned} \tag{2}$$

If the sets $L_{\mathcal{P}}$, I and O are clear, we simply call K controllable.

Observe that for each controllable sublanguage K of $L_{\mathcal{P}}$ two undistinguishable words in $L_{\mathcal{P}}$ are either both in K, or both not in K. This property is also called *normality* in supervisory control. In case each controllable event is also observable (as in our framework) every controllable sublanguage of $L_{\mathcal{P}}$ can be proved to be *normal*.

Lemma 2. *Let $K \subseteq L_{\mathcal{P}}$ be controllable. Then for each $w \in L_{\mathcal{P}}$ either $[w]_T \cap L_{\mathcal{P}} \subseteq K$ or $[w]_T \cap L_{\mathcal{P}} \cap K = \emptyset$.*

Proof. We prove the lemma by contradiction: let $v, w \in L_{\mathcal{P}}$ with $\lambda_T(w) = \lambda_T(v)$ and

$$w \in K \quad \text{and} \quad v \notin K.$$

First observe $v \neq \epsilon$, since $\epsilon \in K$. Assume without loss of generality

$$v = v'x \quad \text{with} \quad v' \in K.$$

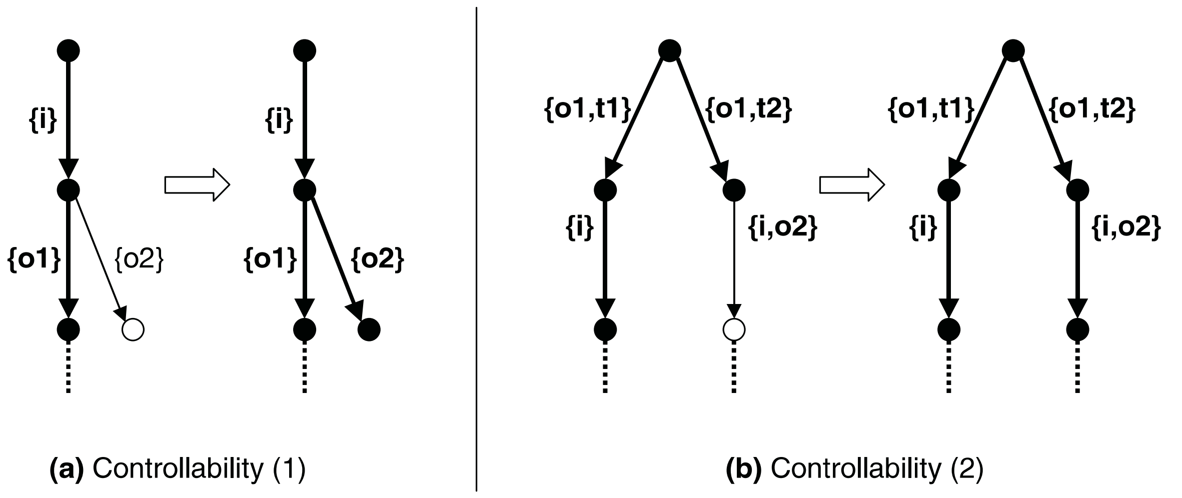


Fig. 8. A step (sequence) consisting of transitions from T (I resp. O) are denoted by (indexed) t's (i's resp. o's). Part (a) (controllability condition (1)): After sending the input i, the output o_2 cannot be avoided. Part (b) (controllability condition (2)): The steps $\{o_1, t_1\}$ and $\{o_1, t_2\}$ cannot be distinguished from the control, because only the output o_1 is visible. Therefore sending the input i should be allowed either in both cases or in none case. Hereby it does not play a role, what effect has this input to the plant (i.e. whether it synchronizes another output or not).

By condition (1) $x \cap I \neq \emptyset$. From $\lambda_T(w) = \lambda_T(v)$ we deduce

$$w = w'y \quad \text{with} \quad \lambda_T(w') = \lambda_T(v') \wedge x \cap I = y \cap I.$$

This contradicts condition (2). □

Observe that the property of normality can also be written in the form

$$\lambda_T^{-1}(\lambda(K)) \cap L_{\mathcal{P}} = K.$$

In supervisory control there exists moreover the notion of observability of sublanguages $K \subseteq L_{\mathcal{P}}$, which essentially states that if the supervisor cannot distinguish between sequences of events (according to some unobservable events), these sequences need the same control action. Observe, that this concept is directly integrated into the above definitions.

As mentioned we are searching for a controllable K, which additionally respects L_c and is maximal with this property:

Definition 14 (Maximally Permissive Controllable Language). *Let L_c be a regular language over the alphabet 2^O and let $K \subseteq L_{\mathcal{P}}$ be controllable w.r.t $L_{\mathcal{P}}$, 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, $L_{\mathcal{P}}$, I and O, if there exists no language K' satisfying $K \subsetneq K' \subseteq L$, which is controllable w.r.t. $L_{\mathcal{P}}$, I and O and fulfills $\lambda_{T \cup I}(K') \subseteq \overline{L_c}$.*

If the sets L_c, $L_{\mathcal{P}}$, I and O are clear, we simply call K maximally permissive controllable.

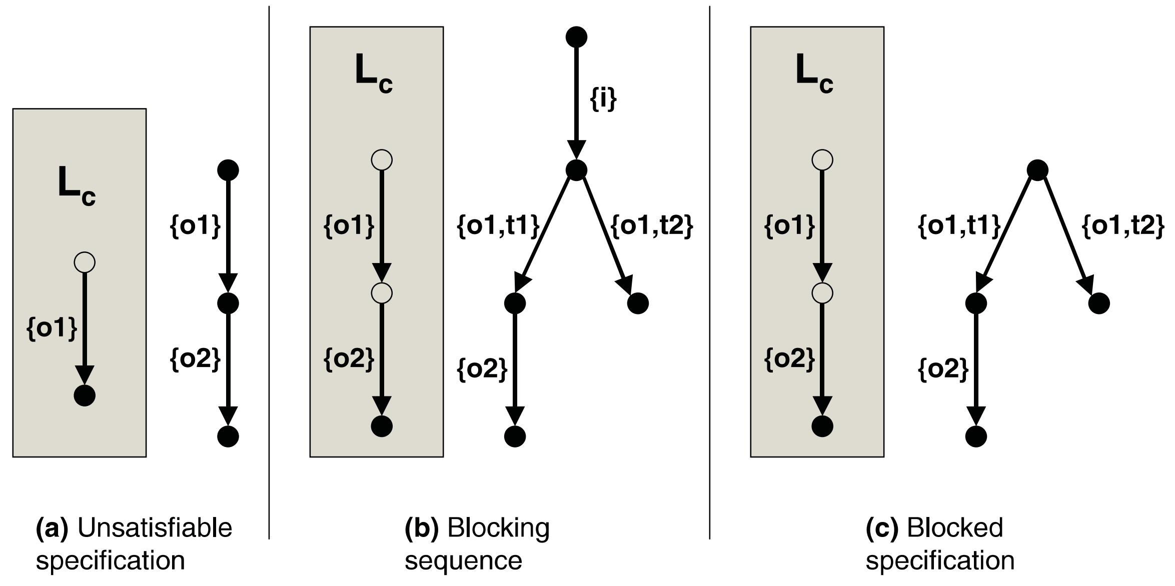


Fig. 9. A step (sequence) consisting of transitions from T (I resp. O) are denoted by (indexed) t's (i's resp. o's). The desired behavior is represented by finite automata, where accepting states are black. Part (a) (condition (3)): The occurrence sequence $\{o_1\}\{o_2\}$ is not a prefix of a word in L_c. So, L_c is unsatisfiable. Part (b) (condition (4)): The occurrence sequence $\{i\}\{o_1, t_2\}$ cannot be completed to a word respecting L_c. This blocking situation can only be avoided by not sending the input i before. Part (c) (condition (5)): The occurrence sequence $\{o_1, t_2\}$ cannot be completed to a word respecting L_c and cannot be avoided, i.e. L_c is only blocking satisfiable.

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 (figure 9 (a)).

Definition 15. *L_c is said to be* unsatisfiable (w.r.t. $L_\mathcal{P}$, I and O)*, if*

$$\exists w \in (2^{O \cup T})^* : \; w \in L_\mathcal{P} \wedge \lambda_T(w) \notin \overline{L_c}. \tag{3}$$

If this is not the case, we call L_c satisfiable (w.r.t. $L_\mathcal{P}$, I and O).

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 (figure 9 (b)):

Definition 16 (Nonblocking Language). *Let $K \subseteq L$ be maximally permissive controllable w.r.t L_c, $L_\mathcal{P}$, I and O and let $M \subseteq K$ be controllable w.r.t. $L_\mathcal{P}$, I and O satisfying*

$$\forall r \in M : \; \exists x \in (2^{O \cup I \cup T})^* \text{ with } rx \in M, \; \lambda_{I \cup T}(rx) \in L_c. \tag{4}$$

We say that M is nonblocking controllable *w.r.t. L_c, $L_\mathcal{P}$, I and O. If it is maximal with this property, M is called* maximally permissive nonblocking controllable language *w.r.t. L_c, $L_\mathcal{P}$, I and O.*

If the sets L_c, $L_{\mathcal{P}}$, I and O are clear, we simply call K maximally permissive non-blocking controllable language.

In classical supervisory control some states in an automaton representing the behavior of the uncontrolled plant are marked. The sequences (words) of events leading to these states describe completed tasks. Nonblocking control requires that every sequence of the language of the controlled plant can complete to a marked state (no dead- or livelock occurs in unmarked states).

In comparison we define nonblocking w.r.t. the given (not prefix closed) specification L_c, thus specifying desired tasks also by L_c and not by an extra given set of states in a fixed automaton representing the uncontrolled behavior. As we will see later, one can construct an automaton recognizing $L_{\mathcal{P}}$, where the set words respecting L_c correspond to certain marked states.

Assume L_c is satisfiable and let K be the maximally permissive controllable. If $K \neq \{\epsilon\}$, it is possible to get $M = \{\epsilon\}$ as maximally permissive *nonblocking* controllable language, but only if $\epsilon \in L_c$. In this case the maximal nonblocking behavior is empty, but without inputs from outside no blocking state can be reached. If even without any input a blocking state can be reached, we call L_c blocked (figure 9 (c)).

Definition 17. *Let $M \subseteq L_{\mathcal{P}}$ be prefix closed. We refer to the the condition*

$$\exists w \in (2^{O \cup T})^* : \, w \in M \wedge (\forall x \in (2^{O \cup I \cup T})^* : \, wx \in M \Rightarrow \lambda_{T \cup I}(wx) \notin L_c), \quad (5)$$

as blocking condition *w.r.t. M.*

Remark 2. If the condition (5) is fulfilled w.r.t. a language M, L_c is said to be *blocked* w.r.t. M.

Lemma 3. *If L_c is blocked w.r.t. $L_{\mathcal{P}}$, then there is no nonblocking controllable sublanguage of $L_{\mathcal{P}}$.*

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 safefy properties can be formalized via a prefix closed specification L_c.

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. In other words: If an occurrence sequence realizes no undesired behavior, all prefixes also do so. That means, safety properties can be formalized by prefix closed specification languages. On the other hand every prefix closed specification language can be regarded as the formalization of safety properties.

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.

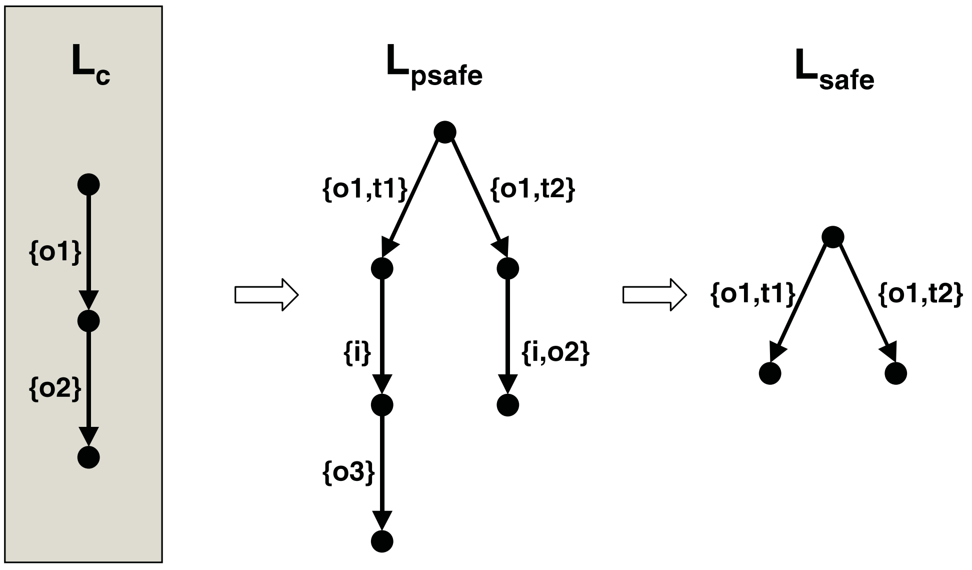


Fig. 10. The languages are represented by finite automata, where accepting states are black (observe that all languages are prefix closed). Sending the input i after observing the output o_1 can cause the not avoidable output o_3. This gives an occurrence sequence not respecting L_c. Therefore allways after observing o_1 the input i should not be sent.

Definition 18. *We define*

$$\begin{aligned} L_{psafe} &= \{w \in L_{\mathcal{P}} \mid \lambda_{I\cup T}(w) \in L_c\} = \lambda_{I\cup T}^{-1}(L_c) \cap L_{\mathcal{P}}, \\ L_{unsafe} &= \{w \in L_{\mathcal{P}} \mid \lambda_{I\cup T}(w) \notin L_c\} = L_{\mathcal{P}} \setminus L_{psafe}. \end{aligned}$$

Observe that

$$L_c \text{ unsatisfiable } \Leftrightarrow \exists w \in L_{unsafe} \cap (2^{O\cup T})^*.$$

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 (condition (1) in definition 13). Such occurrence sequences must be cut at the last possible input (the last possibility of control), if there is one. Due to condition (2) (of controllability) such an input must be avoided at all undistinguishable places. The words ending with these inputs are collected in the language L_{danger}. Deleting the futures of occurence sequences in L_{danger} from L_{psafe} gives the language L_{safe}, which we will prove below to be the searched language M (figure 10).

Definition 19. *We define*

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

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

If L_c is satisfiable, every word from L_{unsafe} has a nonempty prefix in L_{danger}. It is obvious from the definitions of L_{psafe}, L_{unsafe}, L_{danger} and L_{safe} that every set $[w]_T \cap L_{\mathcal{P}}$ is either subset of or disjoint to these languages (see also lemma 2).

The main result of this subsection is the following theorem:

Theorem 1. *L_{safe} is maximally permissive nonblocking controllable w.r.t. L_c, $L_{\mathcal{P}}$, I and O, if L_c is satisfiable w.r.t. $L_{\mathcal{P}}$, I and O.*

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 sophisticated framework presented in [2] could be adapted (since our different notion of controllability is still compatible with the union operation $\cup$). In [2] can also be found some hints to the complexity of the computation.

By definition L_{psafe} and L_{unsafe} are regular (see section 4):

Lemma 4. *L_{psafe} and L_{unsafe} are regular.*

Next we show that L_{danger} is regular. We will give L_{danger} as a simple formula over 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 remaiming words in L_{danger} are of the form $v'j'$ with $\lambda_T(v') = \lambda_T(v)$ 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 $\overline{\lambda}$ defined by

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

Obviously the operators $\overline{\lambda}$ and $(\overline{\lambda})^{-1}$ preserve the regularity of languages, since this is the case for the hiding operator λ as argued in lemma 1 (section 4). We get

Lemma 5. *L_{danger} and L_{safe} are regular.*

Proof. L_{danger} is regular, since it can be constructed by regularity preserving operations in the following way:

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

Then also $L_{safe} = L_{psafe} \setminus \mathit{post}(L_{danger})$ is regular as a formula over regular languages. □

The main theorem 1 now is shown in two steps by the following lemmata.

Lemma 6. *Let L_c be satisfiable. Then L_{safe} is controllable.*

Proof. We show both conditions of controllability by contradiction:

(i) Condition (1):
Assume there is an word $w \in L_{safe}$ and a step $o \in 2^{O \cup T}$ 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 imlies 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.r.t. $L_{\mathcal{P}}$, I and O, wo has a prefix in L_{danger}. This again contradicts $w \in L_{safe}$.

(ii) Condition (2):
Assume there are words $v', vj \in L_{safe}$ and a step j' with $j \cap I = j' \cap I \neq \emptyset$ and $\lambda_T(v') = \lambda_T(v)$ satisfying $v'j' \in L_{\mathcal{P}}$, but $v'j' \notin L_{safe}$. For such vj and $v'j'$ we have according to the definition of L_{danger}:

$$vj \in post(L_{danger}) \Leftrightarrow v'j' \in post(L_{danger}).$$

From this it follows $vj \notin L_{safe}$ analogously to the first case. A contradiction.

$\square$

Lemma 7. *Let L_c be satisfiable. There is no language $K \subseteq L_{\mathcal{P}}$ satisfying $L_{safe} \subsetneq K$, which is controllable and 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 word $w' \in K$ 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 is are words $v' \in L_{\mathcal{P}}, y \in (2^{O \cup T})^*$ and a step j' with $j \cap I = j' \cap I$ and $\lambda_T(v) = \lambda_T(v')$ 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 (condition (2)) and consequently $v'j'y \in K$ (condition (1)).

$\square$

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

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 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.

We now search for a sublanguage L_{nbsafe} of L_{safe}, which is controllable and respecting L_c, nonblocking and maximal with these two properties according to definition 16. As mentioned, in our framework every controllable event is also observable. Therefore, we are able to adapt a result in supervisory control ([2], subsection 3.7.5), which states (under the assumption that every controllable event is also observable): If there is at least one controllable language respecting L_c which is nonblocking, then there is a unique maximal one.

In order to compute L_{nbsafe}, we collect all blocking occurrence sequences of L_{safe} in the set $L_{blocking}$ (observe that every future of a blocking occurrence sequence is blocking, too). We have to cut all occurrence sequences in this set at the last possible input, if there is one. Due to condition (2) (of controllability) such an input must be avoided at all undistinguishable places. The prefixes ending with these inputs are collected in the language $L_{badchoice}$. Deleting the futures of occurrence sequences in $L_{badchoice}$ possibly produces new blocking words. Therefore we have to iterate this procedure. We define (figure 11)

Definition 20. *Let $M \subseteq L_{safe}$. Denote*

$$\begin{aligned} M_{blocking} &= \{w \in M \mid \nexists x \in (2^{O \cup I \cup T})^* : \; wx \in M \wedge \lambda_{I \cup T}(wx) \in L_c\}, \\ &= M \setminus ((M \cap \lambda^{-1}_{I \cup T}(L_c))/(2^{O \cup I \cup T})^*), \end{aligned}$$

$$\begin{aligned} M_{badchoice} = \{vj \in M \mid \; & j \cap I \neq \emptyset, \\ & \exists v' : \; \lambda_T(v') = \lambda_T(v), \\ & \exists j' : \; j' \cap I = j \cap I, \\ & \exists y \in (2^{T \cup O})^* : \; v'j'y \in min(M_{blocking}))\}. \end{aligned}$$

The language $M_{blocking}$ is regular by definition, if M is regular. In analogy to L_{danger}, then $M_{badchoice}$ is regular, too. Observe that

$$L_c \text{ blocked w.r.t. } M \Leftrightarrow \exists w \in M_{blocking} \cap (2^{O \cup T})^*.$$

We are now prepared to state the algorithm to compute L_{nbsafe}:

Input: Language $M^0 = L_{safe}$, Integer $k = 0$.

Step 1:
Compute $M^k_{blocking}$.

Step 2:
If $M^k_{blocking} \cap (2^{O \cup T})^* \neq \emptyset$: **return** *"$L_{nbsafe}$ does not exist"*.
If $M^k_{blocking} = \emptyset$: **return** M^k.

Step 3:
Compute $M^k_{badchoice}$.
$M^{k+1} = M^k \setminus post(M^k_{badchoice})$.
Set $k = k + 1$.
Goto Step 1.

Starting with $M^0 = L_{safe}$ the algorithm iteratively deletes blocking words by cutting them at the last possible inputs (and by additionally cutting all undistinguishable words). This is done until either no new blocking words are produced (in which case L_{nbsafe} is found) or an L_c is blocked w.r.t. the actually computed language (in which

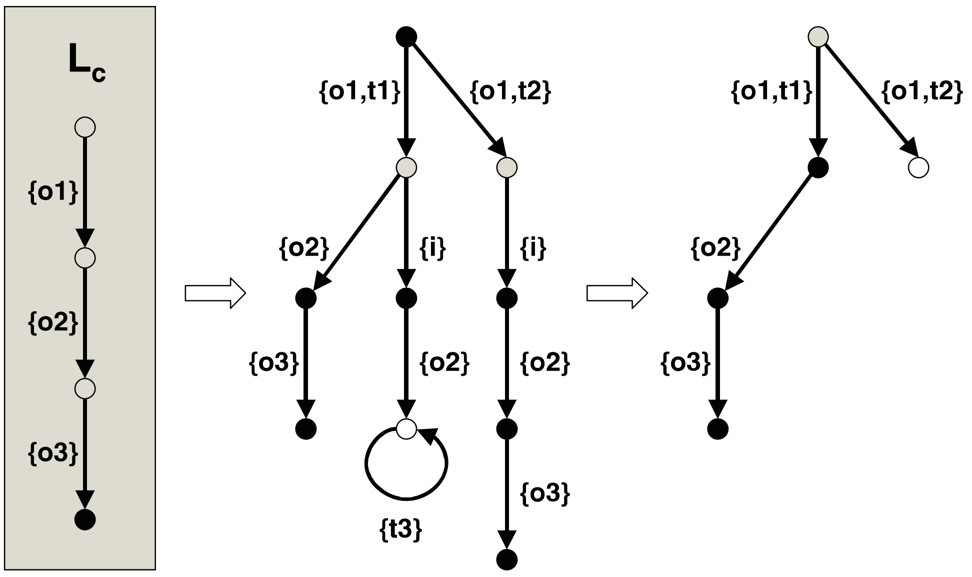


Fig. 11. The black states in the automaton representing L_c are the accepting states. The white states in the other automata (representing languages computed in the nonblocking algorithm) are blocking w.r.t. the given L_c. In the grey states there is the last possibility not to send an input in order to avoid the blocking situation. The input must be avoided at all undistinguishable states. This can cause new blocking situations, which can be even not avoidable.

case no controllable nonblocking language exists). All computed languages M^k are controllable and normal, but possibly not nonblocking. Observe that if $M^k_{blocking} \cap (2^{O\cup T})^* = \emptyset$, then each word in $M^k_{blocking}$ has a nonempty prefix in $M^k_{badchoice}$ and therefore does not belong to M^{k+1}.

Before stating the main result, namely that this algorithm returns L_{nbsafe} if and only if a maximally permissive nonblocking controllable sublanguage exists, we have to verify, that the algorithm allways terminates. For completeness we will give a sketch of the proof below. A detailed proof can be found for example in [3]: the algorithm presented there only slightly differs from ours. The following procedure is repeated: *It first iteratively deletes blocking words by cutting them at the last possible inputs, without additionally cutting all undistinguishable words. This is also done until no new blocking word is found. The resulting language is controllable and nonblocking, but not normal. Then all cuts done so far are also realized for all undistinguishable words, which yields a controllable and normal language, which is possibly not nonblocking*. The whole procedure in repeated until the resulting language is nonblocking. Both algorithms have the same output.

The main idea for showing the termination is to find a deterministic finite automaton $G = (S, (2^{I\cup O\cup T})^*, \delta, F, s_0)$ recognizing L_{safe}, such that *deleting words* from $post(M^k_{badchoice})$ (in the algorithm) corresponds to *deleting edges* in G. A nessecary *and* sufficient condition for this is that the states of G distinguish words in $M^k_{badchoice}$ from words not in $M^k_{badchoice}$, i.e. (see also Figure 13)

$$\delta(s_0, w) = \delta(s_0, v) \Rightarrow (w \in M^k_{badchoice} \Leftrightarrow v \in M^k_{badchoice}). \qquad (6)$$

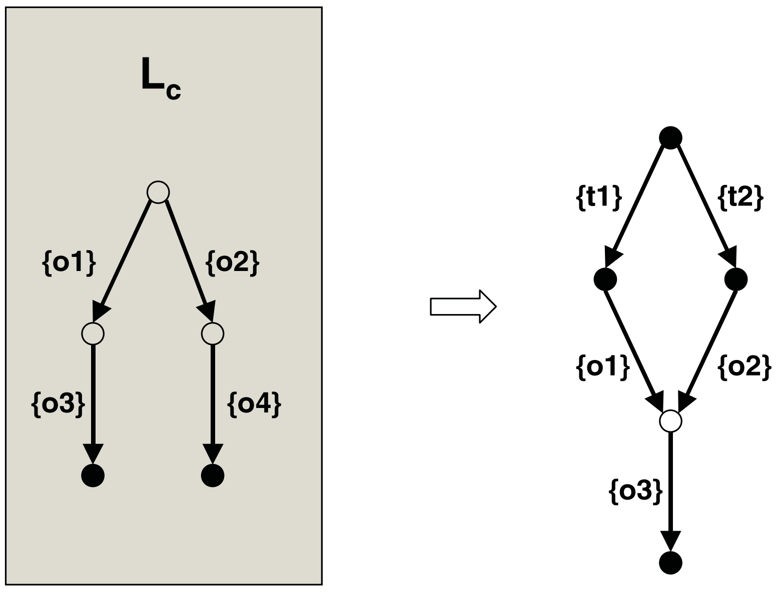


Fig. 12. In general two words $v = \{t_2\}\{o_2\} \notin M^k_{blocking}$ and $w = \{t_1\}\{o_1\} \in M^k_{blocking}$ can have the same follower state in an automaton A recognizing L_{safe}. For the automaton G implementing the nonblocking algorithm however we require that one can distinguish between "*blocking*-states" and "*not blocking*-states". Such states can be splitted appropriately by synchronizing A with an automaton recognizing $\lambda^{-1}_{T \cup I}(L_c)$.

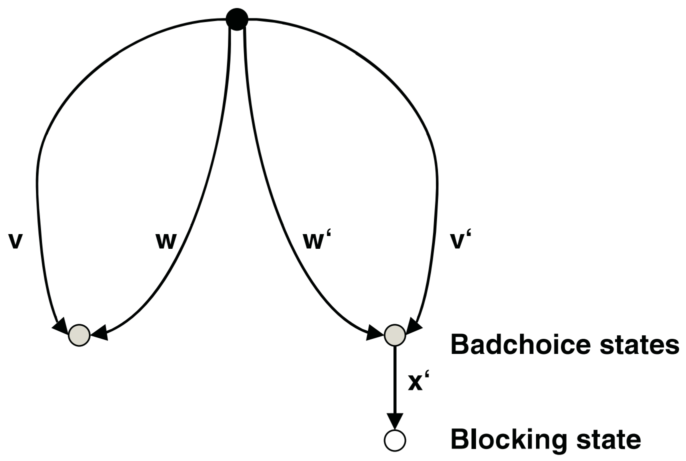


Fig. 13. In general two words $v \notin M^k_{badchoice}$ and $w \in M^k_{badchoice}$ can have the same follower state in an automaton A recognizing L_{safe}. For the automaton G implementing the nonblocking algorithm however we require that one can distinguish between "*badchoice*-states" and "*not badchoice*-states": If there is a w' undistinguishable from w leading to a blocking state, there must also be a word v' undistinguishable from v leading to a blocking state. G is even choosen such that v' and w' have the same follower state.

Taking $v, w \in L_{safe}$ with $\delta(s_0, w) = \delta(s_0, v)$ and $w \in M^k_{badchoice}$, we know that there is a $w' \in M^k_{badchoice}$ with $\lambda_T(w) = \lambda_T(w')$ and there is a $x' \in (2^{OUT})^*$ with $w'x' \in M^k_{blocking}$. To prove (6) it suffices to find $v' \in M^k_{badchoice}$ with $\lambda_T(v) = \lambda_T(v')$ such that $\delta(s_0, v') = \delta(s_0, v)$ and $v'x' \in M^k_{blocking}$ (Figure 13). In other words, the possible futures of words undistinguishable from v and the possible futures of words undistinguishable from w should be the same.

In general a finite automaton A recognizing L_{safe} does not fulfill this property, i.e. does not distinguish by its states words, for which the possible futures of their undistinguishable words are not the same. Such states have to be split appropriately by synchronizing A with another automaton B. B can be constructed from A in such a way that a state $[w]_B$ of B is defined as the set of exactly those states of A which are follower states of words undistinguishable from w:

$$s \in [w]_B \Rightarrow \exists w' \in L_{safe} : \lambda_T(w) = \lambda_T(w') \wedge s = [w']_A.$$

Formally B can be constructed from A in three steps. For this let $(S_A, \Sigma_A, (2^{I \cup O \cup T}))$ be the labelled transition system associated to A:

(1) Replace each edge $s \xrightarrow{x} s' \in \Sigma_A$ by $s \xrightarrow{\lambda_T(x)} s'$ (this yields in particular $s \xrightarrow{\epsilon} s'$ for $x \subseteq T$). The result is a so called nondeterministic ϵ-automaton. By definition of such automata ([11]) the possible follower states of a word $z \in (2^{I \cup O})^*$ in this automaton are exactly the follower states of words $w \in L_{safe}$ in A with $\lambda_T(w) = z$.
(2) Compute the deterministic finite automaton simulating the nondeterministic ϵ-automaton from (1) by the well-known subset construction ([11]): Then exactly the sets of possible follower states of a word $z \in (2^{I \cup O})^*$ in the above ϵ-automaton define the states of this deterministic automaton.
(3) Pump the automaton from (2) by steps of unobservable transitions from 2^T in the following way: For all states s and all $x \subseteq T$ (loop) transitions $s \xrightarrow{x} s$ are added. For all transition $s \xrightarrow{y} s'$ with $y \subseteq I \cup O$ and forall $x \subseteq T$ transitions $s \xrightarrow{x \cup y} s'$ are added.

Finally we have to require the automaton A to distinguish words in $M^k_{blocking}$ from words not in $M^k_{blocking}$ by its states (which is not the case in general, see Figure 12):

$$\forall w, \forall v \in [w]_A : w \in M^k_{blocking} \Leftrightarrow v \in M^k_{blocking}, \tag{7}$$

This can be achieved by building A as the synchronized product of the minimal automata recognizing L_{safe} and $\lambda^{-1}_{I \cup T}(L_c)$. Then A fulfills

$$\forall v \in [w]_A : \lambda_{I \cup T}(w) \in L_c \Leftrightarrow \lambda_{I \cup T}(v) \in L_c. \tag{8}$$

It can be seen as follows, that then property (7) is also satisfied: Take $v \in [w]_A$, $v \neq w$. Assume $w \notin M^k_{blocking}$. There is a $x \in (2^{I \cup O \cup T})^*$, such that $wx \in M^k$ and $\lambda_{I \cup T}(wx) \in L_c$. Since $vx \in [wx]_A$ it follows from property (8), that $\lambda_{I \cup T}(vx) \in L_c$, i.e. $v \notin M^k_{blocking}$.
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 returns a language L_{nbsafe}. In this case L_{nbsafe} is this searched sublanguage.*

Proof. Let $L_{safe} = M^0, \ldots, M^{N_0}$ be the sequence of languages 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}. We will show this by contradiction: Assume there is a controllable language $M \subseteq L_{safe}$ with $M_{blocking} = \emptyset$. Observe that

$$M \cap M^0_{blocking} \subseteq M_{blocking},$$

i.e. the assumption in particular implies

$$M \cap M^0_{blocking} = \emptyset. \tag{9}$$

By **Step 1** M^{N_0} fulfills

$$\exists v_0 \in M^{N_0}_{blocking} \cap (2^{O \cup T})^*.$$

From the controllability of M (condition (1)) we deduce $v_0 \in M$. From (9) it follows $v_0 \notin M^0_{blocking}$. That means v_0 can be extended by some word $y \neq \epsilon$ (remark that $\lambda_{T \cup I}(v_0) \notin L_c$!) to a word $v_0 y \in M^0$ with $\lambda_{I \cup T}(v_0 y) \in L_c$. By the assumption one of these extensions $v_0 y_0$ must be in M:

$$v_0 y_0 \in M \quad \text{and} \quad \lambda_{I \cup T}(v_0 y_0) \in L_c.$$

Since $v_0 \in M^{N_0}_{blocking}$, we have moreover $v_0 y_0 \notin M^{N_0}$. By construction (Step 2) there must be an index $N_1 < N_0$, such that

$$v_0 y_0 \in post(M^{N_1}_{badchoice}).$$

Let $v_0 x i \in M^{N_1}_{badchoice}$ for a prefix xi of y_0 with $i \cap I \neq \emptyset$. By definition of $M^{N_1}_{badchoice}$ there is a $v_1 = v_0' x' i' y \in M^{N_1}_{blocking}$ with

(a) $\lambda_T(v_0' x') = \lambda_T(v_0 x)$,
(b) $i' \cap I = i \cap I$, and
(c) $y \in (2^{O \cup T})^*$.

Remember now that all prefixes of $v_0 y_0$, in particular $v_0 x$ and $v_0 x i$, belong to M. Since M is assumed to be controllable, M contains all words in $L_{\mathcal{P}} \cap [v_0 x]_T$ (lemma 2). In particular $v_0' x' \in M$ (property (a)). From the condition (2) (of controllability) and (b) we get further $v_0' x' i' \in M$, and therefore $v_1 = v_0' x' i' y \in M$ (condition (1) (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 $v_0, v_1, \ldots \in M$, such that $v_i \in M^{N_i}_{blocking}$, $i = 0, 1, \ldots$. Finally $N_k = 0$ for some k, which implies $v_k \in M_{blocking}$, what contradicts our assumption.

Next we consider the “if”-part:
By construction $M^{N_0} = L_{nbsafe}$ is controllable and nonblocking. It remains to show that it is maximally permissive with these two properties. We show this statement by contradiction. Assume another language M to be controllable and nonblocking controllable satisfying $L_{nbsafe} \subsetneq M \subseteq L_{safe}$. In particular, by assumption $M_{blocking} = \emptyset$.

There is a $x \in M \setminus M^{N_0}$. As M^{N_0} and M are prefix closed we can assume (without loss of generality) that x is of the form

$$x = wj \in M,\ w \in M^{N_0},\ j \cap I \neq \emptyset.$$

Since $wj \notin M^{N_0}$, for some step $N_1 < N_0$ it holds $wj \in M^{N_1}_{badchoice}$. By definition of $M^{N_1}_{badchoice}$ there is a $v_0 = w'j'y \in M^{N_1}_{blocking}$ with

(a) $\lambda_T(w') = \lambda_T(w)$,
(b) $j' \cap I = j \cap I$, and
(c) $y \in (2^{O \cup T})^*$.

As above, since M is assumed to be controllable, we follow $v_0 \in M$. By assumption, as in the 'only if'-part, $v_0 \notin M^0_{blocking}$. Therefore v_0 must have an extension within M to a word respecting L_c. Let v_0y_0 be this extension of v_0. Proceed now as in the "only if"-part. □

4.2 Synthesis of Control Modules

In this subsection we show how to synthesize a control module $\mathcal{C}$ from a given behavior $L_{cb} \subseteq L_{\mathcal{P}}$ of $\mathcal{P}$ and to compose this module with $\mathcal{P}$, such that the resulting composed module has exactly this behavior up to transitions of $\mathcal{C}$ which are not in $I \cup O$, whenever possible. Of course, as a first necessary condition, we have to require L_{cb} to be a prefix closed regular language, since the set of occurrence sequences of a module has this property. Formally a control module $\mathcal{C}$ w.r.t. such a language L_{cb} is defined as follows:

Definition 21. *Let $\mathcal{C}$ be a module of a signal net with the set of transitions $T_{\mathcal{C}}$ and denote $U = T_{\mathcal{C}} \setminus (I \cup O)$. Then $\mathcal{C}$ is the* control module *of $\mathcal{P}$ w.r.t. L_{cb}, if there is a composition mapping Ω, such that the set of all occurrence sequences $L_{\mathcal{CP}}$ of the module $\mathcal{C} *_{\Omega} \mathcal{P}$ satisfies $\lambda_U(L_{\mathcal{CP}}) = L_{cb}$.*

We claim that for the existence of such a control module it is sufficient to require L_{cb} to be *controllable* (see definition 13). This gives the main theorem of this subsection:

Theorem 3. *If $L_{cb} \subseteq L_{\mathcal{P}}$ is a regular, controllable, prefix closed language, then there is a control module $\mathcal{C}$ of $\mathcal{P}$ w.r.t. L_{cb}.*

In practice this statement can be applied to $L_{cb} = L_{safe}$ or $L_{cb} = L_{nbsafe}$. We prove the theorem by constructing $\mathcal{C}$. The main idea is to synthesize $\mathcal{C}$ by adding new net structure to $\mathcal{E}$ (see Figure 7). In particular $\mathcal{C}$ is composed with $\mathcal{P}$ via the connections (given by Ω) between $\mathcal{P}$ and $\mathcal{E}$.

For the construction we use a deterministic finite automaton $A = (S, 2^{I \cup O}, \delta, F, s_0)$ recognizing $\lambda_T(L_{cb})$. We denote $\Sigma = \{s \xrightarrow{x} s' \mid \delta(s,x) = s'\}$ the set of edges of A and $l : \Sigma \to 2^{I \cup O}$, $l(s \xrightarrow{x} s') = x$, the labelling of Σ.

Remember that L_{cb} is prefix closed. Without loss of generality we assume that

(a) All states are accepting states: $F = S$. Just omit all edges leading to non-accepting states.

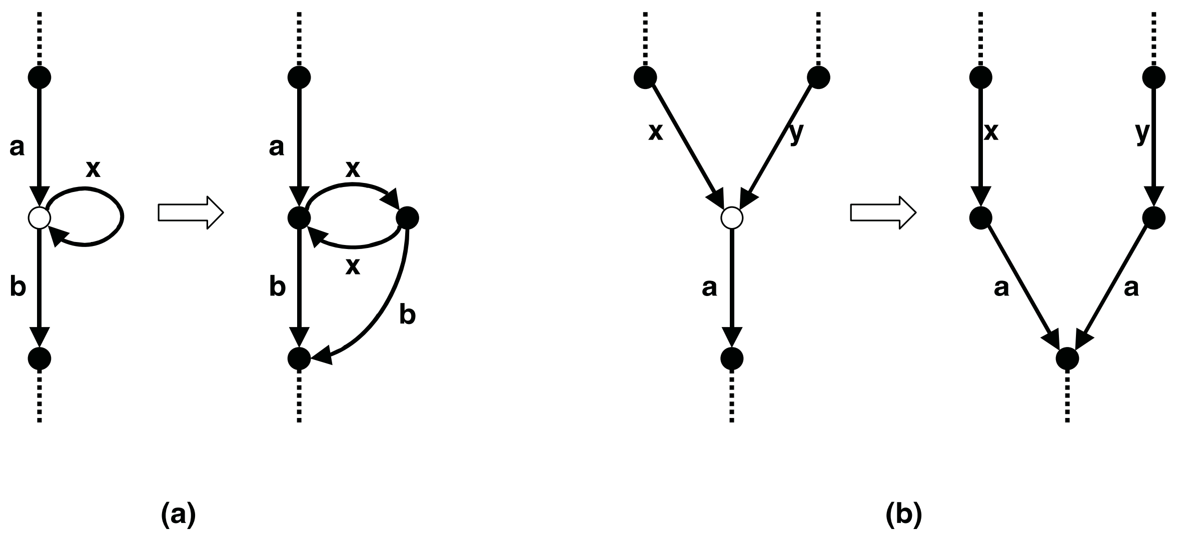


Fig. 14. Part (a): splitting states to avoid loops. Part (b): splitting states to distinguish words according to their last character.

(b) There are no loops in A: $s \xrightarrow{x} s \notin \Sigma$. If this is not the case for a state of A you can think of splitting this state into two copies and thus transforming the loop into a cycle of length 2 (see Figure 14, (a)).

(c) The states of A distinguish words according to their last character: $s' \xrightarrow{x} s, s'' \xrightarrow{y} s \in \Sigma \Rightarrow x = y$. As long as this is not the case for a state s of A, i.e. $x \neq y$, you can think of splitting s into two copies, one for words ending with x and one for words ending with y (see Figure 14, (b)). Hence we get $l(s') = l(s \xrightarrow{x} s') = x$ for $s \xrightarrow{x} s' \in \Sigma$.

Formally $(b), (c)$ can be achieved by synchronizing A with appropriate other finite automata.

We will construct a signal net $N = (P, U \cup I \cup O, F, EN, CN, m_0)$, where P is the set of places, $U \cup I \cup O$ is the set of transitions, F is the flow arc relation, EN is the event arc relation, CN is the context arc relation, and m_0 is the initial marking. N together with input/output structure of $\mathcal{E}$, will give the searched module $\mathcal{C}$. For simplicity we will use two kinds of context arcs: Usual positive context arcs, called *condition arcs* in the context of signal nets, which test places for presence of tokens, and negative context arcs, also called *inhibitor arcs* in literature ([12]), which test places for absence of tokens. It is well known that in elementary nets negative context arcs can be equivalently replaced by a structure using positive context arcs and so-called *co-places* ([12], see Figure 15). So CN splits into the set of condition arcs CN^+ and inhibitor arcs CN^-. We modify some notions for the enabling of steps w.r.t. CN^-: For a transition t we denote $^+t = \{p \mid (p,t) \in CN^+\}$ and $^-t = \{p \mid (p,t) \in CN^-\}$. Given a set $\xi \subseteq T$ of transitions, we extend the above notions to: $^+\xi = \bigcup_{t \in \xi} {}^+t$ and $^-\xi = \bigcup_{t \in \xi} {}^-t$.

Definition 22 (Potentially enabled for negative context). *A step ξ is* potentially enabled *in a marking m if*

- *All $t \in \xi$ are enabled: $^\bullet t \cup {}^+t \subseteq m$, $^-t \cap m = \emptyset$ and $(t^\bullet \setminus {}^\bullet\xi) \cap m = \emptyset$ and*
- *All pairs $t, t' \in \xi$ are not in conflict: $^\bullet t \cap {}^\bullet t' = t^\bullet \cap (t')^\bullet = \emptyset$.*

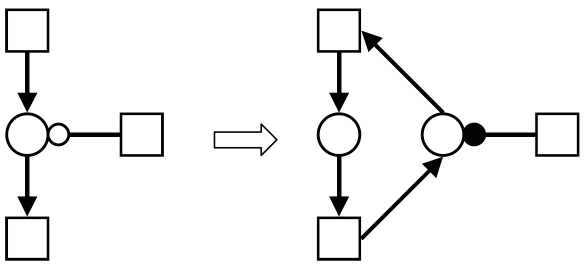


Fig. 15. Introducing coplaces to translate inhibitor arcs into condition arcs.

We set

- $P = \{p_s \mid s \in S\}$.
- $U_{OR} = \{t_s^{empty} \mid s \in S\} \cup \{t_s^{fill} \mid s \in S\} \cup \{t_{s,i} \mid \exists s \xrightarrow{x} s' \in \Sigma : \ i \in x \cap I\}$.
- $U_{AND} = \{t_{s \xrightarrow{x} s'} \mid s \xrightarrow{x} s' \in \Sigma\}$.

Our aim is to identify each state $s \in S$ of A with a unique set of places $P_s \subseteq P$, and each edge $s \xrightarrow{x} s' \in \Sigma$ of A with a unique set of transitions $\xi_{s \xrightarrow{x} s'} \subseteq U \cup I \cup O$, such that

- $\xi_{s \xrightarrow{x} s'}$ is enabled if and only if exactly the places in P_s are marked,
- $\xi_{s \xrightarrow{x} s'}$ together with an appropriate set of transitions of T build a step in $\mathcal{C} *_\Omega \mathcal{P}$, and
- ${}^\bullet\xi_{s \xrightarrow{x} s'} = P_s$ and $\xi^\bullet_{s \xrightarrow{x} s'} = P_{s'}$.

The idea is the following: Assume that $\mathcal{C}$ is in the state P_s. Then for each $s \xrightarrow{x} s' \in \Sigma$ it should be possible to send the input $i \in x \cap I$ (if there is one) to the plant synchronizing a step in the plant (in the case of event inputs) or switching a condition input on/off. In the first case this step in the plant should synchronize the step of outputs $x \cap O$ sent from the plant. If there is no input $i \in x \cap I$, there should be a step in the plant synchronizing the step of outputs $x \cap O$ sent from the plant. Let x be such a set, and let $i \in x \cap I$. Since there are in general also states in which i is not allowed to be sent to the plant, we model the transition $t_{s,i}$ in such a way that

- $t_{s,i}$ is enabled exactly under the marking P_s via condition and inhibitor arcs (Figures 18 and 20), and
- $t_{s,i}$ synchronizes the transition i (Figure 18).

A transition $t_{s \xrightarrow{x} s'}$ is intended to simulate the step of signals x in the control module $\mathcal{C}$, if $\mathcal{C}$ is in state P_s. Therefore $t_{s \xrightarrow{x} s'}$

- is enabled exactly under the marking P_s via condition and inhibitor arcs (Figures 16 and 20),
- synchronizes the transition t_s^{empty} which is intended to empty exactly the places in P_s (Figure 16),
- synchronizes all transitions $t_{s''}^{fill}$, which are intended to mark the places $p_{s''}$ in the follower marking $P_{s'}$ (Figure 16), and
- is synchronized by all outputs in $x \cap O$ and by $t_{s,i}$ for $i \in x \cap I$ (Figures 17 and 18).

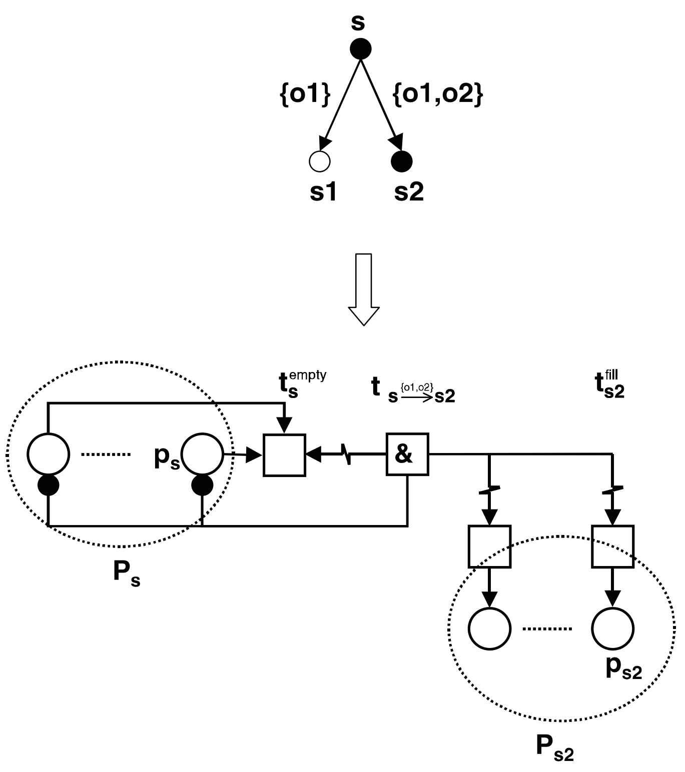


Fig. 16. For clearness we model only the transition $s \overset{\{o_1,o_2\}}{\longrightarrow} s_2$ of the above behavior (the black states). The transition $t_{s\overset{\{o_1,o_2\}}{\longrightarrow}s_2}$ simulates the edge $s \overset{\{o_1,o_2\}}{\longrightarrow} s_2$. $t_{s\overset{\{o_1,o_2\}}{\longrightarrow}s_2}$ synchronizes t_s^{empty} to empty the marking P_s, and synchronizes exactly the transitions t_p^{fill} with $p \in P_{s_2}$ (observe in particular $p_s \in P_s$ and $p_{s_2} \in P_{s_2}$). Moreover $t_{s\overset{\{o_1,o_2\}}{\longrightarrow}s_2}$ tests the places in P_s to be marked. The event arc connections to input and output transitions are omitted (see Figure 17).

The second part of the last condition is necessary, since in general in the same state the same step of outputs can occur spontaneous or can be initiated by an input. Observe that, if $\mathcal{C}$ is in state P_s, a step of outputs $x \cap O$ synchronizes beside the transition $t_{s\overset{x}{\rightarrow}s'}$ also each transition $t_{s\overset{y}{\rightarrow}s''}$ with $y \subset x$. See figure 19.

We are now able to define the sets $\xi_{s\overset{x}{\rightarrow}s'}$ and $P_{s'}$:

- From the event arc relation we deduce (Figure 19) $\xi_{s\overset{x}{\rightarrow}s'} = \{t_{s,i} \mid i \in x \cap I\} \cup x \cup \{t_{s\overset{y}{\rightarrow}s''} \mid y \subseteq x\} \cup \{t_s^{empty}\} \cup \{t_{s''}^{fill} \mid p_{s''} \in P_{s'}\}$.
- Because $t_{s\overset{x}{\rightarrow}s'}$ synchronizes $t_{s'}^{fill}$) the place $p_{s'}$ belongs to $P_{s'}$. For every $s \in S$ with $s \overset{x}{\rightarrow} s' \in \Sigma$ the step of inputs and outputs x also synchronizes beside the transition $t_{s\overset{x}{\rightarrow}s'}$ each transition $t_{s\overset{y}{\rightarrow}s''}$ with $y \subseteq x$. Therefore we get in such cases $P_{s''} \subseteq P_{s'}$. This procedure has to be applied recursively. Therefore we define $P_{s'}$ to be the smallest set satisfying (Figure 20)
  (i) $p_{s'} \in P_{s'}$.
  (ii) $\forall s \overset{x}{\rightarrow} s', s \overset{y}{\rightarrow} s'' \in \Sigma$ $(s' \neq s'')$ with $y \subset x$: $P_{s''} \subseteq P_{s'}$.

Altogether we get formally:

- $F = \{(\mathbf{p}, \mathbf{t_s^{empty}}) \mid s \in S, p \in P_s\} \cup \{(\mathbf{t_s^{fill}}, \mathbf{p_s}) \mid s \in S\}$,
- $CN^+ = \{(\mathbf{p}, \mathbf{t_{s\overset{x}{\rightarrow}s'}}) \mid s \in S, p \in P_s\} \cup \{(\mathbf{p}, \mathbf{t_{s,i}}) \mid s \in S, p \in P_s\}$,

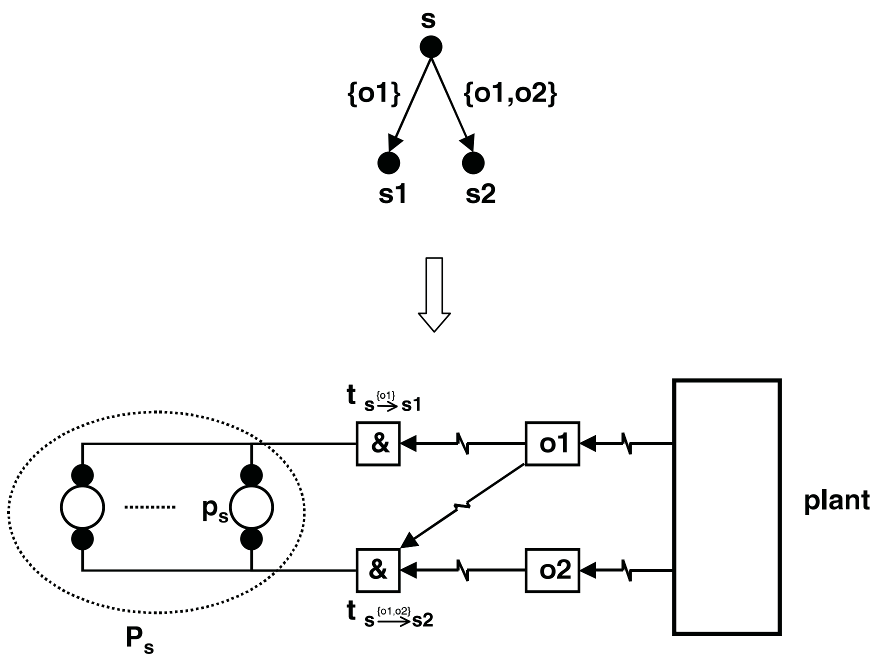

s
{o1}
{o1,o2}
s1
s2
$t_{s\xrightarrow{\{o1\}}s1}$
&
o1
plant
p_s
&
o2
$t_{s\xrightarrow{\{o1,o2\}}s2}$
P_s

Fig. 17. The transition $t_{s \xrightarrow{\{o_1,o_2\}} s_2}$ simulates the edge $s \xrightarrow{\{o_1,o_2\}} s_2$. It is therefore synchronized by the output transitions o_1 and o_2. The occurrence of o_1 together with o_2 synchronizes also $t_{s \xrightarrow{\{o_1\}} s_1}$, since $t_{s \xrightarrow{\{o_1\}} s_1}$ is also enabled under P_s. The occurrence of o_1 without o_2 only synchronizes $t_{s \xrightarrow{\{o_1\}} s_1}$. For clearness the connections to empty- and fill-transitions and places in the follower marking are omitted (see Figure 16), but observe that this implies $p_{s_1} \in P_{s_1} \subset P_{s_2}$. (see also Figure 20).

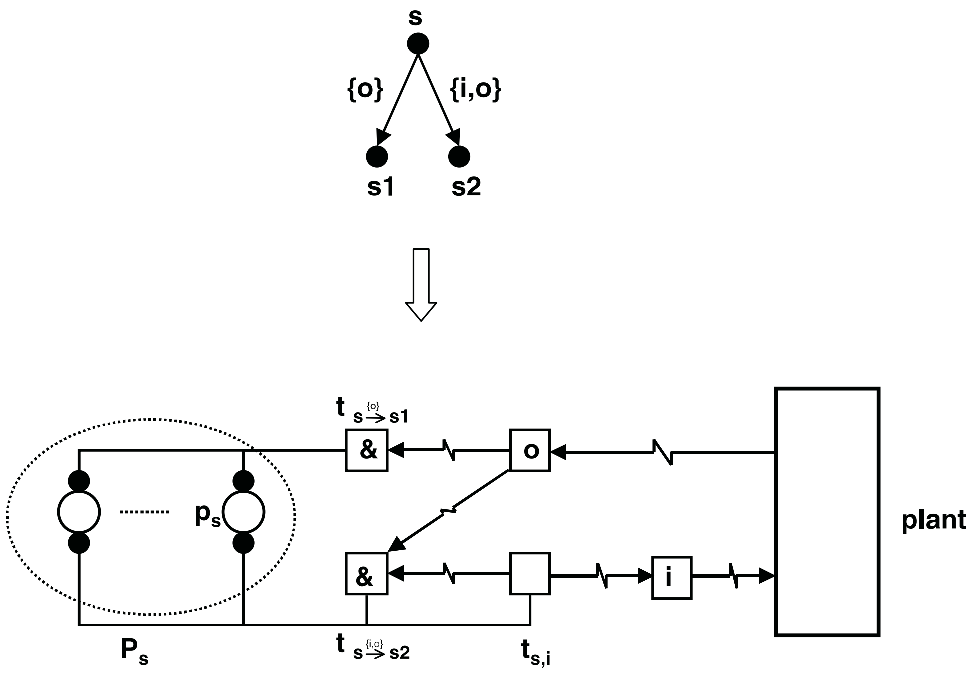
- $CN^- = \{(\mathbf{p}, \mathbf{t}_{\mathbf{s}\xrightarrow{\mathbf{x}}\mathbf{s'}}) \mid \exists \overline{s} \in S : \; P_s \subset P_{\overline{s}} \wedge p \in P_{\overline{s}} \setminus P_s\} \cup \{(\mathbf{p}, \mathbf{t}_{\mathbf{s},\mathbf{i}}) \mid \exists \overline{s} \in S : P_s \subset P_{\overline{s}} \wedge p \in P_{\overline{s}} \setminus P_s\}$, and
- $EN = \{(\mathbf{t}_{\mathbf{s},\mathbf{i}}, \mathbf{i}) \mid s \in S, \; i \in I\} \cup \{(\mathbf{t}_{\mathbf{s},\mathbf{i}}, \mathbf{t}_{\mathbf{s}\xrightarrow{\mathbf{x}}\mathbf{s'}}) \mid s \in S, \; i \in x \cap I\} \cup \{(\mathbf{o}, \mathbf{t}_{\mathbf{s}\xrightarrow{\mathbf{x}}\mathbf{s'}}) \mid s \in S, \; o \in x \cap O\} \cup \{(\mathbf{t}_{\mathbf{s}\xrightarrow{\mathbf{x}}\mathbf{s'}}, \mathbf{t}^{\mathbf{fill}}_{\mathbf{s''}}) \mid s \xrightarrow{x} s' \in \Sigma, \; p_{s''} \in P_{s'}\} \cup \{(\mathbf{t}_{\mathbf{s}\xrightarrow{\mathbf{x}}\mathbf{s'}}, \mathbf{t}^{\mathbf{empty}}_{\mathbf{s}}) \mid s \in S\}$,

Remark 3. Observe that $p_{s'} \in P_s$ implies either $s' = s$ or $l(s') \subset l(s)$. This implies that for all $s, s' \in S$: $p_s \notin P_{s'}$ and/or $p_{s'} \notin P_s$.

Lemma 8. *(a) The mapping $\phi : S \to 2^P$, $\phi(s) = P_s$ is injective.*
(b) $(\xi_{s\xrightarrow{x}s'} \setminus x)^\bullet = P_{s'}$ and ${}^\bullet(\xi_{s\xrightarrow{x}s'} \setminus x) = P_s$.
(c) $\xi_{s\xrightarrow{x}s'} \setminus x$ is potentially enabled in P_s.
(d) $\xi_{s\xrightarrow{x}s'}$ is maximal w.r.t. U and P_s: For each transition $t \in U \setminus \xi_{s\xrightarrow{x}s'}$ with ${}^{\rightsquigarrow}t \cap \xi_{s\xrightarrow{x}s'} \neq \emptyset$ the set $\xi_{s\xrightarrow{x}s'} \cup \{t\}$ is not potentially enabled in P_s.

Proof.
ad (a): see remark 3.
ad (b): Follows from

$$(\xi_{s\xrightarrow{x}s'} \setminus x)^\bullet = \bigcup_{p_{s''} \in P_{s'}} (t^{fill}_{s''})^\bullet \, , \quad {}^\bullet(\xi_{s\xrightarrow{x}s'} \setminus x) = {}^\bullet(t^{empty}_s).$$



s
{o}
{i,o}
s1
s2
$t_{s \xrightarrow{\{o\}} s1}$
&
o
plant
p_s
&
i
P_s
$t_{s \xrightarrow{\{i,o\}} s2}$
$t_{s,i}$

Fig. 18. Here the label of the edge $s \overset{\{i,o\}}{\longrightarrow} s_2$ contains an input transition i. This input transition is synchronized by the spontaneous transition $t_{s,i}$, which is exactly enabled under P_s. If i synchronizes the output transition o via transitions in the plant, the transition $t_{s \overset{\{i,o\}}{\longrightarrow} s_2}$ is synchronized together with the transition $t_{s \overset{\{o\}}{\longrightarrow} s_1}$ (analogously to Figure 17). It is also possible that the plant sends the output o without the input i. In this case $t_{s \overset{\{o\}}{\longrightarrow} s_1}$ is synchronized alone. Observe that no cycles of event arcs are produced.

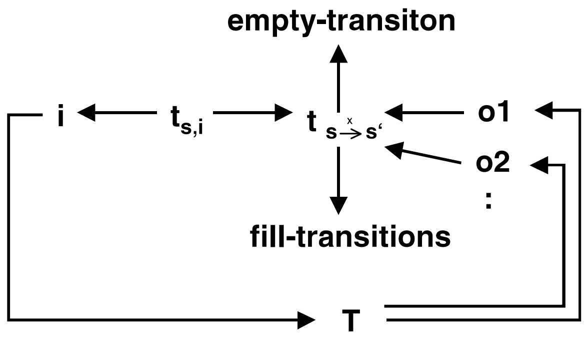


Fig. 19. The event arc relation w.r.t. an edge $s \overset{\{x\}}{\longrightarrow} s' \in E$. The event arc from i to T is optional: it exists only, if i replaces an event input signal, and not in the case i is an on- or off-transition of an condition imput signal.

ad (c): By definition of F, CN^- and CN^+, the only sets of places of the form $^\bullet t$, $t^\bullet$ or ^+t for $t \in \xi_{s \xrightarrow{x} s'} \setminus x$ which are not empty are:

$$^+t_{s \xrightarrow{x} s'} = {}^+t_{s,i} = P_s, \quad {}^\bullet t_s^{empty} = P_s, \quad (t_{s''}^{fill})^\bullet = \{p_{s''}\}.$$

This gives $^\bullet t \cup {}^+t \subseteq P_s$. Moreover, $^-t \cap P_s = \emptyset$ by definition. Finally, from (b) we get $(t^\bullet \setminus {}^\bullet\xi_{s \xrightarrow{x} s'}) \cap P_s = (t^\bullet \setminus P_s) \cap P_s = \emptyset$. By this the first part of the *potentially enabled* definition 22 is fulfilled.

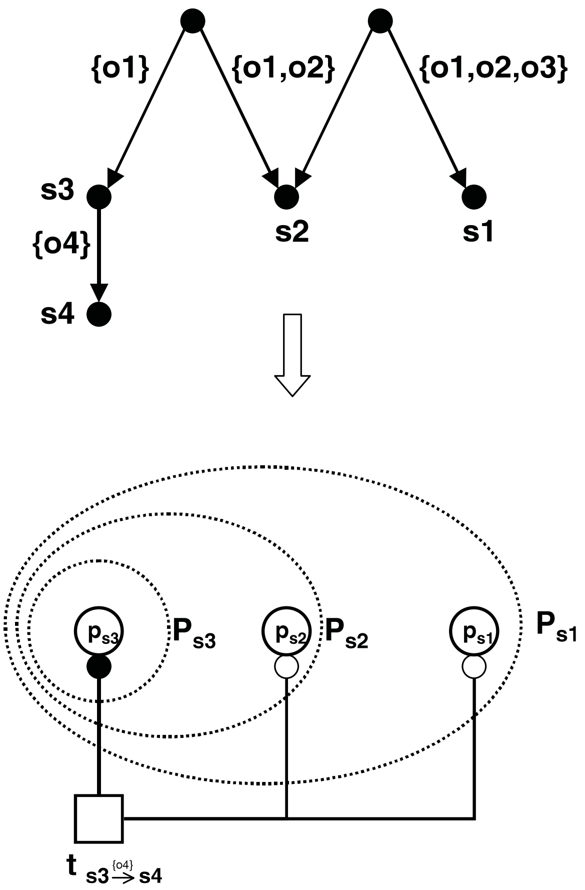


Fig. 20. By definition we get $P_{s_3} \subset P_{s_2} \subset P_{s_1}$. The transition $t_{s_3 \xrightarrow{\{o_4\}} s_4}$ has to test the places in P_{s_3} for the presence of tokens and the places in $P_{s_1} \setminus P_{s_3}$ for the absence of tokens.

It remains to verify that there are no conflicts w.r.t. pre- resp. postsets in $\xi_{s \xrightarrow{x} s'} \setminus x$: The only transition in $\xi_{s \xrightarrow{x} s'} \setminus x$ with nonempty preset is t_s^{empty}. So there are no conflicts w.r.t. presets. The only transitions with nonempty postsets in $\xi_{s \xrightarrow{x} s'} \setminus x$ are of the form $t_{s''}^{fill}$ for $p_{s''} \in P_{s'}$. All postsets of such transitions consist of a unique place, and so are pairwise distinct.

ad (d): Each transition $t \in U \setminus \xi_{s \xrightarrow{x} s'}$ with ${}^{\rightsquigarrow}t \cap \xi_{s \xrightarrow{x} s'} \neq \emptyset$ is of the form (see figure 19) $t = t_{\overline{s} \xrightarrow{x} \overline{s}'}$. There are two cases:

- $\overline{s} \neq s$:
  From $P_s \neq P_{\overline{s}}$ we deduce that t is not enabled. Therefore $\xi_{s \xrightarrow{x} s'} \cup \{t\}$ is not potentially enabled.
- $\overline{s} = s$ and $l(\overline{s}') \not\subseteq l(s')$:
  That means ${}^{\rightsquigarrow}t \not\subseteq \xi_{s \xrightarrow{x} s'}$. Since $t \in U_{AND}$, $\xi_{s \xrightarrow{x} s'} \cup \{t\}$ is not a step. $\square$

We are now going to prove $\lambda_U(L_{\mathcal{CP}}) = L_{cb}$. We need some additional notions: For an occurrence sequence $w = x_1 \dots x_n$ of $\mathcal{E} *_\Omega \mathcal{P}$ we denote

- $w_i = x_1 \dots x_i$,
- m_i the marking of $\mathcal{E} *_\Omega \mathcal{P}$ after the occurrence of w_i,
- $s_i = \delta(s_0, \lambda_T(w_i))$ the state in A after executing $\lambda_T(w_i)$.

Observe $s_\epsilon = s_0$ and $x_i \cap (I \cup O) = \emptyset \Leftrightarrow s_{i-1} = s_i$. For $x_i \cap (I \cup O) \neq \emptyset$ we get $l(s_i) = \lambda_T(x_i)$. Define

$$\eta_i = \begin{cases} x_i \cup \xi_{s_{i-1} \xrightarrow{x} s_i} & \text{if } x = x_i \cap (I \cup O) \neq \emptyset, \\ x_i & \text{else.} \end{cases}$$

For a set of transitions σ and a marking m of $\mathcal{C} *_\Omega \mathcal{P}$ we denote

- $\sigma^{\mathcal{C}} = \lambda_T(\sigma)$ and $m^{\mathcal{C}} = m \cap P$ the $\mathcal{C}$-parts, and
- $\sigma^{\mathcal{EP}} = \lambda_U(\sigma)$ and $m^{\mathcal{EP}} = m \setminus P$ the $\mathcal{E} *_\Omega \mathcal{P}$-parts.

Observe that $\eta_i^{\mathcal{EP}} = x_i$, $\eta_i^{\mathcal{C}} = \xi_{s_{i-1} \xrightarrow{x} s_i}$ resp. $= \emptyset$, $(m_i \cup P_{s_i})^{\mathcal{C}} = P_{s_i}$ and $(m_i \cup P_{s_i})^{\mathcal{EP}} = m_i$. Between the net structure in module $\mathcal{E} *_\Omega \mathcal{P}$ and the additional net structure in $\mathcal{C} *_\Omega \mathcal{P}$ there are only event arc connections. There are no events arc connections between transitions in T and transitions in U. Therefore:

- σ is potentially enabled in $\mathcal{C} *_\Omega \mathcal{P}$ under the marking m *if and only if* both $\sigma^{\mathcal{C}}$ is potentially enabled in $\mathcal{C}$ under the marking $m^{\mathcal{C}}$ and $\sigma^{\mathcal{EP}}$ is potentially enabled in $\mathcal{E} *_\Omega \mathcal{P}$ under the marking $m^{\mathcal{EP}}$
- σ is maximal w.r.t. $T \cup U$ and m in the sense of lemma 8 (d) *if and only if* both $\sigma^{\mathcal{C}}$ is maximal w.r.t. U and $m^{\mathcal{C}}$ and $\sigma^{\mathcal{EP}}$ is maximal w.r.t. T and $m^{\mathcal{EP}}$ in the sense of lemma 8 (d).

Putting this together, observe that σ is an enabled step in $\mathcal{C} *_\Omega \mathcal{P}$ under the marking m *if and only if* σ is a step and $\sigma^{\mathcal{C}}$ and $\sigma^{\mathcal{EP}}$ are potentially enabled and maximal as above.

Lemma 9. $\lambda_U(L_{\mathcal{CP}}) \supseteq L_{cb}$.

Proof. We show by induction on the length of $w = x_1 \dots x_n \in L_{cb}$:

(A) The occurrence sequence $\eta = \eta_1 \dots \eta_n$ is enabled in $\mathcal{C} *_\Omega \mathcal{P}$ under the marking $m_0 \cup P_{s_0}$.
(B) The occurrence of η gives the follower marking $m_n \cup P_{s_n}$.
(C) $x_1 \dots x_n x_{n+1} \in L_{cb}$ implies that the step η_{n+1} is enabled in $\mathcal{C} *_\Omega \mathcal{P}$ under the marking $m_n \cup P_{s_n}$.

First let $w = \epsilon$ ($n = 0$): (A) and (B) are clear. Ad (C): Observe that in each case the $\mathcal{C}$- and $\mathcal{EP}$-parts of η_1 are potentially enabled and maximal in the above sense. According to the above considerations, it remains to show that η_1 is a step. We distinguish three cases:

(i) $x_1 \subseteq T$:
$\eta_1 = x_1$ is clearly a step.
(ii) $x_1 \subseteq T \cup O$ and $x_1 \cap O \neq \emptyset$:
x_1 builds a step in $\mathcal{E} *_\Omega \mathcal{P}$ which includes the set of transitions $x = l(s_1)$. From the transitions in x the transition $t_{s_0 \xrightarrow{x} s_1}$ is synchronized. The transition $t_{s_0 \xrightarrow{x} s_1}$ synchronizes the empty and fill transitions of $\xi_{s_0 \xrightarrow{x} s_1}$ (see figure 19).
(iii) $x_1 \cap I \neq \emptyset$:
analogously to (ii).

Let $w = x_1 \dots x_{n-1}x_n$. By assumption the statements $(A) - (C)$ are valid for w_{n-1}. This implies (A) for w. Statement (B) is clear (since the markings are unions of the $\mathcal{C}$- and $\mathcal{EP}$-parts). (C) can be seen analogously to the above argumentation.

Lemma 10. $\lambda_U(L_{\mathcal{CP}}) \subseteq L_{cb}$.

Proof. To see the statement, the controllability of L_{cb} will play the crucial role: Observe that $\lambda_U(L_{CP}) \subseteq L_{\mathcal{P}}$. Assume there is $\sigma = \sigma_1 \dots \sigma_n \sigma_{n+1} \in \lambda_U(L_{CP}) \setminus L_{cb}$. Without loss of generality $\sigma_1 \dots \sigma_n \in L_{cb}$. Denote $s = \delta(s_0, \lambda_T(\sigma_1 \dots \sigma_n))$ and $s' = \delta(s_0, \lambda_T(\sigma))$.

(i) $\sigma_{n+1} \subseteq T \cup O$:
According to the first condition (1) of controllability it follows $\sigma \in L_{cb}$. A contradiction.

(ii) $\sigma_{n+1} \cap I \neq \emptyset$:
Let $\sigma_{n+1} \cap I = \{i\}$. The sequence $\lambda_T(\sigma_1) \dots \lambda_T(\sigma_n)\lambda_T(\sigma_{n+1}) = \lambda_T(\sigma_1) \dots \lambda_T(\sigma_n) l(s')$ is a path in A from s_0 to s'. That means there is a word $x_1 \dots x_n x_{n+1} \in L_{cb}$ with $\lambda_T(x_i) = \lambda_T(\sigma_i)$ $(i = 1, \dots, n+1)$. In particular we have $\lambda_{T \cup O}(x_{n+1}) = \lambda_O(l(s'))$, since every step contains at most one input. According to the second condition (2) of controllability it follows $\sigma \in L_{cb}$. A contradiction.

□

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 ([21], pp. 15 - 36). To get efficient algorithms one has to restrict the setting in some way, for example by considering only very special kinds of specifications.

As the main result of the paper, we have shown how to synthesize the control module from the behavior of the controlled plant under the paradigm, that outputs of the plant cannot force inputs of the plant via the control module. This paradigm of course (structurally) restricts the class of modules which can be used as control modules. It would be interesting to discuss a generalization of this concept, where the composition of a control module with a plant module is not restricted. That means, in any pair of composed modules both modules can be considered as the control module symmetrically, or even more generally both modules can be considered to control each other. For sake of simplicity, we have restricted the control specification over set of outputs. We are presently working on extension of our methodology for the control specifications including input signals. The methodology for the specifications over observable states (i.e. condition output signals) is also an interesting subject of the further research.

The presented approach considers only Petri nets on a very elementary level. For complex industrial-size systems, these nets tend to be either very large or too abstract. In particular, data and time aspects can not be modelled in a natural way. Therefore, we are working on extension of modules of signal nets by special high-level Petri net features.

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. H. Cho and S.I. Marcus. On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation. em Mathematics of Control, Signals, and Systems, Vol. 2, No. 2, pp. 47-69, 1989.
4. J. Desel, G. Juhás and R. Lorenz. Input/Output Equivalence of Petri Modules. In *Proc. of IDPT 2002*, Pasadena, USA, 2002.
5. P. Dietrich, R. Malik, W.M. Wonham and B.A. Brandin. Omlementation Consideration in Supervisory Control. In [1].
6. H.-M. Hanisch, A. Lüder: Modular Modeling of Closed-Loop Systems, Colloquium on Petri Net Technologies for Modelling Communication Based Systems, Berlin 1999, pp. 103-126.
7. 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.
8. 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.
9. 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.
10. 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.
11. J.E. Hopcroft, R. Motwani and J.D.Ullman. Introduction to Automata Theory, Languages, and Computation. Addison Wesley, 2001.
12. R. Janicki and M. Koutny. Semantics of Inhibitor Nets. *Information and Computations*, 123, pp. 1–16, 1995.
13. 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.
14. G. Juhás and R. Lorenz. Modelling with Petri Modules. In [1].
15. L.E. Pinzon, M.A. Jafari, H.-M. Hanisch and P. Zhao Modelling admissible behavior using event signals submitted
16. P.J. Ramadge, W.M. Wonham: The Control of Discrete Event Systems. Proceedings of the IEEE, 77 (1989) 1, S. 81-98.
17. 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.
18. R.S. Sreenivas und B.H. Krogh. On Condition/Event Systems with Discrete State Realizations. *Discrete Event Dynamic Systems: Theory and Applications*, 2, 1991, 1, 209–236.
19. R. S. Sreenivas and B. H. Krogh. Petri Net Based Models for Condition/Event Systems. In *Proceedings of 1991 American Control Conference*, vol. 3, 2899–2904, Boston, MA, 1991.

20. P. H. Starke. Das Komponieren von Signal-Netz Systemen. In Proc 7. Workshop Algorithmen und Werkzeuge für Petrinetze AWPN 2000, Universität Koblenz - Landau, pages 1–6, 2000.
21. P. Darondeau and S. Kumagai (Eds.). Proceedings of the *Workshop on Discrete Event System Control*. Satellite Workshop of ATPN 2003.
22. M.C. Zhou und F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Adacemic Publishers, Boston, MA, 1993.
23. Zhonghua Zhang and W.M. Wonham. STCT: An Efficient Algorithm for Supervisory Control Design. In [1].