An admissible-behaviour-based analysis of the deadlock in Petri-net controllers

G. Mušič [a,*], D. Matko [a]

[a] *University of Ljubljana, Faculty of Electrical Engineering, 1000 Ljubljana, Tržaška 25, Slovenia*

Abstract

This paper addresses the problem of verifying the discrete control logic that is typically implemented by programmable controllers. Not only are the logical properties of the controller studied during verification, the behaviour of the overall controlled system is also examined. An approach that combines the calculation of the safety-oriented interlock controllers in terms of supervisory control theory (SCT), the corresponding calculation of the admissible behaviour of the system, and the specification of the desired system operation by Petri nets is proposed. A potential deadlock in the controlled system is then verified by taking the admissible-behaviour model as a process model. The analysis of the simultaneously operated supervisory-control-based interlock controller and the Petri-net-based sequential controller is performed with a C-reachability graph. The paper focuses on the calculation of the graph, and the approach is illustrated with an example of a simple manufacturing cell.

Key words: Supervisory control, Petri nets, manufacturing systems, logic controllers

1 Introduction

While the functionality of programmable logic controllers (PLCs) is continuously expanding, discrete control logic remains the core of their operation. For a long time, PLCs have been programmed in a rather intuitive way, using specialised graphical programming languages, such as a ladder diagram [13].

* Corresponding author.
Email addresses: `gasper.music@fe.uni-lj.si` (G. Mušič), `drago.matko@fe.uni-lj.si` (D. Matko).

Recently, much attention has been given to formal methods and their application in the design and verification of PLC programs. This is motivated by the growing complexity of control problems, the demands for reduced development times and the need to reuse existing software modules, on the one hand, and the increasing demands of society for a better control of technological risks, on the other [4,8].

Verification-based approaches deal with the formalization of the specifications and verification of the program against the formal specification [7]. The program passes the verification when the behaviour specified by the designer satisfies a set of properties. The properties can be checked on the control model only, or by considering a model (possibly a partial model) of the process. The latter is a more realistic approach to verification, called model-based verification [4].

To make the results of such a verification approach useful for the control, an appropriate model of the process under control is needed; however, this is not readily available in many cases. Different aspects of plant modelling for the purpose of controller verification have been extensively studied in [6,14,5]. The approach presented there enables detailed and systematic modelling of the controlled processes by employing a special modelling formalism.

In special cases, however, a suitable model for the verification can be obtained by considering a multilevel control structure and adopting a partially controlled plant on the lower level as a plant model for the verification of the upper level. Such a two-level approach is proposed in our previous work [10], and is further elaborated in this paper. In particular, such an approach can be used in applications involving PLCs, where a large portion of the control code is dedicated to safety measures, also called interlocks, and the corresponding part of the logic is sometimes referred to as the locking controller [18]. Assuming a two-stage approach, where the interlock logic is designed first and the sequential part is then added on top of that, the admissible behaviour of the plant imposed by the interlock logic can be adopted as a plant model for the verification of the sequential part.

In the presented approach the interlock part of the control logic is synthesized using supervisory control theory (SCT) [16,1]. The synthesis also gives a model of the admissible behaviour of the process, i.e., the behaviour of the process that complies with the given interlock specifications. The sequential part is then designed using Petri nets [9], which are used in the sense of a formal specification that is verified against the admissible model derived during the interlock synthesis. The basic property of interest is the absence of deadlock. A corresponding reachability-based analysis technique is proposed, which builds a C-reachability graph and enables the detection of any potential deadlock in the system that is controlled by a simultaneously operated supervisory-control-based interlock controller and a Petri-net-based sequential controller.

The motivation for the use of two modelling formalisms is twofold; firstly, the supervisory control theory is well suited to the interlock design. SCT is essentially safety-oriented, i.e., it enables the synthesis of a control policy that prevents any undesired behaviour of the controlled plant. In most applications, however, there are also requirements about the desired behaviour of the plant that should be enforced by the controller. The SCT-based synthesis and implementation of controllers that force the system to exhibit desired behaviour is difficult, although some related results are reported in the literature [2,12]. Secondly, the Petri-net framework provides an intuitive way of modelling the operation sequences, while the Petri-net-based supervisory control methods are less elaborate, especially in terms of event feedback, and few synthesis tools are available. The proposed combined approach exploits the advantages of both frameworks. Compared to other model-based verification approaches that are described in the literature (e.g., [3,6,17], see also survey papers [4,7], and the references therein), the main advantage of the combined approach is that it eliminates the need for an additional plant model for the purpose of verifying the sequential controller. The corresponding model is derived automatically during the interlock design stage.

The remainder of the paper is structured as follows. The proposed combined synthesis/verification approach is introduced in Section 2. The relation between admissible behaviour and the firing of transitions in the Petri-net model of operational procedures is explained in detail. The proposed deadlock-analysis technique is described in Section 3. The C-reachability graph is introduced, the required properties of the graph are examined, and a corresponding graph-calculation algorithm is presented. A simple example is given in Section 4 to illustrate the approach.

2 Combined synthesis/verification appproach

The aim of the verification is to answer the question as to whether a specification model is correct. This is done by examining various properties of the model, such as the stability, the absence of deadlocks, etc. In the presented case, the investigation is limited to the study of a single property, i.e., a check to ensure the Petri-net specification is not blocking the system operation.

The non-blocking property of a Petri net is traditionally regarded as the absence of deadlocks and closely related to the concept of liveness. A Petri net is said to be *live* when it is possible to ultimately fire any transition of the net by progressing through some firing sequence, starting from any marking that is reachable from a given initial marking [9]. A live Petri net guarantees deadlock-free operation.

When examining the Petri-net specification of a logic controller, the property of liveness is insufficient to ensure the non-blocking operation due to external inputs and outputs and their interrelations. The liveness of a PN is a necessary, but not sufficient, condition for the non-blocking operation of a related controller.

To examine the possible blocking of the controller the relation between the inputs and outputs must be taken into account. In other words, instead of analysing the 'open-loop' model of the controller a 'closed-loop' model of the control system has to be studied. Such an approach can be considered as a model-based approach to verification, according to the classification in [4].

2.1 Modelling a process under control

The key to the success of such a verification approach is a suitable model of the process under control. However, building such a model can be a difficult and cumbersome task. But in certain cases models developed during the initial stage of the control logic design can be used.

A two-stage approach to the design of the control logic is schematically shown in Fig. 1. It is a simplified version of the multistage approach proposed in [12]. One of the key points of the approach is that the specifications are split into two parts. The first part involves the prevention of undesired behaviour. It is composed of the so-called interlocks that implement measures to ensure safety, co-ordinate sub-processes, etc. The second part deals with the sequential specification and defines the prescribed order of tasks. The sequencing part of the control logic is synthesized only after the interlock part has been designed. Besides greater modularity, this increases the flexibility of the proposed solution since only the upper layer has to be redesigned when changes to the system operation are required.

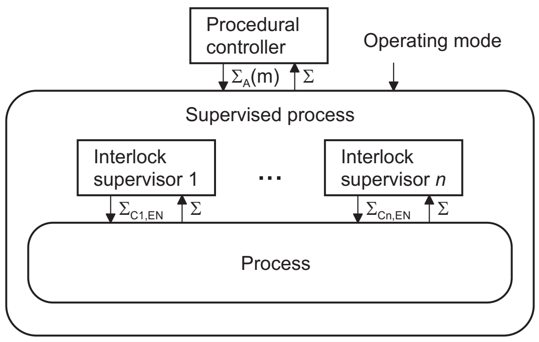


Figure 1. Proposed control structure

The set of interlock supervisors can be designed within the framework of supervisory control theory. The result of the synthesis of the interlock part is a model of the admissible behaviour, i.e., the model of all the possible event sequences in the controlled system that comply with the interlock specification. This model can be used as an 'open-loop' process model when designing the sequencing part of the control logic. This part may be conveniently specified by a Petri net, which is then verified in combination with the open-loop model.

In the combined approach, the evolution of the Petri net is driven by the underlying layer of interlock control logic that is modelled as a finite state machine. The link between the two representations are the input/output (I/O) signals.

2.2 Events, I/O signals, and admissible behaviour

The supervisory control concept [16] deals with restrictions on the behaviour of a discrete event system imposed by an external controller – a supervisor, acting by disabling events. The set of events is partitioned into two disjoint subsets – the controllable and uncontrollable events: $\Sigma = \Sigma_c \cup \Sigma_u$, $\Sigma_c \cap \Sigma_u = \emptyset$. The uncontrollable events cannot be disabled. The supervisor is computed on the basis of the open-loop system model and a specification model. The key issues are the concept of controllability and the concept of the supremal controllable sublanguage [1,20].

The feasible set of input/output (I/O) signal patterns in a supervised system is defined by the supervisor S and is implicitly given by the discrete event model of the supervised plant. This model can be given in the form of a finite automaton, which is interpreted as a deterministic language generator $H_a = (X, \Sigma, \delta, x_0, X_m)$.

Here, X is a set of states, Σ is a set of symbols associated with events, $\delta : X \times \Sigma \to X$ is a state-transition function and is in general a partial function on its domain, x_0 is the initial state and X_m is a set of marker states. A symbol $\sigma_i \in \Sigma$ is generated during every transition. A finite set of symbols is called an event sequence. The language generated by G is $\mathcal{L}(G)$. It is interpreted as the set of all the finite event sequences that can occur in the automaton. The language marked by G is denoted by $\mathcal{L}_m(G)$ and consists of event sequences that end in marker states. Such a generator is derived by the supervisory control synthesis procedure as a model of admissible behaviour.

Let Σ^* denote the set of all the finite sequences of elements of Σ, including the empty sequence, and let st denote the concatenation of sequences $s, t \in \Sigma^*$. A prefix closure of the language $L \subseteq \Sigma^*$ is then defined as $\overline{L} = \{s \in \Sigma^*;\ \exists t \in \Sigma^*, st \in L\}$. The automaton is non-blocking if it is capable of reaching a marker state from any reachable state, i.e., $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$.

As explained in [10], in the proposed approach the blocking is not considered at this point, therefore $X_m = X$. The language $\mathcal{L}(H_a) = L_a$ generated by H_a contains all the admissible event sequences.

An event $\sigma \in \Sigma$ can be regarded either as an external event observed through the change in the state of the corresponding I/O signal or it can be actively triggered by the controller. In any case, a change in the controller input or output signal state is associated with every event $\sigma \in \Sigma$. This will be denoted by $v' = \delta_v(v, \sigma)$ and $u' = \delta_u(u, \sigma)$. The sets of the output and input signal states are denoted as $U := \{u | u : A \to \{0, 1\}\}$ and $V := \{v | v : B \to \{0, 1\}\}$, where A and B are the sets of controller output and input signals, respectively. Next, the set of total states is defined as $W := \{w | w = (x, u, v)\}$.

Considering the event sequences that are generated by the model of the admissible behaviour H_a a new total state automaton $H_w = (W, \Sigma, \xi, w_0, W_m)$ is constructed, where $W \subseteq X \times U \times V$ is as defined above, Σ is the set of events comprising the admissible behaviour, and ξ is a new state-transition function, defined as follows:

$$\xi(w, \sigma) = \begin{cases} (\delta(x, \sigma), u', v') & \text{if } \delta(x, \sigma) \text{ defined} \\ \text{undefined} & \text{if } \delta(x, \sigma) \text{ undefined} \end{cases} \tag{1}$$

where $w = (x, u, v)$, $u' = \delta_u(u, \sigma)$ and $v' = \delta_v(v, \sigma)$, as defined above. For convenience, ξ is extended from the domain $W \times \Sigma$ to $W \times \Sigma^*$ in the usual way. The initial state is $w_0 = (x_0, u_0, v_0)$ and all the states are marked $W_m = W$. Note that $\mathcal{L}(H_w) = \mathcal{L}(H_a) = L_a$, which is evident from (1).

2.3 Specification of the operational procedures

Petri nets as a tool for modelling and the specification of manufacturing systems are described in a number of sources, such as [9,1]. A Place/Transition Petri net can be described as a bipartite graph consisting of two types of nodes, places and transitions. The nodes are interconnected by directed arcs. The state of the system is denoted by the distribution of tokens (called marking) over the places. For the purposes of simulation and possible implementation by industrial controllers, the input/output interpretation can be added. One of these extensions is a class of Petri nets called *Real-Time Petri Nets* (RTPNs) [21]. Formally, a RTPN is defined as an eight-tuple $\mathcal{N} = (P, T, I, O, m_0, D, Y, Z)$ where

- $P = \{p_1, p_2, \ldots, p_k\}, k > 0$ is a finite set of places,

- $T = \{t_1, t_2, \ldots, t_l\}, l > 0$ is a finite set of transitions (with $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$),
- $I : P \times T \rightarrow \mathbb{N}$ is a function that specifies the weights of arcs directed from places to transitions,
- $O : P \times T \rightarrow \mathbb{N}$ is a function that specifies the weights of arcs directed from transitions to places,
- $m : P \rightarrow \mathbb{N}$ is a marking, m_0 is the initial marking,
- $D : T \rightarrow \mathbb{R}^+$ is a firing time-delay function,
- $Y : T \rightarrow \mathcal{B}$ is an input-signal function, where $\mathcal{B}$ is the set of Boolean expressions on the set $B \cup A$ of input/output signals;
- $Z : P \rightarrow 2^{A \times \{0,1\}}$ is a physical output function, where A is the set of output signals [1].

In the following the paper deals only with safe RTPNs, i.e., $m(p) \leq 1, \forall p \in P$. The output function of a place sets the related output signals to the specified values when the place is marked.

2.4 RTPN control of a supervised discrete-event process

To enable a detailed analysis of the potential deadlock in a RTPN that is controlling a process under the restriction of a discrete-event supervisor, the firing rule of the RTPN must be defined. A firing rule from [21] is here adopted with a slight modification.

In standard Petri-net theory the transition $t \in T$ is said to be enabled if $m(p) \geq I(p,t), \forall p \in {}^\bullet t$. Here, ${}^\bullet t \subseteq P$ denotes the set of places that are inputs to the transition $t \in T$. This definition also holds for a RTPN, but such a transition is called a *state-enabled* transition. The set of state-enabled transitions of a RTPN under the marking m is $T_e(m) := \{t | t$ is state enabled under $m\}$.

Next, a transition $t \in T$ is defined as *input enabled* under an I/O state $(v,u) \in V \times U$ when $eval(Y(t), v, u) = 1$. The function $eval(e, v, u)$ denotes an evaluation of the Boolean expression $e \in \mathcal{B}$ by the given I/O state v, u. A set of input-enabled transitions of a RTPN under I/O state v, u is $T_i(v,u) := \{t | t$ is input enabled under $v, u\}$.

A transition is defined as *output enabled* when all the preceding control actions have actually been executed. A transition $t \in T$ is output enabled under an output state $u \in U$ when $Z(p) = \{(a_1, i_1), \ldots, (a_n, i_n)\} \Rightarrow u(a_j) =$

[1] These definitions of input and output functions are slightly changed with respect to [21] and [10].

$i_j, \forall (a_j, i_j) \in Z(p), \forall p \in {}^{\bullet}t$. A set of output-enabled transitions of a RTPN under output state u is $T_o(u) := \{t | t \text{ is output enabled under } u\}$.

The *firing rule* of a RTPN can now be defined as follows:

i) a transition $t \in T$ is enabled if it is state enabled, input enabled and output enabled, i.e., $t \in T_e \cap T_i \cap T_o$,

ii) an enabled transition may or may not fire, depending on the firing time-delay function associated with it:
  - a transition with a zero time delay fires immediately after being enabled,
  - a transition with a non-zero time delay fires immediately after the delay $D(t)$ expires (the corresponding timer starts when the transition is enabled),

iii) the firing of a transition is immediate and it removes a token from each of the input places of the transition and adds a token to each of the output places of the transition.

A no-concurrency firing setting is assumed, i.e., a single transition fires at a time, and $m[t\rangle m'$ denotes that t can fire under m, resulting in m'.

Given Petri-net $\mathcal{N}$ and marking m, a marking m' is said to be immediately reachable, i.e., $m' \in R_1(\mathcal{N}, m)$, if there exists a transition t such that t is state enabled under m and its firing results in m', i.e, $m[t\rangle m'$. A marking m_k is said to be reachable from a marking m_0, i.e., $m_k \in R(\mathcal{N}, m_0)$, if there exists a sequence $\langle m_0 m_1 \dots m_k \rangle$ such that $m_i \in R_1(\mathcal{N}, m_{i-1})$ for $0 < i \le k$. The notion of reachability can be extended by considering the input and output signals of a RTPN:

Definition 1 *Given a RTPN $\mathcal{N}_R$, marking m, input state v, and output state u, marking m' is said to be immediately* IO-reachable *under I/O state v, u, i.e., $m' \in R_1^{IO}(\mathcal{N}_R, m, v, u)$, if there exists a transition t such that t is state enabled, input enabled, and output enabled under m, v, and u, respectively, and the firing of t results in the marking m'.*

The paper is focused on the operation of a controller modelled by a RTPN and acting on a discrete-event system H_w. Therefore, C-reachability (control-reachability) is defined as follows:

Definition 2 *Given a RTPN $\mathcal{N}_R$ with marking m, and coupled to a discrete-event system H_w, marking m' is said to be immediately* C-reachable *under total state w, i.e., $m' \in R_1^C(\mathcal{N}_R, m, H_w, w)$, when it is immediately IO-reachable under I/O state v, u, where $w = (x, u, v)$.*

The admissible firing sequences define the C-reachability set $R^C(\mathcal{N}_R, m_0, H_w)$ of a RTPN $\mathcal{N}_R$ coupled to H_w:

Definition 3 *Given a RTPN $\mathcal{N}_R$ with initial marking m_0, and coupled to a discrete-event system H_w with initial state w_0, marking m' is said to be* C-reachable, *i.e., $m' \in R^C(\mathcal{N}_R, m_0, H_w)$ if there exists a sequence $\langle m_0 m_1 \ldots m_k \rangle$ such that $m_i \in R_1^C(\mathcal{N}_R, m_{i-1}, H_w, w_{i-1})$ and $w_{i-1} = \xi(w_0, s); s \in L_a$ for $0 < i \leq k$. By definition, $m_0 \in R^C(\mathcal{N}_R, m_0, H_w)$.*

H_w is assumed to be in the initial state w_0 when a corresponding RTPN is marked by the initial marking m_0. The changes of the input/output signal state are driven by the evolution of the two models: the total state automaton model of admissible behaviour of the plant and the RTPN model of operational sequences.

Considering the notion of C-reachability the deadlock-free operation of a RTPN controller can now be defined:

Definition 4 *A RTPN system $(\mathcal{N}_R, m_0)$ coupled to H_w is deadlock-free when for every C-reachable marking $m \in R^C(\mathcal{N}_R, m_0, H_w)$ there exists a marking m' that is immediately C-reachable from m, i.e., $m' \in R_1^C(\mathcal{N}_R, m, H_w, w)$, where $w = \xi(w_0, s); s \in L_a$, and L_a is the admissible behaviour.*

3 Analysis of deadlock

To be able to analyse the existence or absence of deadlock in the RTPN controlling a discrete-event process, a new kind of reachability graph is proposed here that enumerates all the admissible event and transition sequences: the C-reachability graph.

3.1 C-reachability graph

The nodes of the graph are pairs (m, w), where m is a marking of the RTPN, while w is the state of the automaton H_w. The construction starts in the initial state (m_0, w_0), where $w_0 = (x_0, u_0, v_0)$. A set of feasible events is then sought. This is a subset of the feasible events $\Gamma(x_0)$ of the automaton H_a. More precisely, the set is composed of two subsets. One is the set of all the events feasible at x_0 and not generated by the RTPN. The other is the set of events generated by the actions of the marked places of the RTPN and defined by the output function Z, which are also feasible at x_0.

Let Σ_{CTRL} denote a set of events triggered by the RTPN, and Σ_{SP} a set of events that are not generated by the RTPN ($\Sigma_{SP} = \Sigma - \Sigma_{CTRL}$). Let $\Sigma_A(m)$ denote the set of events generated by actions of the marked places of the

RTPN. The set of feasible events Σ_F at H_a in the state x and the RTPN marked by m is then given by

$$\Sigma_F(x, m) = \Gamma(x) \cap (\Sigma_{SP} \cup \Sigma_A(m)) \tag{2}$$

Then a node (m_0, w_i) where $w_i = \xi(w_0, \sigma_i)$ is added for $\forall \sigma_i \in \Sigma_F(x_0, m_0)$ and the arc from (m_0, w_0) to (m_0, w_i) is labelled σ_i.

Next, the set of immediately C-reachable markings $R_1^C(\mathcal{N}_R, m_0, H_w, w_0)$ is determined. For every corresponding marking $m_i \in R_1^C(\mathcal{N}_R, m_0, H_w, w_0)$ a node (m_i, w_0) is added to the graph and the arc from (m_0, w_0) to (m_i, w_0) is labelled t_i, where t_i is the transition leading from m_0 to m_i. In the case of conflicting transitions, all possible firing sequences are enumerated as in a standard reachability analysis.

The procedure is repeated for every added node, and duplicate nodes of the graph are merged. The procedure stops when there are no new nodes or all the new nodes are duplicate nodes.

A new kind of reachability graph is derived in the described way . A set of nodes is associated with every reachable marking and the transitions between the nodes are of two types:

– the transitions of a RTPN connect the nodes associated with distinct markings,
– the transitions related to events in a model of admissible behaviour connect the nodes associated with the same marking.

Since the derived graph includes the input and output events of a controller, it is called the *C-reachability graph* of a RTPN controller. It should be noted that only the ordering of events is considered, while the timing information of a RTPN is omitted.

The resulting graph can be interpreted as an automaton where the transitions of a RTPN are considered as additional events in the system. Such an automaton is denoted as $CG = (N, \Sigma_{CG}, \zeta, n_0, N_m)$, where $N \subseteq R^C(\mathcal{N}_R, m_0, H_w) \times W$ is a set of nodes in the graph, $\Sigma_{CG} \subseteq \Sigma \cup T$ is the set of transition labels, $\zeta : N \times \Sigma_{CG} \to N$ is a transition function defined by the arcs of the graph, $n_0 = (m_0, w_0)$ is the initial state, and N_m is the set of marker states.

It is important to note that since the construction is driven by a sequential specification, only a small subset of possible I/O combinations is actually enumerated in CG.

Finally, the C-reachability graph is used to analyse any potential blocking of a controller. The following proposition is applied:

Proposition 1 *A control specification given as a RTPN system $(\mathcal{N}_R, m_0)$ with the transition set T and acting on a discrete-event system H_w is deadlock free if the corresponding C-reachability graph $CG = (N, \Sigma_{CG}, \zeta, n_0, N_m)$:*

(i) contains at least one transition of the RTPN, i.e., one of those transitions appears at least once as a label of an edge in the graph, $\exists t \in T, n, n' \in N, n' = \zeta(n, t)$, and
(ii) can be interpreted as a nonblocking automaton, given $N_m = \{n_0\}$.

Proof: For a non-blocking automaton with $N_m = n_0, \forall s \in \mathcal{L}(CG), \exists s', ss' \in \mathcal{L}(CG), \zeta(n_0, ss') = n_0$. It is, therefore, clear that it can return to the initial state from any reachable state. Consider now the case that the automaton is in state $n = (m, w)$, where $m \neq m_0$. Clearly, if the automaton can return to the initial state $n_0 = (m_0, w_0)$, there exists a firing sequence $\langle mm' \dots m_0 \rangle$ with $m' \in R_1^C(\mathcal{N}_R, m, H_w, w)$. Next, the case when the automaton is in state $n = (m_0, w)$ is considered. In this case the return to the initial state of the automaton is insufficient for the RTPN being deadlock free, as the initial state can be reached without a change in marking and consequently without firing a single transition. But if $\exists t \in T, n' = \zeta(n, t)$, for some $n, n' \in N$ there must also exist $m' \in R_1^C(\mathcal{N}_R, m_0, H_w, w)$, such that $m_0[t\rangle m'$. Therefore, an immediately C-reachable marking can be found for every reachable marking including m_0, which means the RTPN is deadlock free according to Def. 4. ■

There is often a need to extend the requirement for a control specification to be deadlock free. It is also often required for all the parts of the sequential behaviour to be eventually reachable. In terms of Petri-net terminology, this requires any transition within the Petri net to eventually become enabled, starting from any marking reachable from the initial marking. Such a Petri net is live [9]. To adapt this notion to a sequential specification in terms of the RTPN acting on a process under supervision, the following definition is applied.

Definition 5 *A RTPN system $(\mathcal{N}_R, m_0)$ with transition set T and coupled to H_w is C-live when for every C-reachable marking $m \in R^C(\mathcal{N}_R, m_0, H_w)$ any transition $t \in T$ will eventually be fired.*

The C-liveness can also be checked from the C-reachability graph in a similar way to the absence of deadlocks. The difference is that since any transition must be eventually fired, all the transitions must appear in the C-reachability graph. This is summarized in:

Proposition 2 *RTPN system $(\mathcal{N}_R, m_0)$ with transition set T and coupled to H_w is C-live if a corresponding C-reachability graph $CG = (N, \Sigma_{CG}, \zeta, n_0, N_m)$:*

(i) contains all the transitions of the RTPN, i.e., $\forall t \in T, \exists n, n' \in N, n' = \zeta(n, t)$

(ii) can be interpreted as a nonblocking automaton, given $N_m = \{n_0\}$.

Proof: By construction, any node of CG maps to a reachable marking of the RTPN. If all the transitions appear as the labels of the arcs of CG this corresponds to the eventual firing of any transition from the initial marking. If CG can return to the initial state from a given node, the firing sequence can also continue to any other node of the CG, which means any transition of the RTPN can eventually be fired, starting from every reachable marking. ■

Note, that in general, most of the spontaneous events are uncontrollable, in the sense of supervisory control. In addition, most of the controlled events are controllable. It should be noted, however, that this relationship between spontaneous and uncontrollable events, on the one hand, and between controlled and controllable events, on the other, is not strict. An uncontrollable event $\sigma_1 \in \Sigma_u$ can be generated by the RTPN; therefore, $\sigma_1 \in \Sigma_{CTRL}$, e.g., the start of an emergency procedure, which must not be disabled. In contrast, a controllable event $\sigma_2 \in \Sigma_c$ can be generated externally ($\sigma_2 \in \Sigma_{SP}$), e.g., an operator request that may be blocked by the supervisor. All these situations are captured within the C-reachability graph and are taken into account during an analysis of blocking.

3.2 Calculation of the C-reachability graph

To further illustrate the procedure of composing the graph a sketch of the corresponding calculation procedure is given. The graph is represented as $CG = (N, A)$, where N is a set of nodes in the form of ordered pairs (m, w), as described above, and A is a set of arcs, given as $A \subseteq N \times (\Sigma \cup PN.T) \times N$. An arc between the nodes n_1 and n_2 is denoted (n_1, e, n_2), and $e \in \Sigma \cup PN.T$ is either an I/O event or a Petri-net transition. The procedure is summarized in Algorithm 1.

Algorithm 1:

```
$w := (x_0, u_0, v_0)$;
$N_{CG} := (m_0, w)$; (* a node related to the initial marking of a RTPN, the initial state of the automaton $H_w$, and the initial state of the I/O signals *)
CG.N := $\{N_{CG}\}$;
CG.A := $\emptyset$;
RSET := $\{m_0\}$;
(* events not triggered by RTPN *)
$\Sigma_{SP}$ := spontaneous($H_a$, PN);
U := $\{N_{CG}\}$; (* list of unexplored nodes *)
while $U \neq \emptyset$ do
```

```
    choose a node $N_{CG} \in U$;
    (* related marking of RTPN *)
    m := marking($N_{CG}$);
    (* related state of $H_a$ *)
    x := state($N_{CG}$);
    (* related state of $H_w$ *)
    w := total_state($N_{CG}$);
    (* events triggered by actions in marked places *)
    $\Sigma_A$ := actions(PN, m, $N_{CG}$);
    (* feasible events list *)
    $\Sigma_F := \Gamma(x) \cap (\Sigma_{SP} \cup \Sigma_A)$;
    **for** $\forall \sigma \in \Sigma_F$ **do**
        $w' := (\delta(x, \sigma), \delta_u(u, \sigma), \delta_v(v, \sigma))$;
        $newN_{CG} := (m, w')$;
        **if** not a duplicate node **then**
            CG.N := CG.N $\cup$ $\{newN_{CG}\}$;
            CG.A := CG.A $\cup$ $\{(N_{CG}, \sigma, newN_{CG})\}$;
            U := U $\cup$ $\{newN_{CG}\}$;
        **else**
            adjust connections of the duplicated node;
        **end**
    **end**
    (* enabled transitions *)
    $T_{EN}$ := getEnabledTransitions(PN, m);
    (* condition-enabled transitions *)
    $T_{CEN}$ := checkConditions(PN, $T_{EN}$, $N_{CG}$);
    (* check if actions in the input places have been executed: *)
    $T_{ACEN}$ := checkActions(PN, $T_{CEN}$, $N_{CG}$);
    (* transitions that may be triggered *)
    **for** $\forall t \in T_{ACEN}$ **do**
        u := getFiringVector(PN, t);
        (* calculate a new marking *)
        m' := m + (PN.O-PN.I) u;
        $newN_{CG}$ := (m', w);
        **if** not a duplicate node **then**
            CG.N := CG.N $\cup$ $\{newN_{CG}\}$;
            CG.A := CG.A $\cup$ $\{(N_{CG}, t, newN_{CG})\}$;
            U := U $\cup$ $\{newN_{CG}\}$;
        **else**
            adjust connections of the duplicated node;
        **end**
    **end**
    (* remove the node from the list of unexplored nodes *)
    U := U - $\{N_{CG}\}$;
**end**
```

3.3 Complexity

The state size of the constructed C-reachability graph depends heavily on the type and the properties of the process considered and the related operational procedure specification. However, some conclusions can be drawn by considering a typical application.

First, it should be noted that the number of iterations of the main loop of *Algorithm 1* equals the number of nodes (states) in the C-reachability graph. It is also known that the computational complexity of checking the various individual properties, such as the absence of dead-locks, is linear in the size of the state space [19]. The number of nodes in the C-reachability graph will, therefore, be of primary concern.

Recalling the introduction of the C-reachability graph in Section 3.1, the nodes of the graph are pairs (m, w), where m is a marking of the RTPN, while w is the state of the total state automaton H_w. The first, very coarse estimate for the number of nodes in the graph is then

$$N_{nodes} \leq |R(\mathcal{N}_R, m_0)| \cdot |x_{Hw}| \tag{3}$$

where $|R(\mathcal{N}_R, m_0)|$ is the number of possible markings in the RTPN, and $|x_{Hw}|$ is the number of states in the automaton H_w.

The number of nodes in the graph is typically most often related to the complexity of the RTPN. This is because the related specification of the operational procedure extracts only a small subset of states from the admissible behaviour model.

In the following, only the case where all the events in the admissible behaviour model are related to I/O signals of the RTPN is considered. Furthermore, an assumption that the number of states in the total state automaton H_w equals the number of states in the admissible behaviour model H_a will be made. This is true for the majority of applications of SCT to PLC programming. The related interlock design deals with the prevention of the forbidden I/O signal state combinations. When calculating the model of the admissible behaviour, the focus is on distinguishing different I/O states and not different event orderings. Therefore, during the total state automaton calculation there is no need to introduce new states. Instead, the description of the existing states of the admissible behaviour is merely extended by the associated I/O states.

Using such an assumption, a better estimate for the number of nodes in the graph can be obtained by taking possible combinations of the I/O signals at a particular marking into account. At every marking, a set of new nodes in the C-reachability graph is generated, according to the switching of the I/O signals. If the number of signals $N_S(m)$ that can switch at a particular marking m is taken into account, and there is no information on their restrictions, all the possible signal states are considered. The related number of nodes is then $2^{N_S(m)}$, and the estimate is

$$N_{nodes} \leq \sum_{m_i \in R(\mathcal{N}_R, m_0)} 2^{N_S(m_i)} \tag{4}$$

In practice, this number is smaller because certain combinations of signal states are not allowed or not possible. Every interlock defined as a forbidden combination of two signal states lowers the number of nodes by 1/4. Taking N_I such interlocks into account leads to an estimate of $(3/4)^{N_I} 2^{N_S} = 3^{N_I} 2^{(N_S - 2N_I)}$ nodes per marking. The total estimate is then

$$N_{nodes} \leq \sum_{m_i \in R(\mathcal{N}_R, m_0)} 3^{N_I(m_i)} \cdot 2^{(N_S(m_i) - 2N_I(m_i))} \tag{5}$$

where $N_I(m_i)$ is the number of interlocks that apply at m_i.

Clearly, only the proper estimation of N_s can give a useful result. The number of output signals that can switch by a particular marking is known. It is given by the number of actions at the marked places, i.e., the number of elements in the union of $Z(p_i)$ for all the marked places. This union will be denoted by $Z(m)$. It is more difficult to estimate the number of possible events that are not triggered by the RTPN. In Sect. 3.1 these events were denoted by Σ_{SP} (spontaneous events). A further look at equation (2) shows that N_s is actually related to the number of elements in the union of all $\Sigma_F(x, m)$ at fixed m. Since this is rather difficult to estimate, only the maximum number of spontaneous events in a sequence at a given marking is estimated. Let $\Sigma_{SP}(m)$ denote this estimate and let $S_{SP}(m)$ denote the I/O signals related to $\Sigma_{SP}(m)$. The estimate of the upper bound for the number of nodes is then

$$N_{nodes} \leq \sum_{m_i \in R(\mathcal{N}_R, m_0)} 3^{N_I(m_i)} \cdot 2^{(|S_{SP}(m_i)| + |Z(m_i)| - 2N_I(m_i))} \tag{6}$$

Finally, equation (6) can be simplified if the sequential specification is limited to the class of safe Petri nets with no concurrency (state machines). The number of possible markings in such a net equals the number of places. The estimate, therefore, simplifies to

$$N_{nodes} \leq \sum_{p_i \in P} 3^{N_I(p_i)} \cdot 2^{(|S_{SP}(p_i)| + |Z(p_i)| - 2N_I(p_i))} \tag{7}$$

where $N_I(p_i)$ is the number of interlocks that apply when p_i is marked and

$|S_{SP}(p_i)|$ is the number of signals related to the spontaneous events that can occur when p_i is marked. $|Z(p_i)|$ is the number of elements in the output function of p_i.

While the estimate (4) is rather conservative, because all the signal state combinations were considered, estimates (6) and (7) are relatively close to the real number, in particular when a large number of interlocks are applied. The latter two, on the other hand, require a deeper knowledge about the process and control specifications. They are, therefore, more difficult to calculate.

To illustrate the C-reachability graph's complexity, consider a typical small PLC application, involving the control of four pneumatic devices, each equipped with two sensors and driven by two electro-pneumatic valves. Eight interlocks apply, because each pair of sensors cannot be activated simultaneously, and in each pair of actuating signals, at most one can be switched on at a time. Assuming a sequence of 10 steps, taking all the sensor readings as spontaneous events at every step, allowing two actions per step, and imposing no additional restrictions besides interlocks on sensor readings (i.e., four interlocks per step), the equation (6) gives an estimate of 3240 nodes in a corresponding C-reachability graph. This number is quite manageable with the appropriate computer support. Furthermore, the number of nodes is even smaller in real applications, because additional restrictions on signal switchings apply. For example, in [11] a small size practical application is reported, where the complexity of the graph did not exceed 200 nodes.

4 Example

An example is given to illustrate the concept of the C-reachability graph, which is simple enough to exhibit interesting properties, but at the same time based on equipment used in industrial applications. The example deals with part of a laboratory-scale modular production line composed of five working stations controlled by five programmable logic controllers [12]. The stations perform the distribution of workpieces, the testing of workpieces, processing, manipulation and sorting. Every working station is further composed of a set of pneumatic pistons, gears, two state sensors, and electro-pneumatic actuators, which form a mechanical setup that can be controlled by a PLC to perform a required operation.

The central part of the line is considered, i.e., the processing station, consisting of a rotational table that moves a workpiece between consecutive phases, a drilling machine, and a testing device. The next working station includes a manipulator that transports the workpiece further on. The central part of the production line is shown in Fig. 2.

The setup is similar to the one used in [15], except that a much closer view of the process is taken in this paper. In [15] the SCT is used to coordinate the operating phases, while the example presented here deals with the control logic inside a particular phase. The switching of I/O signals in the desired operating procedures is modelled, as is the behaviour relating to erroneous conditions in the process.

To keep the presentation sufficiently simple, a particular detail will be studied, i.e., the interlock between the rotating table and the holder that fixes the workpiece before it is drilled. The table is driven by an electric motor, which is rotating when signal $ar = 1$. When the signal is switched on, event $ar1$ is generated. Similarly, when the signal is switched off, event $ar0$ is generated. The same labelling scheme is used for all the events related to the I/O signals in the modelled process. The prefix of the event label matches the corresponding signal label, while the suffix indicates the direction of the signal switching.

The table has four stop positions, indicated by a proximity switch sp. The switch closes ($sp1$) when the table comes into position, and releases ($sp0$) when the position is left. For the example, only the switching of the actuator ar is considered. The simplified finite-state machine model of the table is shown in Fig. 3.

The holding piston is driven by an electro-pneumatic valve switching the pressure on and off. Initially, the piston is in the forward position, and it moves backwards when $ah = 1$ and forwards when $ah = 0$. The piston is equipped with two limit switches, indicating the backward (sb) and forward (sf) positions. The movement is limited to the distance between the two limit switches. Only one of the two switches (sb) is used in the example. The simplified finite-state machine model of the piston is shown in Fig. 4. An interesting feature of the piston is that it moves forwards in the case of a loss of supply pressure. Since the piston must be moved backwards before the table can start rotating, the potential loss of pressure represents an interesting problem from the

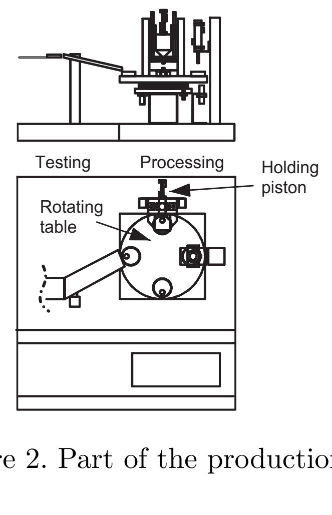


Figure 2. Part of the production line

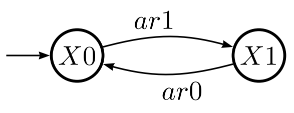


Figure 3. Model of the rotating table

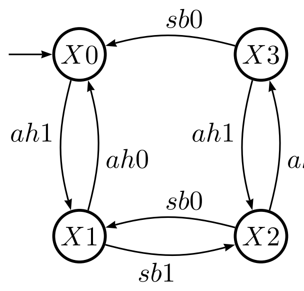


Figure 4. Model of the piston

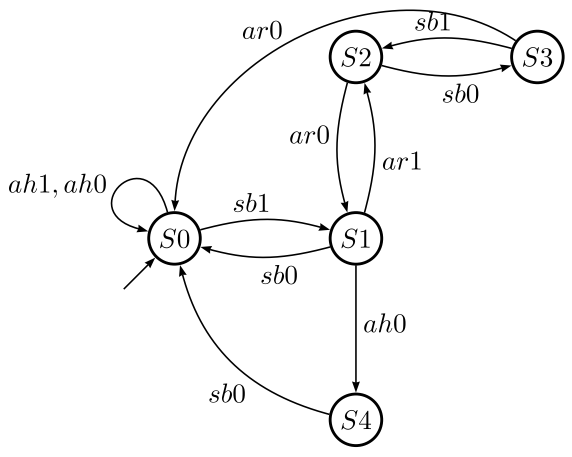


Figure 5. Table - piston interlock specification

control-design viewpoint.

To maintain the interlock between the table and the piston the behaviour represented by the automaton in Fig. 5 is imposed. The start of the table's rotation ($ar1$) is only allowed when the piston is drawn back, i.e., after $sb1$. If the supply pressure is lost, this would result in the forward movement of the piston and the event $sb0$ would be generated. Then, the only allowed action is to stop the rotation ($ar0$). (The requirement to immediately force the rotation stop cannot be carried out by the supervisor.) Another requirement that is imposed by the automaton in Fig. 5 is to prevent the controlled forward movement of the piston ($ah0$) while the table is rotating.

The specification is controllable and results in the admissible behaviour of the process, as shown in Fig. 6

Next, an operational procedure for the process is imposed. An example of the procedural specification is shown in Fig. 7. The interpretation of the places

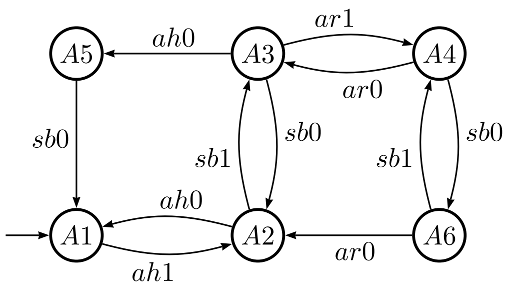


Figure 6. Admissible behaviour

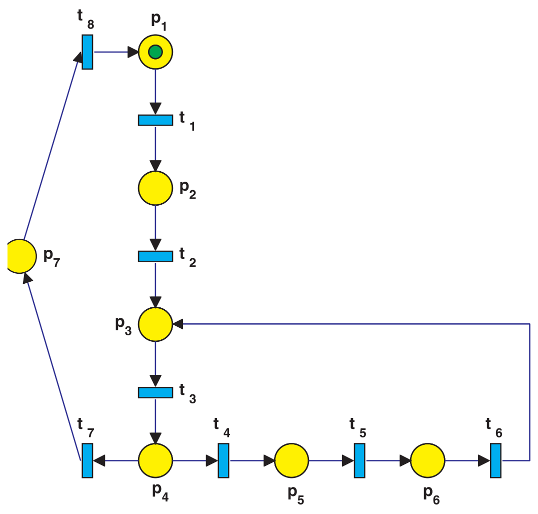


Figure 7. Specification of the operational procedure

and transitions is given in Tables 1 and 2. In the specified procedure, the operation is initiated by an external *start* signal. Then the piston is drawn back and afterwards the controller waits for another external signal (*cycle*), which initiates a cyclic operation. During the cycle, the table first starts rotating and when the position is reached, the piston is released in order to fix the workpiece in position. Next, the piston is drawn back again to release the workpiece and the controller returns to the wait state. If the signal *cycle* is set, the cycle repeats. The transition t_7 models a failure when the piston's position is changed during the table's movement, and the table must stop rotating immediately.

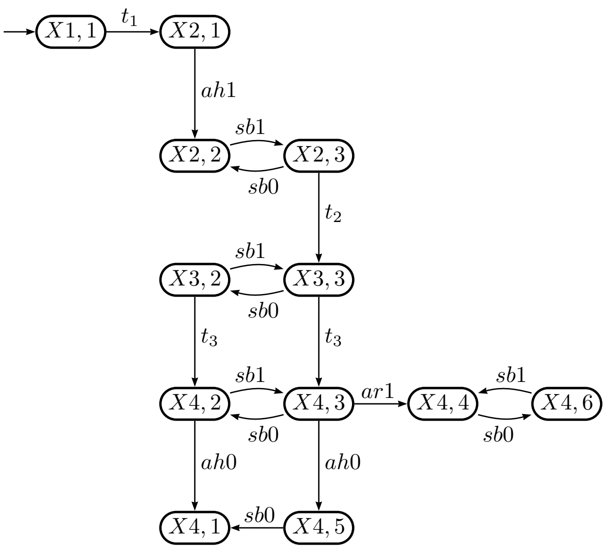
A RTPN defined in this way is verified against the previously derived open-loop process model. The initial states of the input signals that are not part of the admissible behaviour model can be left undefined or can be fixed at a specific value. In our case the signals *ack* and *sp* are set undefined, while *start* and *cycle* are assumed to be 1. The initial position of the piston is assumed to be in front ($sb = 0$). The initial state of all the output signals is assumed to be 0.

For the given case the constructed C-reachability graph consists of 12 nodes

Table 1
RTPN transition conditions

$Y(t_1)$=start
$Y(t_2)$=sb
$Y(t_3)$=cycle
$Y(t_4)$=sp AND NOT ah
$Y(t_5)$=NOT ar
$Y(t_6)$=sb
$Y(t_7)$=NOT sb
$Y(t_8)$=ack AND NOT sb
legend:
ack - error acknowledgement
cycle - start of the cycle
sb - back position sensor
sp - table position sensor
start - start of operation

and 17 transitions, and is shown in Fig. 8. An analysis of the graph shows the system operation is blocking after place p_4 is marked. This is because an attempt has been made to switch the *ah* signal off while the table is rotating. The supervisor blocks the required action and since the transition condition of t_4 implies that *ah* must be switched off before the transition is triggered, the operation is deadlocked.



$$\begin{tikzcd}
\arrow[r] & X1,1 \arrow[r, "t_1"] & X2,1 \arrow[d, "ah1"] & & & \\
& & X2,2 \arrow[r, bend left, "sb1"] & X2,3 \arrow[l, bend left, "sb0"] \arrow[d, "t_2"] & & \\
& & X3,2 \arrow[r, bend left, "sb1"] \arrow[d, "t_3"] & X3,3 \arrow[l, bend left, "sb0"] \arrow[d, "t_3"] & & \\
& & X4,2 \arrow[r, bend left, "sb1"] \arrow[d, "ah0"] & X4,3 \arrow[l, bend left, "sb0"] \arrow[r, "ar1"] \arrow[d, "ah0"] & X4,4 \arrow[r, bend right, "sb0"'] & X4,6 \arrow[l, bend right, "sb1"'] \\
& & X4,1 & X4,5 \arrow[l, "sb0"'] & &
\end{tikzcd}$$

Figure 8. C-reachability graph with deadlock

Table 2
RTPN place actions

$Z(p_1)$={(l_start, 1),(l_error, 0)}
$Z(p_2)$={(l_start, 0),(ah, 1)}
$Z(p_3)=\emptyset$
$Z(p_4)$={(ar, 1),(ah, 0)}
$Z(p_5)$={(ar, 0)}
$Z(p_6)$={(ah, 1)}
$Z(p_7)$={(l_error, 1),(ah, 0),(ar, 0)}
legend:
l_start - initial st. indicator
l_error - error indicator
ah - activate the piston
ar - table rotation

Table 3
Corrected RTPN transition conditions

$Y(t_1)$=start
$Y(t_2)$=sb
$Y(t_3)$=cycle
$Y(t_4)$=sp
$Y(t_5)$=NOT sb AND NOT ar
$Y(t_6)$=sb
$Y(t_7)$=NOT sb
$Y(t_8)$=ack AND NOT sb

To overcome the error, the specification is modified according to Tables 3 and 4. The newly constructed C-reachability graph consists of 25 nodes and 41 transitions, and is shown in Fig. 9. It is clear that the automaton can reach the initial state from any reachable state and that every transition of the RTPN occurs at least once as an event label in the graph. The application of the Propositions 1 and 2 on the graph, therefore, shows that the system operation is now deadlock free and C-live.

For the application of the estimate of the number of nodes for the given case, an estimate of the number of spontaneous events at every marking is needed first. In this case the task is relatively simple, since there are only two spontaneous events, $sb0$ and $sb1$, which are both related to a single input signal

Table 4
Corrected RTPN place actions

$Z(p_1)$={(l_start, 1),(l_error, 0)}
$Z(p_2)$={(l_start, 0),(ah, 1)}
$Z(p_3)=\emptyset$
$Z(p_4)$={(ar, 1)}
$Z(p_5)$={(ar, 0),(ah, 0)}
$Z(p_6)$={(ah, 1)}
$Z(p_7)$={(l_error, 1),(ah, 0),(ar, 0)}

(the events related to the signals *ack*, *cycle*, *sp* and *start* are not part of the admissible behaviour model and are not taken into account). Furthermore, no spontaneous events can occur during the initial marking ($m_0(p_1) = 1$). By also taking into consideration the RTPN output function, and again considering only events that take part in the model of the admissible behaviour, (4) gives an estimate of 31 for the number of nodes in the graph. Obviously, this is only a rather coarse estimate of the real number. If the interlock between $ar1$ and $ah0$ is taken into account, which applies when p_5 or p_7 is marked, (7) gives an estimate of 27, which is very close to the actual number of nodes in the constructed C-reachability graph for the given case.

5 Conclusions and future work

The algorithm for the calculation of the C-reachability graph presented in the paper enables a detailed analysis of the potential deadlock in discretely controlled processes. The prerequisite is that a discrete-event model of the plant should be available. Such a model can be easily obtained when the interlock layer is designed using the supervisory control theory, which gives a model of the admissible behaviour.

The potential applicability of the algorithm is limited by the complexity of the graph. Therefore, an estimate of the number of nodes in the graph was given. The proposed calculation makes it possible to estimate whether the construction of the graph is feasible. The exploration of techniques for the reachability analysis that do not involve the explicit enumeration of the state space is planned in a future work.

The combined synthesis/verification approach proposed, makes possible a relatively high level of automation of the control synthesis for manufacturing systems. Once the model of the plant and the specification models are developed an appropriate computer tool can perform all the necessary calculations

$$\begin{tikzcd}
\arrow[r] & (X1,1) \arrow[r, "t_1"] & (X2,1) \arrow[d, "ah1"] & \\
(X7,5) \arrow[r, "sb0"'] & (X7,1) \arrow[u, "t_8"] & (X2,2) \arrow[r, bend left, "sb1"] & (X2,3) \arrow[l, bend left, "sb0"] \arrow[d, "t_2"] \\
(X7,3) \arrow[u, "ah0"] \arrow[r, bend right, "sb0"'] & (X7,2) \arrow[u, "ah0"] \arrow[l, bend right, "sb1"'] & (X3,2) \arrow[r, bend left, "sb1"] \arrow[d, "t_3"] & (X3,3) \arrow[l, bend left, "sb0"] \arrow[d, "t_3"] \\
(X7,4) \arrow[u, "ar0"] \arrow[r, bend right, "sb0"'] & (X7,6) \arrow[u, "ar0"] \arrow[l, bend right, "sb1"'] & (X4,2) \arrow[r, bend left, "sb1"] & (X4,3) \arrow[l, bend left, "sb0"] \arrow[d, "ar1"] \\
 & & (X4,6) \arrow[ul, bend left, "t_7"] \arrow[r, bend left, "sb1"] \arrow[d, "t_4"] & (X4,4) \arrow[l, bend left, "sb0"] \arrow[d, "t_4"] \\
 & & (X5,6) \arrow[r, bend left, "sb1"] \arrow[d, "ar0"] & (X5,4) \arrow[l, bend left, "sb0"] \arrow[d, "ar0"] \\
 & & (X5,2) \arrow[r, bend left, "sb1"] \arrow[d, "ah0"] & (X5,3) \arrow[l, bend left, "sb0"] \arrow[d, "ah0"] \\
 & & (X5,1) \arrow[d, "t_5"] & (X5,5) \arrow[l, "sb0"] \\
 & & (X6,1) \arrow[d, "ah1"] & \\
 & & (X6,2) \arrow[r, bend left, "sb1"] & (X6,3) \arrow[l, bend left, "sb0"] \arrow[uuuuuuu, bend right, "t_6"']
\end{tikzcd}$$

Figure 9. C-reachability graph for the corrected case

and even generate the control code. Only a small amount of additional programming is then needed to obtain an operating logic controller.

References

[1] C. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Kluwer Academic Publishers, Dordrecht, 1999.

[2] V. Chandra, S. Mohanty, R. Kumar, Automated control synthesis for an assembly line using discrete event system control theory, in: Proceedings of the American Control Conference, Arlington, VA, 2001, pp. 4956–4961.

[3] G. Frey, L. Litz, Verification and validation of control algorithms by coupling of interpreted petri nets, in: Proc. 1998 IEEE International Conference on Systems, Man, and Cybernetics, vol. 1, San Diego, CA, 1998, pp. 7–12.

[4] G. Frey, L. Litz, Formal methods in plc-programming, in: Proc. 2000 IEEE International Conference on Systems, Man, and Cybernetics, vol. 4, Nashville, TN, 2000, pp. 2431–2436.

[5] H.-M. Hanisch, A. Lobov, J. M. Lastra, R. Tuokko, V. Vyatkin, Formal validation of intelligent automated production systems towards industrial applications, International Journal of Manufacturing Technology and Management 8 (2006) 892–904.

[6] H.-M. Hanisch, A. Lüder, J. Thieme, A modular plant modelling technique and related controller synthesis problems, in: Proc. 1998 IEEE International Conference on Systems, Man, and Cybernetics, vol. 1, San Diego, CA, 1998, pp. 686–691.

[7] T. Johnson, Improving automation software dependability: A role for formal methods?, Control Engineering Practice 15 (2007) 1403–1415.

[8] G. Morel, P. Valckenaers, J. Faure, C. Pereira, C. Diedrich, Manufacturing plant control challenges and issues, Control Engineering Practice 15 (2007) 1321–1331.

[9] T. Murata, Petri nets: Properties, analysis and applications, Proc. IEEE 77 (1989) 541–580.

[10] G. Mušič, D. Matko, Petri net control of systems under discrete-event supervision, in: ECC'03 European Control Conference, Cambridge, UK, 2003.

[11] G. Mušič, D. Matko, Combined synthesis/verification approach to programmable logic control of a production line, in: Preprints of the 16th IFAC World Congress, IFAC, Prague, Czech Republic, 2005.

[12] G. Mušič, B. Zupančič, D. Matko, Model based programmable control logic design, in: Preprints of the 15th Triennial IFAC World Congress, Barcelona, Spain, 2002.

[13] S. Peng, M. Zhou, Ladder diagram and petri-net-based discrete-event control design methods, IEEE Trans. on Systems, Man, and Cybernetics - Part C 34 (2004) 523–531.

[14] L. Pinzon, A. Jafari, H.-M. Hanisch, Modelling admissible behaviour using event signals, IEEE Trans. on Systems, Man, and Cybernetics - Part B 34 (2004) 1435–1448.

[15] M. Queiroz, J. Cury, Synthesis and implementation of local modular supervisory control for a manufacturing cell, in: Proc. 6th International Workshop on Discrete Event Systems (WODES'02), Zaragoza, Spain, 2002, pp. 377–382.

[16] P. Ramadge, W. Wonham, Supervisory control of a class of discrete event processes, SIAM J. Control and Optimization 25 (1987) 206–230.

[17] M. Rausch, B. Krogh, Formal verification of plc programs, in: Proceedings of American Control Conference, vol. 1, Philadelphia, PA, 1998, pp. 234–238.

[18] M. Rausch, A. Lüder, H.-M. Hanisch, Combined synthesis of locking and sequential controllers, in: Proc. 3rd International Workshop on Discrete Event Systems (WODES'96), Edinburgh, UK, 1996, pp. 133–138.

[19] A. Valmari, The state explosion problem, in: W. Reisig, G. Rozenberg (eds.), Lectures on Petri Nets I: Basic Models, vol. 1491 of Lecture Notes in Computer Science, Springer, 1998, pp. 429–528.

[20] W. Wonham, Notes on Control of Discrete Event Systems: ECE 1636F/1637S 2003-2004, Systems Control Group, Dept. of ECE, University of Toronto, 2003.

[21] M. Zhou, E. Twiss, Design of industrial automated systems via relay ladder logic programming and petri nets, IEEE Trans. on Systems, Man, and Cybernetics - Part C 28 (1998) 137–150.