ADMISSIBLE BEHAVIOUR BASED ANALYSIS OF DEADLOCK IN PETRI NET CONTROLLERS

Gašper Mušič[1], **Drago Matko**[1]

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

gasper.music@fe.uni-lj.si(Gašper Mušič)

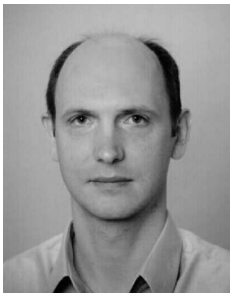
Abstract

The paper addresses the problem of verification of discrete control logic that is typically implemented by programmable controllers. To make the results of such a verification approach useful for the control, an adequate model of the process under control is needed, which is not readily available in many cases. To facilitate the derivation of the process model an approach is proposed in the paper, which combines the calculation of safety oriented interlock controllers in terms of supervisory control theory (SCT), the corresponding calculation of admissible behaviour of the system, and specification of desired system operation by Petri nets. The interlock part of the logic is designed by SCT while operational procedures are specified by a Petri net, extended by input and output mappings. A potential deadlock in the controlled system is then verified taking the admissible behaviour model as a process model. The analysis of the simultaneously operated supervisory control based interlock controller and Petri net based sequential controller is based on the C-reachability graph. The paper is focused on the calculation of the graph. A corresponding algorithm is presented and some remarks about computational complexity are given. The application of the algorithm is illustrated by a simple manufacturing cell example.

Keywords: Petri nets, modelling, manufacturing systems, logic controllers.

Presenting Author's Biography

Gašper Mušič received B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Ljubljana, Slovenia in 1992, 1995, and 1998, respectively. He is Associate Professor at the Faculty of Electrical Engineering, University of Ljubljana. His research interest are in discrete event and hybrid dynamical systems, supervisory control, and applications in industrial process control.



ISBN 978-3-901608-32-2

1 Introduction

While the functionality of programmable logic controllers (PLCs) is continuously expanding, the 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 ladder diagram [1]. Recently, much attention has been given to formal methods and their application in design and verification of PLC programs. This is motivated by the growing complexity of the control problems, demands for reduced development time and need for reuse of existing software modules on one hand, and on the other hand by increasing demand of society for a better control of technological risks.

Verification based approaches are dealing with formalization of the specifications and verification of the program against the formal specification [2]. The program passes the verification when the behaviour specified by the designer satisfies a set of properties. Properties can be checked on the control model only or by considering a model (possibly partial) of the process. The later is a more realistic approach of verification, called model-based [3].

To make the results of such a verification approach useful for the control, an adequate model of the process under control is needed, which is not readily available in many cases. Different aspects of plant modelling for the purpose of controller verification have been extensively studied in [4, 5, 6]. An approach is presented there, which enables a detailed and systematic modelling of controlled processes employing a special modelling formalism.

In special cases, however, a suitable model for verification may 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 [7] and is further elaborated in this paper. In particular, such an approach may 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 in sometimes referred as locking controller [8]. Assuming a two stage approach, where the interlock logic is designed first and the sequential part is then added atop of that, the admissible behaviour of the plant imposed by the interlock logic may be adopted as a plant model for verification of the sequential part.

In the presented approach the interlock part of the control logic is synthesized by the use of the supervisory control theory (SCT) [9, 10]. 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 by Petri nets [11], which are used in a sense of 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 a detection of any potential deadlock in the system that is controlled by simultaneously operated supervisory control based interlock controller and Petri net based sequential controller.

The motivation for the use of two modelling formalisms is twofold: First, the supervisory control theory is well suited for the interlock design. SCT is essentially safety oriented, i.e., it enables the synthesis of a control policy that prevents the undesired behaviour of the controlled plant. In most applications, however, there are also requirements about 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 [12, 13]. Secondly, the Petri net framework provides an intuitive way of modelling operation sequences, while the Petri net based supervisory control methods are less elaborated, especially in terms of event feedback, and little synthesis tools are available. The proposed combined approach exploits the advantages of both frameworks. Compared to other approaches that are described in the literature, the main advantage of the combined approach is that it eliminates the need for an additional plant model for the purpose of verification of the sequential controller. The corresponding model is derived automatically during the interlock design stage.

2 Combined synthesis/verification approach

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 input/output (I/O) signals.

2.1 Events, I/O signals, and admissible behaviour

The supervisory control concept [9] 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 – controllable and uncontrollable events: $\Sigma = \Sigma_c \cup \Sigma_u$, $\Sigma_c \cap \Sigma_u = \emptyset$. The uncontrollable events can not be disabled. The supervisor is computed based on the open-loop system model and a specification model. The key issues are the concept of controllability and the concept of supremal controllable sublanguage [10, 14].

The feasible set of input/output (I/O) signal patterns is defined by the supervisor S and is implicitly given by the discrete event model of the supervised plant in a form of a deterministic generator $H_a = (X, \Sigma, \delta, x_0, X_m)$, which is derived by the supervisory control synthesis procedure as a model of admissible behaviour.

Here X is a set of states, Σ is a set of symbols associated with events, $\delta : X \times \Sigma \rightarrow X$ is a state transition function and is in general a partial function on its do-

main, x_0 is the initial state and X_m is a set of marker states. A symbol $\sigma_i \in \Sigma$ is generated at 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 a set of all finite event sequences that may 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. Let Σ^* denote a set of all finite sequences of elements of Σ including the empty sequence, and let st denote a concatenation of sequences $s, t \in \Sigma^*$. A prefix closure of a 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 to reach a marker state from any reachable state, i.e., $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$.

As explained in [7], blocking is not considered at this point, therefore $X_m = X$. Language $\mathcal{L}(H_a) = L_a$ generated by H_a contains all admissible event sequences.

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

Considering 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 $W \subseteq X \times U \times V$ as defined above, Σ is the set of events composing the admissible behaviour, and ξ is the new state transition function defined as follows:

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

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 domain $W \times \Sigma$ to $W \times \Sigma^*$ in the usual way. Initial state is $w_0 = (x_0, u_0, v_0)$ and all states are marked, $W_m = W$. Note that $\mathcal{L}(H_w) = \mathcal{L}(H_a) = L_a$, which is evident from (1).

2.2 Specification of operational procedures

Petri nets as a tool for modelling and specification of manufacturing systems are described in a number of sources, such as [11, 10]. A Place/Transition Petri net can be described as a bipartite graph consisting of two types of nodes, places and transitions. Nodes are interconnected by directed arcs. State of the system is denoted by distribution of tokens (called marking) over the places. For the purpose of simulation and possible implementation by industrial controllers, the input/output interpretation can be added. One of such extensions is a class of Petri nets called *Real-Time Petri Nets* (RTPN) [15]. Formally, a RTPN is defined as an eight tuple $RTPN = (P, T, I, O, m_0, D, Y, Z)$ 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 $P \cap T = \emptyset$); $I : P \times T \to \mathbb{N}$ is a function that specifies weights of arcs directed from places to transitions; $O : P \times T \to \mathbb{N}$ is a function that specifies weights of arcs directed from transitions to places; $m : P \to \{0, 1, 2, \ldots\}$ is a marking, m_0 is the initial marking. $D : T \to \mathbb{R}^+$ is a firing time-delay function; $Y : T \to \mathcal{B}$ is an input signal function, where $\mathcal{B}$ is the set of Boolean expressions on the set B of input signals; $Z : P \to 2^{A \times \{0,1\}}$ is a physical output function, where A is the set of output signals.

In the following the paper only deals with safe RTPN, 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.

2.3 RTPN control of discrete-event process

To enable a detailed analysis of potential deadlock in a RTPN that is controlling a process under a restriction of a discrete event supervisor, the firing rule of a RTPN must be defined. A firing rule from [15] is here adopted with a slight modification.

In a standard Petri net theory a 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 which are inputs to a transition $t \in T$. This definition also holds for a RTPN but such a transition is called a *state enabled* transition. A set of state enabled transitions of a RTPN under marking m is $T_e(m) := \{t | t \text{ is state enabled under } m\}$.

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

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 $Z(p) = \{(a_1, i_1), \ldots, (a_n, i_n)\} \Rightarrow u(a_j) = i_j, \forall (a_j, i_j) \in Z(p), \forall p \in {}^\bullet t$. 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, which depends on the firing time-delay function associated with it: a transition with zero time delay fires immediately when enabled, a transition with non-zero time delay fires immediately after delay $D(t)$ expires (the corresponding timer starts when transition is enabled); (*iii*) a firing of a transition is immediate and 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 single transition firing at a time is assumed and $m[t\rangle m'$ denotes that t may fire under m, resulting in m'.

Given Petri net $\mathcal{N}$ and marking m, a marking m' is said to be immediately reachable ($m' \in R_1(\mathcal{N}, m)$) if there

ISBN 978-3-901608-32-2

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 ($m_k \in R(\mathcal{N}, m_0)$) 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 $0 < i \leq k$. The notion od reachability can be extended by considering 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 a 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 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 $\langle m_0 m_1 \dots m_k\rangle$ 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 $w = \xi(w_0, s); s \in L_a$.*

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 that enumerates all admissible event and transition sequences.

3.1 C-reachability graph

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 searched for. This is a subset of 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 events feasible at x_0 and not generated by the RTPN. Second is the set of events generated by 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 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 RTPN marked by m is then given by

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

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 case of conflicting transitions, all possible firing sequences are enumerated as in 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 new nodes are duplicate nodes.

In the described way a new kind of reachability graph is derived. A set of nodes is associated with every reachable marking and the transitions between nodes are of two types: (*i*) transitions of a RTPN connect nodes associated with distinct markings, (*ii*) transitions related to events in a model of admissible behaviour connect nodes associated with the same marking. Since the derived graph includes input and output events of a controller it is called the *C-reachability graph* of a RTPN controller. It must be noted that only the ordering of events is considered, while timing information of a RTPN is omitted.

The resulting graph can be interpreted as an automaton where 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 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 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 a 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 transition set T and acting on a discrete-event system H_w is deadlock free if a 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 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) may 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 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 not sufficient for the RTPN being deadlock free as the initial state may 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. Commonly it is also required for all 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 initial marking. Such a Petri net is live [11]. To adapt this notion to sequential specification in terms of 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 as the absence of deadlocks. The difference is that since any transition must be eventually fired, all 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 transitions of the RTPN, i.e. $\forall t \in T, \exists n, n' \in N, n' = \zeta(n,t)$

(ii) may 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 transitions appear as the labels of arcs of CG this corresponds to 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 RTPN can eventually be fired, starting from every reachable marking. ■

Note that although in general most of the spontaneous events is uncontrollable in the sense of supervisory control and most of the controlled events is also controllable, this correspondence is not strict. An uncontrollable event $\sigma_1 \in \Sigma_u$ may be generated by the RTPN, therefore $\sigma_1 \in \Sigma_{CTRL}$, e.g., the start of an emergency procedure, which must not be disabled. On the other hand a controllable event $\sigma_2 \in \Sigma_c$ may be generated externally ($\sigma_2 \in \Sigma_{SP}$), e.g., an operator request that may be blocked by the supervisor.

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 a 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 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 Petri net transition. The procedure is summarized in the following algorithm:

Algorithm 1:

w := (x_0, u_0, v_0);
N_{CG} := (m_0, w); (* a node related to initial marking of a RTPN, initial state of the automaton, and initial state of I/O signals *)
CG.N := $\{N_{CG}\}$;
CG.A := $\emptyset$;
RSET := $\{m_0\}$;
(* events that are not triggered by RTPN *)
Σ_{SP} := spontaneous(G, 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 *)
Σ_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 properties of the process considered and the related operational procedure specification. However, some conclusions can be made 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. The number of nodes will therefore be of the primary concern.

The number of nodes in the graph is typically mostly related to the complexity of 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 events in the admissible behaviour model are related to I/O signals of the RTPN is considered. Furthermore, the given estimation 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.

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 that can switch at particular marking is taken into account, and there is no information on their restrictions, all possible orderings are considered. The maximum number of added nodes is then $N_s!$. An additional node is inserted for every transition firing. Let n_p denote the number of places in the RTPN, and let N_{ti} denote the number of outgoing transitions from place p_i. The estimated upper bound for the number of nodes in the graph is then

$$N_{nodes} \leq \sum_{i=1}^{n_p} N_s! \cdot N_{t_i} \qquad (3)$$

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 place, i.e the number of elements in $Z(p_i)$. 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). Further look at equation (2) shows that N_s is actually 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 $N_{SP}(p_i)$ denote this estimate and let $N_A(p_i)$ denote a number of actions related to marked place p_i. The estimate of the upper bound for the number of nodes is then

$$N_{nodes} \leq \sum_{i=1}^{n_p} (N_{SP}(p_i) + N_A(p_i))! \cdot N_{t_i} \qquad (4)$$

The estimate is rather conservative, because all event orderings were considered. To achieve a better estimate, a restrictions on the event orderings given by admissible behaviour model should be taken into account.

4 Example

To illustrate the concept of C-reachability graph an example is given, which is simple enough to exhibit interesting properties, but in the same time based on the equipment used in industrial applications. The example deals with a part of a laboratory scale modular production line composed of five working stations controlled by five programmable logic controllers [13].

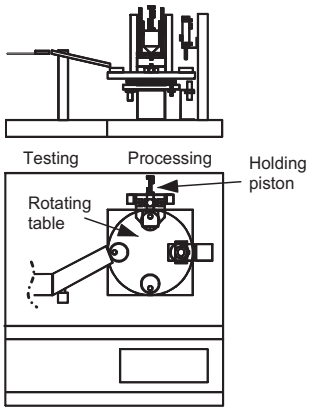


Fig. 1 Part of the production line

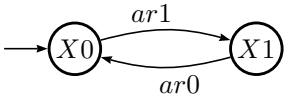


Fig. 2 Model of the rotating table

The stations perform distribution of workpieces, testing of workpieces, processing, manipulation and sorting. Every working station is further composed of a set of pneumatic pistons, gears, two state sensors, electropneumatic 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 consequent phases, a drilling machine, and a testing device. Next working station includes a manipulator that transports the workpiece further. The setup is shown in Fig. 1.

The setup is similar to the one used in [16], except that a much closer view to the process is taken in this paper. In [16] the SCT is used to coordinate the operating phases, while the example presented here is dealing with the control logic inside a particular phase. Switching of I/O signals in desired operational procedures is modelled as well as the behaviour in erroneous conditions in the process.

To keep the presentation simple enough a particular detail will be studied, i.e. the interlock between rotating table an a holder that fixes a workpiece before it is drilled. The table is driven by an electric motor, switched on ($ar1$) and off ($ar0$). The table has four stop positions, indicated by a proximity switch sp. The switch closes ($sp1$) when the table comes into a 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. 2.

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 $af = 1$ and forwards when $af = 0$. The piston is equipped by two limit switches, indicating backward (sb) and forward (sf) position. 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. 3. The interesting feature of the piston is that moves forwards in case of the loss of supply pressure. Since the piston must be moved backwards before the table can start rotating, the potential loss of pressure presents an interesting problem from the control design viewpoint.

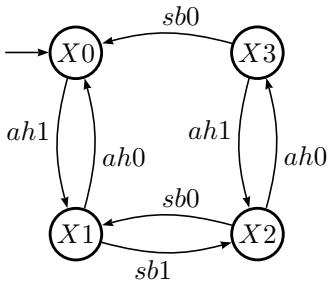


Fig. 3 Model of the piston

To maintain the interlock between the table and the piston the behaviour presented by automaton in Fig. 4 is imposed. The start of the table rotation ($ar1$) is only allowed when the piston is drawn back ($sb = 1$). If the supply pressure is lost, this would result in $sb0$. The only allowed action is then to stop rotation ($ar0$). (The requirement to immediately force the rotation stop can not be achieved by the supervisor). Another requirement that is imposed by automaton in Fig. 4 is to prevent controlled forwards 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. 5

Next, an operational procedure for the process is imposed. An example of the procedural specification is shown in Fig. 6. The interpretation of places and transitions is given in Tabs. 1 and 2.

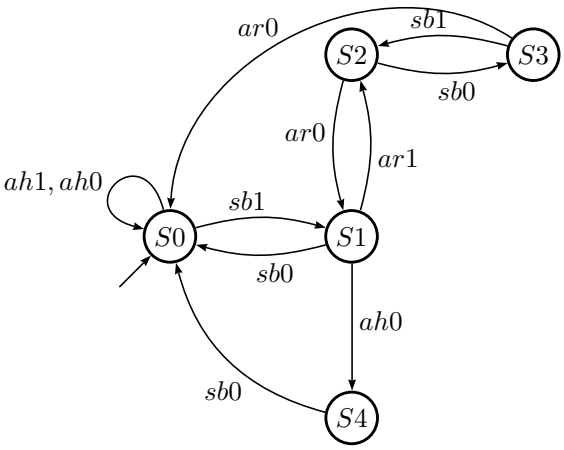


Fig. 4 Table - piston interlock specification

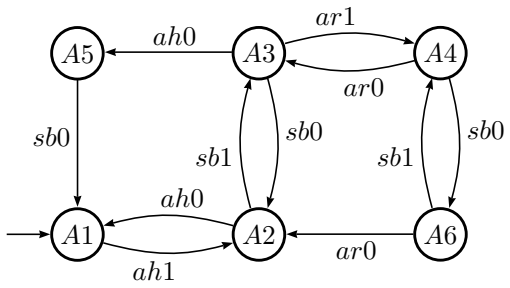


Fig. 5 Admissible behaviour

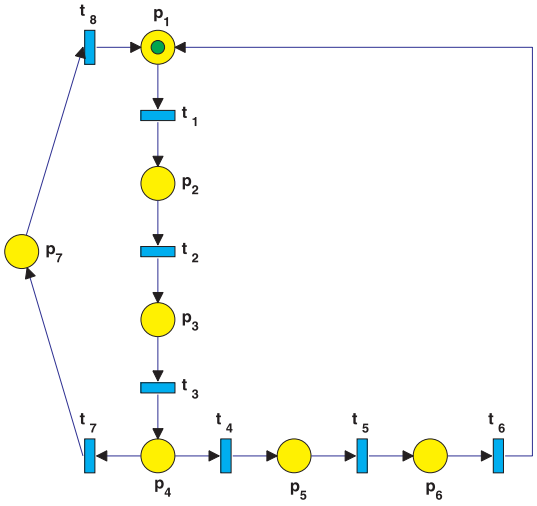


Fig. 6 Specification of the operational procedure

Tab. 1 RTPN transition conditions

$Y(t_1)$=start
$Y(t_2)$=sb
$Y(t_3)$=cycle
$Y(t_4)$=sp
$Y(t_5)$=NOT sb
$Y(t_6)$=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

Tab. 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

A RTPN defined 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 may be left undefined or may be fixed at specific value. In our case signals ack and sp are set undefined, while $start$ and $cycle$ are assumed to be 1. The initial position of the piston is assumed in front ($sb = 0$). The initial state of all output signals is assumed to be 0.

For the given case the constructed C-reachability graph consists of 12 nodes and 17 transitions, and is shown in Fig. 7. The 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 firing rule of the RTPN assumes all actions are completed before an outgoing transition is triggered, the operation is deadlocked.

To overcome the error, the specification is modified according to Tab. 3. The newly constructed C-reachability graph consists of 25 nodes and 41 transitions, and is shown in Fig. 8. It can be observed 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 the system operation is now deadlock free and C-live.

For the application of the estimate of the number of nodes to the given case, an estimate of the number of spontaneous events at every marking is first needed. In this case the task is relatively simple since there are only two spontaneous events: $sb0$ and $sb1$ (events related to 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 at the initial marking ($m_0(p_1) = 1$). By taking into consideration also the RTPN output function, and again considering only events that take part in the model of the admissible behaviour, (4) gives an estimate of 75 for the number of nodes in the graph. Obviously, this is only a very coarse estimate of the real number.

Tab. 3 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)}

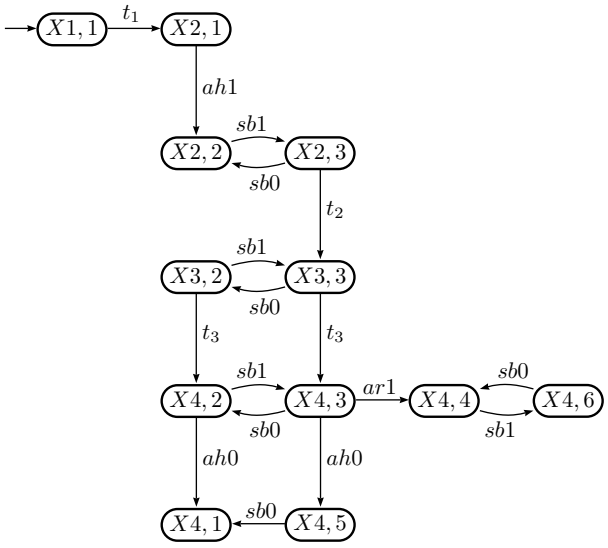


Fig. 7 C-reachability graph with deadlock

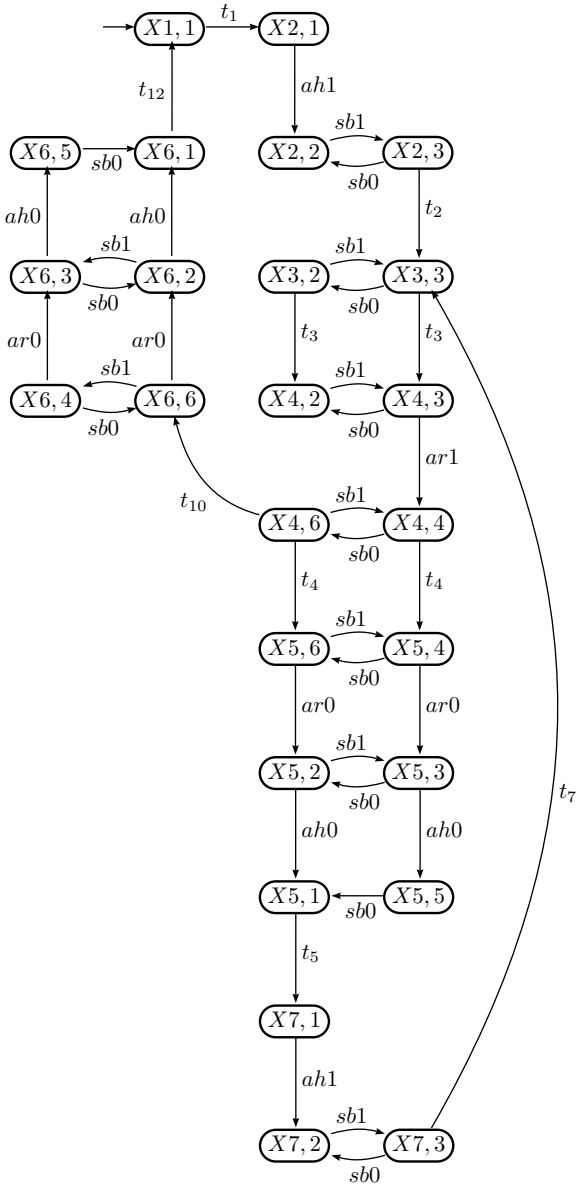


Fig. 8 C-reachability graph for the corrected case

5 Conclusions and future works

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

The potential applicability of the algorithm is limited by the complexity of the graph. Therefore an estimate for the number of nodes in the graph was given. Although very coarse, it enables to estimate whether the construction of the graph is feasible. An improvement of the estimate is foreseen for the future work. Another issue for the future work is to explore techniques for reachability analysis without the explicit enumeration of the state-space.

The combined synthesis/verification approach enables a relatively high automation of the control synthesis for the manufacturing systems. Once the model of the plant and the specification models are developed an appropriate computer tool may perform all the necessary calculations and even generate the control code. Only a small amount of additional programming is then needed to obtain an operating logic controller.

6 References

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

[2] G. Frey and L. Litz. Formal methods in plc-programming. In *Proc. of the SMC'2000*. 2000.

[3] M. Rausch and B.H. Krogh. Formal verification of plc programs. In *Proceedings of American Control Conference*, pages 234–238, Philadelphia, PA, USA, June 1998.

[4] H.-M. Hanisch, A. Lüder, and J. Thieme. A modular plant modelling technique and related controller synthesis problems. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, pages 686–691, October 1998.

[5] L. Pinzon, A. Jafari, and H.-M. Hanisch. Modelling admissible behaviour using event signals. *IEEE Trans. on Systems, Man, and Cybernetics - Part C*, 34:1435–1448, 2004.

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

[7] G. Mušič and D. Matko. Petri net control of systems under discrete-event supervision. In *ECC'03 European Control Conference*. Cambridge, UK, 2003.

[8] M. Rausch, A. Lüder, and H.-M. Hanisch. Combined synthesis of locking and sequential controllers. In *Int. Workshop on Discrete Event Systems (WODES96)*, pages 133–138, Edinburgh, UK, Aug. 1922 1996.

[9] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25:206–230, 1987.

[10] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Dordrecht, 1999.

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

[12] V. Chandra, S.R. Mohanty, and R. Kumar. Automated control synthesis for an assembly line using discrete event system control theory. In *Proceedings of the American Control Conference*, pages 4956–4961. Arlington VA, 2001.

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

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

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

[16] M.H. Queiroz and J.E. Cury. Synthesis and implementation of local modular supervisory control for a manufacturing cell. In *Proc. 6th International Workshop on Discrete Event Systems (WODES'02)*, pages 377–382, Zaragoza, Spain, October 2002. IEEE Computer Society.

ISBN 978-3-901608-32-2