Model Checking Verification of Logic Control Programs in SIPN

Xiying Weng and Lothar Litz

Institute of Process Automation
Dept. of Electrical and Computer Engineering
University of Kaiserslautern
PO 3049, D-67653 Kaiserslautern, Germany
Email:weng@eit.uni-kl.de

Abstract: Symbolic model checking is a promising verification technique for industrial applications. In the area of automation, this technique appears also useful for verifying logic control programs. In this contribution, applying this technique to the verification of a logic control program, which is described in the SIPN (Signal Interpreted Petri Net) framework, is addressed. Based on the isomorphism between Boolean algebra and the SIPN framework, a Boolean formula to represent transition relations of an SIPN is derived, so that applying symbolic model checker for verifying an SIPN program is made possible. Finally, an industrial example is used to illustrate this approach.

Keywords: CTL, symbolic model checking, Petri net, logic control program.

1. Introduction

Today software systems are widely used in automation field, such as logic control programs, so the reliability of software becomes very important. The traditional ways to verify the correctness are testing and simulation, which usually involve providing certain inputs and observing the corresponding outputs. These methods can be cost-efficient ways to find many errors. However, the consideration of all the possible input patterns and interactions is rarely possible. Due to the mathematical exactitude and completeness, formal methods are receiving more and more interest in both academic research and industry. Now one of the most promising automatic verification techniques for industrial applications is symbolic model checking [5][12], which can treat the state explosion problem in some extent.

On the other hand, a Signal Interpreted Petri Net (SIPN) framework was introduced for logic control algorithm design [9] [8]. This framework has shown several advantages. It has not only preserved the ordinary Petri net properties, such as compact description of concurrency and causality of the transition relations, but has also intuitively defined input/output facilities to the controlled process. Additionally, an SIPN can be implemented automatically in a standard logic control programming language [7] (so here we use a logic control program and a logic control design indistinctly). The usual way to determine the properties of an SIPN is using the Petri net theory, which is based on the traversal of the reachability graph explicitly[6], that may suffer from the state explosion problem. Furthermore, many dynamic features of a control system are difficult to be expressed in such standard properties of a Petri net.

In [14], experimental case studies have been performed to determine the properties of an SIPN design through temporal logic model checking approach. In this contribution, the theoretical background of this approach is explored. The paper is organized as follows: In section 2, the SIPN framework is briefly introduced. Section 3 sketches the mathematical fundamentals, which are required to establish the isomorphism between an SIPN and Boolean algebra. The symbolic representation of transition relations of an SIPN is described in section 4. Model checking of the properties of an SIPN program expressed in CTL is discussed in section 5. In section 6, an example of control system is presented to illustrate this approach. We conclude in section 7 with a discussion.

2. SIPN framework

Definition: An SIPN is given by an eight-tuple SIPN = (P, T, F, M_0, I, O, C, A) with the subset (P, T, F, M_0) as the ordinary PN, where $P = \{p_1, p_2, \cdots p_n\}$ and $T = \{t_1, t_2, \cdots t_m\}$ are sets of places and transitions satisfying $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$. F: $(P\times T) \cup (T\times P)$ is set of directed arcs, and M_0 an initial marking. Sets I, O, C, A are defined as follows:

I: a non-empty finite set of binary input signals;

O: a non-empty finite set of binary output signals with $I \wedge O = \emptyset$;

C: a mapping associating every transition $t_i \in T$ as a firing condition in a Boolean expression in I;

A: a mapping associating every place $p_i \in P$ with an action $A(i) \in \{0,1,-\}^{|o|}$. A(i)[j] specifies the value of the output signal o_j in the place p_i. And "-" means "don't care".

A firing of a transition leads to the evolving of system states, which are represented by a series of markings. A transition is said to be enabled when each input place holds a token and each output place holds no token (**strict firing rule**) (not included is the self-loop case). The firing of an enabled transition is governed by its associated firing condition, which is usually a Boolean function of input signals. As soon as the condition is evaluated to be 1, the enabled transition fires immediately. The firing of a transition removes a token from each input place and adds a token in each of its output places. For a place in a self-loop case, that is this place is both an input and output place of a transition, the firing of this transition causes no change of its token. Thus, the change of tokens in a place p_i can be given in (1) with respect to the firing of transition t:

$$\forall p_i \in P: \qquad m'(p_i) = \begin{cases} m(p_i) - 1 & p_i \in {}^{\bullet}t \setminus t^{\bullet} \\ m(p_i) + 1 & p_i \in t^{\bullet} \setminus {}^{\bullet}t \\ m(p_i) & \text{otherwise} \end{cases} \qquad (1)$$

Here $m'(p_i)$ denotes the successor marking reached from the marking $m(p_i)$ by one step. ${}^{\bullet}t$ denotes the set of input places of transition t, while $t^{\bullet}$ the set of output places of transition t. The set operator "\" is defined as $S\setminus T = \{ x \mid x \in S \text{ and } x \notin T \}$, in which S,T are two sets. So under this strict firing rule an SIPN is always **safe**. This also holds for the self-loop case.

In order to avoid the generation of the explicit reachability graph, which may turn out to be very large, we try to describe the net behavior symbolically, so that symbolic model checking verification techniques can be applied. For this, Boolean functions will be used. The applied main features of mathematical fundamentals [1] are given in the following section.

3. Mathematical fundamentals

3.1 Boolean Algebra

A Boolean algebra is defined as a quintuple (B, +, ·, 0, 1), in which B is a set, called the carrier; + and · are binary operations on B; and 0 and 1 are distinct members of B. Thus, the system ({0, 1}, +, · 0, 1) is a Boolean algebra with + and · defined as logic OR and logic AND operations, respectively. It is also called switching algebra. Because our application limits to Boolean functions, so we always assume that B = {0, 1}.

3.2 n-Variable Boolean Functions

For all $(x_1, x_2, \cdots x_n) \in B^n$, the Boolean function $f(x_1, x_2, \cdots x_n)$ is a mapping

$$f(x_1, x_2, \cdots x_n): B^n \to B.$$

Let $F_n(B)$ be the set of n-variable logic functions on B, then the system $(F_n(B), +, \cdot, 0, 1)$ is also a Boolean algebra, in which + and · signify addition and multiplication of Boolean functions, respectively. And 0 and 1 signify the zero- and one-functions, respectively ($f(x_1, x_2, \cdots x_n) = 0$ and $f(x_1, x_2, \cdots x_n) = 1$).

3.3 Boole's Expansion Theorem

Theorem: If $f : B^n \to B$ is a Boolean function, then for all $(x_1, x_2, \cdots x_n) \in B^n$:

$$f(x_1, x_2, \cdots x_n) = x_i' \cdot f(x_i') + x_i \cdot f(x_i)$$

in which

$$f(x_i') = f(x_1, x_2, \cdots, x_{i-1}, 0, x_{i+1}, \cdots x_n)$$
$$f(x_i) = f(x_1, x_2, \cdots, x_{i-1}, 1, x_{i+1}, \cdots x_n)$$

are called cofactors of f with respect to x_i' and x_i respectively.

The Quantified Boolean Formulas QBF [4] can be defined using this expansion theorem. For a Boolean formula $f(x_1, x_2, \cdots x_n)$:

$$\exists x_i: \quad f = f|_{x_i=0} + f|_{x_i=1}$$
$$\forall x_i: \quad f = f|_{x_i=0} \cdot f|_{x_i=1}$$

Based on the above mathematical fundamentals, the isomorphism between the set of markings of an SIPN and Boolean functions can be established.

3.4 Fundamental Isomorphism

As known, a marking m in an SIPN can be represented by a set of places p_i, where every place holds a token. We denote a set of markings as M, then M is a subset of the set of places in a Petri net $M \subset P$, the number of all possible markings in M is $|M| = 2^{|P|}$. So the system $(2^M, \cup, \cap, \emptyset, M)$ is a Boolean algebra of the set of markings, in which 2^M denotes the power set of the set M, $\cup$ denotes set-union, $\cap$ denotes set-intersection. This system is therefore isomorphic with the Boolean algebra of n-variable Boolean functions. Since there is a one-to-one correspondence between markings in M and vertices of B^n, we use p_i to denote both a variable in the Boolean function and a place in P indistinctly. A marking can be encoded by the cube of B^n as follows:

$$M = \bigcap_{i=1}^{|P|} \begin{cases} p_i & \text{if } p_i \text{ has a token} \\ p_i' & \text{if } p_i \text{ has no token} \end{cases} \tag{2}$$

For example, if an SIPN has 5 places, then a marking $m_i = \{p_1, p_3\}$ can be represented as $m_i = p_1 p_2' p_3 p_4' p_5'$.

A set of markings can be described with its characteristic function. A characteristic function of a set is defined as follows:

Definition: Let $S \subseteq B^n$ be a set of vertices in the Boolean algebra of n-variable logic functions. The characteristic function λ_S of a set S is defined as a logic function, which is 1 for those vertices of B^n that are in S, i.e.

$$s \in B^n, s \in S \iff \lambda_S(s) = 1$$

Thus, for a set of markings $M = \{\{p_1, p_2\}, \{p_4\}, \{p_3, p_4, p_5\}\}$ of an SIPN with 5 places, the characteristic function of this set is given as:

$\lambda_M = p_1 p_2 p_3' p_4' p_5' + p_1' p_2' p_3' p_4 p_5' + p_1' p_2' p_3 p_4 p_5$.

Therefore, by means of the characteristic function of a set, it is possible to convert a set manipulation into a Boolean algebra calculation.

4. Boolean Representation of Transition Relations

Based on the above encoding method, transition relations of an SIPN can be described in an algebra form. Here two sets of variables are used: $P=\{p_1, p_2, \ldots p_n\}$ to represent the marking m before transition t fires, and $Q=\{q_1, q_2, \ldots q_n\}$ for the marking m' after the firing of transition t. According to the strict firing rule, the marking evolution in the place p_i can be described with respect to the firing of transition t_j:

$$\varepsilon_i(p_1, p_2, \cdots p_n, t_j) = \begin{cases} p_i' q_i & p_i \in t_j^{\bullet} \setminus {}^{\bullet}t_j \\ p_i q_i' & p_i \in {}^{\bullet}t_j \setminus t_j^{\bullet} \\ p_i q_i & p_i \in {}^{\bullet}t_j \cap t_j^{\bullet} \\ q_i = p_i & \text{otherwise} \end{cases} \tag{3}$$

ε_i denotes the change of the marking in place p_i, when transition $t_j \in T$ fires. And $q_i = p_i$ stands for the equivalence and is defined as $p_i q_i + p_i' q_i'$. For a place $p_i \in {}^{\bullet}t_j \cap t_j^{\bullet}$, it means that p_i and t_j is a self-loop.

So the transition relation with respect to t_j in an SIPN can be specified as:

$$R_{t_j}(p_1, p_2, \cdots p_n, q_1, q_2, \cdots q_n) = \bigcap_{i=1}^{|P|} \varepsilon_i(p_1, p_2, \cdots p_n, t_j) \cdot C_{t_j}(I) \tag{4}$$

where $C_{t_j}(I)$ denotes the Boolean function of input signals $I = \{i_1, i_2, \cdots i_n\}$ labeled on the transition t_j.

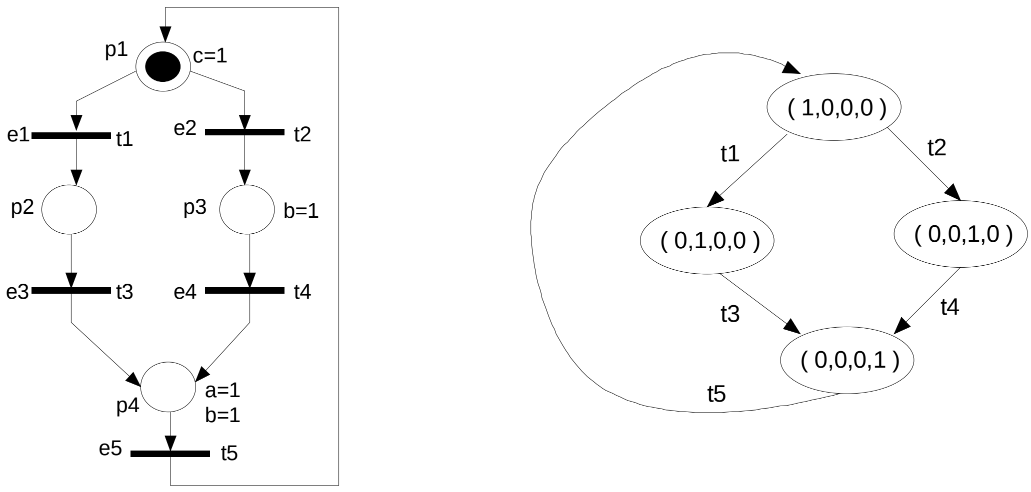
Consequently, the whole transition relations of an SIPN can be given as follows:

$$R_t(p_1, p_2, \cdots p_n, q_1, q_2, \cdots q_n) = \bigcup_{\forall t_j \in T} R_{t_j} = \bigcup_{\forall t_j \in T} [\bigcap_{i=1}^{|P|} \varepsilon_i(p_1, p_2, \cdots p_n, t_j) \cdot C_{t_j}(I)\,] \tag{5}$$

Based on this transition relation, the successor markings m' can be determined according to the following function:

$$m' = \exists p_1 \cdots \exists p_n [\, R_t(p_1, p_2 \cdots p_n, q_1, q_2 \cdots q_n) \cdot m] \tag{6}$$

The following example of an SIPN design as shown in Fig.1 (a) will illustrate how successor markings from the initial marking $m_0 = \{\, p_1 \,\} = p_1 p_2' p_3' p_4'$ can be determined.



(a) SIPN

(b) reachability graph

Fig. 1

According to the transition relation given in (4), the following functions can be obtained:

$$R_{t_1} = p_1 q_1' \cdot p_2' q_2 \cdot (q_3 = p_3) \cdot (q_4 = p_4) \cdot e_1$$
$$R_{t_2} = p_1 q_1' \cdot (q_2 = p_2) \cdot p_3' q_3 \cdot (q_4 = p_4) \cdot e_2$$
$$R_{t_3} = (q_1 = p_1) \cdot p_2 q_2' \cdot (q_3 = p_3) \cdot p_4' q_4 \cdot e_3$$
$$R_{t_4} = (q_1 = p_1) \cdot (q_2 = p_2) \cdot p_3 q_3' \cdot p_4' q_4 \cdot e_4$$
$$R_{t_5} = p_1' q_1 \cdot (q_2 = p_2) \cdot (q_3 = p_3) \cdot p_4 q_4' \cdot e_5$$

So the whole transition relation in the SIPN according to (5) is:

$$R_t = \bigcup_{t_i \in T} R_{t_i}$$

Assuming $e_1 = e_2 = e_3 = e_4 = e_5 = 1$, then

$$\begin{aligned} m_1 &= \exists p_1 \exists p_2 \exists p_3 \exists p_4 \,(R_t \cdot m_0) \\ &= \exists p_1 \exists p_2 \exists p_3 \exists p_4 (p_1 q_1' \cdot p_2' q_2 \cdot p_3' q_3' \cdot p_4' q_4' + p_1 q_1' \cdot p_2' q_2' \cdot p_3' q_3 \cdot p_4' q_4') \\ &= q_1' q_2 q_3' q_4' + q_1' q_2' q_3 q_4' \\ &= \{p_2\} + \{p_3\} \end{aligned}$$

$$\begin{aligned} m_2 &= \exists p_1 \exists p_2 \exists p_3 \exists p_4 (R_t \cdot m_1) \\ &= \exists p_1 \exists p_2 \exists p_3 \exists p_4 (p_1' q_1' \cdot p_2 q_2' \cdot p_3' q_3' \cdot p_4' q_4 + p_1' q_1' \cdot p_2' q_2' \cdot p_3 q_3' \cdot p_4' q_4) \\ &= q_1' q_2' q_3' q_4 \\ &= \{p_4\} \end{aligned}$$

$$
\begin{aligned}
m_3 &= \exists p_1 \exists p_2 \exists p_3 \exists p_4 (R_t \cdot m_2) \\
&= \exists p_1 \exists p_2 \exists p_3 \exists p_4 (p_1' q_1 \cdot p_2' q_2' \cdot p_3' q_3' \cdot p_4 q_4') \\
&= q_1 q'_2 q_3' q'_4 \\
&= \{p_1\}
\end{aligned}
$$

The first iteration m_1 produces all the markings reached from m_0 by one step. The second iteration generates all the successors from m_1. The third iteration m_3 generates the initial marking again, that is, no new markings can be reached. This can be viewed as a breadth-first search. The set of all reachable markings is depicted in Fig.1 (b).

From above we see that the transition relations in an SIPN can be successfully described with Boolean functions, and the reachable states can also be calculated symbolically.

5. CTL Symbolic Model Checking and tool SMV

CTL (Computation Tree Logic) [3] is based on the propositional logic and extended with temporal operators and path quantifiers. CTL can express the temporal properties of a system. The case study in [14] has shown that the functional properties as well as the properties relating to the formalism of an SIPN can be expressed in CTL. The output signals labeled on a place of an SIPN can be treated as atomic propositions. In this section the determining of the truth of an CTL formula based on the transition relations of an SIPN will be explained

In CTL two path quantifiers A (for all computation path), E (for some computation path) precede directly the usual forward-time operators of linear temporal logic[10]: G (globally or invariantly), F (sometime in the future), X (next time) and U (until). Here are some basic CTL formulas: $EX(\varphi)$ holds in a state, if there exits a successor of that state, in which φ holds; $EG(\varphi)$ means that there exists at least one path, along which the formula φ holds globally starting from that state; $E(\phi \text{ U } \varphi)$ holds in a state if and only if there is a path, in which ϕ holds until φ holds. The other typical CTL formulas can be expressed in terms of the above three formulas:

$$
\begin{gathered}
EF\varphi = E(\text{TRUE} \cup \varphi) \\
AX\varphi = \neg EX \neg \varphi \\
AF\varphi = \neg EG \neg \varphi \\
AG\varphi = \neg EF \neg \varphi \\
A(\phi U \varphi) = E[\neg \varphi U \neg \phi \wedge \neg \varphi] \wedge \neg EG \neg \varphi
\end{gathered}
$$

Therefore, it is only necessary to illustrate the computation of $EX(\varphi), EG(\varphi), E(\phi \text{ U } \varphi)$ in this paper.

The operators in CTL have simple characterizations as fixed points that allow us to compute efficiently whether a formula is satisfied in a given state in a given finite model. This calculation is termed model checking. By symbolic model checking, a finite system is implicitly represented with a Boolean formula. The model checking algorithms in available tools are usually manipulated based on OBDD representations of Boolean functions [2] [11], such as the model checker SMV [13]. Because of the limitation of the scope, the OBDD aspect will be omitted here, but this will not prevent us from explaining the basic concepts how a model checker determines the truth of CTL formulas using fixed point characterizations. For the example in section 4, we will verify the truth of the following three formulas with respect to the given initial marking $m_0 = \{ p_1 \} = p_1 p_2' p_3' p_4'$, since the dynamic behavior of an SIPN depends on the initial marking. Here a, b and c are output signals labeled on some places (see Fig.1 (a)).

(1) $EX(b)$

EX(b) is valid for the marking m_0 if and only if there are some successors of m_0, in which b holds. That is

$$m_0 \models EX(b) \qquad \text{iff} \qquad m_0 \in (\exists Q(R_t \wedge b(P \leftarrow Q)) \tag{7}$$

Now we calculate the set of markings that satisfy the $\exists Q(R_t \wedge b(P \leftarrow Q)$ using the characteristic function:

$$\begin{aligned} \lambda_{SAT(EX(b))} &= \exists Q[\lambda_{SAT(b)}(Q) \wedge R_t] \\ &= \exists Q[p_1 q_1' \cdot p_2' q_2 \cdot p_3 q_3 \cdot (p_4 = q_4) \cdot e_1 + p_1 q_1' \cdot (p_2 = q_2) \cdot p_3' q_3 \cdot (p_4 = q_4) \cdot e_2 \\ &\quad + p_1' q_1 \cdot (q_2 = p_2) \cdot p_3 q_3 \cdot p_4 q_4' \cdot e_5 + p_1 q_1' \cdot p_2' q_2 \cdot (q_3 = p_3) \cdot p_4 q_4 \cdot e_1 \\ &\quad + (q_1 = p_1) \cdot p_2 q_2' \cdot (q_3 = p_3) \cdot p_4' q_4 \cdot e_3 + (q_1 = p_1) \cdot (q_2 = p_2) \cdot p_3 q_3' \cdot p_4' q_4 \cdot e_4] \end{aligned} \tag{8}$$

where SAT (ϕ) denotes a collection of elements, which satisfy ϕ. Assuming $e_1 = e_2 = e_3 = e_4 = e_5 = 1$, the following table shows all the possible markings, which satisfy $EX(b)$ and the corresponding value of Q. Therefore, the initial marking is also among the set (see the first line). That is, $m_0 \models EX(b)$.

p_4	**p_3**	**p_2**	**p_1**	**q_4**	**q_3**	**q_2**	**q_1**
0	**0**	**0**	**1**	**0**	**1**	**0**	**0**
0	0	1	0	1	0	0	0
0	0	1	1	0	1	1	0
0	1	0	0	1	0	0	0
0	1	0	1	0	1	1	0
				1	0	0	1
0	1	1	0	1	1	0	0
				1	0	1	0
0	1	1	1	1	1	0	1
				1	0	1	1
1	0	0	1	1	1	0	0
				1	0	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	0	1	1	1

Table 1

(2) $E(a\,U\,b)$

To verify the validity of $E(a\,U\,b)$ in the initial marking m_0 means that there is a computation beginning in the m_0, in which b is true in some future marking m, and a is true in all the markings preceding m. This means that either b is true in the initial marking m_0, or a is true in

m_0 and there exists a successor marking, in which $E(a\,U\,b)$ is true. Generally, it can be computed using the following least fixed point calculation [3]:

$m_0 \models E(\phi\ U\ \varphi)$ iff $m_0 \in \mu Y.(\varphi \vee (\phi \wedge EX\,Y))$ (9)

Here μY denotes the least fixed point, and ϕ and φ are atomic propositions. There is a standard algorithm for computing the least fixed point of a monotonic function. This can be done by starting with false and iterating the functional until a fixed point is reached. The iterating algorithm is given as follows:

$$Z_1 = SAT(\varphi)$$
$$Z_{i+1} = Z_i \cup (SAT(\phi) \cap SAT(EX(\lambda_{Z_i}))$$

So for the example in section 5:

$$Z_1 = SAT(b) = \{p_3\} + \{p_4\}$$
$$\begin{aligned} Z_2 &= Z_1 \cup (\{p_4\} \cap SAT(EX(p_3 + p_4))) \\ &= \{p_3\} + \{p_4\} + \{p_1 p_3' p_4\} + \{p_1 p_2' p_4\} + \{p_1' p_3 p_4\} \\ &= \{p_3\} + \{p_4\} \end{aligned}$$

Since Z_2 produces the same result as $Z_{1,}$ the least fixed point is reached. Because the initial marking m_0 is not in the set Z_2, $E(a\,U\,b)$ does not hold for it.

(3) $EG(c)$

Now we check the formula $EG(c)$, which means there exists a computation beginning with the initial marking in which c is globally true. With other words, c is true in m_0, and EG c is true in some successor markings. This condition is the greatest fixed point calculation [3]. Generally holds:

$m_0 \models EG(\varphi)$ iff $m_0 \in \gamma Y.(\varphi \wedge EXY)$ (10)

Here γY denotes the greatest fixed point. This greatest fixed point can be computed through the following iterations:

$$Z_1 = SAT(\varphi)$$
$$Z_{i+1} = SAT(\varphi) \cap SAT(EX(\lambda_{Z_i}))$$

Now we return to the example. For this example we have

$$Z_1 = SAT(c) = \{p_1\}$$

$$\begin{aligned} Z_2 &= Z_1 \cap SAT(EX\,p_1) \\ &= \{p_1 p_2 p_4'\} + \{p_1 p_3 p_4'\} \end{aligned}$$

$$\begin{aligned} Z_3 &= \{p_1\} \cap SAT(EX(\lambda_{Z_2})) \\ &= \{p_1\} \cap (\{p_1' p_2 p_4\} \cup \{p_1' p_3 p_4\}) \\ &= \Phi \end{aligned}$$

So EG(c) does not hold in initial marking m_0.

6. Example

This example illustrates the model checking verification for an SIPN program using the tool SMV. Fig. 2 shows an air chamber (compressed air accumulator) control problem. Consumers can draw off compressed air via a valve from the chamber. Two binary sensors ps1 and ps2 are used to monitor the pressure in the chamber. Via two compressor motors (Ma and Mb) air is fed into the chamber. The compressors give a signal when they are disturbed.

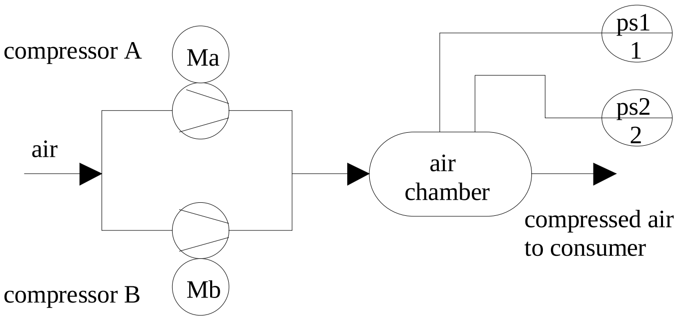


Fig. 2 air chamber

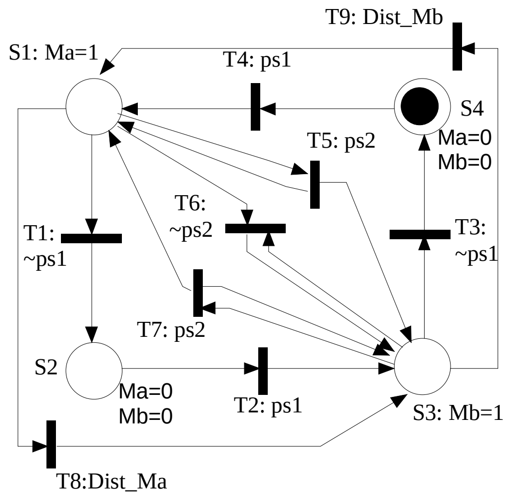


Fig. 3

Fig. 3 is a supposed SIPN solution, which should realize the following functions:

(1) If the pressure is less than 6.1 bar (ps1 switches on), one compressor should run.
(2) If a compressor is disturbed (Dist_Ma or Dist_Mb), the other one should substitute it.
(3) If the pressure is less than 5.9 bar (ps2 switches on), both compressors should run.
(4) If the pressure arises again above 5.9 bar (ps2 switches off), compressor A should be shut down.

The corresponding CTL forms for these requirements are presented as follows (see also [14]):

(1) property1: SPEC AG (ps1 & ~ps2 -> AX ((Ma & ~ Mb) | (~Ma & Mb)));
(2) property2: SPEC AG ((Ma & ~Mb & ps1 & ~ps2 & Dist_Ma -> AX (Mb & ~ Ma))
& (Mb & ~Ma & ps1 & ~ps2 & Dist_Mb -> AX (Ma & ~ Mb)));
(3) property3: SPEC AG ((ps1 & ps2 & Ma & ~Dist_Ma -> AX (Ma & Mb))
& (ps1 & ps2 & Mb & ~Dist_Mb -> AX (Ma & Mb)));
(4) property4: SPEC AG (Ma & Mb & ~ps2 -> AX (~ Ma & Mb));

Now we check the first assertion property 1. The result turns out to be false as shown in Fig. 4

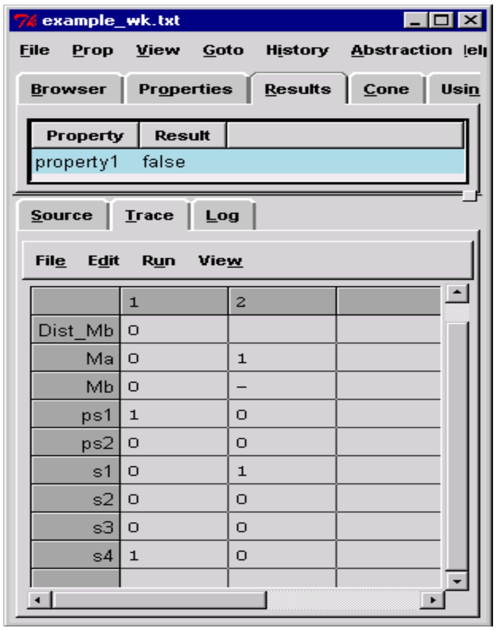


Fig. 4

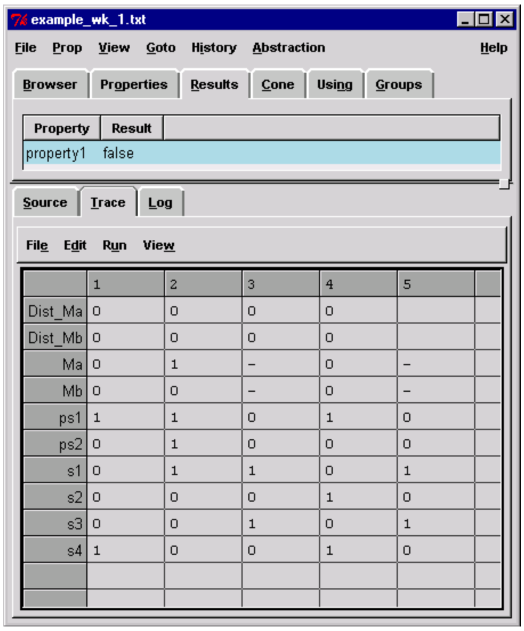


Fig. 5

The counterexample shows that in the second state, where S1 is marked, motor B is not defined. This indicates the incompletely defined output signals.

Now if the output signals on the place S1 are changed with Ma=1, and Mb=0, and on the place S3 with Ma=0, Mb=1, then we check it again with the same assertion. The result turns out to be still false (see Fig. 5). The counterexample shows another problem. In state 5 in Fig. 5, when S1 and S3 are both marked, then both Ma and Mb are not defined. The reason of this result is that the two variables have contradictory values when both S1 and S3 are marked. So this problem can not be solved by relabeling the output signals, it must be solved by changing of the structure of the design.

Fig. 6 shows a revised design. Now we check all the properties above. The results are shown in Fig. 7, which turn out to be all true. That is, this control program can satisfy the functional requirements listed above. From Fig. 7 we see that an additional reversibility property is checked. The CTL form for this property is AG EF S4, which checks whether the program can always return to the initial state, in which S4 is marked. This is an important property, because a logic control program should work as a cyclic one. This also results in true. More details about the case study using symbolic model checking can refer to [14].

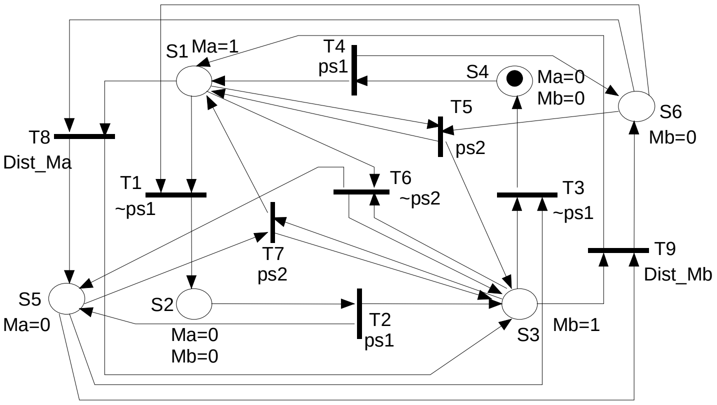


Fig. 6

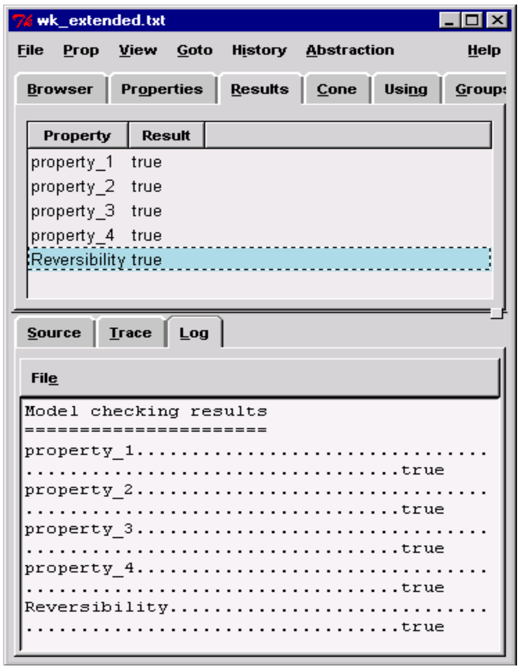


Fig. 7

7. Conclusion

In this paper, an encoding method of an SIPN program is proposed, so that a logic control program in SIPN can be automatically verified using symbolic model checking technology. Based on this mathematical background, an experimental example is given, which has shown that

1) Functional properties, which are usually derived from the informal requirement specifications, can be expressed in CTL formulas.
2) Properties of the formalism, such as correct output signals assignment, reversibility, etc. can also be translated into CTL formulas.

Verifying a logic control program in SIPN by symbolic model checking approach can also take the following advantages:

1) This approach checks the program behavior under all possible input patterns. In this sense it is complete and reliable.
2) The state explosion problem can be tackled in some extent by avoiding the explicit exploring of the reachability graph.

Further investigating of SIPN properties using model checking approach is necessary, so that systematical methods for the verification of an SIPN program can be developed.

Reference

[1] Brown, F. M. (1990). *Boolean Reasoning: the Logic of Boolean Equations.* Kluwer academic publishers, Boston, Dordrecht, London, ISBN 0-7923-9121-7.

[2] Bryant, R.E. (1986). Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers, Vol. C-35(8), page 677-691.

[3] Clarke, E., Grumberg, O. and Peled, D.A.(1999). *Model Checking*. The MIT Press, Cambridge, Massachusetts, London, England. ISBN 0-262-03270-8.

[4] Clarke, E., Grumberg, O. and Long, D. (1994). Verification tools for finite-state concurrent systems. In A Decade of Concurrency- Reflections and Perspectives. Lecture Notes in Computer Science, 803, page 124-175.

[5] Filkorn, Th., Hölzlein, M., Warkentin, P. and Weiß, M.(1999). Formal verification of PLC-programs. In proceedings of the 14th IFAC, Beijing, China, page 503-508.

[6] Frey, G.(2000). Analysis of Petri net based control algorithms - Basic properties. Appear in: Proceedings of the American Control Conference, ACC' 2000, June 28.-30. 2000, Chicago, USA.

[7] Frey, G.(2000): Automatic implementation of Petri net based control algorithms on PLC. Proceedings of the American Control Conference ACC 2000, June 28.-30. 2000, Chicago, USA.

[8] Jörns, C. (1996). *Ein Integriertes Steuerungsentwurfs – und Verifikationskonzept mit Hilfe Interpretierter Petri-Netze*. VDI Reihe 8, Nr. 641. (Ph.D. Thesis, Institute of Automation, Uni. of Kaiserslautern, Germany)

[9] König, R. and Quäck, L. (1988). *Petri-Netze in der Steuerungs- und Digitaltechnik.* Oldenbourg, München, Wien.

[10] Manna, Z. and Pnueli, A. (1992). *The Temporal Logics of Reactive and Concurrent Systems - Specification*. Springer-Verlag.

[11] McMillan, K.L. (1993). *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic.

[12] Moon, I. (1994). Modeling programmable logic controllers for logic verification. IEEE Control Systems Magazine, 14(2), page 53-59.

[13] SMV model checker. http://wwwcad.eecs.berkeley.edu/~kenmcmil/psdoc.html.

[14] Weng, X. and Litz, L. (2000). Verification of logic control design using SIPN and model checking – Methods and case study. In Proceeding of American Control Conference, ACC'2000. June 28. –30. 2000, Chicago, USA.