Int. J. Appl. Math. Comput. Sci., 2004, Vol. 14, No. 1, 113–121

DETECTION OF DEADLOCKS AND TRAPS IN PETRI NETS BY MEANS OF THELEN'S PRIME IMPLICANT METHOD[†]

AGNIESZKA WĘGRZYN[*], ANDREI KARATKEVICH[*], JACEK BIEGANOWSKI[*]

[*] Institute of Computer Engineering and Electronics
University of Zielona Góra
ul. Podgórna 50, 65–246 Zielona Góra, Poland
e-mail: {A.Wegrzyn, A.Karatkevich, J.Bieganowski}@iie.uz.zgora.pl

A new method of detecting deadlocks and traps in Petri nets is presented. Deadlocks and traps in Petri nets can be represented by the roots of special equations in CNF form. Such equations can be solved by using the search tree algorithm proposed by Thelen. In order to decrease the tree size and to accelerate the computations, some heuristics for Thelen's method are presented.

Keywords: Petri net, deadlock, trap, analysis, liveness

1. Introduction

Petri nets (Murata, 1989) are a mathematical model describing parallel discrete systems. They are used for the modelling, simulation, analysis and synthesis of a wide range of systems, such as parallel algorithms, asynchronous digital circuits, networks, multitasking operating systems, communication protocols, distributed software, industrial control systems, etc. (Girault and Valk, 2003). Several languages for the description of logical control algorithms, e.g. a standard industrial language SFC (Lewis, 1995), are based on Petri nets or similar models. The Petri net approach has also been used for simulating computation processes, neural networks and logical reasoning.

The analysis of Petri nets is important in most of their applications. It often turns out to be a time- and memory-consuming task. The number of reachability states of a Petri net depends exponentially on its size, so the brute force approach does not allow a practical analysis of big nets. Thus many advanced methods of a Petri net analysis have been developed. Generally, those methods can be divided into three classes: those based on the analysis of the state space, reduction of nets (keeping the properties to be analyzed) and solving systems of equations, linear or logical. The main tasks of a Petri net analysis are the checking of some properties of the nets, i.e. the liveness, boundedness, persistence, etc., and also the reachability analysis.

The calculation of deadlocks and traps is one of the most important analysis tasks. For some classes of Petri nets (and their extensions) properties like liveness and reversibility can be checked by an analysis of deadlocks and traps (Barkaoui and Minoux, 1992; Jiao *et al.*, 2002; Schmidt, 1997; Ohta *et al.*, 1999; Best *et al.*, 1990).

The detection of deadlocks and traps is also important as an independent task. In this paper a general form of this task is considered, i.e. detecting all the deadlocks and traps of a given net. Deadlocks and traps of a Petri net correspond to the decisions of some logical equations, which can be represented in conjunctive normal form. Thelen's method (Thelen, 1988; Mathony, 1989) enables efficient calculation of prime implicants of a Boolean function represented in such a form. In this paper applying Thelen's method to the above-mentioned logical equations is discussed. It makes it possible to obtain sets of deadlocks and traps in the form of ternary vectors. Some heuristics for Thelen's method are suggested and computer experiments are performed.

2. Definitions

2.1. Petri Nets

Formally, a Petri net (Murata, 1989) is a triple $PN = (P, T, F)$, where P is a set of places, T is a set of transitions, $P \cap T = \emptyset$, $F \subseteq (P \times T)(T \times P)$.

[†] The paper contains some results from the Ph.D. dissertation of A. Węgrzyn (2003) and results of new research partially supported by the State Committee for Scientific Research in Poland under the grant no. 4T11C006 24.

The sets of input and output transitions of place p are defined respectively as follows:

$$^\bullet p = \{t \in T : (t,p) \in F\},$$

$$p^\bullet = \{t \in T : (p,t) \in F\}.$$

For example, for the net shown in Fig. 1 we have $^\bullet p_1 = \{t_4, t_5\}$ and $p_1{}^\bullet = \{t_1\}$.

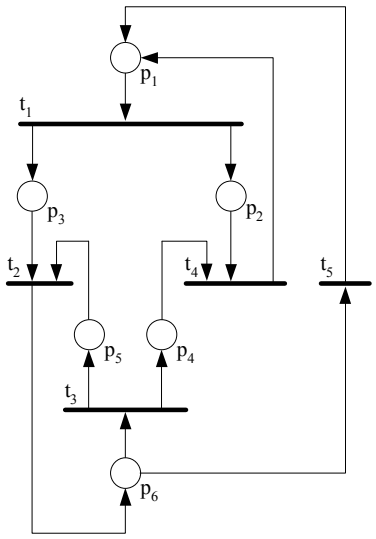

Fig. 1. Petri net (Kotov, 1984).

The sets of input and output places of transition t are defined respectively as follows:

$$^\bullet t = \{p \in P : (p,t) \in F\},$$

$$t^\bullet = \{p \in P : (t,p) \in F\}.$$

For example, for the net shown in Fig. 1 we have $^\bullet t_1 = \{p_1\}$ and $t_1{}^\bullet = \{p_2, p_3\}$.

Graphically, a Petri net is represented as a bipartite oriented graph whose nodes correspond to places and transitions, and arcs go from transitions to their output places and from places to their output transitions (cf. Fig. 1).

A marking of a net is defined as a function $M : P \to \{0, 1, 2, \dots\}$. It can be considered as a number of tokens situated in the net places. The number of tokens in a place p for marking M is denoted by $M(p)$. A transition t is enabled and can fire if $M(p) > 0$, $\forall p \in {}^\bullet t$. Transition firing removes one token from each input place and adds one token to each output place of it. A marking can be changed only by transition firing. The initial marking M_0 is usually specified.

2.2. Deadlocks and Traps in Petri Nets

A deadlock is a set of places such that every transition which outputs to one of the places in the deadlock also inputs from one of these places. This means that once all of the places in the deadlock become unmarked, the entire set of places will always be unmarked; no transition can place a token in the deadlock because there is no token in the deadlock to enable a transition which outputs to a place in the deadlock (Peterson, 1981) (Fig. 2). Formally, a nonempty subset of places S of a net N is a deadlock if $^\bullet S \subseteq S^\bullet$.

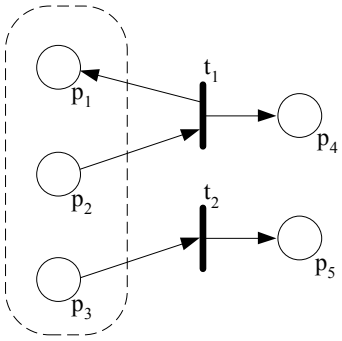

Fig. 2. Example of a deadlock $S = \{p_1, p_2, p_3\}$, $^\bullet S = \{t_1\}$, $S^\bullet = \{t_1, t_2\}$.

A trap is a set of places such that every transition which inputs from one of these places also outputs to one of these places. This means that once any of the places in a trap has a token, there will always be a token in one of the places of the trap. Firing transitions may move the token between places but cannot remove a token from the trap (Peterson, 1981) (Fig. 3). Formally, a nonempty subset of places Q of a net N is a trap if $Q^\bullet \subseteq {}^\bullet Q$.

The union of two deadlocks (traps) is again a dead-

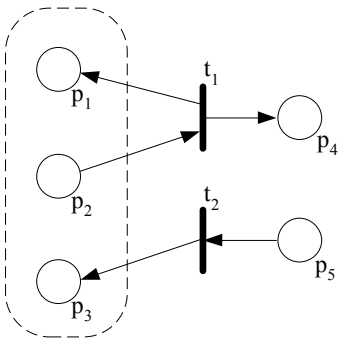

Fig. 3. Example of a trap $Q = \{p_1, p_2, p_3\}$, $Q^\bullet = \{t_1\}$, $^\bullet Q = \{t_1, t_2\}$.

lock (trap). A deadlock (trap) is called a basic deadlock (basic trap) if it cannot be represented as a union of other deadlocks (traps). So all the deadlocks (traps) in a Petri net can be generated by the union of some basic deadlocks (traps). A deadlock (trap) is said to be minimal if it does not contain any other deadlock (trap). Minimal deadlocks (traps) are basic deadlocks (traps), but not all basic deadlocks (traps) are minimal.

The Petri net shown in Fig. 1 contains 10 deadlocks and 10 traps (excluding the entire set of places, which is both a deadlock and a trap). In Table 1 all those deadlocks and traps are listed; minimal deadlocks and traps are marked.

Table 1. Sets of deadlocks and traps of the Petri net shown in Fig. 1.

Deadlock	Minimal deadlock	Trap	Minimal trap
$\{p_5, p_6\}$	Y	$\{p_1, p_2\}$	Y
$\{p_4, p_5, p_6\}$	N	$\{p_1, p_2, p_5, p_6\}$	N
$\{p_1, p_2, p_3, p_6\}$	Y	$\{p_1, p_2, p_4\}$	N
$\{p_1, p_2, p_5, p_6\}$	N	$\{p_1, p_2, p_4, p_6\}$	N
$\{p_1, p_2, p_3, p_5, p_6\}$	N	$\{p_1, p_2, p_3, p_5, p_6\}$	N
$\{p_1, p_3, p_4, p_6\}$	Y	$\{p_1, p_2, p_4, p_5, p_6\}$	N
$\{p_1, p_2, p_3, p_4, p_6\}$	N	$\{p_1, p_3, p_4, p_6\}$	Y
$\{p_1, p_4, p_5, p_6\}$	N	$\{p_1, p_2, p_3, p_4, p_6\}$	N
$\{p_1, p_2, p_4, p_5, p_6\}$	N	$\{p_1, p_3, p_4, p_5, p_6\}$	N
$\{p_1, p_3, p_4, p_5, p_6\}$	N	$\{p_1, p_3, p_5, p_6\}$	Y

2.3. Normal Forms and Implicants of Boolean Expressions

A literal is either a propositional letter p (a positive literal) or the negation $\overline{p}$ of a propositional letter p (a negative literal).

A conjunctive normal form (CNF) formula is the one which is a conjunction of disjunctions of literals, i.e. a formula of the form

$$(l_{11} \vee \ldots \vee l_{1n1}) \wedge \ldots \wedge (l_{m1} \vee \ldots \vee l_{mnm}), \qquad (1)$$

where each l_{ij} is either of the form p, or of the form $\overline{p}$ for some propositional letter p.

A disjunctive normal form (DNF) formula is the one which is a disjunction of conjunctions of literals, i.e. a formula of the form

$$(l_{11} \wedge \ldots \wedge l_{1n1}) \vee \ldots \vee (l_{m1} \wedge \ldots \wedge l_{mnm}), \qquad (2)$$

where each l_{ij} is either of the form p, or of the form $\overline{p}$ for some propositional letter p.

An implicant k of a Boolean expression F is a conjunction of literals such that $k \rightarrow F$. A prime implicant k_p of a Boolean expression F is an implicant of F such that removing any literal from k_p leads to a conjunction k' such that k' is not an implicant of F.

3. Known Methods of Calculating Deadlocks and Traps

3.1. Detecting Deadlocks and Traps by Solving Logical Equations

Deadlocks and traps do not depend on the initial markings and can be detected by a structural analysis of the net. A deadlock S has to satisfy the set of conditions $\forall t_i \in T: t_i^\bullet \cap S \neq \emptyset :\Rightarrow {}^\bullet t_i \cap S \neq \emptyset$. Analogous conditions exist for a trap. They can be represented in a natural way by logical equations.

Consider a set of Boolean variables $X = \{x_1, x_2, \ldots, x_m\}$, where $m = |P|$ and $x_i = 1$ if and only if p_i belongs to a subset of places. Denote by X_i and X_i^* the subsets of X corresponding to ${}^\bullet t_i$ and $t_i^\bullet$, respectively. Denote by $\vee[X_i]$ and $\vee[X_i^*]$ the disjunctions of the corresponding variables. Then the condition for the transition t_i can be expressed by the formula $\vee[X_i^*] \rightarrow \vee[X_i] = 1$.

Then all the deadlocks of the Petri net (P, T, F) are specified by the roots of a logical equation.

Affirmation 1 (Aff. 4.9 from (Zakrevskij, 1999)). *All the deadlocks of a Petri net PN are defined by the roots of the logical equation*

$$\wedge_{i=1}^{n}(\vee[X_i^*] \rightarrow \vee[X_i]) = 1, \qquad (3)$$

where $n = |T|$.

The situation with traps is analogous:

Affirmation 2 (Aff. 4.8 from (Zakrevskij, 1999)). *All the traps of a Petri net PN are defined by the roots of the logical equation*

$$\wedge_{i=1}^{n}(\vee[X_i] \rightarrow \vee[X_i^*]) = 1, \qquad (4)$$

where $n = |T|$.

So, deadlocks and traps of a Petri net can be calculated by solving logical equations. This methodology is described in (Zakrevskij, 1999). How to solve efficiently equations of such kind? One of the methods is the elimination of implications and transformation of the equation into a DNF, from which the roots can be obtained. Another approach allows simplifying the calculations, taking

into account the fact that in (3) and (4) all the variables are met without negation. In that approach the ternary matrices are used to represent the equations and the roots are obtained by combinatorial operations on those matrices. For details, see (Zakrevskij, 1999). Another approach to symbolic calculation of deadlocks and traps is described in (Schmidt, 1996a; 1996b).

Let us consider some more transformations of the equations which can simplify their solving. Let $m = |t_i^\bullet|$ and $x_1^{i^*}, \ldots, x_m^{i^*}$ be the variables corresponding to the output places of t_i.

$$\begin{aligned} \vee[X_i^*] \rightarrow \vee[X_i] &= \overline{\vee[X_i^*]} \vee (\vee[X_i]) \\ &= \wedge_{i=1}^m \overline{x_i^{j^*}} \vee (\vee[X_i]) \\ &= \wedge_{i=1}^m \Big(\overline{x_i^{j^*}} \vee (\vee[X_i]) \Big). \end{aligned} \quad (5)$$

So the equation is transformed into the conjunctive normal form. Note that changing the interpretation of the variables so that the belonging of a place to a set means that the corresponding variable has value 0 would lead to the inversion of all the literals in (5) and the obtained expression turns out to be a conjunction of Horn clauses.

There is an efficient polynomial-time algorithm solving the satisfability of a Horn expression and finding a decision with a minimal number of variables having value 1 (Papadimitriou, 1994). So for a given subset of places it is possible to calculate efficiently the *maximal deadlock contained in it*. This approach is used in (Barkaoui and Minoux, 1992). There are also methods of finding minimal deadlocks or minimal deadlocks containing given places (Tanimoto *et al.*, 1996; Yamauchi *et al.*, 1996; Yamauchi and Watanabe, 1999). But in this paper the task of generating all the deadlocks is considered. Applying the abovementioned approach to it would require a time-consuming combinatorial search.

Note that the approach described in this subsection does not lead to finding basic deadlocks (traps) only, unlike the approaches described below. But this does not mean that all the deadlocks (traps) are calculated and represented explicitly; it would lead to non-compact representations. The Boolean equations (3) and (4) can be solved in such a way that the set of their roots (deadlocks, traps) is represented by a ternary matrix, every row of which describes several solutions (see the example below). That is how we are going to solve the task.

3.2. Other Approaches to Detect Deadlocks and Traps

There are methods in which deadlocks and traps are calculated by solving systems of linear equations or inequalities. Those methods can be grouped into three classes (Ezpeleta *et al.*, 1993):

1. A particular class of Boolean equation systems can be transformed into systems of linear inequalities, by solving which the deadlocks or traps can be detected (Silva, 1985).

2. The so-called p-invariants can be obtained by solving a linear equation $Ax = 0$, where A is the incidence matrix of a net (Murata, 1989). The set of states corresponding to a p-invariant is an st-component (a deadlock and a trap at the same time). But not every deadlock and trap has a corresponding p-invariant. In the second approach (Lautenbach, 1987) the net is transformed in such a way that for every deadlock (or trap) of the initial net there is a corresponding p-invariant of the transformed net. So, the detection of the deadlocks (traps) of the initial net can be performed by solving linear equations generated according to the transformed net.

3. The support of a vector being a solution of the linear inequality $Ax \geqslant 0$ is a trap in the corresponding net, but not every trap corresponds to such a solution. In the third approach (Sifakis, 1979) the incidence matrix is transformed in such a way that it describes the same underlying graph of the net, but with weighted arcs. Solving the obtained inequality allows us to detect the traps of the net. As far as the tasks of detecting deadlocks and traps are equivalent (a deadlock of a net N is a trap in the net obtained from N by reversing all arcs), the approach can be used for detecting deadlocks, too.

The paper is dedicated to detecting deadlocks and traps by solving logical equations, so the methods using linear algebra are not considered here in detail.

4. Thelen's Prime Implicant Algorithm

Thelen (1988) proposed an efficient algorithm finding all the prime implicants of a Boolean expression in the conjunctive normal form. It is based on building a search tree, such that every level of it corresponds to a clause of the CNF, and the outgoing arcs from a node correspond to the literals of the disjunction. The conjunction of all the literals corresponding to the arcs down the path from the root of the tree to a node is associated with a node. The tree is searched in the DFS order, and several pruning rules are used to minimize it (Fig. 4). The rules are listed below:

R_1**:** An arc is pruned if its predecessor node-conjunction contains the complement of the arc-literal.

R_2**:** A disjunction is discarded if it contains a literal which appears also in the predecessor node-conjunction.

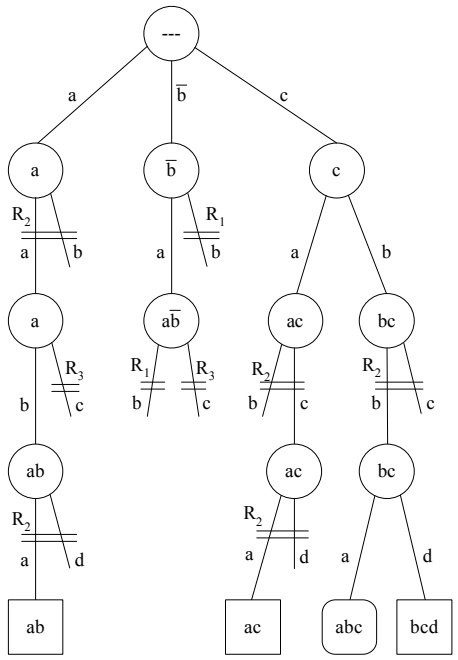

Fig. 4. Tree example $f = (a \vee \bar{b} \vee c) \wedge (a \vee b) \wedge (b \vee c) \wedge (a \vee d)$.

R_3**:** An arc is pruned if another non-expanded arc on a higher level still exists which has the same arc-literal.

Leaf nodes of the tree are the elementary conjunctions being the prime implicants of the expression or the implicants absorbed by the prime implicants, *calculated before*. So there is no need of calculating *all* products and keeping them in memory before one can decide whether the given product is a prime implicant. Thelen even claims that his method has a linear space-complexity. This claim is based on the fact that for searching the tree it is enough to keep in memory only the current path from the root, and its length depends linearly on the number of clauses. But for every leaf of the tree it is necessary to check whether the corresponding conjunction is absorbed by any of the previously calculated, so the algorithm has to keep in memory all the prime implicants calculated before. And it is easy to show that in the worst case the number of prime implicants exponentially depends on the expression size (for example, when the literals never repeat in the expression), hence Thelen's method requires an exponential space in the worst case. But due to the reduction of the search tree, it requires less space than the straightforward multiplication of the clauses. Concerning time complexity, Thelen did not show any theoretical evaluation of it, but he showed that his method is faster than other known methods, e.g. (Reusch, 1975; Das and Khabra, 1972). It is evident, however, that the worst-case time-complexity of this algorithm is exponential: it cannot be greater, because even without any pruning the size of the search tree exponentially depends on the expression size, and cannot be less, because the number of prime implicants may be exponential.

Mathony (1989) proposed the additional fourth reduction rule, which reduces the search tree up to 25%. But using this rule complicates the algorithm remarkably, and in such a variant of the algorithm linear memory is not enough even for the tree searching. Another way of improving Thelen's method is reordering clauses in the expression and literals into clauses. The next section is dedicated to such heuristics.

5. Improvements of Thelen's Method

As far as the algorithm requires exponential time, the minimization of the search tree is important. Some heuristics (Mathony, 1989) are known for doing this. One of them is sorting the disjunctions by their size in ascending order.

Heuristic 1. ***(Sort by Length)*** *Choose a disjunction D_j with the smallest number of literals.*

Consider a complete search tree (without arc pruning). Its size (the number of nodes) can be calculated according to the formula

$$|V| = 1 + \sum_{i=1}^{n} \prod_{j=1}^{i} L_j, \tag{6}$$

where L_j is the number of literals in the j-th clause. It is easy to see that the size of the tree depends on the order of clauses in the formula. So sorting clauses can reduce the tree. Let a Horn formula consist of 5 clauses, each having a distinct number of literals, from 2 to 6. If they are sorted from the maximal to the minimal length, the complete search tree will contain 1237 nodes. If the same is performed from the minimal to the maximal lenghts, the tree will contain only 873 nodes. In the second case the cost is by 30% smaller. So sorting clauses influences the tree size remarkably. Of course, for reduced search trees this relation may differ. In Table 1 the results of computer experiments are shown.

Now let us turn to the pruning rules. Note that every rule can be implemented only if the disjunction under consideration contains the same variables as the disjunctions corresponding to the predecessor nodes. So we may suppose that sorting the clauses according to the variables may also lead to tree reduction. Consider the next sorting: at each position in the Horn formula insert the conjunction which has a minimal number of variables that do not belong to any previous clause.

Heuristic 2. ***(Sort by Variables)*** *Choose a disjunction D_j with the smallest number of literals that do not appear in the disjunctions chosen before.*

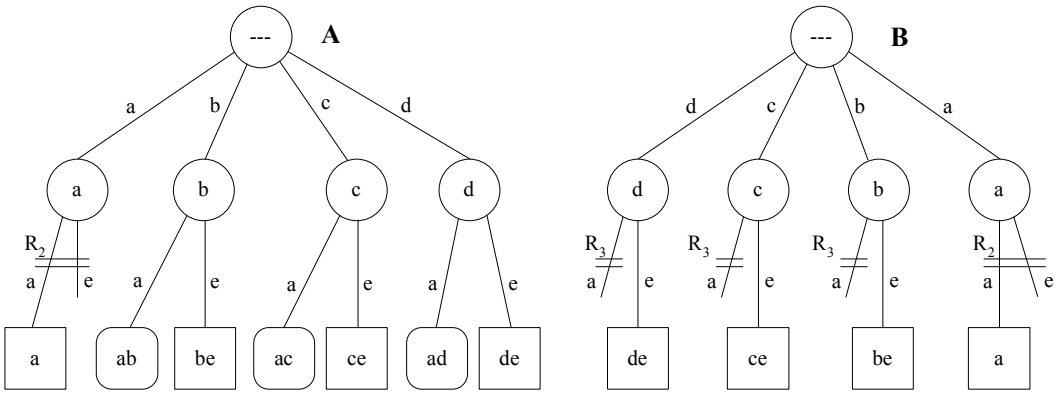

Fig. 5. Order of literals in disjunctions, formula $f = (a \vee b \vee c \vee d) \wedge (a \vee e)$: (a) worst case, rule R_2 blocks all rules R_3, (b) best case, all rules R_3 occur before rule R_2.

Table 2. Results of computer experiments

Net Param.	PI	No Sort		Sort by Length			Sort by Variables			Reordering Literals			SV + RL		
		TS	NPI	TS	NPI	%	TS	NPI	%	TS	NPI	%	TS	NPI	%
20x15	144	42194	6336	7680	1296	18.2	7522	1296	17.8	2536	0	6.0	702	0	1.7
20x18	145	7755	420	9504	420	122.6	6391	420	82.4	2596	0	33.5	2191	0	28.3
20x18	12	1422	150	645	24	45.4	599	24	42.1	247	0	17.4	283	0	19.9
20x18	105	2486	319	3166	319	127.4	3166	319	127.4	702	0	28.2	1067	0	42.9
20x18	70	3024	317	1543	125	51.0	1435	125	47.5	816	0	27.0	738	0	24.4
20x18	28	1346	168	671	84	49.9	635	84	47.2	333	0	24.7	247	0	18.4
20x20	36	4356	377	1638	48	37.6	1575	48	36.2	705	0	16.2	654	0	15.0
20x20	16	2650	92	660	56	24.9	826	56	31.2	881	0	33.2	391	0	14.8
20x20	117	3418	406	4181	406	122.3	4181	406	122.3	773	0	22.6	1139	0	33.3
25x20	946	51275	6111	26030	6111	50.8	26075	4895	0.9	6365	0	12.4	2910	0	5.7
25x20	144	55608	9026	4222	540	7.6	1770	540	3.2	2283	0	4.1	438	0	0.8
25x20	560	63234	5622	22826	6226	36.1	18831	6226	29.8	4170	0	6.6	2125	0	3.4
25x20	91	6838	288	4603	288	67.3	4109	288	60.1	1917	0	28.0	970	0	14.2
Average:						**58.5**			**53.7**			**20.0**			**17.1**

To the best of our knowledge, this is an original heuristic. It also allows a remarkable reduction of the search tree. The results of computer experiments are presented in Table 2.

Sorting clauses in an expression can be used together with reordering literals in disjunctions. Let us consider the next heuristic which reorders literals to optimize the usage of pruning rules R_2 and R_3. These rules depend on each other on every level. As is shown in Fig. 5, rule R_2 blocks the usage of rule R_3 when R_2 occurs before R_3. This causes an unnecessary growth of the tree and a possibility to produce non-prime implicants. In most cases the effect of blocking rules R_3 by rule R_2 can be avoided.

Heuristic 3. ***(Reordering Literals)*** *Split the set of literals of every disjunction D_i into two parts. One part (C) is formed from literals that appear in any of the clauses $D_{i+1}, \ldots, D_k$ (where k is the number of clauses) and the second part (NC) contains the remaining literals. The optimized disjunction contains literals in order*

$$D_i = \{\underbrace{P_1 \vee \ldots \vee P_n}_{NC} \vee \underbrace{P_{n+1} \vee \ldots \vee P_m}_{C}\}.$$

The literals in (C) are sorted in the order of the growing frequencies in $D_{i+1}, \ldots, D_k$.

An additional positive feature of reordering literals in clauses is that almost all implicants produced by the algorithm are prime (see Table 2). This effect is similar to the effect of the fourth pruning rule mentioned in Section 4 (Mathony, 1989), but can be obtained without any complication of the algorithm of building the tree itself. Note that there are still expressions for which the non-prime implicants are generated, e.g. $(a \vee b \vee c) \wedge (d \vee b \vee c) \wedge (a \vee x \vee y) \wedge (d \vee u \vee z)$.

The results of computer experiments are summarized in Tab. 2. For the tests the expressions corresponding to deadlocks and traps of Petri nets (5) were used. The Petri nets were randomly generated by the tool Paral, using the algorithm described in (Pottosin, 1995) (the EFC nets were generated). In the first column, the number of places and the number of transitions of a net are given (e.g. 20×18). TS means the tree size (the number of nodes), PI stands for the number of prime implicants, NPI denotes the number of non-prime implicants being the leaves of the search tree. For every heuristic the column '%' shows the percentage of the tree size versus the size in the case when no heuristic is used. The experiments show that the best way is sorting disjunctions according to Heuristic 2, and literals in the disjunctions according to Heuristic 3.

6. Algorithm of Calculating Deadlocks and Traps

As is shown in Subsection 3.1, a Boolean formula describing deadlocks or traps of a Petri net can be easily transformed into CNF (5). So Thelen's method can be efficiently applied to calculating deadlocks and traps. The corresponding algorithm is presented below.

Input: A Petri net
Output: The ternary matrix of (deadlocks) traps

BEGIN

1. Generate a Boolean formula for deadlocks (traps) from the Petri net according to (3) and (4).
2. Transform the formula into CNF applying (5).
3. Sort disjunctions and literals in disjunctions to minimize the search tree.
   (a) Sort disjunctions by variables (Heuristic 2).
   (b) Sort literals in disjunctions (Heuristic 3).
4. Find all prime implicants of the Boolean formula using Thelen's method.
5. Represent the set of prime implicants as a ternary matrix. Each row corresponds to one prime implicant and represents one or more deadlocks (traps).

END

If necessary, the compactness of the representation of the sets of traps (deadlocks) can be increased by using minimization methods of Boolean functions (De Micheli, 1994).

7. Example

7.1. Proposed Approach

Let us find all deadlocks for the Petri net from Fig. 6. First, a Boolean formula is generated according to (3):

$$(x_1 \rightarrow x_6)(x_2 \vee x_3 \rightarrow x_1)(x_1 \rightarrow x_2 \vee x_4),$$
$$(x_6 \rightarrow x_3 \vee x_5)(x_4 \vee x_5 \rightarrow x_6) = 1. \qquad (7)$$

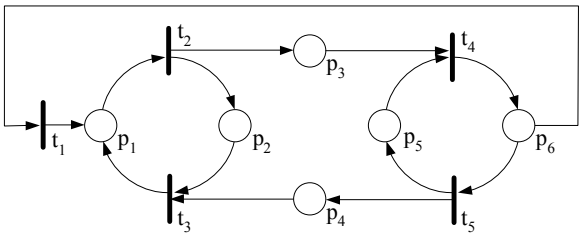

Fig. 6. Petri net (Zakrevskij, 1999).

Next the formula is transformed into CNF using (5):

$$(\overline{x}_1 \vee x_6)(\overline{x}_2 \vee x_1)(\overline{x}_3 \vee x_1)(\overline{x}_1 \vee x_2 \vee x_4),$$
$$(\overline{x}_6 \vee x_3 \vee x_5)(\overline{x}_4 \vee x_6)(\overline{x}_5 \vee x_6) = 1. \qquad (8)$$

All prime implicants are found and the tree size for different heuristics is shown below:

No heuristic – order like in (8)	37 nodes,
Sort by Length	27 nodes,
Sort by Variables	25 nodes,
Reordering Literals	37 nodes,
SV + RL	25 nodes.

The prime implicants are represented as a ternary matrix. For the analyzed Petri net, there are 11 deadlocks, e.g. the first row defines 2 deadlocks $\{p_5, p_6\}$ and $\{p_4, p_5, p_6\}$,

$$\begin{array}{c} p_1\ p_2\ p_3\ p_4\ p_5\ p_6 \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & - & 1 & 1 \\ - & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & - & - & 1 \\ 1 & 1 & - & - & 1 & 1 \\ 1 & - & 1 & 1 & - & 1 \\ 1 & - & - & 1 & 1 & 1 \end{array}\right] \end{array}. \qquad (9)$$

7.2. Some Other Symbolic Approaches

Using the method described in (Zakrevskij, 1999), the task would be solved as follows: The expression in the left part of (7) is transformed into a conjunction of DNFs and then into DNF by removing parentheses:

$$\begin{aligned}(\overline{x}_1\overline{x}_4\overline{x}_5 \vee x_6)(\overline{x}_2\overline{x}_3 \vee x_1)(\overline{x}_1 \vee x_2 \vee x_4)(\overline{x}_6 \vee x_3 \vee x_5)\\ = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4\overline{x}_5\overline{x}_6 \vee \overline{x}_1\overline{x}_2\overline{x}_3x_5x_6 \vee \overline{x}_2\overline{x}_3x_4x_5x_6\\ \vee x_1x_2x_3x_6 \vee x_1x_2x_5x_6 \vee x_1x_3x_4x_6\\ \vee x_1x_4x_5x_6 = 1. \qquad (10)\end{aligned}$$

The DNF represents the set of deadlocks similarly as (9) does. The transformation in (10) is performed by multiplying directly the DNFs in the parentheses. Thelen's method cannot be used here because the disjunctions to be multiplied are not elementary. For this example such a transformation requires 31 multiplications of the elementary conjunctions (compare with 24 multiplications of an elementary conjunction and a literal in the best variant of our method).

In another approach, also described in (Zakrevskij, 1999), a ternary matrix N_r is constructed corresponding to (3). For this example it looks as follows:

$$N_r = \begin{array}{c} \begin{array}{cccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{array} \\ \left[\begin{array}{cccccc} 0 & - & - & - & - & 1 \\ 1 & 0 & 0 & - & - & - \\ 0 & 1 & - & 1 & - & - \\ - & - & 1 & - & 1 & 0 \\ - & - & - & 0 & 0 & 1 \end{array}\right] \end{array}. \qquad (11)$$

In this matrix the column minors not containing the so-called negative rows (rows with 0-elements, but without 1-elements) correspond to deadlocks. To find them, a combinatorial search is necessary. For example, the last two columns satisfy the above-mentioned property and correspond to the deadlock $\{p_5, p_6\}$.

7.3. Linear Algebraic Approach

The linear algebraic approach (Ezpeleta *et al.*, 1993) leads to solving the system of linear inequalities $y^T W_\Theta \geqslant \mathbf{0}$, where y is a vector indexed by places of a net, and W_Θ is a transformed incidence matrix which, for our example, may look as follows:

$$\begin{array}{c} \begin{array}{cccccc} & t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \left[\begin{array}{ccccc} -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ 1 & 0 & 0 & -1 & 2 \end{array}\right] \end{array}. \qquad (12)$$

The support of any integer non-negative solution of the above-mentioned system corresponds to a deadlock. For example, $(0, 0, 0, 0, 1, 1)$ is a solution, and it corresponds to the deadlock $\{p_5, p_6\}$.

Note that the last two approaches lead to obtaining the sets of deadlocks (traps) in an explicit representation, and the first two approaches represent those sets in a compact form of ternary matrices.

8. Conclusions and Further Work

A symbolic method of calculating all deadlocks and traps of a Petri net has been discussed. It is close to the known methods solving this or similar tasks (Sifakis, 1979; Barkaoui and Minoux, 1992; Zakrevskij, 1999), but differs in the fact that it uses Thelen's prime implicant algorithm which, being applied to the corresponding formula describing a Petri net structure, allows the time- and memory-efficient obtaining of all the solutions without complex symbolic transformations or a combinatorial search on the matrices. In (Węgrzyn, 2003) it was suggested to apply Thelen's method to the Petri net analysis. Our paper contributes by suggesting new efficient heuristics for finding prime implicants. Those heuristics can be applicable not only for a Petri net analysis, but also for solving other tasks where Thelen's method is used. Further work will concentrate on a deeper investigation of the heuristics, their comparative analysis (experimental and theoretical) and methods of estimating the search tree size for a more precise use of the heuristics. Also, we are going to study the applicability of binary decision diagrams (BDD) for the representation of deadlocks and traps.

References

Barkaoui K. and Minoux M. (1992): *A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets*, In: *Lecture Notes in Computer Science* (K. Jensen, Ed.). — Berlin: Springer-Verlag, Vol. 616, pp. 62–75.

Best E., Cherkasova L., Desel J. and Esparza J. (1990): *Characterisation of home states in free choice systems*, In: Semantics for Concurrency (M.Z. Kwiatkowska, M.W. Schields, R.M. Thomas, Eds.). — Proc. Int. BCS-FACS Workshop, Leicester, UK, pp. 16–20, London: Springer.

Das S. and Khabra N. (1972): *Clause-column table approach for generating all the prime implicants of switching functions*. — IEEE Trans. Comp., pp. 1239–1246.

Ezpeleta J., Couvreur J.M. and Silva M. (1993): *A new technique for finding a generating family of siphons, traps and st-components. Application to colored Petri Nets*, In: Advances in Petri Nets (G. Rozenberg, Eds.). — Berlin: Springer-Verlag, Vol. 674.

Girault C. and Valk R. (2003): *Petri Nets for Systems Engineering*. — Berlin: Springer.

Jiao L., Cheung T.-Y. and Lu W. (2002): *Characterizing liveness of Petri nets in terms of siphons*. — Proc. 23rd Int. Conf. *Applications and Theory of Petri Nets*, London: Springer-Verlag, Vol. 2360.

Kotov V.Ye. (1984): *Petri Nets*. — Moscow: Nauka, (in Russian).

Lautenbach K. (1987): *Linear algebraic calculation of deadlocks and traps*, In: Concurrency and Nets, Advances of Petri Nets (K. Voss, H.J. Genrich and G. Rozenberg, Eds.). — Berlin: Springer-Verlag.

Lewis R.W. (1995): *Programming industrial control systems using IEC 1131-3*, In: *IEE Control Engineering Series*, London, Vol. 50.

Mathony H.-J. (1989): *Universal logic design algorithm and its application the synthesis of two-level switching circuits*. — IEE Proc., Vol. 136, No. 3, pp. 171–177.

De Micheli D. (1994): *Synthesis and Optimization of Digital Circuits*. — New York: McGraw-Hill.

Silva M. (1985): *Las redes de Petri en la Informatica y la Automatica*. — Technical Report, MAdrid.

Murata T. (1989): *Petri nets: Properties, analysis and applications*. — Proc. IEEE, Vol. 77, No. 4, pp. 541–580.

Ohta A., Tsuji K. and Hisamura T. (1999): *On liveness of extended partially ordered condition nets*. — IEICE Trans. Fund. Electron. Commun. Comput. Sci., Vol. E82–A, No. 11, pp. 2576–2578.

Papadimitriou Ch.H. (1994): *Computational Complexity*. — Reading, MA: Addison Wesley.

Peterson J.L. (1981): *Petri Net Theory and The Modeling of Systems*. — Englewood Cliffs: Prentice-Hall.

Pottosin Yu.V. (1995): *Generating of parallel automata*, In: Methods and algorithms of logical design (A.D. Zakrevskij, Ed.). — Institute of Engineering Cybernetics, the Academy of Sciences of Belarus, Minsk, (in Russian), pp. 132–142.

Reusch B. (1975): *Generation of prime implicants from subfunctions and a unifying approach to the covering problem*. — IEEE Trans. Comp., Vol. C-24, No. 9, pp. 924–930.

Schmidt K. (1996a): *How to calculate symbolically siphons and traps of algebraic Petri nets*. — Techn. Rep. No. A39, Helsinki University of Technology pp. 1–40.

Schmidt K. (1996b): *Siphons and traps for algebraic Petri nets*. — Proc. Workshop CSP, Berlin, pp. 157–168.

Schmidt K. (1997): *Characterizing liveness of Petri nets in terms of siphons*. — Proc. 18th Int. Conf. *Application and Theory of Petri Nets*, pp. 271–289.

Sifakis J. (1979): *Controle des systemes asynchrones: concepts, proprietes, analyse statique*. — I'Universite Scientifique et Medical de Grenoble.

Tanimoto S., Yamauchi M. and Watanabe T. (1996): *Finding minimal siphons in general Petri nets*. — IEICE Trans. Fundam. Electron. Commun. Comput. Sci., Vol. E79–A, No. 11, pp. 1817–1824.

Thelen B. (1988): *Investigations of algorithms for computer-aided logic design of digital circuits*. — Univ. of Karlsruhe, (in Germany).

Węgrzyn A. (2003): *Symbolic analysis of logical control devices using selected methods of Petri net analysis*. — Warsaw Univ. Technol., (in Polish).

Yamauchi M., Tanimoto S. and Watanabe T. (1996): *Finding a minimal siphon containing specified places in a general Petri net*. — IEICE Trans. Fundam. Electron. Commun. Comput. Sci., Vol. E79–A, No. 11, pp. 1825–1828.

Yamauchi M. and Watanabe T. (1999): *Time complexity analysis of the minimal siphon extraction problem of Petri nets*. — IEICE Trans. Fundam. Electron. Commun. Comput. Sci., Vol. E82–A, No. 11, pp. 2558–2565.

Zakrevskij A.D. (1999): *Parallel logical control algorithms*. — Institute of Engineering Cybernetics, Academy of Sciences of Belarus, Minsk, (in Russian).

Received: 24 April 2003
Revised: 3 November 2003