Parallel algorithm for computation of deadlocks and traps in Petri nets

Agnieszka Węgrzyn
Uniweristy of Zielona Gora
ul. Podgorna 50
65-246 Zielona Gora, Poland
A.Wegrzyn@iie.uz.zgora.pl

Abstract

In the paper the method of computation all deadlocks and traps in the Petri net is presented. This method is based on Thelen method [9] and it was proposed in [10].

Methods of calculation of all deadlocks and traps in Petri nets are very time consuming. Therefore it is very important to optimize a computation. The parallel computation method for the time reduction is proposed. Experimental results of presented method are discussed, as well.

1. Introduction

Petri nets are modelling and analysis tool for representing digital circuits, especially concurrent controllers, because Petri net in natural way exhibit concurrency and parallelism [7].

Analysis of Petri nets, which represent such controllers, is important. So there are developed many advanced methods of Petri net analysis. The model is formally verified based on well-known Petri net theory. The main tasks of Petri net analysis are to check some properties of the net, i.e. liveness, boundedness, persistence etc. There exists a lot of method for analysis of Petri nets, but majority of them is checking properties and give answer to question whether Petri net is or not live, bounded, etc. without presenting exactly place of error [7,8].

Calculation of deadlocks and traps is one of the most important analysis tasks, because a good system should not contain events, which can never occur. Testing the liveness of Petri net depends on finding deadlock and traps and researching dependencies between them.

In presented approach, for checking liveness of Petri net Thelen’s method is used. This method allows efficient calculation of prime implicants of a Boolean function represented in such form. In presented system the Thelen’s method for the mentioned logical equations is applied; it allows obtaining sets of deadlocks and traps in form of ternary vectors. Presented method is based on generating and searching tree for CNF [9,5]. Such approach is very time consuming, because number of deadlocks and traps in Petri nets increases exponentially with the number of places and transitions [1]. For this purpose, parallelism of presented algorithm is proposed.

In the following section theoretical basis related to Petri nets especially deadlocks and traps, is presented. In section 3, the method of computation of all deadlocks and traps in Petri net is described. In this section, the method of time consuming decreasing of proposed algorithm is presented, too.

2. Definitions, theoretical basis

In this section, some basic information of general aspects of Petri nets, deadlocks and traps, is presented. In addition, specification of deadlocks and traps in a Petri net by means of certain logical equations, called Horn formulae, is described.

2.1. Petri net

Petri Nets model can be presented as an oriented bipartite graph with two subsets of nodes called places and transitions, where each node is linked with another one by arc. Petri nets model can be described as [7,8]:

\[ PN = (P, T, F) \]

where:

- \( P \) - a finite set of places,
- \( T \) - a finite state of transitions,
- \( F \) - a finite set of arcs.

A marking of a net is defined as a function \( M: P \rightarrow \{0, 1, 2, \ldots\} \). It can be considered as a number of tokens situated in the net places. Number of tokens in a place \( p \) for marking \( M \) is denoted as \( M(p) \).

A transition \( t \) is enabled and can fire if each input places have a token. Transition firing removes one token from each input place and adds one token to its output place.

In graphical form of Petri Nets each place represents a local state of digital circuit. Currently active place is marked. Every marked place represents an active local state. The set of places, which are marked at the same time, defines the global state of the controller. Each transition describes logical condition of controllers' state change. On fig.1 example of Petri net is presented.
2.2. Deadlock and trap

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 [8].

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 [8].

Deadlocks and traps can be used for liveness, boundedness checking and P-invariants finding [7,6].

2.3. Horn formulae

Deadlocks and traps of a Petri net correspond to the decisions of certain logical equations, which can be represented in conjunctive normal form.

Deadlocks and traps can be represented by a special conjunctive form - Horn formulae [6].

A Horn formula is a conjunction of basic Horn formulae. The basic Horn formula (Horn clause) is a disjunction of literals, with at most one positive literal. A literal is either a propositional letter P (a positive literal) or the negation /P of a propositional letter P (a negative literal).

Each basic Horn formula is equivalent to a clause of one of three types:

- Q, a propositional letter;
- /P1 ∨ ... ∨ /Pq where q≥1 and P1,...,Pq are distinct propositional letters;
- /P1 ∨ ... ∨ /Pq ∨ Q where q≥1 and P1,...,Pq are distinct propositional letters and Q is a propositional letter.

Equation (1) described deadlock in Petri net:

\[ \Pi t \in T \Pi p_i \in t(y_i + \sum_j p_j \in \bullet t /y_j) \]  

where:
- t, T - (respectively) transition, set of transitions;
- t, t\bullet - input and output place to transition t;
- p_i, p_j - considered places p_i \in t, p_j \in t;  
- y_i - variable representing place p_i, i.e. p_i \rightarrow y_i=0.

**Theorem:** Deadlocks in Petri net [6]  
All the y vectors satisfying (1) are in 1-1 correspondence with the deadlocks.

Equation (2) described trap in Petri net:

\[ \Pi t \in T \Pi p_i \in \bullet (y_i + \sum_j p_j \in t /y_j) \]  

where:
- t, T - (respectively) transition, set of transitions;
- t, t\bullet - input and output place to transition t;
- p_i, p_j - places;
- y_i - variable representing place p_i, i.e. p_i \rightarrow y_i=0.

**Theorem:** Traps in Petri net [6]  
All the y vectors satisfying (2) are in 1-1 correspondence with the traps.

3. The algorithm for finding all deadlocks and traps

In this section, the algorithm for finding all deadlocks and traps proposed in [10] is presented. In addition, a new approach to optimize the time of computation of it is introduced.

3.1. Thelen method

Thelen has proposed an efficient algorithm for converting a conjunctive form into the sum of all prime implicants [9]. 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 clause. Conjunction of all the literals corresponding to the arcs on the path from the root of the tree to a node is associated with the node. The tree is searched in DFS order, and several pruning rules are used to minimize it. The rules are listed below.
R1: An arc is pruned, if its predecessor node-conjunction contains the complement of the arc-literal.

R2: An arc is pruned, if another non-expanded arc on a higher level still exists which has the same arc-literal.

R3: A disjunction is discarded, if it contains a literal which appears also in the predecessor node-conjunction.

The next rule R4 was added by Mathony [5]. This rule leads application to a significant reduction of the search tree.

R4: An arc j is pruned, if another already expanded arc k with the same arc-literal exists on a higher level i and if rule R2 was not applied in the subtree of arc k with respect to arc p on level i which leads to arc j.

This method has an exponential time-complexity and a linear space-complexity [5,9]. For decreasing of time calculation of all prime implicants, firstly, heuristic methods have been used. A size of the tree can be reduced by sorting clauses and literals in the input formula. There have been proposed three methods [11]:

- H1 - Heuristic 1 (Sort by Length): Choose disjunction \( D_i \) with the smallest number of literals.
- H2 - Heuristic 2 (Sort by Variables): Choose disjunction \( D_i \) with the smallest number of literals that do not appear in the disjunctions chosen before.
- H3 - Heuristic 3 (Reordering Literals): Split the set of literals of every clause \( D_i \) into two parts. One part (C) is formed from literals that appear in any of the clauses \( D_{i+1}...D_k \) (where \( k \) is the number of clauses) and the second part (NC) contains remaining literals. Optimized disjunction contains literals in order

\[
D_i = \{ P_1 \lor ... \lor P \lor P_{n+1} \lor ... \lor P_m \}
\]

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

The experiments show that the best way is sorting disjunctions according to Heuristic 2, and literals in the disjunctions according to Heuristic 3. But time reduction is not enough, especially for bigger model of system. The result of computer experiments was presented in [11].

The next section described a new approach to reduce time of computation of all prime implicants (i.e. deadlocks and traps).

### 3.2. The parallel approach to the Thelen method

It is very important to findout methods of accelerations of computational algorithms. Even though a power of computers is bigger and bigger, nevertheless a complexity problem is still bigger and bigger. Most analysis methods of a calculation of Petri nets’ deadlocks and traps have an exponential-time complexity. Similarly, the presented method of calculation of prime implicants has an exponential-time complexity. Therefore, researches on methods of acceleration of calculation of all prime implicants have been carried out. In the presented method, a parallel approach has been applied for decreasing a calculation time. The search tree was decomposed in vertical way.

During a decomposition of the tree, the processes are generated on the basis of partially obtained tree. Therefore, it is necessary, at the beginning of the computation to run program sequentially as long as the number of nodes will be equal or bigger than number of declared processes (Fig.2). In the case, if the number of nodes is bigger than the number of processes, then processes, which compute the left-side of the tree, get more input data, because a degree of the optimization is the biggest for these processes. Such solution leads to a balanced load of the processes.

![Fig.2. Decomposition of sequential program](image)

Data are sent from the main process to the child processes. The communication between child processes has been eliminated using the suitable structure of an optimization function. The function uses two methods of communication, i.e. by means of files and messages. The files are used for an input data transfer to processes and a transfer results from processes. The messages are used to transfer status of the child process from this process to the main process.

Fig.3 presents an example of a tree decomposition. Firstly, a Horn clause is sorted using one of the heuristic methods, which was described in previous section. Then, the first clause \((a'+b+c)\) and the second one \((a+b')\) are processed by sequential program. In the next step, the tree is divided into the three processes, because the three nodes were declared. Each process has got the own data (the first process - \(a'b'\) and \(ab\), the second one - \(ac\) and the third one - \(b'c\)), data for the rules R2 and R4, and the remaining clauses: \((a+c'+d)\) \((b'+c)\). After the calculation of prime implicants, the data from each process, i.e. \((a'b'+c)+(a'b'd)\) from the first process, \((ac)\) from the second process, and \((b'cd)\) from the third process, are joined.
F = (a' + b + c)(a + b')(a + c' + d)(b' + c)

Generation of input files for the processes

F = a'b'c' + a'b'd + ac + b'cd

Fig. 3 Example of decomposition of tree
3.3. Experimental result

The algorithm has been implemented in C language on the Linux operating system.

Sequential program was tested on Pentium IV 2.4 GHz 256 MB RAM.

Parallel program was tested on Linux cluster consisting of four PC with Linux operating system (Red Hat and Mandrake) and Mosix:
- Pentium IV 2.4 GHz, 256 MB RAM,
- AMD Athlon 1800 XP, 256 MB RAM,
- AMD Duron 600 MHz, 128 MB RAM,
- Pentium II 400 MHz, 192 MB RAM.

Why Mosix?

Mosix is a tool for a Unix-like kernel, such as Linux, consisting of adaptive resource sharing algorithms. It allows multiple Uni-processors and SMP’s running the same kernel to work in close cooperation. The resource sharing algorithms of Mosix are designed to respond on-line to variations in the resource usage among the nodes. This is achieved by migrating processes from one node to another, preemptively and transparently, for load-balancing and to prevent thrashing due to memory swapping [2,3].

On Fig.4 and Fig.5 results of computer experiments are shown, where Petri nets’ (PN) parameters mean numbers of places and transitions. Petri nets were randomly generated by the Paral tool [4]. Fig.4 presents, as a solution, the number of tree nodes, which was obtained in the parallel program in reference to the selected heuristic method. Fig.5 shows times of computations of deadlocks and traps using the sequential and parallel versions of the programs. The time is present in seconds (s).

<table>
<thead>
<tr>
<th>PN parameters</th>
<th>Sequential program [s]</th>
<th>Parallel program [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 / 20</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>20 / 30</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20 / 40</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>25 / 10</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>25 / 20</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>25 / 30</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>25 / 40</td>
<td>46</td>
<td>10</td>
</tr>
<tr>
<td>30 / 20</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>30 / 30</td>
<td>47</td>
<td>11</td>
</tr>
<tr>
<td>30 / 40</td>
<td>166</td>
<td>17</td>
</tr>
<tr>
<td>35 / 20</td>
<td>57</td>
<td>13</td>
</tr>
<tr>
<td>35 / 30</td>
<td>155</td>
<td>17</td>
</tr>
<tr>
<td>35 / 40</td>
<td>795</td>
<td>47</td>
</tr>
<tr>
<td>40 / 20</td>
<td>360</td>
<td>45</td>
</tr>
<tr>
<td>40 / 30</td>
<td>611</td>
<td>79</td>
</tr>
<tr>
<td>40 / 40</td>
<td>1983</td>
<td>176</td>
</tr>
</tbody>
</table>

Fig.4 Result of computer experiments – number of node of tree

<table>
<thead>
<tr>
<th>PN parameters</th>
<th>Sequential program [s]</th>
<th>Parallel program [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 / 20</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>20 / 30</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20 / 40</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>25 / 10</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>25 / 20</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>25 / 30</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>25 / 40</td>
<td>46</td>
<td>10</td>
</tr>
<tr>
<td>30 / 20</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>30 / 30</td>
<td>47</td>
<td>11</td>
</tr>
<tr>
<td>30 / 40</td>
<td>166</td>
<td>17</td>
</tr>
<tr>
<td>35 / 20</td>
<td>57</td>
<td>13</td>
</tr>
<tr>
<td>35 / 30</td>
<td>155</td>
<td>17</td>
</tr>
<tr>
<td>35 / 40</td>
<td>795</td>
<td>47</td>
</tr>
<tr>
<td>40 / 20</td>
<td>360</td>
<td>45</td>
</tr>
<tr>
<td>40 / 30</td>
<td>611</td>
<td>79</td>
</tr>
<tr>
<td>40 / 40</td>
<td>1983</td>
<td>176</td>
</tr>
</tbody>
</table>

Fig.5 Result of computer experiments – time of computation

An effectiveness of the presented parallel algorithm increases together with growing a size of input data (Fig.5). In the case of a small size of input data, the sequential program executes faster than the parallel program, because a decomposition of data for some processes takes more time than an execution time of the sequential program.

The size of the tree depends on both, a number of clauses and a number of literals in a Horn formula. The Horn formula is specified by the structure of a Petri net (the number of places and transitions, and the degree of parallelism of the net). The number of clauses (for the same number of places) is increasing along with a growing the number of transitions. Therefore, an analytical study of the time reduction as a function of the Petri net size is difficult. In addition, determining of communication time between processes is a hard problem as well.
4. Conclusions

In the paper, the parallel algorithm for a computation of all deadlocks and traps in Petri nets, has been presented. The algorithm is based on Thelen method of a calculation prime implicants. The method uses a symbolic way of analysis methods. For acceleration of computing, an advantage of parallel approaches was taken. The search tree has been decomposed in a vertical way. This solution has been tested using Linux cluster. The presented approach and the experiments show that a time reduction increases with growing the size of input data, i.e. the size of analyzed Petri nets. The presented method is used for testing of some Petri nets’ properties, namely liveness and boundedness.

A horizontal division of the search tree is considered for the future work.

Acknowledgement

The research has been financially supported by (Polish) Committee of Scientific Research (KBN) in 2004-2006 (grant No. 3 T11C 046 26).

References