Parallel Computation of Reachable Dead States in a Free-choice Petri Net

W.M.P. van der Aalst
Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, telephone: -31 40 2474295, e-mail: wsinwa@win.tue.nl

Keywords: Petri nets, Parallel algorithms, Free-choice Petri nets, Analysis of Petri nets.

Abstract. Free-choice Petri nets are a prominent tool for modeling and analyzing communication protocols, multiprocessor systems, parallel programs, flexible manufacturing systems and workflow scripts. Unfortunately, analysis is often hampered by the state-explosion phenomenon. Even for free-choice Petri nets, the reachability problem is known to be EXPSPACE-hard. In this paper we discuss a technique for the parallel computation of reachable dead states. This technique is based on the partitioning of tokens instead of the Petri net.

1 Introduction

Petri nets are used by both theoreticians and practitioners. From a theoretical point of view, Petri nets provide a formal model for concurrency with many elegant mathematical properties. From a practical point of view, Petri nets are a graphical, easy to use, technique for the modeling of systems. A very important feature of Petri nets is the fact that the Petri net representation can be used as a starting point for various kinds of analysis. The construction of the coverability graph is a well-know tool for the analysis of Petri nets. Inherently difficult problems can be tackled with this tool. Unfortunately, algorithms for constructing the coverability graph require a lot of computing power. For many Petri nets, it is not possible to construct the coverability graph in reasonable time on sequential computers. Consequently, the use of parallelism to speed up the construction of the coverability graph could be attractive.

In this paper we focus on the analysis of *dead states* in *free-choice Petri nets*. Free-choice Petri nets are a very interesting class of Petri nets for which strong theoretical results and efficient analysis techniques exist. A state s is dead if no transitions are enabled in s, i.e. nothing can happen in state s. From an analysis point of view, dead states are very important. Consider for example the situation shown in Figure 1. A production process can be represented by a free-choice Petri net. The process transforms raw materials into end-products. Given an initial state, it is interesting to know the set of dead states reachable from the initial state. Based on the set of reachable dead states, we are able to establish the correctness of the production process. If we model an administrative procedure in terms of a free-choice Petri net, then the set of dead states reachable from some initial state contains important information with regards to the correctness of the administrative procedure. At the moment, the construction of the coverability graph is the obvious way to determine the set of reachable dead states. Even for free-choice Petri nets the construction of the coverability graph may be intractable. Therefore, an effective parallel construction of the coverability graph to obtain dead states is desirable.

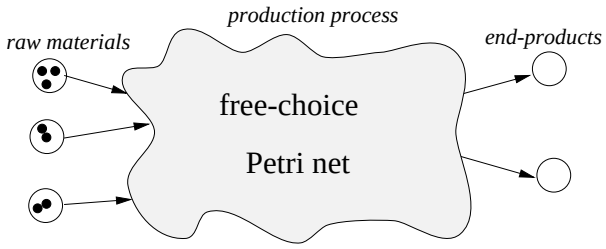

Figure 1: A production process modeled in terms of a free-choice Petri net.

In the context of Generalized Stochastic Petri nets, some attempts have been made to construct the Tangible Reachability Set (TRS) in parallel. In Caselli, Conte, Bonardi & Fontanesi (1994) and Caselli, Conte & Marenzoni (1995) both SIMD and MIMD programming models are considered for the parallel construction of the TRS. In our opinion, these approaches can also be used for the construction of the coverability graph. For an MIMD programming model, the master-slave paradigm is the obvious approach to construct the coverability graph in parallel. The master manages the set of nodes in the coverability graph computed so far and distributes new nodes over the slaves. The slaves compute new nodes, which are returned to the master. One of the starting points for such a straightforward approach, is the assumption that nodes (i.e. reachable states) are indivisible.

In this paper we present an approach which is entirely different. The reachable states represented by the nodes in the coverability graph are no longer atomic, i.e. the calculation of a new state may be distributed among several processors. We are able to use this approach by exploiting the structure of the free-choice Petri net. In fact, we have 'discovered' a new property (State-split property) with respect to the dynamics of free-choice Petri nets. Based on this property we have developed two algorithms for the parallel computation of dead states. The algorithms are named *TIGRA-I* and *TIGRA-II*. In this paper, TIGRA-I is presented. For TIGRA-II and the application of these algorithms to larger examples, the reader is refered to Aalst (1996).

The remainder of this paper is organized as follows. In Section 2 we introduce some of the basics for free-choice Petri nets. Section 3 deals with the construction of the coverability graph to obtain all dead states. In Section 4 we present the State-split theorem. This theorem serves as the basis for the TIGRA algorithm presented in Section 5.

2 Free-choice Petri nets

The classical Petri net is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

Definition 1 (Petri net) *A Petri net is a triplet* (P, T, F)*:*

- P *is a finite set of places,*
- T *is a finite set of transitions* ($P \cap T = \emptyset$)*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation)*

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. We use $\bullet t$ to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g. $p\bullet$ is the set of transitions sharing p as an input place.

Places may contain zero or more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places. We will represent a state as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. We can also represent this state as follows: $p_1 + 2p_2 + p_3$.

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

(1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.

(2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

Given a Petri net (P, T, F) and an initial state M_1, we have the following notations:

- $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ leads from state M_1 to state M_n, i.e. $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$
- $M_1 \xrightarrow{*} M_n$: there is a firing sequence which leads from M_1 to M_n

A state M_n is called *reachable* from M_1 iff $M_1 \xrightarrow{*} M_n$. A state M is a *dead state* iff no transition is enabled in M. For a state M and a place p, we use $M(p)$ to denote the number of tokens in p in state M. For two states M and N, $M \leq N$ iff for each place p: $M(p) \leq N(p)$. A Petri net (PN, M) is *bounded* iff for each place p there is a natural number n such that for every reachable state the number of tokens in p is less than n.

In this paper we focus on a restricted class of Petri nets. The results presented in this paper apply to Petri nets satisfying the so-called *free-choice property*.

Definition 2 (Free-choice) *A Petri net is a free-choice Petri net iff, for every two places* p_1 *and* p_2 *either* $(p_1\bullet \cap p_2\bullet) = \emptyset$ *or* $p_1\bullet = p_2\bullet$.

Figure 2 shows two Petri nets. Petri net (a) is a free-choice Petri net. Petri net (b) is not a free-choice Petri net, since $t2$ and $t3$ share the input place $p2$ and $t3$ is the only output transition of $p3$.
Free-choice Petri nets have been studied extensively (Desel & Esparza, 1995; Esparza, 1990) because they seem to be a good compromise between expressive power and analyzability. It is a class of Petri nets for which strong theoretical results and efficient analysis techniques exist.
One of the fundamental properties of a free-choice Petri net is the fact that it can be partitioned into *clusters*.

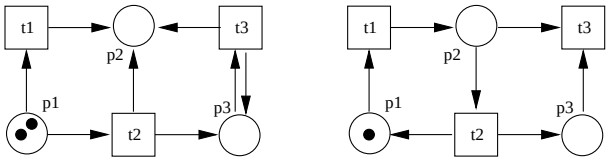

Figure 2: Petri net (a) is a free-choice Petri net, (b) is not a free-choice Petri net.

Definition 3 (Cluster) *Let t be a transition in a free-choice Petri net. The cluster of t, denoted by $[t]$, is the set $\bullet t \cup \{t' \in T \mid \bullet t' = \bullet t\}$. The cluster of a place p, also denoted by $[p]$, is the set $p\bullet \cup \{p' \in P \mid (p'\bullet \cap p\bullet) \neq \emptyset\}$.*

Note that a place p and a transition t belong to the same cluster (i.e. $[p] = [t]$) iff $p \in \bullet t$. For free-choice Petri nets, we have the following property. If transition t is enabled, then every transition in $[t]$ is enabled. A cluster c is called *enabled* iff the transitions in c are enabled.

3 Computing dead states

For bounded Petri nets it is possible to construct the so-called *reachability graph*. This graph contains a node for each state reachable from the initial state. Two nodes in the reachability graph M and N are connected by a directed arc if and only if there is a transition enabled in M whose firing results in state N. If we are able to construct the reachability graph, then it is easy to find all dead states reachable from the initial state. A node in the reachability graph corresponds to a reachable dead state if and only if it has no outgoing arcs.
Unfortunately, it is not possible to use the reachability graph for an unbounded Petri net. If we try to construct the reachability graph for an unbounded net, then this graph will grow infinitely large. This is the reason we resort to the use of the so-called *coverability graph*. For any Petri net having an arbitrary initial state, the corresponding coverability graph is finite. To obtain the coverability graph, the symbol ω is introduced. This symbol can be thought of as 'infinity' and for any integer n the following properties hold: $\omega > n$, $\omega + n = \omega$, $\omega - n = \omega$ and $\omega \leq \omega$. For a more detailed description of the algorithm to construct the coverability graph the reader is referred to Peterson (1981) or Murata (1989).
For complex Petri-net models the construction of the coverability graph may be very time consuming. The complexity of the algorithm to construct the coverability graph can be worse than primitive recursive space. For free-choice Petri nets the reachability problem is known to be EXPSPACE-hard (Cheng, Esparza & Palsberg, 1993). Moreover, even for bounded free-choice Petri nets the problem of finding all dead states is NP-hard. Therefore, it is interesting to investigate whether a parallel computer can be used to find all dead states in a free-choice Petri net. In the remainder of this paper we present a parallel algorithm for the calculation of dead states in a free-choice Petri net. The algorithm exploits the structure of a free-choice Petri net. We will show that in most situations a significant speedup is possible. Even if we execute the algorithm on a single processor, a significant speedup (compared to the standard algorithm for the construction of the coverability graph) is possible.

4 State-split theorem

The algorithm for the calculation of dead states presented in this paper exploits the structure of a free-choice Petri net. For this purpose we present a new result for free-choice Petri nets. This result is embedded in a theorem called the *State-split theorem*. This theorem shows that it is possible to distribute the construction of the coverability graph of a free-choice Petri net if we are only interested in the reachable dead states. The State-split theorem is the core of this paper. We need this theorem to prove the correctness of the parallel algorithm presented in this paper. In order to prove the State-split theorem we need some preliminary results.

The first preliminary result we present is the *Advance lemma*. This lemma shows that given a firing sequence it is possible to advance the firing of certain transitions.

Lemma 1 (Advance lemma) *Let $\sigma = t_1 t_2 \ldots t_k$ be a firing sequence of a free-choice Petri net such that σ leads from state M to state M', i.e. $M \xrightarrow{\sigma} M'$. If a cluster c is enabled in state M and t_i is the first transition in σ such that $t_i \in c$, then $M \xrightarrow{\sigma'} M'$ with $\sigma' = t_i t_1 t_2 \ldots t_{i-1} t_{i+1} \ldots t_k$.*

Proof.
In state M each of the transitions in c is enabled, i.e. t_i is enabled in state M. The transitions t_j with $1 \leq j < i$ are not disabled by the advanced firing of t_i, because they belong to different clusters. Therefore, the firing sequence $\sigma' = t_i t_1 t_2 \ldots t_{i-1} t_{i+1} \ldots t_k$ is possible. Since σ' is a permutation of σ, we deduce that $M \xrightarrow{\sigma'} M'$. □

We use the Advance lemma to prove the *Substate-ordering lemma*. The Substate-ordering lemma captures the essence of the State-split theorem. The Substate-ordering lemma is illustrated in Figure 3.

Lemma 2 (Substate-ordering lemma) *Let PN be a free-choice Petri net and N and N' states of PN such that $N \xrightarrow{*} N'$ and N' is dead. For any substate M of N (i.e. $M \leq N$),*

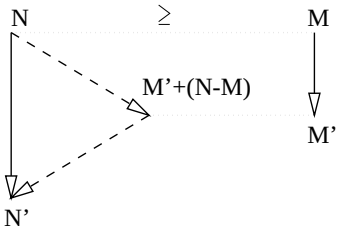

Figure 3: The Substate-ordering lemma.

there is a dead state M' such that $M \xrightarrow{*} M'$ and $M' + (N - M) \xrightarrow{*} N'$.

Proof.
Let $\sigma = t_1 t_2 \ldots t_k$ be an arbitrary firing sequence leading from N to N' ($N \xrightarrow{\sigma} N'$). We use induction upon the length k of σ.
If $k = 0$, then $N = N'$. Since N is dead ($N = N'$) and $M \leq N$, M is also dead. Hence, $M' = M$ is a dead state such that $M \xrightarrow{*} M'$ and $M' + (N - M) \xrightarrow{*} N'$.
Assume $k > 0$. If M is dead, then for $M' = M$ the lemma holds. Therefore, we may assume that M is not dead. Let t_i be the first transition in σ which is enabled in M, i.e. t_i is enabled in M and for all $1 \leq j < i$: t_j is not enabled in M. Note that such a transition exists, because $M \leq N$, M is not dead and N' is dead. The cluster $[t_i]$ is enabled in N and t_i is the first transition in σ which belongs to $[t_i]$. We can use lemma 1 to prove that $N \xrightarrow{\sigma'} N'$ with $\sigma' = t_i t_1 t_2 \ldots t_{i-1} t_{i+1} \ldots t_k$. Let N_1 and M_1 be states such that $N \xrightarrow{t_i} N_1$ and $M \xrightarrow{t_i} M_1$. By the induction hypothesis we can show that there is a dead state M' such that $M_1 \xrightarrow{*} M'$ and $M' + (N_1 - M_1) \xrightarrow{*} N'$. By the definition of N_1 and M_1 we conclude that $M \xrightarrow{*} M'$ and $M' + (N - M) \xrightarrow{*} N'$. □

We use the Substate-ordering lemma to prove the State-split theorem illustrated in Figure 4.

Theorem 1 (State-split theorem) *Let $PN = (P, T, F)$ be a free-choice Petri net and let M and and M' be two states such that $M \xrightarrow{*} M'$ and M' is dead. If we split M up in n substates $M_1, M_2, \ldots M_n$ (i.e. $M = M_1 + M_2 + \ldots + M_n$), then there exist n dead states $M'_1, M'_2, \ldots M'_n$ such that for any i ($1 \leq i \leq n$) $M_i \xrightarrow{*} M'_i$ and $M'_1 + M'_2 + \ldots M'_n \xrightarrow{*} M'$.*

Proof.
Let $M_1, M_2, \ldots M_n$, M and M' be states of PN such that $M = M_1 + M_2 + \ldots + M_n$ and $M \xrightarrow{*} M'$. We need to prove that there exist n dead states $M'_1, M'_2, \ldots M'_n$ such that for any i ($1 \leq i \leq n$) $M_i \xrightarrow{*} M'_i$ and $M'_1 + M'_2 + \ldots M'_n \xrightarrow{*} M'$. Since $(M_1 + M_2 + \ldots M_{n-1}) + M_n \xrightarrow{*} M'$, we can use

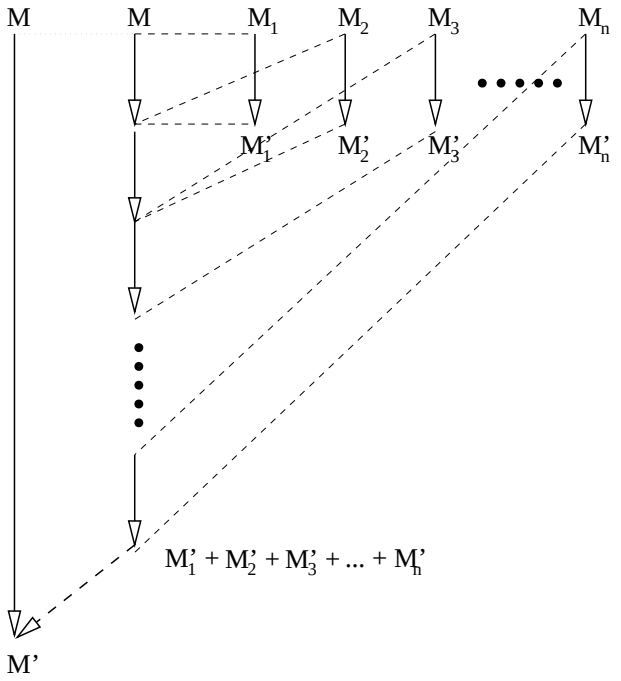

Figure 4: The State-split theorem.

Lemma 2 to deduce that there is a dead state M'_n such that $M_n \xrightarrow{*} M'_n$ and $M'_n + (M_1 + M_2 + \ldots M_{n-1}) \xrightarrow{*} M'$. If $n > 1$, then we can repeat this step. Since $(M'_n + M_1 + M_2 + \ldots M_{n-2}) + M_{n-1} \xrightarrow{*} M'$, we can use Lemma 2 to deduce that there is a dead state M'_{n-1} such that $M_{n-1} \xrightarrow{*} M'_{n-1}$ and $M'_{n-1} + (M'_n + M_1 + M_2 + \ldots M_{n-2}) \xrightarrow{*} M'$. This process can be repeated until all M_i have been removed, i.e. there are dead states $M'_1, M'_2, \ldots M'_n$ such that for any i ($1 \leq i \leq n$) $M_i \xrightarrow{*} M'_i$ and $M'_1 + M'_2 + \ldots M'_n \xrightarrow{*} M'$. □

The State-split theorem shows that any firing sequence which leads to a dead state, can be composed of $n + 1$ firing sequences which are relatively independent. In fact the first n firing sequences can be calculated in parallel. This property is used to define the parallel algorithm for the calculation of dead states.
To conclude this section we focus attention on the following lemma which holds for any Petri net.

Lemma 3 *Let M be the initial state of a Petri net $PN = (P, T, F)$. If we split M up in n substates $M_1, M_2, \ldots M_n$ (i.e. $M = M_1 + M_2 + \ldots + M_n$), then any state $M' = M'_1 + M'_2 + \ldots M'_n$ such that for any i ($1 \leq i \leq n$) $M_i \xrightarrow{*} M'_i$ is reachable from M (i.e. $M \xrightarrow{*} M'$).*

Lemma 3 is a well-known result, therefore the proof of this lemma has been omited.

5 Parallel computation of dead states

In Section 3 we showed that the coverability graph can be used to characterize all reachable dead states. If the number of dead states is finite, then is is possible to construct a coverability graph which contains all reachable dead states. Unfortunately, only for small or very simple Petri nets the construction of the coverability graph is feasible. Based on these observations, we started a quest for a parallel algorithm to speed up the generation of dead states.

If we observe the coverability graph algorithm described in Section 3, then a simple parallel algorithm which uses a workpool containing nodes which should be examined seem to be the obvious choice. In Caselli, Conte & Marenzoni (1995) multiple workpools are used, i.e. each processor generates a private set of reachable states and periodic synchronization points are used to distribute the union of all known reachable states. It is also possible to use a single workpool by adopting the master-slave paradigm. The master manages a pool containing nodes which need to be examined. These nodes are distributed over slaves, which return new nodes to the master. In both cases nodes or states are indivisible, i.e. elements in a workpool correspond to complete nodes or states instead of parts of nodes or states.

We propose a less straightforward approach which exploits the property recorded in the State-split theorem. Nodes in the coverability graph are no longer indivisible, i.e. multiple processors may be working on the calculation of parts of a state. In fact, we propose an approach where tokens instead of states are distributed over the processors.

The first parallel algorithm, named *TIGRA*, is quite simple. Given a parallel system with n processors and an initial state M, the tokens in M are partitioned over n substates M_1, M_2, $\ldots$ M_n. Each of the processors stores one of these substates in its private memory, i.e. there is a one-to-one correspondence between the processors and the substates M_1, M_2, $\ldots$ M_n. For each of the substates a coverability graph is constructed and the set of dead states reachable from the corresponding substate is recorded. This can be done in parallel, without any need for intermediate synchronization. When each of the processors has completed the construction of the local set of dead states, all possible combinations of dead states are stored in a workpool. The states in the workpool are distributed over the processors. For each state one of the processors calculates the set of reachable dead states. The program terminates when all states in the workpool have been evaluated. The TIGRA algorithm can be sketched as follows.

TIGRA Algorithm

(1) Partition the set of tokens in state M into n states M_1, M_2, $\ldots$ M_n, i.e. $M = M_1 + M_2 + \ldots + M_n$:

$M_1, M_2, \ldots, M_n \leftarrow$ *PARTITION*(M)

(2) For each processor i compute the set of dead states DS_i reachable from state M_i:

$DS_i \leftarrow$ *CONSTRUCT_DEAD_STATES*(M_i)

(The set DS_i contains all dead states M_i' reachable from M_i.)

(3) Construct the set of states S which contains all possible combinations of dead states, i.e. $S = \{M_1' + M_2' + \ldots + M_n' \mid \text{for all } i : M_i' \in DS_i\}$:

$S \leftarrow$ *CONSTRUCT_ALL_COMBINATIONS* $(M_1', M_2', \ldots M_n')$

(4) Compute the set of dead states DS reachable from any state in S.

$DS \leftarrow$ *CONSTRUCT_DEAD_STATES*(S)

(The states in the set S are distributed over the processors.)

The correctness of the TIGRA algorithm can easily be verified using the State-split theorem.

Theorem 2 *Let $PN = (P, T, F)$ be a free-choice Petri net and let M be the initial state. The set DS constructed using the TIGRA algorithm contains all dead states reachable from state M.*

Proof.
Let M' be an arbitrary dead state reachable from the initial state M. We have to prove that M' is an element of DS. The first step of the TIGRA algorithm partitions state M into $M_1, M_2, \ldots M_n$ such that $M = M_1 + M_2 + \ldots + M_n$. By Theorem 1 we know that there exist n dead states $M_1', M_2', \ldots M_n'$ such that for any i $(1 \leq i \leq n)$ $M_i \stackrel{*}{\rightarrow} M_i'$ and $M_1' + M_2' + \ldots M_n' \stackrel{*}{\rightarrow} M'$. These dead states are computed in the second step of the algorithm. In the third step all possible states of the form $M_1' + M_2' + \ldots M_n'$ are constructed. In the fourth step all dead states reachable from these constructed states are calculated including state M'. □

5.1 Performance of the TIGRA algorithm

It is difficult to evaluate the performance of the TIGRA algorithm. First of all, the performance of the algorithm highly depends on the partitioning in step (1) of the algorithm. Secondly, the size of the Petri net is not a good measure for the size of the corresponding coverability graph. Even for moderate size Petri nets, the coverability graph may be very large. On the other hand, there are large Petri nets for which the coverability graph is surprisingly small. In other words the structure of the Petri net and the initial state may influence the size of the corresponding coverability graph dramatically. Finally, we are faced with the problem that the 'best' sequential algorithm for the computation of reachable dead states is not known. This makes it difficult to determine the speedup of the TIGRA algorithm. Nevertheless, we can make some statements about the performance of the TIGRA algorithm.

Let us assume that we have a free-choice Petri net *PN* and an initial state *M* such that the number of reachable dead states is finite. Moreover, we assume that the number of nodes in the corresponding coverability graph is equal to T. T is a good measure for the time required to construct the coverability graph. Therefore, we define T to be the time required to find all dead states using the traditional approach on a single processor system.
For the free-choice Petri net *PN* with the initial state *M* we define T_n' to be the sum of the number of nodes of the coverability graphs constructed in step (2) and step (4) of the TIGRA algorithm. The time required to process step (3) is proportional to the time required to process step (2) and step (4). Moreover, the time required to process step (3) is small compared to the time required to process step (2) and step (4). Therefore, T_n' is a good measure for the time required to compute all reachable dead states on a single processor system using the TIGRA algorithm. In other words T_n' is a measure for the processing time if we emulate n processors on a single processor system using the TIGRA algorithm.
Finally, we define T_n to be the maximum number of nodes in one of the coverability graphs constructed in step (2) plus the maximum number of nodes handled by one of the n processors in step (4) of the TIGRA algorithm. Clearly, T_n is a reasonable measure for the time required to compute all reachable dead states using the TIGRA algorithm on an MIMD system with n processors.

Since the 'best' sequential algorithm is unknown, it is difficult to define a speedup measure to evaluate the improvement in time performance of the TIGRA algorithm on a system with n processors compared to single processor system. Therefore we define two speedup measures. The first speedup measure S_n compares the performance of the TIGRA algorithm on a system with n processors with the standard technique based on one coverability graph: $S_n = \frac{T}{T_n}$. The second speedup measure S_n' compares the performance of the TIGRA algorithm on a system with n processors with the performance of the TIGRA algorithm on a single processor system: $S_n' = \frac{T_n'}{T_n}$. We will use both speedup measures to characterize the time performance of the TIGRA algorithm. If we use the measure S_n a superlinear speedup (i.e. $S_n > n$) is possible. This is a result of the fact that we compare two alternative algorithms. We also define two measures for the efficiency of the TIGRA algorithm. The first measure of efficiency is based on S_n: $E_n = \frac{T}{nT_n}$. We can also define an efficiency measure based on S_n': $E_n' = \frac{T_n'}{nT_n}$.

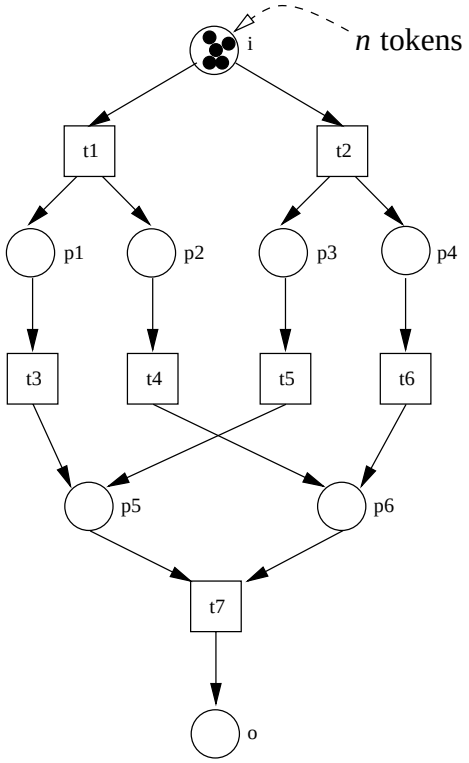

Figure 5: A free-choice Petri net with n tokens in place i.

To show the potential of the TIGRA algorithm, we will analyze the Petri net shown in Figure 5. The number of tokens in place i is variable and equal to the number of processors n. In step (1) of the TIGRA algorithm each token is assigned to a private processor. The size of the coverability graph for the Petri net shown in Figure 5 highly depends in the initial number of tokens in place i. If place i contains just one token, then the corresponding coverability graph contains only 9 nodes. If place i contains just 10 token, then the corresponding coverability graph contains 24815 nodes. Ta-

ble 1 shows some results for the Petri net shown in Figure 5. If the number of processors is equal to 8 and we start with 8 tokens in place i, then the speedup S_n is equal to 1027.10, i.e. the TIGRA algorithm is more than 1000 times as fast as the conventional algorithm based on the construction of one coverability graph. This example shows that a superlinear speedup is possible if we use the measure S_n. If we use the measure S'_n the speedup is linear, i.e., the speedup is equal to the number of processors.

n	T	T_n	S_n	E_n	S'_n	E'_n
1	9	9	1.00	1.00	1.00	1.00
2	45	9	5.00	2.50	2.00	1.00
3	159	9	17.67	5.89	3.00	1.00
4	450	9	50.00	12.50	4.00	1.00
5	1090	9	121.11	24.22	5.00	1.00
6	2354	9	261.55	43.59	6.00	1.00
7	4654	9	517.11	73.87	7.00	1.00
8	8579	9	1027.10	128.39	8.00	1.00
9	14939	9	1659.89	184.43	9.00	1.00
10	24815	9	2757.22	275.72	10.00	1.00

Table 1: Some results for the Petri net shown in Figure 5.

The Petri net shown in Figure 5 is just an example of a 'problem instance'. It is difficult to estimate the speedup for an arbitrary problem instance. Nevertheless, the following lemma holds.

Lemma 4 *For the TIGRA algorithm the following relation holds:* $T_n \leq T$.

Proof.
By Lemma 3 we know that any state generated in step (2), (3) or (4) of the TIGRA algorithm, is also present in the standard coverability graph. □

Lemma 4 shows that $1 \leq S_n$, i.e. the effect of the TIGRA algorithm is never negative if we abstract from overhead. By the definitions of S'_n and S_n we also deduce that $S'_n \leq n$ and $S'_n \leq S_n$. For practical situations: $T'_n \leq T$. This means that in practise even for a single processor system the TIGRA algorithm turns out to be fruitful.

6 Conclusion

The algorithm presented in this papers allows for the efficient calculation of dead states in a free-choice Petri net. The TIGRA algorithm allows for superlinear speedups compared to the traditional approach. Unfortunately, the TIGRA algorithm is not very robust. Therefore, we developed the more robust TIGRA-II algorithm (Aalst, 1996). Experiments show that tremendous speedups are possible. In fact, we can also use the two algorithms on a single processor system and obtain remarkable speedups compared to the traditional technique of constructing one coverability graph. These results are possible by exploiting the fundamental property retained in the State-split theorem.

References

AALST, W.M.P. VAN DER (1996), Parallel Computation of Reachable Dead States in a Free-choice Petri Net, Computing Science Reports 96/03, Eindhoven University of Technology, Eindhoven.

CASELLI, S., G. CONTE, F. BONARDI, ET AL. (1994), Experiences on SIMD Massively Parallel GSPN Analysis, in: G. Haring and G. Kotsis (eds.), *Proceedings of the 7th International Conference of Modelling Techniques and Tools for Computer Performance Evaluation*, Lecture Notes in Computer Science 794, Springer-Verlag, Berlin, 265–283.

CASELLI, S., G. CONTE, AND P. MARENZONI (1995), Parallel State Space Exploration for GSPN Models, in: G. De Michelis and M. Diaz (eds.), *Application and Theory of Petri Nets 1995*, Lecture Notes in Computer Science 935, Springer-Verlag, Berlin, 181–200.

CHENG, A., J. ESPARZA, AND J. PALSBERG (1993), Complexity results for 1-safe nets, in: R.K. Shyamasundar (ed.), *Foundations of software technology and theoretical computer science*, Lecture Notes in Computer Science 761, Springer-Verlag, Berlin, 326–337.

DESEL, J. AND J. ESPARZA (1995), *Free choice Petri nets*, Cambridge tracts in theoretical computer science 40, Cambridge University Press, Cambridge.

ESPARZA, J. (1990), Synthesis rules for Petri nets, and how they can lead to new results, in: J.C.M. Baeten and J.W. Klop (eds.), *Proceedings of CONCUR 1990*, Lecture Notes in Computer Science 458, Springer-Verlag, Berlin, 182–198.

HACK, M.H.T. (1972), Analysis production schemata by Petri nets, Master's thesis, Massachusetts Institute of Technology, Cambridge, Mass.

MURATA, T. (1989), Petri Nets: Properties, Analysis and Applications, *Proceedings of the IEEE* **77**, 541–580.

PETERSON, J.L. (1981), *Petri net theory and the modeling of systems*, Prentice-Hall, Englewood Cliffs.