Efficient Verification using Generalized Partial Order Analysis

Steven Vercauteren*, Diederik Verkest*, Gjalt de Jong† and Bill Lin‡
* IMEC Laboratory, Kapeldreef 75, B-3001 Leuven, Belgium
† Alcatel Telecom, F.Wellesplein 1, B-2018 Antwerpen, Belgium
‡ ECE Department, University of California, San Diego, La Jolla, CA

Abstract

This paper presents a new formal method for the efficient verification of concurrent systems that are modeled using a safe Petri net representation. Our method generalizes upon partial-order methods to explore concurrently enabled conflicting paths simultaneously. We show that our method can achieve an exponential reduction in algorithmic complexity without resorting to an implicit enumeration approach.

1 Introduction

Most modern embedded systems are notoriously difficult to design. Despite utmost care exercised by designers, initial design specifications often contain subtle, difficult to detect, errors that result from unanticipated interactions between the concurrent parts. Traditional analysis methods such as simulation are often inadequate for uncovering such errors, especially those that only occur under rare conditions. Thus, automated formal verification tools are becoming an indispensable part of a designer's tool-box.

In [16] we explained how both specification and implementation of an embedded system can be formally represented by the Petri net formalism [12]. In this paper, we will focus on a novel formal method for efficiently verifying concurrent systems modeled as a Petri net. Conventional analysis for Petri nets mainly involves a reachability analysis of the underlying state space. However, there are two primary sources of combinatorial explosion that makes this conventional approach intractable for many problem instances.

The first source is due to concurrently enabled actions. Due to the underlying unbounded delay assumption of Petri nets, concurrently enabled actions may fire in any order. This interleaving semantics requires the analysis to enumerate all possible orderings, which has a factorial complexity with respect to the number of concurrently enabled actions. To circumvent this problem, partial-order analysis (also referred to as stubborn-set or anticipation analysis) techniques have been developed where it has been shown that only one interleaved sequence needs to be analyzed for deadlock and liveness checks [6, 9, 14].

The second source is due to concurrently marked conflict places. A conflict place specifies a choice in a Petri net. In conventional analysis, each branch of a conflict place must be traversed independently. When there are multiple conflict places marked concurrently, all possible combinations of paths must be enumerated, which has an exponential complexity with respect to the number of concurrently marked conflict places. This source of complexity is not avoided by partial-order analysis techniques, thus leaving many problem instances still intractable.

In this paper, we describe a *generalized partial-order analysis* technique that can enumerate conflicting paths simultaneously, thus extending the partial-order analysis approach to tackle also the second source of combinatorial explosion. Our technique is based on a modified representation of markings to distinguish the different conflicting paths. The firing rules have been modified in combination with the partial-order analysis technique to enumerate conflicting paths simultaneously. This new analysis technique can demonstrate exponential reduction in complexity, as illustrated by the examples shown in Section 4.

Another technique for tackling the combinatorial explosion problem is symbolic analysis. Symbolic analysis approaches [2, 3, 5, 11] that implicitly enumerate the state space have been used to tackle the complexity problem; they are effective when the state space being traversed can be efficiently encoded using binary decision diagrams. We believe this approach is complementary to our method that aims to address specifications whose state space cannot be efficiently encoded.

The remainder of this paper is organized as follows. Section 2 reviews the basic definitions and properties of Petri nets, as well as conventional, partial-order and symbolic analysis techniques. Section 3 presents our generalized partial-order analysis approach. Section 4 discusses the implementation aspects and the results. Conclusions are drawn in Section 5.

2 Background

In this section we provide some background material necessary for the exposition of our work. In Section 2.1 we review basic definitions and properties of (classical) Petri nets [12]. In Section 2.2 we discuss a straightforward approach to verification and the problems involved. In Section 2.3 partial-order analysis techniques are discussed, as well as their limitations. In Section 2.4 the symbolic techniques are elaborated.

2.1 Petri Nets

Definition 2.1 (Petri Net) *A Petri net is a tuple* $\Sigma = \langle P, T, F, m_0 \rangle$, *with* $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ *and* $m_0 : P \to \mathbb{N}$.

In the above definition P denotes a set of places, T a set of transitions, F a flow relation and m_0 an initial marking. In Figure 1(a), a simple Petri net (PN) is shown. The places are depicted with the open circles, the transitions are depicted with the annotated bars, and the flow relation is represented by the arcs. The black dots represent tokens, and the initial token configuration represents the initial marking. In the sequel, these notations will be kept. For a place (transition) x, $\bullet x$ and $x\bullet$ denote the preset and postset of x, that are referred to as the set of input and output transitions (places) of x, respectively.

Two transitions are *conflicting* or are said to be *in conflict* when they share common input places. In a *maximal conflict(ing) set* (MCS) all transitions that are in conflict with a transition of the set, are also included in the set.

Definition 2.2 *Let* $t, u \in T$.
$conflict(t, u) \equiv (\bullet t \cap \bullet u \neq \emptyset)$
$mcs(T) \equiv \{T' | \forall t \in T \setminus T' : \forall u \in T' : \neg conflict(t, u)\}$

Besides the structure of a Petri net, there is also an associated dynamics. A *state* or marking, is the mapping of the places to the natural numbers $P \to \mathbb{N}$, indicating the number of tokens in the places. Transitions between states are dictated by the following firing rule. In the sequel M_P denotes the set of all states (markings) of a Petri net with $|P|$ places.

Definition 2.3 (Enabling Rule) *Let* $t \in T$ *and* $m \in M_P$.
$enabled(t, m) \equiv \forall p \in \bullet t : m(p) \geq 1$

Definition 2.4 (Firing Rule) *Let* $t \in T$, $m \in M_P$ *and* $enabled(t, m) = true$.

$$nextstate(m(p), t) = \begin{cases} m(p) \Leftrightarrow 1 & \textit{if } p \in \bullet t \setminus t\bullet \\ m(p) + 1 & \textit{if } p \in t\bullet \setminus \bullet t \\ m(p) & \textit{otherwise} \end{cases}$$

Definition 2.3 states that a transition t can fire if all its input places contain at least one token. Definition 2.4 states that firing of t removes one token in all its input places and adds a new token in all its output places.

The set of all reachable states is represented in a *reachability graph*, as shown in Figure 1(b). In such a reachability graph all vertices correspond to a valid marking of the Petri net and all arcs correspond to a transition from one marking to another due to firing of some transition in the net. The reachability graph of a Petri net N, denoted as $RG(N)$, can then be interpreted as the reflexive transitive closure of the next-state relation defined in Definition 2.4.

Two important properties of Petri nets are *liveness* and *safeness*. Liveness concerns the question whether a transition can ever be fired, and is opposed to deadlock. Safeness means that a place *should not* contain more than one token at any time. In this paper only safe Petri nets are considered.

2.2 Conventional Analysis

A straightforward approach to verification is to explicitly enumerate all reachable states. Reachability analysis, also known as exhaustive simulation or state space generation is indeed a powerful formal method for detecting errors in concurrent and distributed systems that have a finite state space. A deadlock is then said to occur when there is a reachable state from which the system (Petri net) cannot perform any action (transition). This approach however suffers from the *state explosion problem*, that is an exponential increase in the number of reachable states. The source of this exponential complexity are concurrently enabled actions (transitions). In Petri net terms this is illustrated in Figure 1 by means of an example.

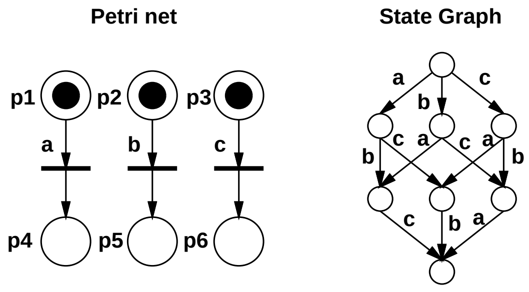

Figure 1: (a) Marked Petri net (b) Reachability graph

In the example of Figure 1(a), a simple Petri net is depicted residing in an initial state or marking where transitions A, B and C are enabled. In this state, all three transitions can be fired separately, each firing resulting in a new state. In each of these new states, the two other transitions remain enabled and can be fired, in turn leading to two new states. In this example, we then end up with 3! different firing sequences or *interleavings*, as can be seen in Figure 1(b). This factorial blow up causes the state space explosion problem: especially for large systems that exhibit lots of concurrency, the state space can be too large with respect to the time and other resources needed to inspect all states in the space.

2.3 Partial Order Analysis

To verify the liveness properties of a Petri net, it can be shown [6] that it is not necessary to build the complete reachability graph for a net. Instead of constructing all interleaving execution sequences, only a few are needed to extract the external behavior and these still cover the internal non-determinism. Continuing with the example of Figure 1(a), suppose we are in the (depicted) initial state where transitions A, B, C are all enabled. Instead of firing all 3 transitions, in this state it is only necessary to pursue with one transition, at least if liveness is the main concern of our analysis. Indeed, the firing of A will keep transitions B and C enabled. In turn, in the next state it is sufficient to fire only B, for example. Finally, we only have selected one path from the full reachability graph. So we went from N! factorial interleaving to N linear interleavings, which is quite substantial.

The method can also be lifted to transitions that are in conflict, that is transitions that share common input places. Suppose in a certain state, T' is a maximal (i.e. it cannot be extended) set of conflicting transitions that are all enabled. Then it is clear, that from this particular state, it is sufficient to fire only those transitions belonging to T', "anticipating" all other enabled transitions. Indeed, the latter transitions remain enabled after firing the transitions of T', therefore not affecting the liveness properties of the net.

The method has been presented in different variants [6, 9, 14]; they are all based on the observations described above. By using partial-order semantics for state-transition based systems, they abstract from the interleaving semantics. Although originally presented to preserve liveness properties, the method can also be used to deduce *safeness* properties of the net and is even partially applicable to model checking as described in [6, 9].

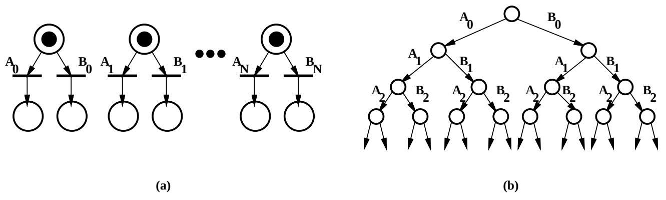

Figure 2: (a) Petri net (b) "Anticipated" reachability graph

Problem. These methods, however, still have problems with concurrently enabled *conflict places*. Suppose a marked Petri net as shown in Figure 2. In this example Petri net, we have N pairs of conflicting transitions (the marked places are called conflict places). For this application, the partial-order methods compute the reachability graph depicted in Figure 2(b). It first selects and fires $\{A_0, B_0\}$ — i.e. a maximal set of conflicting transitions that is enabled — then $\{A_1, B_1\}$, and so forth. Although the computed reachability graph is significantly smaller than the full reachability graph it still results in a number of states equal to $2^{N+1} \Leftrightarrow 1$.

From all this, it is clear that it would be desirable to not restrict the analysis to firing each transition sequentially, as with classical firing rules of Petri nets. Instead we would like to order the transitions, and evaluate points at which series of transitions can be fired, since we do not need to consider intermediate steps, but only final reachability information.

2.4 Symbolic Reachability Analysis

Another approach to reachability analysis is to use Ordered Binary Decision Diagrams (OBDD's) [2] to represent the state graph symbolically. OBDD's are known to be compact representations for symmetric functions. For applications with non-linear communication patterns, however, symbolic techniques generally perform worse [4], as the non-linear structure makes it difficult to find a good variable ordering for the OBDD's. Thus, the state explosion problem can still be present, especially as the encoding of the transition function is based on the interleaving semantics.

Recently, an approach was described [1] that incorporated partial-order reduction into an OBDD-based symbolic reachability analysis. While this method improves over standard symbolic reachability analysis, it still requires an efficient encoding of the state space, which may not exist. We therefore believe that [1] is complementary to our method that aims to address specifications whose state space cannot be efficiently encoded.

3 Generalized Partial Order Analysis

In this section the key ideas of our generalized partial-order analysis approach are elaborated in detail. Section 3.1 presents an intuitive overview. Section 3.2 formally defines Generalized Petri nets, the working vehicle of our analysis approach. Section 3.3 discusses the analysis procedure itself.

3.1 Rationale

As mentioned in the discussion on the partial-order methods, for the Petri net shown in Figure 2, the firing rules of classical Petri nets restrict the analysis to firing each transition sequentially. In this work, we overcome this problem, by enumerating conflicting paths simultaneously. Simply putting a token in the output places of the fired transitions will not suffice, as this would lead to execution sequences that are not possible in the "original" reachability graph, possibly hiding the presence of deadlock situations. It is clear that the representation of the markings has to be modified to distinguish the different conflicting paths. In this paper we therefore present a modified Petri net model, called a *Generalized Petri Net (GPN)*, that is clarified in Figure 3.

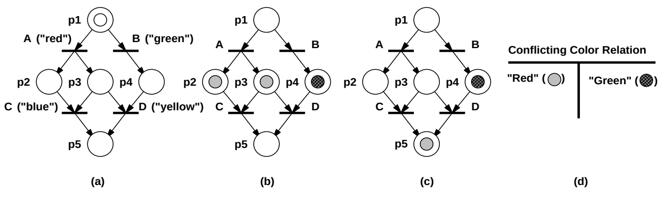

Figure 3: Successive states of a Generalized Petri Net (a) initial state (b) after firing A and B simultaneously (c) after firing C

In Figure 3(a) a simple GPN is shown; place $p1$ is filled with a "white" token, and the transitions A, B, C and D are tagged with the colors "red", "blue", "green" and "yellow", respectively[1]. In this initial state, both A are B are enabled, and can fire simultaneously to arrive in the state depicted in Figure 3(b). Transition A removes the white token from its input place $p1$, "paints" it red, and moves it to each of its output places. Thus, $p2$ and $p3$ get filled with a red token. Similarly, transition B removes the white token from its input place $p1$, paints it green, and moves it to its output place $p4$. The input places of D, viz. $p3$ and $p4$, contain tokens with mutual *conflicting colors* (as they correspond to conflicting transitions). Thus, we don't allow transition D to fire. Transition C, on the other hand, can fire. This results in the state depicted in Figure 3(c). As C is the only transition that is fired at this stage, there is no need for an extra coloring by C; the firing of C removes a red token from its input places $p2$ and $p3$, and puts a red token in its output place $p5$. Note that the states, obtained as such, capture all the information that is related to the "original" reachability graph.

This approach is compatible with the techniques of the partial-order methods described in Section 2.3. Suppose the example Petri net shown in Figure 2(a). Following the reasoning of the partial-order methods, in this state it is sufficient to fire only those transitions that belong to one of the N concurrently enabled conflicting sets. In turn, in the next states, another conflicting set of transitions can then be selected. When combined with our approach of firing a conflicting set of transitions simultaneously, this comes down to only enumerating one possible interleaving, or alternatively, firing all conflicting sets at the same time. For this example, we then go from $2^{N+1} \Leftrightarrow 1$ to only 2 computed states!

In the following, this intuitive introduction to Generalized Petri nets is more formalized.

3.2 Generalized Petri Nets

In Section 3.1 we gave an intuitive introduction to Generalized Petri Nets (GPNs). We explained the GPN model in terms of places that are marked with colored tokens. In a formal setting, these colors will be represented by *transition sets*, to distinguish the different conflicting paths. We also indicated that colors may be conflicting. For example, in Figure 3(d) we suggested a conflicting color relation, that states that the colors "red" and "green" are indeed conflicting. In the following, this color relation will be represented – in the reverse sense – by a *set of valid transition sets.*

Definition 3.1 (Generalized Petri Net) *A Generalized Petri Net is a tuple $\langle P, T, F, m_0, r_0 \rangle$ with $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$, $m_0 : P \to 2^{2^T}$ the initial marking and $r_0 \subseteq 2^{2^T}$ the valid transition sets.*

Besides the structure of the GPN there is also an associated dynamics. A *state* is a tuple $\langle m, r \rangle$ where m denotes the *marking* or the mapping of the places to the sets of transition sets $P \to 2^{2^T}$, and r denotes the set of all *valid* transition sets. In the sequel P represents a set of places, T a set of transitions, and S_P^T the set of all states of a GPN with $|P|$ places and $|T|$ transitions.

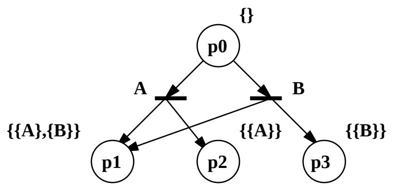

Figure 4: Marking of GPN after firing A and B simultaneously

To clarify the above, imagine a example GPN marking as illustrated in Figure 4. In this example, a GPN marking or state is shown after firing A and B simultaneously. The places $p2$ and $p3$ get filled with $\{\{A\}\}$ and $\{\{B\}\}$, respectively, while place $p0$ becomes empty — or better contains an empty set. The need for a set of sets notation is clear when looking at place $p1$. This place has two incoming paths, and as a result gets "filled" with two sets $\{A\}$ and $\{B\}$. Associated with this depicted state, there is also a set of valid sets $r = \{\{A\}, \{B\}\}$. For this example, $\{A, B\}$ will not be a valid set – and therefore not included in r – because A and B are conflicting transitions. Intuitively, a valid set of transitions denotes a set of transitions that can "act" together to enable and fire a certain transition. Imagine a transition E that has $p2$ and $p3$ as its input places. The conflicting transitions A and B cannot act together to fire transition E, and therefore $\{A, B\}$ is not included in the set of valid sets. This information can then be used to "guard" the enabling of transition E, effectively preventing the latter transition from being fired. The classical PN enabling condition and firing rules have to be changed as such, as will become clear below.

Let us return to Figure 3. The actions of firing A and B simultaneously, and of firing C, are treated differently. In

[1]Generalized Petri nets should not be confused with Colored Petri Nets [15]. The latter are used in a different context. The notion of colors is introduced just to convey the idea.

the former case, the tokens that arrive in the output places of A and B are colored, as to distinguish their mutual exclusive origin. In the latter case, no extra coloring is performed, as we don't need to distinguish from a conflicting transition that may be fired at the same time (transition D cannot fire!). Thus, for the firing of C, we can apply a straightforward extension of the "original" PN firing rule: a red token is removed from each of its input places $p2$ and $p3$, and put into its output place $p5$. From all this, it is clear, that in our framework we need two firing semantics, namely a *multiple firing semantics* and a *single firing semantics*. These will be explained hereafter.

Single Firing Semantics. According to the single firing semantics a transition can fire once it is *single enabled*. Remember from the above discussion, that the marking of a place in a GPN can be interpreted as to represent all possible histories of transition firings that lead to the involved marking. We then say a transition is single enabled when its input places contain a "common" history, or alternatively, when the intersection of the transition sets, contained in its input places, is not empty. If so, the firing of that transition will remove that common history from its input places, and put it in its output places.

Definition 3.2 (Single Enabling Rule) *Let* $\langle m, r\rangle \in S_P^T$, $t \in T$. $s\_enabled(t, \langle m, r\rangle) \equiv \bigcap_{p\in \bullet t} m(p) \cap r$

Suppose a GPN in a state depicted in Figure 5(a). r denotes the set of valid transition sets; its derivation will be described later in more detail. For now, it suffices to know that $\{A, B\}$ is not a valid set because A and B are conflicting transitions. Transition B is not single-enabled as $s\_enabled(B, m, r) = \{\}$. Transition A, on the contrary, is single-enabled as $s\_enabled(A, \langle m, r\rangle) = \{\{A\}\}$. Thus, transition A can fire to go to the next state, according to the *single firing rule*.

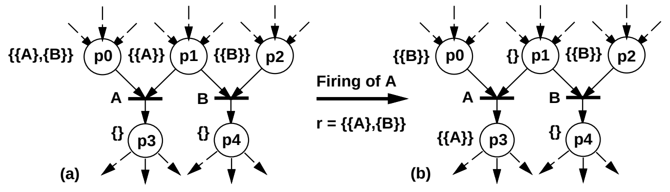

Figure 5: Illustration of the single firing semantics. (a) Example GPN marking m with valid sets $r = \{\{A\}, \{B\}\}$ (b) Next marking m' after firing transition A.

Definition 3.3 (Single Firing Rule) *Let* $s = \langle m, r\rangle \in S_P^T$, $t \in T$. $s\_update(m, \langle m, r\rangle) \equiv \langle m', r\rangle$
where

$$m'(p) = \begin{cases} m(p) \setminus s\_enabled(t, s) & \text{if } p \in \bullet t \setminus t\bullet \\ m(p) \cup s\_enabled(t, s) & \text{if } p \in t\bullet \setminus \bullet t \\ m(p) & \text{otherwise} \end{cases}$$

Continuing with the example of Figure 5(a), if A fires we arrive in a state depicted in Figure 5(b). Following definition 3.3 the set $s\_enabled(A, \langle m, r\rangle) = \{\{A\}\}$ is removed from its input places $p0$ and $p1$ and added to its output place $p3$. Note that this firing rule is still "consistent" with the classical PN firing rule, defined in Definition 2.4. Imagine, for example, two classical PN markings by placing in Figure 5(a) a token in each place containing $\{A\}$ or $\{B\}$, respectively. This results in the two classical PN markings shown in Figure 6(a). An analogous *mapping* of the GPN marking depicted in Figure 5(b), results in the two classical PN markings shown in Figure 6(b). Note that these markings are exactly those markings that could be reached from the markings shown in Figure 6(a) by firing transition A using the classical PN firing rule. We can formally define the above *mapping* as follows.

Definition 3.4 *Let* $\langle m, r\rangle \in S_T^P$. $mapping(\langle m, r\rangle) \equiv \{m' \in 2^P | \exists v \in r : m' = \{p \in P | v \in m(p)\}\}$

Note that the marking of a safe PN can be represented by a set of places m, where $pi \in m$ indicates that there is a token in pi. As a result, the mapping function can be interpreted as to *map* between a state of a GPN and a set of states of a safe PN with the same structure. For Figure 5, this gives $mapping(\langle m, r\rangle) = \{\{p0, p1\}, \{p0, p2\}\}$ and $mapping(m', r) = \{\{p3\}, \{p0, p2\}\}$.

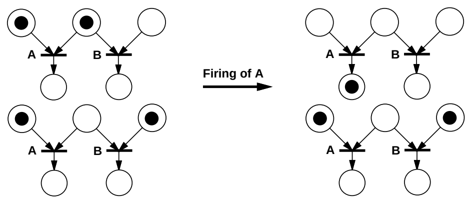

Figure 6: Classical PN markings "equivalent" to (a) Figure 5(a) (b) Figure 5(b)

Multiple Firing Semantics According to the multiple firing semantics a set of transitions - that may be conflicting - can be fired simultaneously, provided that each transition is *multiple enabled*.

Definition 3.5 (Multiple Enabling Rule) *Let* $s = \langle m, r\rangle \in S_P^T$, $t \in T$ *and* $T' \subseteq T$.
$m\_enabled(t, s) \equiv \{v \in \bigcap_{p_i \in \bullet t} m(p_i) | t \in v\}$
$m\_enabled(T', s) \equiv \forall t \in T' : m\_enabled(t, s) \neq \emptyset$

Suppose a GPN in a state depicted in Figure 7(a). The derivation of the set r_0 of valid transition sets will be described later in more detail. For now, it suffices to know that $\{A, B\}$ and $\{C, D\}$ cannot be included in r_0, because A and B, as well as C and D are conflicting transitions. We say a transition t is *multiple enabled* in a state $\langle m, r\rangle$ if $m\_enabled(t, \langle m, r\rangle) \neq \emptyset$. Transitions A and

B are then multiple enabled in the depicted state $\langle m_0^G, r_0\rangle$ as $m\_enabled(A, m_0^G, r_0) = \{\{A,C\}, \{A,D\}\} \neq \emptyset$, and $m\_enabled(B, \langle m_0^G, r_0\rangle) = \{\{B,C\}, \{B,D\}\} \neq \emptyset$. Note that when a transition is multiple enabled, it is also single enabled (the reverse, however, is not always true). Thus, in the state $\langle m_0^G, r_0\rangle$ both A and B are multiple enabled, and can be fired simultaneously, according to the *multiple firing rule*.

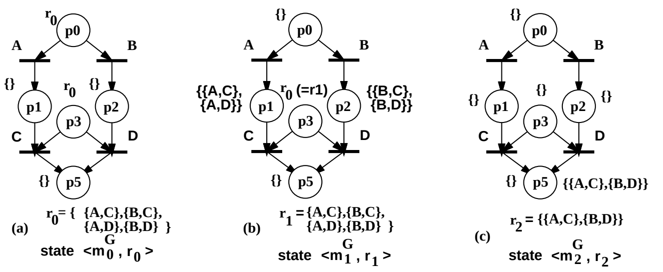

Figure 7: Subsequent markings of a GPN with maximal conflicting sets $\{A,B\}$ and $\{C,D\}$.(a) initial state $\langle m_0^G, r_0\rangle$ (b) $\langle m_1^G, r_1\rangle = m\_update(\langle m_0^G, R_0\rangle, \{A,B\})$ (c) $\langle m_2^G, r_2\rangle = m\_update(\langle m_1^G, R_1\rangle, \{C,D\})$

Definition 3.6 (Multiple Firing Rule) *Let* $s = \langle m, r\rangle \in S_P^T$*, and* $T' \in mcs(T)$ *such that* $m\_enabled(T', s) = true$*.*
$m\_update(\langle m, r\rangle, T') \equiv \langle m', r'\rangle$
where $m'(p) =$

$$\begin{cases} (m(p) \setminus \bigcup_{t\in T'\cap p\bullet} m\_enabled(t,s)) \cap r' & p \in \bullet T' \setminus T'\bullet \\ (m(p) \cup \bigcup_{t\in T'\cap \bullet p} m\_enabled(t,s)) \cap r' & p \in T'\bullet \setminus \bullet T' \\ (m(p) \setminus \bigcup_{t\in T'\cap p\bullet} m\_enabled(t,s) \\ \quad \cup \bigcup_{t\in T'\cap \bullet p} m\_enabled(t,s)) \cap r' & p \in \bullet T' \cap T'\bullet \\ (m(p) \cap r') & \textit{otherwise} \end{cases}$$

$$r' = \bigcup_{t\in(T\setminus T')} s\_enabled(t,s) \cup \bigcup_{t\in T'} m\_enabled(t,s)$$

Continuing with the example of Figure 7(a), the simultaneous firing of transitions A and B moves $m\_enabled(A, \langle m_0^G, r_0\rangle)$ (= $\{\{A,C\}, \{A,D\}\}$) and $m\_enabled(B, \langle m_0^G, r_0\rangle)$ (= $\{B,C\}, \{B,D\}\}$) from their common input place $p0$, to their output places $p1$ and $p2$, respectively. As such, we arrive in the state $\langle m_1^G, r_1\rangle$ depicted in Figure 7(b). The updating of the set of valid sets has no effect in this case, as $r_1 = r_0$. As with the single firing semantics, one can also observe the consistency with classical PN dynamics. Indeed, as $mapping(m_0^G, r_0) = \{\{p0, p3\}\}$ marking m_0^G can be mapped to the marking of a classical (safe) PN where places $p0$ and $p3$ contain a token. The marking m_1^G, however, can be mapped to 2 classical PN markings $mapping(m_1^G, r_1) = \{\{p1, p3\}, \{p2, p3\}\}$. Note that the latter markings can be reached from $\{\{p0, p4\}\}$ by separately firing A and B using the classical PN firing rule.

The same observation can be made when going to the next state; transitions C and D are both multiple enabled in state $\langle m_1^G, r_1\rangle$ and can be fired simultaneously to go to the next-state $\langle m_2^G,\ r_2\rangle$, depicted in Figure 7(c). Due to the extra conditioning of r_1 by s_update, the new set of valid sets now becomes $r_2 = \{\{A,C\}, \{B,D\}\}$. As a result, $mapping(m_2^G, r_2) = \{\{p5, p3\}\}$, and the new state $\langle m_2^G, r_2\rangle$ can be mapped to the marking of a classical (safe) PN where only places $p5$ contains a token; i.e. the marking that can be reached from $mapping(m_1^G, r_1) = \{\{p1, p3\}, \{p2, p3\}\}$ by firing D and E using the classical PN firing rule. The extra conditioning of the set of valid sets rules out $\{A,D\}$ and $\{B,C\}$, in fact modeling an "extended conflict" relation between A and D, and between B and D, respectively. Indeed, if $A(B)$ *precedes* $C(D)$ and C *conflicts* with D, then $A(B)$ conflicts with $D(C)$ and $\{A,D\}(\{B,C\})$ cannot be included in any valid set.

3.3 Analysis Procedure

Suppose now a safe classical PN $N = \langle P, T, F, m_0\rangle$ and a Generalized Petri net with the same structure $N^G = \langle P, T, F, m_0^G, r_0\rangle$ with

$$m_0^G(p) = \begin{cases} r_0 & \text{if } p \in m_0 \\ \{\} & \text{otherwise} \end{cases}$$
$$r_0 = \{v \in 2^T | \forall t, u \in T : conflict(t,u) \Rightarrow \{t,u\} \not\subseteq v\}$$

then it is clear that $mapping(m_0^G, r_0) = \{m_0\}$ and no two conflicting transitions can be part of any valid set. One can then proceed with the reachability analysis as follows. In state $\langle m_0^G, r_0\rangle$ we search for candidate Maximal Conflicting Sets (MCS's). A candidate MCS T' must be multiple enabled, and firing T' will not disable any other MCS that was already multiple enabled, as well as any other transition $t \notin T'$ that was single enabled. As a result, the same reduction techniques of the partial-order methods can be applied: selecting only one interleaving sequence, or alternatively, firing all candidate MCS's at the same time and postponing the possible firing of the other transitions to a future state. Moreover, one is guaranteed that the reached state can be mapped to a set of states of the "original" reachability graph $RG(N)$.

If in state $\langle m_0^G, r_0\rangle$ no candidate MCS's can be found, one has to fall back on the single firing semantics and again apply partial-order reductions, if possible. Similarly, with the single firing rule, one is guaranteed to be "in track" with classical PN dynamics.

The same reasoning applies to all next reachable states. By induction, one can then conclude that the following algorithm[2] computes enough of the reachable states of a

[2]As to not clutter the algorithm, we left out the code that checks a.o. that the firing of an enabled transition is not postponed “forever”.

GPN to decide about the behavior of a safe PN with the same structure.

A property that is often checked for is the freedom of deadlock. In this framework, the definition of a deadlock is characterized by : $deadlock(\langle m^G, r\rangle) \Leftrightarrow$

$$\exists m \in mapping(m^G, r) : \forall t \in T : \neg enabled(t, m)$$

Effectively, this boils down to checking upon

$$\sum_{t \in T} s\_enabled(t, \langle m, r\rangle) \neq r$$

Algorithm Generalized Partial-Order Reachability Analysis

```
global: RG /* reachability graph */

reachability-graph (N, m_0)
in: a tuple N = ⟨ P, T, F ⟩, a marking m_0 such that ⟨ P, T, F, m_0 ⟩ is a safe
Petri net
begin
let $m_0^G(p) = \begin{cases} r_0 & \text{if } p \in m_0 \\ \{\} & \text{otherwise} \end{cases}$
let $r_0 = \{v \in 2^T | \forall t_1, t_2 \in T : conflict(t_1, t_2) \Rightarrow \{t_1, t_2\} \not\subseteq v\}$
let $s_0 = \langle m_0^G, r_0 \rangle$
RG = [ { $s_0$ } , ∅, $s_0$ ]
reach (N, $s_0$)
end

reach (N, s)
in: a tuple N = ⟨ P, T, F ⟩, a state s = ⟨ m, r ⟩
begin
let single-enabled-trans = { t ∈ T | s_enabled(t, s) }
Search for candidate-mcs
if $\bigcup_{t \in T}$ s_enabled(t,s) ≠ r
then report deadlock possibility at state s
else if candidate-mcs ≠ ∅
    then multiple-execute (N, s, candidate-mcs)
    else if ∃ T' ⊆ single-enabled-trans: T' ∈ mcs(T))
        then forall t ∈ T' do
                single-execute(N, s, t)
            od
        else forall t ∈ single-enabled-trans do
                single-execute (N, s, {t})
                od
        fi
    fi
fi
end

multiple-execute (N, s, mcs)
in: a tuple N =⟨ P, T, F ⟩, a state s = ⟨ m, r ⟩, a set mcs of
maximal conflicting sets that are multiple enabled in state s
begin
let T' = $\bigcup_{T_i \in mcs} T_i$
let s' = m_update(s, T')
add (s, T, s') to RG
if s' ∉ RG
then reach (N, s')
fi
end

single-execute (G, s, t)
in: a net N =⟨ P, T, F, ⟩, a state s = ⟨ m, r ⟩, a transition t that is single
enabled in state s
begin
let s' = s_update (s, t)
add (s, t, s') to RG
if s' ∉ RG
then reach (N, s')
fi
end
```

4 Implementation and Results

The techniques presented in this paper have been implemented in a tool, called JULIE, in about 9000 lines of C code. To assess the viability of our approach, we have carried out a number of experiments that suffer from the state explosion problem. The analyzed examples were the non-serialized version of the well-known Dining Philosophers Problem (NSDP) [6], the Asynchronous Arbiter Tree (ASAT) [1], the Overtake Protocol (OVER) [4], and Readers and Writers (RW) [4].

To compare with state-of-the-art verification tools, we have chosen SPIN extended with the Partial-Order Package (SPIN+PO) [8], and the Symbolic Model Verifier (SMV)(Release 2.4.4) [10], as representatives of the partial-order methods and the symbolic techniques, respectively. We used both packages, as well as our prototype Generalized Partial Order Analysis tool, to test the examples for deadlock situations. However, obtained results are also valid for safety checks, since the verification of a safety property can always be reduced to a check for deadlock [9]. A numeric overview of the results is shown in Table 1.

In Table 1 the number of states of the complete reachability graph, the number of states of the reachability graphs derived using (generalized) partial-order analysis, as well the peak BDD sizes encountered during symbolic reachability analysis, are listed for various instances of the parameterized examples. CPU times, measured on a HP K260 with 896 MB RAM, are also included.

For NSDP, ASAT and OVER, generalized partial-order analysis outperforms both SPIN+PO and SMV. A drastic improvement is observed for NSDP and ASAT. For NSDP 3 states are sufficient to detect all deadlock situations, independent of the number of philosophers. As can be observed, CPU times increase linearly with problem size.

For RW, generalized partial-order analysis performs better than SPIN+PO, but slightly worse than SMV. This is because RW exhibits a lot of conditional behavior where non of the classical partial-order reduction techniques can be applied – this is also visible in the reduced state space which equals the complete state space. For this problem, OBDD's apparently provide an efficient encoding of the state space.

Table 1: Results of Generalized Partial Order Analysis (GPO)

Problem(size)	States	SPIN+PO		SMV		GPO	
		States	Time(s)	Peak BDD-size	Time (s)	States	Time(s)
NSDP(2)	18	12	0.08	1068	0.04	3	0.01
NSDP(4)	322	110	0.13	10018	0.22	3	0.03
NSDP(6)	5778	1422	1.07	52320	8.97	3	0.04
NSDP(8)	103682	19270	25.62	687263	1169.30	3	0.05
NSDP(10)	$1.86 * 10^6$	239308	453.16	$>$ 24 hours		3	0.06
ASAT(2)	88	33	0.08	1587	0.05	8	0.01
ASAT(4)	7822	192	0.11	117667	79.61	14	0.06
ASAT(8)	$1.58 * 10^6$	3598	1.12	$>$ 24 hours		23	0.35
OVER(2)	65	28	0.09	3511	0.08	6	0.01
OVER(3)	519	107	0.13	10203	0.19	7	0.02
OVER(4)	4175	467	0.44	11759	0.64	8	0.04
OVER(5)	33460	2059	2.05	24860	3.59	9	0.06
RW(6)	72	72	0.06	3689	0.09	2	0.05
RW(9)	523	523	1.51	9886	0.16	2	0.20
RW(12)	4110	4110	16.89	10037	0.28	2	0.61
RW(15)	29642	29642	194.33	10267	0.43	2	1.50

5 Conclusions

In this paper we presented a novel *generalized partial-order analysis* technique that can enumerate conflicting paths simultaneously, thus extending the partial-order techniques [6, 9, 14] to tackle also the explosion problem due to concurrently marked conflict places. The approach is based on a *Generalized Petri Net* model that has a modified marking representation to distinguish the different conflicting paths.

The experimental results are very promising. As illustrated in Section 4, for certain examples one can demonstrate exponential reduction in complexity. The method has been applied to some real-life applications in embedded system design (e.g. a Quadrature Amplitude Modulation modem), where it also provided significant gain [16].

Recently, a number of interesting methods [7, 13] have been proposed for the efficient timing verification of concurrent systems, modeled as *Timed Petri nets*. We are currently investigating how these methods can be leveraged to our work.

References

[1] R. Alur, R.K. Brayton, T.A. Henzinger, and S. Qadeer amd S.K. Rajamani. Partial-order Reduction in Symbolic State Space Exploration. In *Proceedings of the International Conference on Computer Aided Verification*, 1997.

[2] R. R. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential Circuit Verification Using Symbolic Model Checking. In *Proceeding of the 27th ACM/IEEE Design Automation Conference*, Orlando, June 1990.

[4] J.C. Corbett. Evaluating Deadlock Detection Methods for Concurrent Software. *IEEE Transactions on Software Engineering*, 22(3):161–180, March 1996.

[5] O. Coudert, J. C. Madre, and C. Berthet. Verifying Temporal Properties of Sequential Machines Without Building their State Diagrams. In *Workshop on Computer-Aided Verification*, Rutgers, June 1990.

[6] G.G. de Jong. *Generalized Data Flow Graphs Theory and Applications*. Ph.D. Thesis, Technical University of Eindhoven, October 1993.

[7] G. de Jong E.Verlind and B. Lin. Efficient Partial Enumeration for Timing Analysis of Asynchronous Systems. In *Proceedings of the 33rd Design Automation Conference*, pages 55–58, June 1996.

[8] P. Godefroid and D. Pirottin. Refining Dependencies Improves Partial-Order Verification Methods. In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification*, pages 438–449, Elounda, Greece, 1993.

[9] P. Godefroid and P. Wolper. Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties. *Lecture Notes in Computer Science*, 575(10):332–342, Aalborg, Denmark, July 1-4 1991.

[10] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, 1993.

[11] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia. Petri Net Analysis using Boolean Manipulation. *Lecture Notes in Computer Science 815, Springer Verslag*, pages 416–435, June 1994.

[12] J.L. Peterson. *Petri Net Theory and Modelling of Systems*. Prentice Hall, 1981.

[13] A. Semenov and A. Yakovlev. Verification of Asynchronous Circuits using Time Petri Net Unfolding. In *Proceedings of the 33rd Design Automation Conference*, pages 55–58, June 1996.

[14] A. Valmari. A Stubborn Attack on State Explosion. In 2^{nd} *Workshop on Computer Aided Verification*, pages 156–165, New Brunswick, NJ, June 18-21 1990.

[15] J. Vautherin. Parallel Systems Specifications with Colored Petri Nets and Algebraic Specifications. *Advances in Petri Nets 1987, Lecture Notes in Computer Science 266*, pages 293–308, 1987.

[16] S. Vercauteren, D. Verkest, G. de Jong, and Bill Lin. Derivation of Formal Representations from Process-based Specification and Implementation Models. In *The Proceedings of the International Symposium on System Synthesis*, pages 16–23, September 1997.