Faster Verification of Partially Ordered Runs in Petri Nets Using Compact Tokenflows

Robin Bergenthum

Department of Software Engineering and Theory of Programming,
FernUniversität in Hagen
robin.bergenthum@fernuni-hagen.de

Abstract. In this paper we tackle the problem of verifying whether a labeled partial order (LPO) is executable in a Petri net. In contrast to sequentially ordered runs an LPO includes both, information about dependencies and independencies of events. Consequently an LPO allows a precise and intuitive specification of the behavior of a concurrent or distributed system. In this paper we consider Petri nets with arc weights, namely marked place/transition-nets (p/t-nets). Accordingly the question is whether a given LPO is an execution of a given p/t-net.

Different approaches exist to define the partial language (i.e. the set of executions) of a p/t-net. Each definition yields a different verification algorithm, but in terms of runtime all these algorithms perform quite poorly for most examples. In this paper a new compact characterization of the partial language of a p/t-net will be introduced, optimized with respect to the verification problem. The goal is to develop an algorithm to efficiently decide the verification problem.

1 Introduction

Specifications of concurrent or distributed systems are often formulated in terms of scenarios [31, 10, 35, 20]. A scenario can be represented by a labeled partial order (LPO), i.e. a partially ordered set of events. In many cases it is part of a system's specification that given LPOs should or should not be executable by the system. P/t-nets [30, 32], i.e. Petri nets with arc weights, are well suited for modelling concurrent or distributed systems and have a huge range of theoretical and practical applications [29, 2, 13, 11]. Thus, it is a natural question whether an LPO is an executions of a p/t-net. We refer to this question as the verification problem. Deciding the verification problem can help to check conformance of a system, uncover system faults or requirements, validate the system and evaluate design alternatives.

The set of all executable LPOs of a p/t-net is called its partial language. To define this language there exist several equivalent characterisations [18, 17, 19, 21, 28]. Each characterisation yields a different algorithm deciding the verification problem. The characterisations of the partial language are as follows:

– The set of runs. A run is an LPO which includes the transitive order relation between events of an occurrence net of the given p/t-net [18, 17].
– The set of executions. An LPO is an execution if each cut (i.e. a maximal set of unordered events) of the LPO is enabled after the occurrence of its prefix in the given p/t-net [19].

– The set of LPOs for which valid tokenflows describing the flow of tokens between events exist [21].
– The set of LPOs for which valid interlaced tokenflows describing a quadruple of flows of tokens between events on the skeleton arcs of the LPO exist [28].

The set of runs coincides with the set of executions of a p/t-net [23, 34]. The set of runs can be described using the definition of valid tokenflows [21]. A tokenflow is a function assigning non-negative integers to the set of arcs of an LPO. Each tokenflow describes a distribution of tokens with respect to a place of the p/t-net. Given two events a and b and a fixed place p, the value of a tokenflow function x on the arc (a, b) describes that the occurrence of a produces $x(a, b)$ tokens in place p that will be consumed by the occurrence of b. A tokenflow is valid if it respects the firing rule of the p/t-net in a sense that each event receives enough tokens to occur and no event has to pass more tokens then its occurrence produces. An LPO is an execution if there is a valid tokenflow for each place of a p/t-net. The notion of interlaced tokenflows and its equivalence to normal tokenflows is presented in [28].

The different characterisations of the partial language of the p/t-net lead to different algorithms solving the verification problem. Unfortunately, each known verification algorithm has big drawbacks dependent on the structure of the given LPO or the given p/t-net. The main ideas of the corresponding verification algorithms are as follows:

Algorithms using the notion of runs decide the verification problem by considering the set of occurrence nets of the given p/t-net. They check if the given LPO includes the partial order of events for one of the occurrence nets. The set of all occurrence nets can be calculated using so called unfolding algorithms [27, 15, 8]. Even if the number of occurrence nets is infinite these algorithms are able to calculate a set of occurrence nets such that the number of events of each net is equal to the number of events of the given LPO. The main problem is that the number of occurrence nets may be exponential in the number of events such that the runtime complexity of such an algorithm is in exponential time.

Algorithms using the notion of executions decide the verification problem by checking if each cut of the given LPO is enabled after the occurrence of its prefix. This time the number of cuts can be exponential in the number of events such that the runtime complexity of such an algorithm is in exponential time. However, the number of cuts is small if the given LPO is dense. In such a case the LPO describes a lot of dependencies between events and the verification algorithm is applicable. Given a thin LPO or an LPO describing reasonable concurrency between events this verification algorithm is insufficient.

Algorithms using the notion of valid tokenflows are by now the most elegant way to decide the verification problem. As stated above a tokenflow is a function describing the distribution of tokens between events with respect to a place. This distribution respects the transitive ordering of event in the LPO, such that a valid tokenflow can be constructed if and only if the LPO is a run of the p/t-net. The first implementation of a verification algorithm using the notion of valid tokenflows is given in [3, 4]. Each construction of a valid tokenflow is done by constructing maximal flows [16, 1] in so called associated flow networks. The runtime of this algorithm is in polynomial time. The main disadvantage of this algorithms is that its runtime highly depends on the size of the order relation of the LPO. If the LPO describes a lot of dependencies the verification algorithm is hardly applicable. A second approach to solve the verification problem using valid tokenflows is given in [24]. This approach is able to reduce the runtime by reducing the number of maximal flow problems to be solved. Still, its runtime is poor if the LPO is dense.

The most recent characterisation of the partial language of a given p/t-net, so called valid interlaced tokenflows, was given in [28]. An interlaced tokenflow is a quadruple of special tokenflows defined on every skeleton arc of the given LPO. The interplay between the components of a valid interlaced tokenflow ensures that they correspond to a valid tokenflow defined on every arc of the given LPO. The main advantage is that the number of skeleton arcs is the smallest representation of the number of all dependencies specified by the LPO. By now it is not known if the resulting verification algorithm is faster than the algorithm described in [24] since both algorithms are not yet implemented.

In this paper we introduce compact tokenflows as a new characterisation of the partial language of a p/t-net. Compact tokenflows are optimized to efficiently solve the verification problem. The main ideas leading to the concept of compact tokenflows are as follows:

– Compact tokenflows are based on normal tokenflows. A Tokenflow is defined on the arcs of an LPO and the number of arcs does not increase exponentially with the number of events. This is a necessary condition to receive an algorithm having polynomial worst case runtime complexity.
– LPOs include the complete transitive relation between events. In contrast to tokenflows a compact tokenflow is defined on the skeleton of an LPO. The skeleton is the smallest representation of the transitive relation.
– In contrast to tokenflows a compact tokenflow abstracts from the history of tokens. Since in a compact tokenflow events are able to pass received tokens to later events, a compact tokenflow describes the sum of tokens produced by sets of events and not the number of tokens produced by each event.
– An interlaced tokenflow respects the history of each tokens by considering a four component flow of tokens. Using compact tokenflows only a single tokenflow is needed.

We will show that compact tokenflows can be constructed adopting the ideas presented in [24]. We will be able to introduce a new verification algorithm having an efficient runtime independent from the structure of the given LPO and p/t-net. This algorithm will be much faster than any known verification algorithm.

The paper is organized as follows. In Section 2 we describe LPOs, p/t-nets and their partial languages. In Section 3 we recapitulate the concepts of tokenflows and introduce the new concept of compact tokenflows. We prove that compact tokenflows are equivalent to normal tokenflows. In section 4 we describe the resulting verification algorithm. In Section 5 we present runtime experiments of the new verification algorithm comparing it to all alternative verification algorithms. Finally, Section 6 concludes the paper.

2 Labeled Partial Orders, p/t-Nets and Partial Languages

In this paper the following notations will be used. $\mathbb{N}$ denotes the non-negative integers. Given a finite set A, 2^A denotes the power set of A. $\mathbb{N}^A$ denotes the set of multisets over A. For $m \in \mathbb{N}^A$ we write $m = \sum_{a \in A} m(a) \cdot a$.

Definition 1. *A labeled partial order (LPO) is a triple $lpo = (V, <, l)$, where V is a finite number of events, $< \subseteq V \times V$ is a transitive and irreflexive relation over V and $l : V \to T$ is a labeling function assigning a label $t \in T$ to each event.*

Given an LPO $lpo = (V, <, l)$ and an event $e \in V$, the preset of e is denoted by $\bullet e = \{v \in V | v < e\}$. Given a set of events E, the preset of E is denote by $\bullet E = \bigcup_{v \in E} \bullet v$. A prefix is a set of events E such that $E = (E \cup \bullet E)$ holds. The maximal set of events having an empty preset is denoted by $min(V)$. A set of events C is called a co-set if $(v, v' \in C \Rightarrow v \not< v')$ holds. A cut is a co-set which is not included in any other co-set.

The transitive closure of a relation $\to$ is denoted by $\to^*$, the transitive reduction of a relation $\to$ is denoted by $\to^\diamond$. Given a finite relation the transitive reduction is the smallest relation such that its transitive closure equals the primary LPO. Given an LPO $lpo = (V, <, l)$ the transitive reduction $<^\diamond$ is called the skeleton of lpo. The graph $(V, <^\diamond)$ forms a Hasse diagram and we call the labeled Hasse diagram $(V, <^\diamond, l)$ the compact LPO of lpo.

In this paper concurrent or distributed system will be given by p/t-nets, i.e. Petri nets with arc weights.

Definition 2. *A marked place/transition-net (p/t-net) is a tuple $N = (P, T, W, m_0)$, where P and T are finite sets of places and transitions fulfilling $P \cap T = \emptyset$, $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a multiset of edges and $m_0 : P \to \mathbb{N}$ is an initial marking.*

Given a p/t-net $N = (P, T, W, m_0)$, a transition $t \in T$ is enabled in N iff $\forall\ p \in P : W(p, t) \leq m_0(p)$ holds. If a transition is enabled it may fire and change the given marking m_0 to a new marking m which is for each $p \in P$ given by $m(p) = m_0(p) + W(t, p) - W(p, t)$. A multiset of transitions $\tau \in \mathbb{N}^T$ (called a step of N) is enable in N iff $\forall\ p \in P : \sum_{t \in \tau} \tau(t) \cdot W(p, t) \leq m_0(p)$ holds. If a step is enabled it may fire and change the given marking m_0 to a new marking m which is for each $p \in P$ given by $m(p) = m_0(p) + \sum_{t \in \tau} \tau(t) \cdot (W(t, p) - W(p, t))$.

Given a p/t-net a sequential run is a sequence of consecutively enabled and fired transitions or transition steps. Instead of the set of sequential runs we consider the partial language of a p/t-net. In contrast to sequentially ordered runs an LPO includes arbitrary dependencies and independencies between events.

Definition 3. *Given a p/t-net $N = (P, T, W, m_0)$ and an LPO $lpo = (V, <, l)$ with $l(V) \subseteq T$. The lpo is an execution of N iff for each $p \in P$ and each cut C of lpo : $m_0(p) + \sum_{e \in \bullet C}(W(l(e), p) - W(p, l(e))) \geq \sum_{e \in C} W(p, l(e))$ holds. We denote the set of all executions of N by $L(N)$, the so called partial language of N.*

Following Definition 3, we say that an LPO is executable with respect to a place p, if $m_0(p) + \sum_{e \in \bullet C}(W(l(e), p) - W(p, l(e))) \geq \sum_{e \in C} W(p, l(e))$ holds.

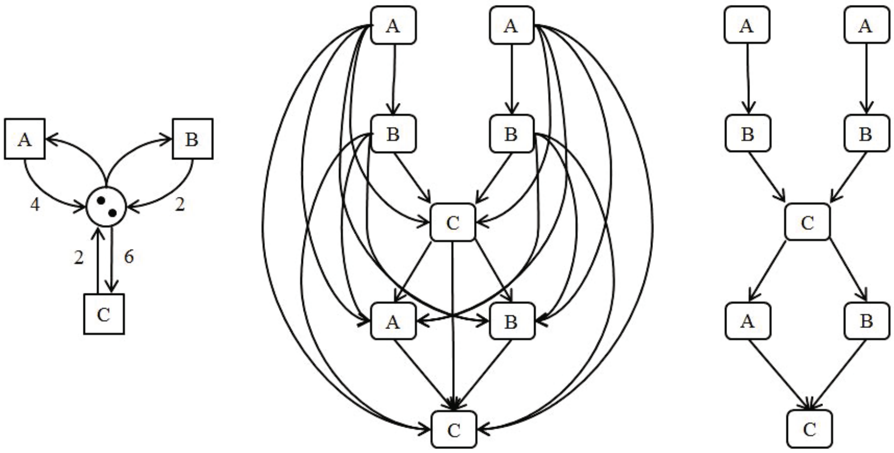

Fig. 1. A p/t-net, an executable LPO and its compact LPO

As stated in the introduction there are other equivalent characterizations of the partial language of a p/t-net. Any characterization leads to the same verification problem: Given a p/t-net N and an LPO lpo, decide if $lpo \in L(N)$ holds.

Figure 1 shows a p/t-net having only one place which is marked by two tokens in the initial marking, three transitions A, B, C as well as several weighted arcs. Arcs without attached numbers have the arc weight one. The LPO in the middle consists of eight events labeled with the transitions of the p/t-net. This LPO is an execution of the p/t-net according to Definition 3. The right side of Figure 1 depicts the corresponding compact LPO.

3 Tokenflows

The definition of valid tokenflows (see [21]) was originally invented to more easily prove the equivalence between runs and executions of a p/t-net. The original proof is quite complex and was introduced in [23, 34]. In [21] additionally an algorithm deciding the verification problem in polynomial time was obtained which was first implemented in [4] yielding a fast verification algorithm in case of a thin LPOs. [24] describes a second algorithm having $|V|$-times faster runtime (V denotes the number of events of the given LPO). This was the first step in the direction of an efficient verification algorithm, but still this algorithm performs poorly in case of a dense LPOs. After this tokenflows have been used for the synthesis of p/t-nets from LPOs [5, 6], synthesis from infinite sets of LPOs [7, 33, 28], for the definition of the partial language of more general classes of Petri nets [25], as well as for the unfolding of the partial language of p/t-nets [9, 8]. All in all the verification problem took a back seat. The idea of this paper is to find a characterisation of the partial language based on tokenflows which is optimized for the verification problem.

First, we will recall the concept of tokenflows invented in [21]. Given an LPO $lpo = (V, <, l)$ and a p/t-net $N = (P, T, W, m_0)$ such that $T \subseteq l(V)$ holds, a tokenflow x

is a function assigning a non-negative integer to each event and each arc of the LPO. Given an arc (v, v') of *lpo* the value $x(v, v')$ describes the number of tokens produced by the occurrence of v which are consumed by the occurrence of v'. Given an event v, the value $x(v)$ describes the number of tokens consumed by v from the initial marking. A tokenflow is valid with respect to a place $p \in P$, if it respects the occurrence rule for each event v of *lpo* in a sense that: the sum of ingoing tokens of v equals $W(p, l(v))$, the sum of outgoing tokens of v is less than $W(l(v), p)$ and the sum of tokens consumed from the initial marking by all events does not exceed the initial marking $m_0(p)$. Since such a valid distribution of tokens along the arcs of an LPO respects the described dependencies, a valid tokenflow coincides with a run of a p/t-nets considering p. If there exists a valid tokenflow for each place p of N a corresponding occurrence net of N can be found and vice versa.

Definition 4. *Given a p/t-net $N = (P, T, W, m_0)$ and an LPO $lpo = (V, <, l)$, such that $l(V) \subseteq T$ holds. A tokenflow is a function $x : (< \cup V) \to \mathbb{N}$. Given an event $v \in V$, by $in(v) = x(v) + \sum_{v' < v} x(v', v)$ we denote the inflow of v and by $out(v) = \sum_{v < v'} x(v, v')$ we denote the outflow of v.*

Given a place $p \in P$ a tokenflow is valid with respect to p iff the following conditions hold.

(I) $\forall v \in V : in(v) = W(p, l(v))$,
(II) $\forall v \in V : out(v) \leq W(l(v), p)$,
(III) $\sum_{v \in E} x(v) \leq m_0(p)$.

lpo is called a valid LPO of N iff for every $p \in P$ there is a valid tokenflow.

The main result of [21] is that the partial language of a p/t-net can be characterised using valid tokenflows.

Theorem 1. ([21]) *Given a p/t-net N. The partial language $L(N)$ coincides with the set of valid LPOs of N.*

Theorem 1 leads to the verification algorithms given in [4] and [24]. Given a fixed place both algorithms construct a valid tokenflow by constructing maximal flows in associated flow networks derived from the LPO and the p/t-net. The big disadvantage of both algorithms is that the size of the corresponding flow networks is directly related to the number of arcs of the LPO. Although the runtime complexity of both algorithms is in polynomial time, both algorithms perform worse than an algorithm checking if each cut of an LPO is enabled after the occurrence of its prefix if the given LPO is dense.

In the following we want to introduce a new characterisation of the partial language of a p/t-net yielding an efficient verification algorithm. A compact tokenflow is defined on the skeleton of an LPO, i.e. its corresponding compact LPO. A (normal) tokenflow describes the complete distribution of tokens between events using the given transitive order relation of the LPO. In a compact tokenflow each event v can receive more tokens than its transition $l(v)$ consumes and v is allowed to pass those additional tokens on to later events. The main idea is that losing the set of transitive arcs is fine since every transitive arc can be represented by a path of skeleton arcs.

Definition 5. *Given a p/t-net $N = (P, T, W, m_0)$, an LPO $lpo = (V, <, l)$ such that $l(V) \subseteq T$ holds and the skeleton $\lhd$ of lpo. A compact tokenflow is a function $x : (\lhd \cup V) \to \mathbb{N}$. Given an event $v \in V$, by $in^{\lhd}(v) = x(v) + \sum_{v' \lhd v} x(v', v)$ we denote the inflow of v and by $out^{\lhd}(v) = \sum_{v \lhd v'} x(v, v')$ we denote the outflow of v.*

Given a place $p \in P$ a compact tokenflow is valid with respect to p iff the following conditions hold.

(i) $\forall v \in V : in^{\lhd}(v) \geq W(p, l(v))$,
(ii) $\forall v \in V : out^{\lhd}(v) \leq in^{\lhd}(v) - W(p, l(v)) + W(l(v), p)$,
(iii) $\sum_{v \in E} x(v) \leq m_0(p)$.

lpo is called a valid compact LPO of N iff for every $p \in P$ there is a valid compact tokenflow.

Please note that the definition of a compact tokenflow can be generalised to using not only the skeleton arcs of an LPO, but using any subset of the partial order relation including the skeleton. Only the skeleton arcs are needed, but if an LPO is given by some acyclic graph all definitions and algorithms shown in this paper may be directly applied to this graph instead of its transitive reduction.

The left side of Figure 2 shows a valid tokenflow for the place and the p/t-net shown in Figure 1. Transitive arcs are not depicted if their value is equal to zero. The right side shows a valid compact tokenflow with respect to the same place.

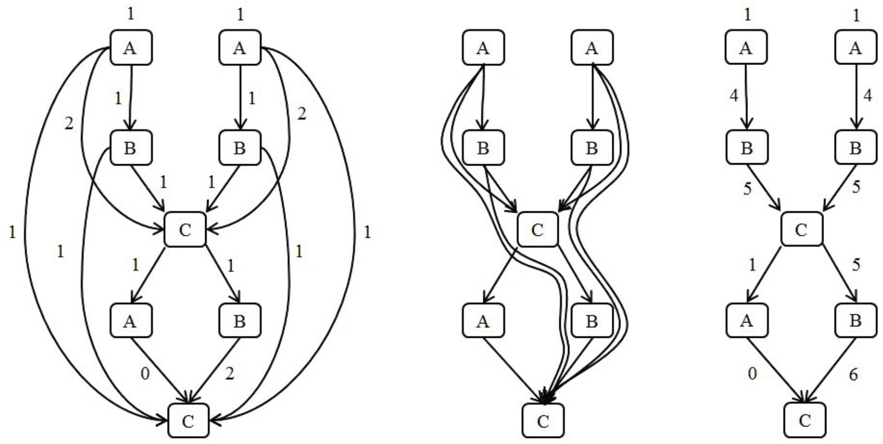

Fig. 2. A valid tokenflow and a valid compact tokenflow of the p/t-net shown in Figure 1

In the next step we will prove that the set of valid compact LPOs of a p/t-net N coincides with its partial language $L(N)$. Since the compact representation abstracts from the concrete distribution of tokens, in the sense that the history of a token is lost while it is passed on by events, this proof cannot be done similarly to the proof concerning valid LPOs given in [21]. We will show that for every valid tokenflow in an LPO there is a valid compact tokenflow. Given a place p and a valid tokenflow we will fold this tokenflow into a compact tokenflow. We will show that every possible folding yields a valid compact tokenflow. After that, given an LPO and a valid compact tokenflow for every place of N, we will proof that this LPO is an execution of N (in the sense of Definition 3).

Definition 6. *Given an LPO $lpo = (V, <, l)$, the skeleton $\lhd$ of lpo and $x : (< \cup V) \to \mathbb{N}$ a tokenflow. For each arc $(v, v') \in <$ there exists a path of skeleton arcs from v to v'. Consequently a function $\rho : < \to 2^{\lhd}$ can be found which maps every arc (v, v') of lpo to a path from v to v' in $\lhd$. Given a fixed function ρ for every skeleton arc (e, e') we define $(e, e')^* = \{(v, v') \in < \,|(e, e') \in \rho(v, v')\}$. Depending on ρ we are now able to define the associated compact tokenflow $x^{\lhd} : (\lhd \cup V) \to \mathbb{N}$.*

$$x^{\lhd}(a) = \begin{cases} \sum_{(v,v') \in (e,e')^*} x(v, v'), & \textit{if } a = (e, e') \in \lhd, \\ x(e), & \textit{if } a = e \in V. \end{cases}$$

The middle part of Figure 2 depicts a possible function ρ mapping every transitive arcs to a path of skeleton arcs. There may be different choices for ρ since often for some transitive arcs it is possible to choose between different paths of skeleton arcs using concurrent parts of the LPO. Given a skeleton arc, its compact tokenflow is the sum of tokenflow of all attached transitive arcs.

Theorem 2. *Given a p/t-net $N = (P, T, W, m_0)$, an LPO $lpo = (V, <, l)$, such that $l(V) \subseteq T$ holds, a place $p \in P$ and a tokenflow x which is valid with respect to p. Every possible associated compact tokenflow $x^{\lhd}$ is valid with respect to p.*

Proof. Given a fixed event $v \in V$, the image set $\rho(<)$ contains four different kinds of paths. Paths leading to v, paths beginning at v, paths including v and paths not including v. The inflow of v with respect to $x^{\lhd}$ is given by the sum of values of paths leading to v, paths including v and the value $x(v)$. By $\delta(v)$ we denote the sum of tokenflow reaching v via paths including v whereby $in^{\lhd}(v) = in(v) + \delta(v)$ holds. The outflow of v with respect to $x^{\lhd}$ is given by the sum of values of paths starting at v and paths including v whereby $out^{\lhd}(v) = out(v) + \delta(v)$ holds. The conditions (i), (ii) and (iii) of Definition 5 hold as follows:

$$\begin{aligned} in^{\lhd}(v) &= in(v) + \delta(v) \\ &\geq in(v) \\ &\overset{(I)}{=} W(p, l(v)), \end{aligned}$$

$$\begin{aligned} out^{\lhd}(v) &= out(v) + \delta(v) \\ &\overset{(I)}{=} out(v) + \delta(v) + in(v) - W(p, l(v)) \\ &= out(v) + in^{\lhd}(v) - W(p, l(v)) \\ &\overset{(II)}{\leq} W(l(v), p) + in^{\lhd}(v) - W(p, l(v)), \end{aligned}$$

$$\begin{aligned}\textstyle\sum_{v\in V} x^{\lhd}(v) &= \textstyle\sum_{v\in V} x(v)\\ &\overset{(III)}{\leq} m_0(p).\end{aligned}$$

Theorem 2 states that each valid tokenflow can be folded into a valid compact tokenflow. In a next step we need to prove the opposite direction. Since it is not clear in which way a compact tokenflow needs to be unfolded to get a valid tokenflow, we prove that if there exists a compact tokenflow which is valid with respect to a place p the LPO is an execution with respect to p.

Definition 7. *Given an LPO $lpo = (V, <, l)$, the skeleton $\lhd$ of lpo and a compact tokenflow $x^{\lhd}$. Given a set of events $E \subseteq V$, by $E^{\ll} = \lhd \cap ((V\backslash E) \times E)$ we denote the set of skeleton arcs leading to E and by $E^{\gg} = \lhd \cap (E \times (V\backslash E))$ we denote the set of skeleton arcs leaving E. By $IN(E) = \sum_{(e,e')\in E^{\ll}} x^{\lhd}(e,e') + \sum_{e\in E} x^{\lhd}(e)$ we denote the inflow of E and by $OUT(E) = \sum_{(e,e')\in E^{\gg}} x^{\lhd}(e,e') + \sum_{e\in V\backslash E} x^{\lhd}(e)$ we denote the outflow of E.*

Lemma 1. *Given a p/t-net $N = (P, T, W, m_0)$, an LPO $lpo = (V, <, l)$ with $l(V) \subseteq T$, a place $p \in P$, a compact tokenflow $x^{\lhd}$ which is valid with respect to p and a prefix $E \subseteq V$ of lpo. The following inequation holds:*

$$OUT(E) \leq m_0(p) + \textstyle\sum_{v\in E}(W(l(v), p) - W(p, l(v))).$$

Proof. We will prove this by induction. Let $E = \emptyset$ be the empty prefix.

$$\begin{aligned}OUT(\emptyset) &= \textstyle\sum_{v\in V} x^{\lhd}(v)\\ &\overset{(iii)}{\leq} m_0(p)\\ &= m_0(p) + \textstyle\sum_{v\in\emptyset}(W(l(v), p) - W(p, l(v))).\end{aligned}$$

Let E' be a prefix of lpo and let $OUT(E') \leq m_0(p) + \sum_{v\in E'}(W(l(v), p) - W(p, l(v)))$ hold. Given a event $e \in V\backslash E'$ which is minimal in $V\backslash E'$. We define a new prefix $E = (E' \cup e)$.

$$\begin{aligned}OUT(E) &= OUT(E') - in^{\lhd}(e) + out^{\lhd}(e)\\ &\overset{(ii)}{\leq} OUT(E') - in^{\lhd}(e) + in^{\lhd}(e) - W(p, l(e)) + W(l(e), p)\\ &\leq m_0(p) + \textstyle\sum_{v\in E'}(W(l(v), p) - W(p, l(v))) - W(p, l(e)) + W(l(e), p)\\ &= m_0(p) + \textstyle\sum_{v\in E}(W(l(v), p) - W(p, l(v)))\end{aligned}$$

Every prefix E of lpo can be constructed by consecutively appending events to the empty prefix (each event minimal in the set of not added events).

The next lemma states a relation between the number of tokens flowing into a cut C of events of an LPO and the number of tokens leaving the prefix given by ${}^{\bullet}C$.

Lemma 2. *Given an LPO $lpo = (V, <, l)$, the skeleton $\lhd$ of lpo and a compact tokenflow $x^{\lhd}$. For each cut C of lpo the following inequation holds: $IN(C) \leq OUT({}^{\bullet}C)$.*

Proof. Every skeleton arc of lpo leading to C starts at an event in ${}^{\bullet}C$. Since C is a cut every skeleton arc leaving ${}^{\bullet}C$ leads to C. Thereby intuitively both values should be equal. We get an inequation, because of the values of the tokenflow defined on events.

$$\begin{aligned} IN(C) &= \textstyle\sum_{(v,v')\in C^{\ll}} x^{\lhd}(v,v') + \sum_{v\in C} x^{\lhd}(v) \\ &= \textstyle\sum_{(v,v')\in {}^{\bullet}C^{\gg}} x^{\lhd}(v,v') + \sum_{v\in C} x^{\lhd}(v) \\ &\leq \textstyle\sum_{(v,v')\in {}^{\bullet}C^{\gg}} x^{\lhd}(v,v') + \sum_{v\in \{V\setminus {}^{\bullet}C\}} x^{\lhd}(v) \\ &= OUT({}^{\bullet}C). \end{aligned}$$

At this point, with the help of Lemma 1 and 2, we are able to prove the opposite direction to Theorem 2.

Theorem 3. *Given a p/t-net $N = (P, T, W, m_0)$, an LPO $lpo = (V, <, l)$, such that $l(V) \subseteq T$ holds, a place $p \in P$ and a compact tokenflow $x^{\lhd}$ valid with respect to p. lpo is executable with respect to p.*

Proof. For each cut C of lpo the following inequation hold:

$$\begin{aligned} \textstyle\sum_{v\in C} W(p, l(v)) &\overset{(i)}{\leq} \textstyle\sum_{v\in C} in^{\lhd}(v) \\ &\overset{\text{(Lemma 1)}}{\leq} OUT({}^{\bullet}C) \\ &\overset{\text{(Lemma 2)}}{\leq} m_0(p) + \textstyle\sum_{v\in {}^{\bullet}C}(W(l(v), p) - W(p, l(v))). \end{aligned}$$

We conclude this section with its main theorem. It states that the set of valid compact LPOs coincide with the partial language of a p/t-net. This theorem follows directly from Theorem 2 and Theorem 3.

Theorem 4. *Given a p/t-net N, the partial language $L(N)$ coincides with the set of valid compact LPOs of N.*

We have shown that compact tokenflows are equivalent to all other characterisations of the partial language of a p/t-net. In the next section we will present an efficient way to solve the verification problem using compact tokenflows. This algorithm will construct a valid compact tokenflow for each place of the p/t-net if such a tokenflow exists. This algorithm will only need to regard skeleton arcs, hence it is much faster then the algorithms presented in [4] and [24]. In contrast to normal tokenflows the new definition abstracts from the concrete distribution of tokens between events. Compared to interlaced tokenflows [28] only a single tokenflow is needed for every arc of the LPO.

4 Verification Algorithm

In this section we provide an algorithm solving the verification problem using compact tokenflows. Given a p/t-net $N = (P, T, W, m_0)$ and an LPO $lpo = (V, <, l)$ we will construct a flow network for every place of N such that constructing a maximal flow in each network will decide the verification problem.

A flow network (see for example [1]) is a directed graph with two special nodes. A source, the only node having no ingoing arcs, and a sink, the only node having no outgoing arcs. Each arc has a capacity. A flow is a function from the arcs to the non-negative integers assigning a value of flow to each arc. This flow function needs to respect the capacity of each arc and the so called flow conservation. The flow conservation says that the sum of flow reaching a node is equal to the sum of flow leaving a node for every inner node of the flow network. Thus, flow is only generated at the source and flows along different paths till it reaches the sink. The value of a flow function in a flow network is the sum of flow reaching the sink. The maximal flow problem is to find a flow function having a maximal value.

For a given place $p \in P$ we construct the associated flow network. Just like a tokenflow in an LPO a flow in the associated flow network describes the propagation of tokens between events. Each flow in the associated flow network coincides with a compact tokenflow. For each event in lpo we insert two nodes into the flow network. The first node is called the top-node, the second is called the bottom-node. A pair of such nodes represent an event $v \in V$. The flow at the top-node of v describes the value of tokenflow received by v and this value has to be greater than $W(p, l(v))$. Therefore, if flow arrives at the top-node of v a value of $W(p, l(v))$ is routed to the sink representing tokens consumed by the occurrence of $l(v)$. Additional flow can be distributed further flowing from the top-node of v to its bottom-node. The bottom-node of v distributes flow. The maximal number of flow this node can pass on is the number of flow received from its top-node plus $W(l(v), p)$. Therefore, flow in a value of $W(l(v), p)$ is routed from the source to the bottom-node of v representing tokens produced by the occurrence of $l(v)$. Additionally, nodes are connected according to the skeleton arcs of the lpo. For each skeleton arc (v, v') we add a corresponding arc in the associated flow network leading from the bottom-node of v to the top-node of v'. The value of a compact tokenflow on each event is considered by an additional node of the flow network. It can be seen as the bottom-node related to the initial marking. First, we add an arc from the source to this bottom-node having the capacity $m_0(p)$. Second, we add arcs from this bottom-node to all top-nodes associated to minimal events of lpo. Flow leaving this bottom-node is able to reach any top-node of the flow network via paths of skeleton arcs. This flow represents tokens consumed from the initial marking.

Figure 3 depicts the associated flow network of the LPO and the p/t-net show in Figure 1. Pairs of top- and bottom-nodes are drawn in rounded boxes labeled by the corresponding events. Inner arcs have no number attached, their capacity is equal to an upper bound for the maximal value of a possible flow functions. Such an upper bound will be defined in the following definition.

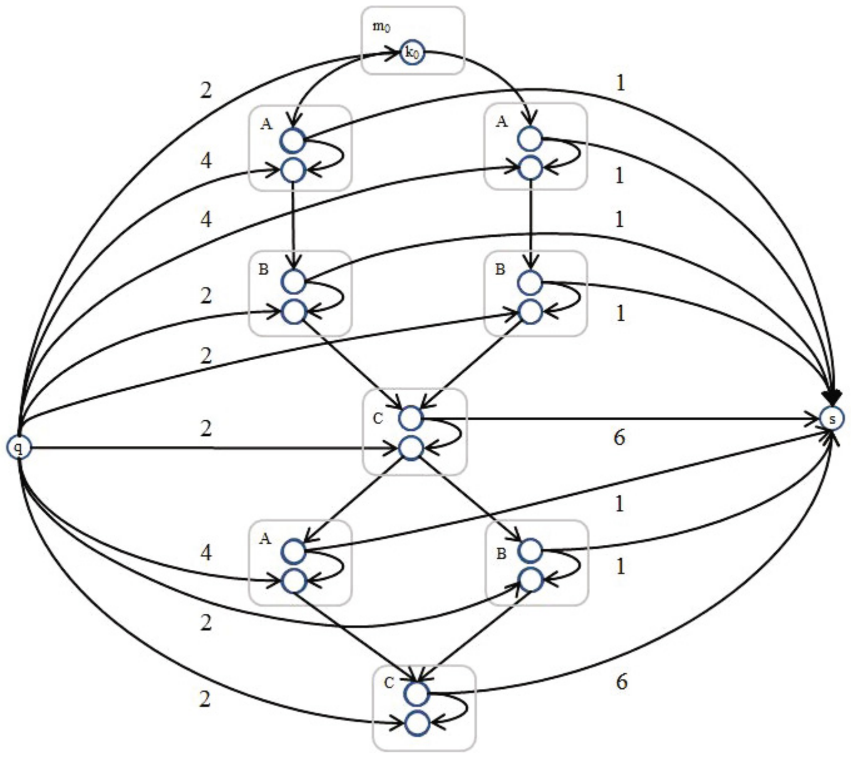

Fig. 3. The associated flow network of the p/t-net and the LPO shown in Figure 1

Definition 8. *Given a p/t-net $N = (P, T, W, m_0)$, an LPO $lpo = (V, <, l)$ with $l(V) \subseteq T$, the skeleton $\lhd$ of lpo and $p \in P$ a place. Let $M_p(lpo, N) = \sum_{v \in V} W(p, l(v))$ denote the sum of tokens consumed by all events of lpo. The associated flow network $G = (K, F, c, q, s)$ is defined by:*

$$K = \{k_0\} \cup \{t_v, b_v | v \in V\},$$

$$F \ = F_m \cup F_v \cup F_\lhd \cup F_q \cup F_0 \cup F_s \text{ with}$$

$$F_m = \{(k_0, t_v) |\, v \in min(V)\}\,,$$

$$F_v \ = \{(t_v, b_v) |\, v \in V\}\,,$$

$$F_\lhd \ = \{(b_v, t_{v'}) | (v, v') \in \lhd\}\,,$$

$$F_q \ = \{(q, b_v) | v \in V\}\,,$$

$$F_0 \ = \{(q, k_0)\},$$

$$F_s \ = \{(t_v, s) |\, v \in V\}.$$

$$c(k,k') = \begin{cases} W(l(v),p), & \textit{if } k = q, k' = b_v, \\ m_0(p), & \textit{if } (k,k') = (q,k_0), \\ W(p,l(v)), & \textit{if } k' = s, k = t_v \\ M_p(lpo,N), & \textit{otherwise.} \end{cases}$$

By construction flow on the set of arcs $F_\lhd \cup F_m$ directly corresponds to a valid tokenflow if the value of the flow function is $M_p(lpo,N)$. In this case arcs leading to s are saturated and each event satisfies condition (i). Since the outflow of events is given by $F_v \cup F_q$ each event satisfied condition (ii). The outflow of k_0 is restricted by the capacity of F_0 such that condition (iii) holds. On the other hand, if a valid compact tokenflow exists it can be translated into a maximal flow function of the associated flow network. This flow function is maximal, because it saturates all arcs leading to s and will have the value $M_p(lpo,N)$.

Constructing a maximal flow function in a flow network is the well known maximal flow problem (see for example [1]). There exist various algorithms solving the maximal flow problem in polynomial time. For the application of calculating the value of a maximal flow in an associated flow network we consider two of them, each having a good average case complexity: the algorithm of Dinic [14] and a preflow push algorithm [22, 26] using a so called gap heuristic. Given an LPO $lpo = (V,<,l)$ and a associated flow network constructed from lpo the worst case complexity of the algorithm of Dinic is in $O(|V|^2\ |<|)$, the worst case complexity of the preflow push algorithm is in $O(|V|^3)$. Both algorithms perform much better in most cases. All in all this leads to the following verification algorithm using compact tokenflows.

```
Data: LPO $lpo = (V,<,l)$, p/t-net $N = (P,T,W,m_0)$.
Result: Decides if $lpo \in L(N)$ holds.

for each $p \in P$ do
    $G \leftarrow$ associated compact flow network($p$);
    $w \leftarrow$ value of maximal Flow($G$);
    if ($w < M_p(lpo,N)$) then
        RETURN $false$;
    end
end
RETURN $true$;
```

Algorithm 1. Verification algorithm using compact tokenflows

Remark, that using a preflow push algorithm the runtime of this verification algorithm is in $O(|P| \cdot |V|^3)$.

5 Experimental Results

In this section we will discuss some experimental results. To do so we implemented several existing and new verification algorithms into the tool VipTool [12]. We did extensive runtime tests and for this paper we show the most interesting results considering a selection of the implemented verification algorithms. For the new tokenflow algorithms we show results using both mentioned maximal flow algorithms, the algorithm of Dinic and the preflow push algorithm. We selected the algorithms as follows:

- Algorithm I checks if the LPO is an execution by considering all cuts,
- Algorithm II constructs a tokenflow as implemented in [4],
- Algorithm III constructs a tokenflow as described in [24] using the algorithm of Dinic,
- Algorithm IV constructs a tokenflow as described in [24] using the preflow push algorithm,
- Algorithm V constructs a compact tokenflow using the algorithm of Dinic,
- Algorithm VI constructs a compact tokenflow using the preflow push algorithm.

The following two experiments were performed using a Dual Core Prozessor, 1.7 GHz and 4GB RAM. In both experiments we consider the p/t-net shown in figure 4. It models a simple workflow performed by a group of students working within a collaborative learning environment. The p/t-nets structure is exemplary for simple workflows and we omit a description of this special workflow in this paper.

Figure 5 depicts an LPO of the partial language of the p/t-net shown in Figure 4. The LPO describes a cycle in the p/t-net by executing all the transitions once. In both experiments we will repeat and compose copies of this cycle of events to bigger and more complex LPOs. In the first experiment we will consider thin LPOs while in the second we will consider dense LPOs.

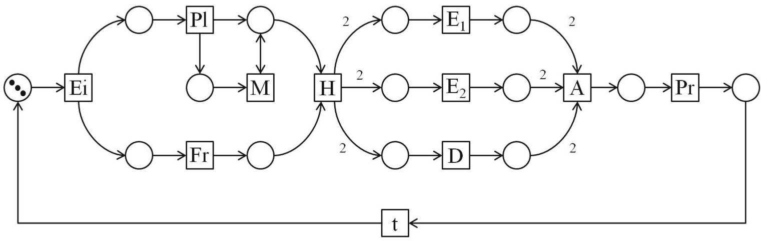

Fig. 4. A p/t-net describing a workflow

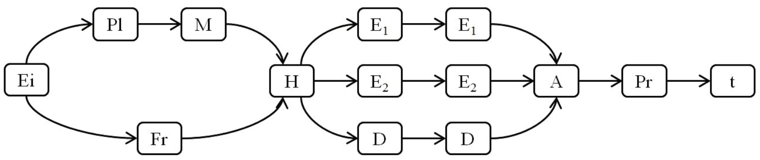

Fig. 5. Skeleton of an LPO of the partial language of the p/t-net shown in Figure 4

Experiment 1. *We consider 5 LPOs $lpo_1, \ldots, lpo_5$ of the partial language of the p/t-net of Figure 4. Each LPO is a composition of copies of the LPO shown in Figure 5. The number of copies varies from 6 to 120. The resulting LPOs are connected as shown in Figure 6, resulting in three parallel threads of events. We decide the verification problems using the algorithms I to VI.*

	lpo_1	lpo_2	lpo_3	lpo_4	lpo_5
number of copies	*6*	*15*	*30*	*60*	*120*
number of events	*252*	*630*	*420*	*1260*	*5040*
runtime in ms					
Alg I (cuts)	*480*	*15091*	-	-	-
Alg II (tokenflows, [4])	*170*	*3002*	*25364*	*269862*	-
Alg III (tokenflows, [24], Dinic)	*3*	*15*	*55*	*250*	*1336*
Alg IV (tokenflows, [24], preflow-push)	*3*	*11*	*47*	*216*	*1130*
Alg V (compact tokenflows, Dinic)	*4*	*11*	*43*	*173*	*871*
Alg VI (compact tokenflows, preflow-push)	*3*	*12*	*36*	*148*	*699*

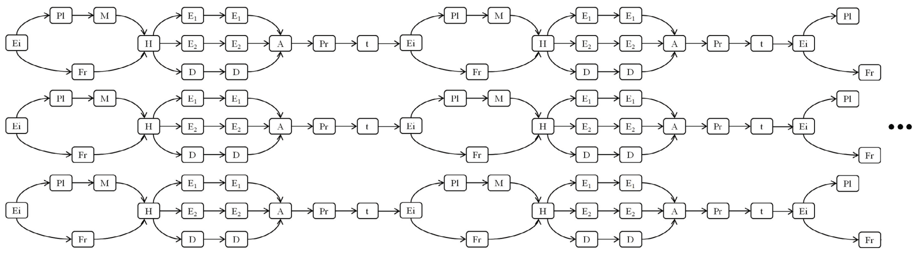

Fig. 6. A composition of copies of the LPO shown in Figure 5 leading to a thin LPO

In Experiment 1 algorithm I performs quite poorly. Within 10 minutes this algorithm only decides the first two of the given verification problems. The main reason for this is the huge number of cuts existing in the specified LPOs. Algorithm II performs better than algorithm I. As stated in [4] using tokenflows is reasonable if the LPO describes some concurrent behaviour. Algorithm III, as described in [24], is $|V|$-times faster then algorithm II. Algorithm II needs to construct a maximal flow for each place of the p/t-net and each event of the LPO. Algorithm III and IV only construct one maximal flow for each place. Algorithms V and VI use the new compact definition of tokenflows. As shown in the previous sections compact tokenflows regard only skeleton arcs instead of the transitive relation of the specified LPO. Experiment 1 contains a lot of concurrency, but even in this case the difference between the skeleton and the transitive relation matters. Both algorithms V and VI perform much better than the algorithms III and IV.

To further investigate the difference between all considered algorithms we provide a second experiment and increase the number of described dependencies between events of the tested LPOs.

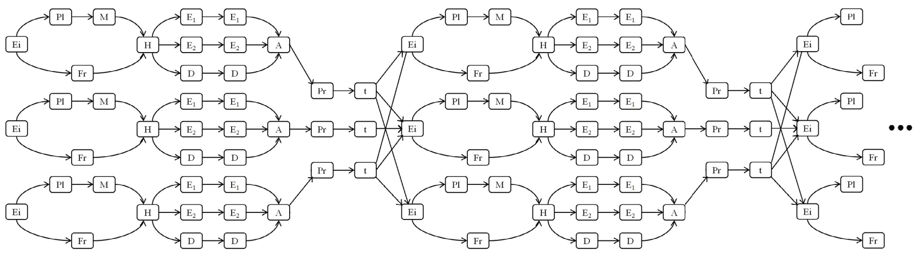

Fig. 7. A composition of copies of the LPO shown in Figure 5 leading to a dense LPO

Experiment 2. *We consider 5 LPOs $lpo_1, \ldots, lpo_5$ of the partial language of the p/t-net of Figure 4. Each LPO is a composition of copies of the LPO shown in Figure 5. The number of copies varies from 6 to 120. The resulting LPOs are connected as shown in Figure 7. We again decide the verification-problem using the algorithms I to VI.*

	lpo_1	lpo_2	lpo_3	lpo_4	lpo_5
number of copies	*6*	*15*	*30*	*60*	*120*
number of events	*252*	*630*	*420*	*1260*	*5040*
runtime in ms					
Alg I (cuts)	*126*	*558*	*2424*	*12175*	*84605*
Alg II (tokenflows, [4])	*194*	*3602*	*32806*	*334647*	*-*
Alg III (tokenflows, [24], Dinic)	*3*	*16*	*70*	*334*	*1955*
Alg IV (tokenflows, [24], preflow-push)	*3*	*14*	*65*	*342*	*1936*
Alg V (compact tokenflows, Dinic)	*4*	*11*	*42*	*170*	*881*
Alg VI (compact tokenflows, preflow-push)	*4*	*9*	*35*	*146*	*648*

In Experiment 2 algorithm I preforms better then in Experiment 1. The number of cuts is reduced and the algorithm is able to solve the first four tests within 10 minutes. Nevertheless, its runtime is poor. This time algorithm II performs worse then algorithm I. The number of transitive arcs to be considered is much higher than in Experiment 1. The notion of tokenflows highly depend on the number of arcs of a given

LPO. By the same reason Algorithm III and IV perform worse than in Experiment 1. Again, the runtime of algorithm II is $|V|$-times the runtime of algorithm III. Algorithm V and VI use the new compact definition of tokenflows. Their runtime almost equals the runtime given in Experiment 1. The LPOs shown in Figure 6 and 7 have almost an identical set of skeleton arcs. The runtime of algorithm V and VI is by this means independent of the number of described transitive dependencies between events. Again, both algorithms are by far the fastest algorithms in this experiment.

A comparison of algorithm III and IV shows that the choice of a the maximal flow algorithm does not matter that much while using normal tokenflows. For both algorithms the associated flow networks contain a big number of arcs, but only short paths form source to sink. The maximal length of each path is 3 (see [24]). In that case choosing a simple straightforward maximal flow algorithm like the algorithm of Dinic is sufficient. The comparison of algorithm V and VI, both using the new compact tokenflows, shows that the choice of the maximal flow algorithm matters. The structure of the associated flow network matches the structure of the skeleton of the LPO. Dealing with such flow networks using a preflow push algorithms with a gap-heuristic leads to the best runtime results. Notice, that the gap heuristic leads to an algorithm that only calculates the value of a maximal-flow and not the flow function itself. If a compact tokenflow should be constructed the algorithm of Dinic is the best choice.

6 Conclusion

We have shown a new compact definition of the partial language of a p/t-net. The new definition is based on the idea of tokenflows, since the number of arcs does not grow exponentially in the number of given events of an LPO. This new definition is only defined on the skeleton arcs of a given LPO such that it is not prone to the number of described dependencies. Compact tokenflows abstracts from the distribution of token as far as possible. In contrast to interlaced tokenflows only a single tokenflow is needed. All this leads to a definition which is optimized for deciding the verification problem. We have presented a corresponding verification algorithm and experimental results of its implementation in VipTool. We compared these results to all existing reasonable alternative algorithms.

An important topic of future research is to investigate if the new definition is applicable in the field of synthesis of p/t-net or unfolding a given p/t-net to its set of runs. In both fields algorithms using tokenflow lead to fast algorithms if the number of dependencies of the occurring LPOs are small. We hope that the new compact definition of tokenflows leads to similar results as for the verification problem, i.e. fast algorithms not prone to the structure of the given LPO or p/t-net.

References

[1] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network flows: Theory, algorithms, and applications. Prentice-Hall, Englewood Cliffs (1993)

[2] Baumgarten, B.: Petri-Netze: Grundlagen und Anwendungen. Spektrum, Heidelberg (1996)

[3] Bergenthum, R.: Algorithmen zur Verifikation von halbgeordneten Petrinetz-Abläufen: Implementierung und Anwendungen. Diplomarbeit, Katholische Universität Eichstätt-Ingolstadt (2006)
[4] Bergenthum, R., Desel, J., Juhás, G., Lorenz, R.: Can I Execute My Scenario in Your Net? VipTool Tells You! In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 381–390. Springer, Heidelberg (2006)
[5] Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. Fundamenta Informaticae 88(4), 437–468 (2008)
[6] Bergenthum, R., Desel, J., Mauser, S.: Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language. In: Jensen, K., Billington, J., Koutny, M. (eds.) ToPNoC III, LNCS, vol. 5800, pp. 216–243. Springer, Heidelberg (2009)
[7] Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Synthesis of Petri Nets from Term Based Representations of Infinite Partial Languages. Fundamenta Informaticae 95(1), 187–217 (2009)
[8] Bergenthum, R., Juhás, G., Lorenz, R., Mauser, S.: Unfolding Semantics of Petri Nets Based on Token Flows. Fundamenta Informaticae 94(3-4), 331–360 (2009)
[9] Bergenthum, R., Lorenz, R., Mauser, S.: Faster Unfolding of General Petri Nets Based on Token Flows. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 13–32. Springer, Heidelberg (2008)
[10] Best, E., Devillers, R.: Sequential and concurrent behaviour in Petri net theory. Theoretical Computer Science 55(1), 87–136
[11] Desel, J., Juhás, G.: "What Is a Petri Net?" Informal Answers for the Informed Reader. In: Ehrig, H., Juhás, G., Padberg, J., Rozenberg, G. (eds.) Unifying Petri Nets, LNCS, vol. 2128, pp. 1–25. Springer, Heidelberg (2001)
[12] Desel, J., Juhás, G., Lorenz, R., Neumair, C.: Modelling and Validation with VipTool. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 380–389. Springer, Heidelberg (2003)
[13] Desel, J., Reisig, W.: Place/Transition Petri Nets. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 122–173. Springer, Heidelberg (1998)
[14] Dinic, E.A.: Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. Soviet Math Doklady 11, 1277–1280 (1970)
[15] Esparza, J., Römer, S., Vogler, W.: An Improvement of McMillan's Unfolding Algorithm. Formal Methods in System Design 20(3), 285–310 (2002)
[16] Ford, L.R., Fulkerson, D.R.: Maximal Flow Through A Network. Canadian Journal of Mathematics 8, 399–404 (1956)
[17] Goltz, U., Reisig, W.: Processes of Place/Transition-Nets. In: Díaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 264–277. Springer, Heidelberg (1983)
[18] Goltz, U., Reisig, W.: The Non-sequential Behavior of Petri Nets. Information and Control 57(2/3), 125–147 (1983)
[19] Grabowski, J.: On partial languages. Fundamenta Informaticae 4(2), 427–498 (1981)
[20] Janicki, R., Koutny, M.: Structure of concurrency. Theoretical Computer Science 112(1), 5–52 (1993)
[21] Juhás, G., Lorenz, R., Desel, J.: Can I Execute My Scenario in Your Net? In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 289–308. Springer, Heidelberg (2005)
[22] Karzanov, A.V.: Determining the maximal flow in a network by the method of preflows. Soviet Mathematics Doklady 15, 434–437 (1974)
[23] Kiehn, A.: On the Interrelation Between Synchronized and Non-Synchronized Behaviour of Petri Nets. Elektronische Informationsverarbeitung und Kybernetik 24(1/2), 3–18 (1988)
[24] Lorenz, R.: Szenario-basierte Verifikation und Synthese von Petrinetzen: Theorie und Anwendungen, Katholische Universität Eichstätt-Ingolstadt. Habilitation (2006)

[25] Lorenz, R., Mauser, S., Bergenthum, R.: Testing the executability of scenarios in general inhibitor nets. In: Basten, T., Juhás, G., Shukla, S.K. (eds.) Proc. of Application of Concurrency to System Design 2007, pp. 167–176. IEEE Computer Society (2007)

[26] Malhotra, V.M., Kumar, M.P., Maheshwari, S.N.: An $O(|V|^{3})$ Algorithm for Finding Maximum Flows in Networks. Information Processing Letters 7(6), 277–278 (1994)

[27] McMillan, K.L., Probst, D.K.: A Technique of State Space Search Based on Unfolding. Formal Methods in System Design, 45–65 (1992)

[28] de Oliveira Oliveira, M.: Hasse Diagram Generators and Petri Net. Fundamenta Informaticae 105(3), 263–289 (2012)

[29] Peterson, J.L.: Petri net theory and the modeling of systems. Prentice-Hall, Englewood Cliffs (1981)

[30] Petri, C.A.: Kommunikation mit Automaten, Dissertation, Technische Universität Darmstadt (1962)

[31] Pratt, V.: Modelling Concurrency with Partial Orders. International Journal of Parallel Programming 15 (1986)

[32] Reisig, W.: Petrinetze: Eine Einführung. Springer, Berlin (1986)

[33] Solé, M., Carmona, J.: Process Mining from a Basis of State Regions. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 226–245. Springer, Heidelberg (2010)

[34] Vogler, W. (ed.): Modular Construction and Partial Order Semantics of Petri Nets. LNCS, vol. 625. Springer, Heidelberg (1992)

[35] Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)