Can I Execute My Scenario in Your Net?

Gabriel Juhás, Robert Lorenz, and Jörg Desel

Lehrstuhl für Angewandte Informatik,
Katholische Universität Eichstätt, 85071 Eichstätt, Germany
{gabriel.juhas, robert.lorenz, joerg.desel}@ku-eichstaett.de

Abstract. In this paper we present a polynomial algorithm to decide whether a scenario (given as a Labelled Partial Order) is executable in a given place/transition Petri net while preserving at least the given amount of concurrency (adding no causality). In the positive case the algorithm computes a process net that respects the concurrency formulated by the scenario. We moreover present a polynomial algorithm to decide whether the amount of concurrency given by a Labelled Partial Order is maximal, i.e. whether the Labelled Partial Order precisely matches a process net w.r.t. causality and concurrency of the events, if this process net represents a minimal causality of events among all process nets.

1 Introduction

Specifications of distributed systems are often formulated in terms of scenarios. In other words, it is often part of the specification that some scenarios should be executable by the system. Given the system, a natural question is whether a scenario can be executed. In this paper we consider Petri net models instead of systems, and we restrict our consideration to place/transition Petri nets. Transforming the above question to this model, we ask whether a given scenario represents a possible execution of a given Petri net. If the answer is positive for all specified scenarios then the Petri net model can be used as a design specification of the system to be implemented. We have not been precise w.r.t. scenarios and executions yet. In general, there are different ways to represent single executions of Petri nets. The most prominent concepts are occurrence sequences, i.e., sequences of transition names that can occur consecutively, and process nets ([4, 5]), i.e., Petri nets representing transition occurrences by events (transitions of process nets) with explicit pre- and post-conditions representing token occurrences of the original net (places of process nets). Playing the token game, it is very easy to check whether a given sequence of transition names is in fact an occurrence sequence of a given net with initial marking. For process nets, we can easily verify the defining conditions of a process net, which reads for marked place/transition Petri nets as follows:

- The underlying net has no cycles, hence the transitive closure of the relation given by arcs is a partial order,
- conditions are not branched,
- no event has an empty pre-set or an empty post-set,
- events are labelled by transitions and conditions by places,

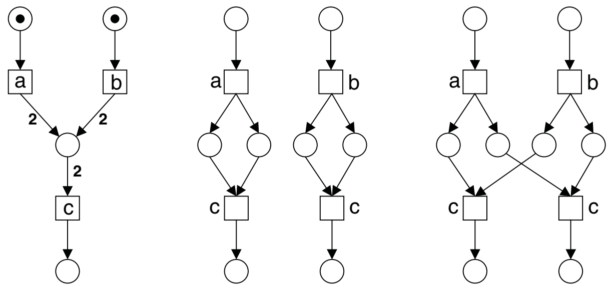

Fig. 1. A p/t-net (left figure) with two of its process nets

- the set of conditions with empty pre-set corresponds to the initial marking (if place p has $m_0(p)$ initial tokens then $m_0(p)$ conditions of this set are labelled by p),
- the pre- and post-sets of events respect the pre- and post-sets of the corresponding transitions (if a transition t consumes k tokens from a place p then each event labelled by t has k pre-conditions labelled by p, and similarly for post-conditions).

Clearly, deciding whether an occurrence sequence can be executed in a place/transition net as well as deciding whether an acyclic labelled net is a process net of a place/transition net can be done in linear time (w.r.t. the size of the occurrence sequence / process net) if the size of the original net is assumed to be constant. For motivation purpose, consider the following example. In Figure 1 a place/transition net is shown. One possible occurrence sequence is $a\ b\ c\ c$. Figure 1 also shows two process nets.

Occurrence sequences lack any information about independence and causality. So it is impossible to specify that events should occur concurrently by an occurrence sequence. Therefore, as soon as concurrency of events has to be specified, occurrence sequences cannot be used for specification of scenarios. Process nets are also not very suitable for specification purposes for two reasons. First, conditions are labelled by names of places of the model specified. So it is not possible to specify that two events have to occur in some order but it is rather also necessary to state which place is responsible for establishing this order. So the specification includes already details of an implementation. The second disadvantage is that a process net determines the precise causality between events. Hence it is not possible to specify a scenario with two events that may either occur (causally) ordered or concurrently. One way to overcome these problems is to specify scenarios in terms of Labelled Partial Orders of events, where the labels refer to the transitions of the specified model. These *LPO*s are called *pomsets* (partially ordered multisets) in [9], emphasizing their close relation to partially ordered sets (we have multisets here because the same transition can occur more than once in a pomset, formally represented by two distinct events labelled by the same transition name). LPOs are called *partial words* in [6], emphasizing their close relation to words or sequences; the total order of elements in a sequence is replaced by a partial order. Actually, pomsets and partial words do not distinguish isomorphic LPOs, because the order of transition occurrences only depends on the labels. So LPOs are somehow in-between occurrence sequences and process nets.

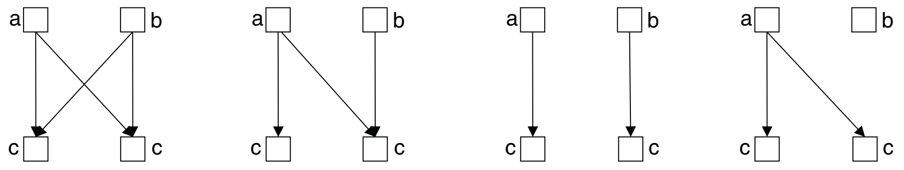

Fig. 2. Four labelled partial orders. All except the most right one are executions of the net in Figure 1. The third one (from the left) is the only strict execution of this net. The most left one refers to both process nets shown in Figure 1

An LPO represents a scenario that can (or cannot) be executed by a marked Petri net. The order defined between events of a so-called *executable* LPO is interpreted as follows: If two events e_1 and e_2, labelled by transitions t_1 and t_2 respectively, are ordered ($e_1 < e_2$) then either t_1 occurs causally before t_2 or both occur concurrently. If e_1 and e_2 are not ordered, then concurrent executions of t_1 and t_2 is demanded. Another interpretation of the order between events defines *strictly executable* LPOs: If e_1 and e_2 are ordered ($e_1 < e_2$) then t_1 is demanded to occur causally before t_2. If e_1 and e_2 are not ordered, then concurrent executions of t_1 and t_2 is demanded.

It is immediate to see that occurrence sequences are special cases of LPOs: A sequence $t_1 t_2 \dots t_n$ can be viewed as a partially ordered set of events $e_1 < e_2 < \dots < e_n$ where e_i is labelled by t_i for $1 \leqslant i \leqslant n$. Process nets can be translated to LPOs by removing all conditions and keeping the partial order for the events. We will call such LPOs runs. Formally, an LPO is executable by a given marked place/transition net if it includes (is more sequentialized than) a run of the net. An LPO is strictly executable, if it equals a minimal run (w.r.t. to set inclusion).

Figure 2 shows four LPOs. The first three represent executions of the net in Figure 1. The third one (from the left) is the only strict execution (it is the run given by the first process given in Figure 1). The fourth one is not an execution of this net.

The aim of this paper is to provide an efficient algorithm for deciding whether a given LPO is executable by a given place/transition net. We will provide such an algorithm and prove that its runtime is $O(n^4|P|)$, where n is the number of events of the LPO and $|P|$ is the number of places of the net. In the positive case, the algorithm computes a run included in the LPO, and thus an underlying process net. Moreover, we provide a (polynomial) characterization which tells us whether the computed run equals the LPO.

The surprising message of this paper might not be the existence of a polynomial algorithm but conversely the fact that this is not a trivial problem. In fact, for elementary Petri nets or 1-safe place/transition nets there exists an immediate algorithm to decide the problem because a unique corresponding process net can be constructed from an LPO – if it exists. The crucial point for place/transition nets is that in general there is not a unique process net corresponding to a given LPO (i.e. an LPO can include different runs). For example, the first LPO given in Figure 2 refers to both process nets of Figure 1.

Astrid Kiehn [8] and Walter Vogler [11] showed that an LPO can be executed if and only if for each cut of the LPO the marking reached by firing all transitions corresponding to events smaller than the cut enables the multiset of transitions given by the cut.

Unfortunately, this result does not lead to an efficient algorithm because the number of cuts grows exponentially with the size of the LPO in general. This result seems to be not very surprising. However, its proof in [8] is quite complicated, and the shorter proof in [10, 11] employs a version of the nontrivial Marriage Theorem known from Graph Theory.

The construction of the algorithm for testing executability and the proof of our result is based on Flow Theory [2]. We will transform a part of our construction to the problem of finding a maximal flow in a flow network associated to the LPO. This maximal flow problem was extensively studied for decades. In [7] an algorithm is presented running in cubic time, and there are several improvements since then (see [3] for an overview). We obtain our complexity result by repeated transformations to flow networks and computations of the maximal flow.

The structure of the remainder of this paper is as follows. In section 2 we provide standard definitions of place/transition nets, occurrence sequences, process nets and LPOs. In Section 3 we establish the so-called flow property of an LPO as a necessary and sufficient condition for its executability. Section 4 is the core of the paper. By applying a maximum flow algorithm we decide whether a given LPO satisfies the flow property. Section 5 introduces the strong flow property which characterizes LPOs corresponding exactly to a run of a process net and briefly presents a polynomial test of strict executability of LPOs.

2 Place/Transition Nets

We use $\mathbb{N}$ to denote the nonnegative integers. Given a function f from A to B and a subset C of A we write $f|_C$ to denote the restriction of f to the set C. Given a finite set A, the symbol $|A|$ denotes the cardinality of A. The set of all multi-sets over a set A is denoted by $\mathbb{N}^A$. Given a binary relation $R \subseteq A \times A$ over a set A, the symbol R^+ denotes the transitive closure of R.

2.1 Place/Transition Net Definitions

Definition 1 (Net).
A net *is a triple* (P, T, F)*, where* P *is a finite set of* places*,* T *is a finite set of* transitions*, satisfying* $P \cap T = \emptyset$*, and* $F \subseteq (P \cup T) \times (T \cup P)$ *is a* flow relation.

Let (P, T, F) be a net and $x \in P \cup T$ be an element. The *preset* $\bullet x$ is the set $\{y \in P \cup T \mid (y, x) \in F\}$, and the *post-set* $x\bullet$ is the set $\{y \in P \cup T \mid (x, y) \in F\}$. Given a set $X \subseteq P \cup T$, this notation is extended by $\bullet X = \bigcup_{x \in X} \bullet x$ and $X\bullet = \bigcup_{x \in X} x\bullet$. For technical reasons, we consider only nets in which every transition has a nonempty pre-set and a nonempty post-set.

Definition 2 (Place/transition net).
A place/transition net *(shortly* p/t-net*)* N *is a quadruple* (P, T, F, W)*, where* (P, T, F) *is a net and* $W : F \to \mathbb{N}^+$ *is a* weight function.

We extend the weight function W to pairs of net elements $(x, y) \in (P \times T) \cup (T \times P)$ satisfying $(x, y) \notin F$ by $W((x, y)) = 0$.

A *marking* of a net $N = (P, T, F, W)$ is a function $m : P \to \mathbb{N}$, i.e. a multi-set over P.

Definition 3 (Occurrence rule).
Let $N = (P, T, F, W)$ *be a p/t-net. A transition* $t \in T$ *is* enabled to occur in a marking m *of* N *iff* $m(p) \geq W((p, t))$ *for every place* $p \in \bullet t$*. If a transition* t *is enabled to occur in a marking* m*, then its* occurrence *leads to the new marking* m' *defined by* $m'(p) = m(p) - W((p, t)) + W((t, p))$ *for every* $p \in P$.

Definition 4 (Marked p/t-net).
A marked p/t-net *is a pair* (N, m_0)*, where* N *is a p/t-net and* m_0 *is a marking of* N *called* initial marking.

2.2 Labelled Partial Orders

In this section we recall the definition of semantics of p/t-nets based on labelled partial orders, also known as partial words [6] or pomsets [9]. For proofs of the presented results see e.g. [10].

Definition 5 (Directed graph, (Labelled) partial order).
A directed graph *is a pair* $(V, \to)$*, where* V *is a finite* set of nodes *and* $\to \subseteq V \times V$ *is a binary relation over V called the* set of arcs*. As usual, given a binary relation* $\to$ *we write* $a \to b$ *to denote* $(a, b) \in \to$.

A partial order *is a directed graph* $po = (V, <)$*, where* $<$ *is an irreflexive and transitive binary relation on* V.

For a set $S \subseteq V$ *and a node* $v \in V \setminus S$ *we write* $v < S$*, if* $v < s$ *for a node* $s \in S$.

Two nodes v, v' *of a partial order* $(V, <)$ *are called* independent *if* $v \not< v'$ *and* $v' \not< v$*. By* $co \subseteq V \times V$ *we denote the set of all pairs of independent nodes of* V*. A* co-set *in a partial order* $(V, <)$ *is a subset* $S \subseteq V$ *fulfilling:* $\forall x, y \in S : x \, co \, y$*. Clearly the relation* co *is symmetric and reflexive. A* cut *is a maximal co-set.*

Given partial orders $po_1 = (V, <_1)$ *and* $po_2 = (V, <_2)$*, we say that* po_2 *is a* sequentialization of po_1 *if* $<_1 \subseteq <_2$.

A labelled partial order *is a triple* $lpo = (V, <, l)$*, where* $(V, <)$ *is a partial order, and* l *is a* labelling function *on* V*. If X is a set of labels of lpo, i.e.* $l : V \to X$*, then for a cut* $S \subseteq V$*, we define the multi-set* $\mu(S) \subseteq \mathbb{N}^X$ *by* $\mu(S)(x) = |\{v \in V \mid v \in S \wedge l(v) = x\}|$.

We use the above notation defined for partial orders also for labelled partial orders.

Definition 6 (Enabledness of LPOs).
A labelled partial order $lpo = (V, <, l)$ *with* $l : V \to T$ *is called* enabled to occur in a marking m *if the following statement holds: For every cut* S *of* $<$ *and every* $p \in P$*:* $m(p) + \sum_{v \in V \wedge v < S}(W((l(v), p)) - W((p, l(v)))) \geq \sum_{v \in S} W((p, l(v)))$*. Its occurrence leads to the marking* $m'(p)$ *given by* $m'(p) = m(p) + \sum_{v \in V}(W((l(v), p)) - W((p, l(v))))$.

A labelled partial order $lpo = (V, <, l)$ *enabled in* m *is said to be* minimal *iff there exists no labelled partial order* $lpo' = (V, <', l)$ *enabled in* m *with* $<' \subset <$.

Proposition 1. *If a labelled partial order is enabled in m and leads to m', then every sequentialization is enabled in m and leads to m', too.*

2.3 Processes, Runs and Executability of LPOs

Definition 7 (Occurrence net).
An occurrence net *is a net $O = (B, E, G)$ such that $| \bullet b|$, $|b \bullet | \leqslant 1$ for every $b \in B$ (places are* unbranched*) and O is* acyclic*, i.e. the transitive closure G^+ of G is a partial order. Places of an occurrence net are called* conditions *and transitions of an occurrence net are called* events.

The set of conditions of an occurrence net $O = (B, E, G)$ which are minimal (maximal) according to G^+ is denoted by $Min(O)$ ($Max(O)$). Clearly, $Min(O)$ and $Max(O)$ are cuts w.r.t. G^+ (recall that events have nonempty pre- and post-sets by assumption).

Definition 8 (Process).
Let (N, m_0) be a marked p/t-net, $N = (P, T, F, W)$. A process *of (N, m_0) is a pair $K = (O, \rho)$, where $O = (B, E, G)$ is an occurrence net and $\rho : B \cup E \to P \cup T$ is a labelling function, satisfying*

(i) $\rho(B) \subseteq P$ *and* $\rho(E) \subseteq T$.
(ii) $\forall e \in E, \forall p \in P : \ |\{b \in \bullet e \mid \rho(b) = p\}| = W((p, \rho(e)))$ *and*
$\forall e \in E, \forall p \in P : \ |\{b \in e\bullet \mid \rho(b) = p\}| = W((\rho(e), p))$.
(iii) $\forall p \in P : \ |\{b \in Min(O) \mid \rho(b) = p\}| = m_0(p)$.

Definition 9 (Run).
Let $K = (O, \rho)$ be a process of a marked p/t-net (N, m_0). The labelled partial order $lpo_K = (E, G^+|_{E\times E}, \rho|_E)$ is called run *of (N, m_0) representing K.*

A run $lpo = (E, <, l)$ of (N, m_0) is said to be minimal *iff there exists no other run $lpo' = (E, <', l)$ of (N, m_0) with $<' \subset <$.*

Definition 10 (Executability of LPOs). *A labelled partial order $lpo = (V, \prec, l)$ is called* executable in a marked p/t-net (N, m_0) *if there exists a run $(V, <, l)$ of (N, m_0) with $< \subseteq \prec$.*

A labelled partial order $lpo = (V, \prec, l)$ is called strictly executable in a marked p/t-net (N, m_0) *if it is a minimal run of (N, m_0).*

Directly from the definition of processes we obtain:

Proposition 2. *Every run of (N, m_0) is enabled in m_0.*

From proposition 1 and proposition 2 follows:

Proposition 3. *If a labelled partial order is executable in (N, m_0), then it is also enabled in m_0.*

The important result completing the relationship between enabled and executable labelled partial orders was proven in [8, 10].

Theorem 1. *If a labelled partial order is enabled in m_0 in a p/t-net N, then it is executable in (N, m_0).*

3 Flow Property

As described in the introduction, the definition of enabledness of LPOs is inherently exponential, since an LPO can have exponentially many cuts in the number of nodes. That means, the definition is not appropriate to develop a test of executability.

Instead, we introduce the so called *flow property* of labelled partial orders w.r.t. a marked p/t-net (N, m_0). In this section we show: A labelled partial order fulfills the flow property w.r.t (N, m_0) if and only if it is executable in (N, m_0). In the next section we will give a polynomial test of fulfilling the flow property for a labelled partial order. In the positive case, this test will compute a run included in this labelled partial order.

We fix a marked p/t-net (N, m_0) and a place p of N. Given a labelled partial order *lpo* $= (V, <, l)$ with $l(V) = T$ we assign non-negative integers to its edges through a so called *flow function*. The aim is to find a flow function assigning values $x(v, v')$ to edges (v, v') in such a way that there is a process with exactly $x(v, v')$ post-conditions of v labelled by p which are also pre-conditions of v'. Thus, such a flow function of *lpo* abstracts from individuality of conditions of a process and encodes the flow relation of this process by natural numbers. Clearly, finding such a flow function for every place means that *lpo* includes the run of this process.

In order to simplify the formal definition of the flow property, let us define an extension of *lpo* $= (V, <, l)$ by adding an initial node which is smaller than all nodes from V and which is labelled by a new label.

Definition 11 (0-extension of a labelled partial order). *Let lpo* $= (V, <, l)$ *be a labelled partial order. Then a labelled partial order lpo*$^0 = (V^0, <^0, l^0)$*, where* $V^0 = (V \cup \{v_0\})$*,* $v_0 \notin V$*,* $<^0 = < \cup (\{v_0\} \times V)$*,* $l^0(v_0) \notin l(V)$ *and* $l^0|_V = l$*, is called* 0-extension of *lpo.*

Assigning natural numbers to the arcs of a 0-extension of a labelled partial order we define a so called flow function of this labelled partial order (with v_0 as its unique smallest element).

Definition 12 (Flow function of a labelled partial order). *Let lpo* $= (V, <, l)$ *be a labelled partial order and lpo*$^0 = (V^0, <^0, l^0)$ *be a* 0*-extension of lpo. A function* $x : <^0 \to \mathbb{N}$ *is called* flow function *of lpo. For* $v \in V$*, we denote*

- $\sum_{v' <^0 v} x((v', v))$ *the* ingoing flow of v w.r.t. x*, and*
- $\sum_{v <^0 v'} x((v, v'))$ *the* outgoing flow of v w.r.t. x.

Let *lpo* $= (V, <, l)$ be a run representing a process of (N, m_0). For every place p define the *canonical flow function of the run w.r.t.* p, by counting for every $v < v'$ the number of post-conditions of v labelled by p which are pre-conditions of v' in the process. The outgoing flow of the source event v_0 represents the the number of minimal conditions labelled by p which are used by further events.

Definition 13 (Canonical flow function of a run). *Let* $K = (O, \rho)$ *be a process of* (N, m_0) *with* $O = (B, V, G)$ *and let lpo* $= (V, <, l)$ *be the run representing* K*. Let lpo*$^0 = (V^0, <^0, l^0)$ *be a* 0*-extension of lpo. Define* $v_0\bullet = Min(O)$ *for the unique* *smallest element v_0 of $(V^0, <^0)$. We define for every place $p \in P$ the* flow function *$x_p :<^0\rightarrow \mathbb{N}_0$ of lpo as follows:*

$$x_p(v, v') = |\{b \in B \mid \rho(b) = p \wedge b \in v \bullet \cap \bullet v'\}|.$$

By definition, this canonical flow function respects the weight function and the initial marking of (N, m_0) in the following sense:

(A) The ingoing flow of an event v equals the number of tokens consumed from place p by the occurrence of transition $l(v)$.

(B) The outgoing flow of an event v (i.e. the number of post-conditions of v labelled by p which are used as pre-conditions of other events) is less than or equal to the number of tokens which are produced by the occurrence of transition $l(v)$ in place p. In particular, the outgoing flow of the source event v_0 is less or equal to the number of tokens in place p of the initial marking m_0.

In general, we say that an arbitrary labelled partial order, whose labels are transitions of (N, m_0), fulfils the flow property w.r.t. (N, m_0), if for every place there exists a flow function which fulfils the properties (A) and (B).

Definition 14 (Flow property). *Let lpo $= (V, <, l)$ be a labelled partial order with $l(V) = T$ and let lpo$^0 = (V^0, <^0, l^0)$ be a 0-extension of lpo. Denote $W((l(v_0), p)) = m_0(p)$ for each place p. We say that lpo fulfils the flow property w.r.t. (N, m_0) if the following statement holds: For every place $p \in P$ there exists a flow $x_p :<^0\rightarrow \mathbb{N}$ such that*

(A) For every $v' \in V$: $\sum_{v<^0v'} x_p(v, v') = W((p, l(v')))$.

(B) For every $v \in V^0$: $\sum_{v<^0v'} x_p(v, v') \leqslant W((l(v), p))$.

The ingoing flow of a node v w.r.t. x_p is also called (A)-sum of *x_p* w.r.t. *v and the outgoing flow of a node v w.r.t. x_p is also called (B)*-sum of x_p w.r.t. v.

Given a run *lpo* of (N, m_0), it follows directly from the definitions of processes and runs that for every $p \in P$ the canonical flow function x_p of *lpo* fulfils the statements (A) and (B):

Lemma 1. *Every run of (N, m_0) fulfils the flow property w.r.t. (N, m_0).*

By the definition of the flow property, given a labelled partial order *lpo* $= (V, <, l)$ fulfilling (A) and (B) w.r.t. a place p and a flow function x_p and a labelled partial order *lpo*$' = (V, <', l)$ with $<\subset<'$ we have: *lpo*$'$ fulfils (A) and (B) w.r.t. to the place p and the flow function x'_p given by $x'_p|_< = x_p$ and $x'_p|_{<'\setminus<} = 0$. Therefore:

Lemma 2. *Every labelled partial order executable in (N, m_0) fulfils the flow property w.r.t. (N, m_0).*

The following lemma states the converse:

Lemma 3. *Let lpo $= (V, \prec, l)$ be a labelled partial order which fulfils the flow property w.r.t. (N, m_0). Then lpo is executable in (N, m_0), i.e. there exists a run $(V, <, l)$ of (N, m_0) such that $<\subseteq\prec$.*

Proof. From the definition of the flow property, for every place $p \in P$ there exists a function x_p which fulfils (A) and (B). We will fix these functions and use them to construct a process $K = (O, \rho)$ of (N, m_0) with $O = (B, V, G)$ and $\rho|_V = l$, satisfying $<= G^+|_{V \times V} \subseteq \prec$. According to the definition of runs, this will conclude the proof.

For convenience, denote $V = \{v_1, \ldots, v_{|V|}\}$ such that $v_i \prec v_j$ implies $i < j$. First define the set of conditions and the labelling of conditions. For every event $v \in V^0$ we define the set of post-conditions of v labelled by $p \in P$:

$$B_p^v = \{p_v^1, \ldots p_v^{W((l(v),p))}\}.$$

Thus, the number of these post-conditions equals the value $W((l(v), p))$. Especially, the number of post-conditions of v_0 labelled by $p \in P$ equals $m_0(p)$. Denote $B_p = \cup_{v \in V^0} B_p^v$ the set of all conditions labelled by p. Define the labelling of conditions by $\rho(b) = p$ for $b \in B_p$. Finally, the set of all conditions of the process is given by $B = \cup_{p \in P} B_p$.

It remains to define the flow relation G. It is the union of all ingoing and outgoing arcs of all events $v \in V$. An event $v \in V$ has an outgoing arc to each of its post-conditions (observe that $v_0 \notin V$). Thus, the set of outgoing arcs of an event $v \in V$ labelled by $p \in P$ is

$$G_p^{v\bullet} = \{v\} \times B_p^v.$$

The ingoing arcs are defined w.r.t. the flow functions. If $x_p(v, v_m) > 0$, then we connect exactly $x_p(v, v_m)$ post-conditions of v labelled by p with v_m. In order to avoid branching of conditions, we connect the post-conditions $p_v^1, \ldots, p_v^{x_p(v,v_m)}$ with the event v_m which has the smallest index m from all events v_m with $x_p(v, v_m) > 0$, and so on. Formally, define the set of ingoing arcs from conditions labelled by $p \in P$ to an event $v_m \in V$ by

$$\begin{aligned} G_p^{\bullet v_m} = \{(p_v^i, v_m) \mid v \in V_0,\ & x_p(v, v_m) > 0, \\ & \sum_{j<m} x_p(v, v_j) < i \leqslant \sum_{j \leqslant m} x_p(v, v_j)\}. \end{aligned}$$

Because x_p fulfils (B), i.e. the number of post-conditions of an event $v \in V^0$ is not less than the outgoing flow of v, by this construction any event $v_m \in V$ is connected with exactly $x_p(v, v_m)$ post-conditions of v labelled by p whenever $x_p(v, v_m) > 0$. Because of this and because x_p also fulfils (A), by this construction every $v_m \in V$ has exactly $W((p, l(v_m)))$ pre-conditions labelled by $p \in P$. Finally denote $G_p = \cup_{v \in V}(G_p^{\bullet v} \cup G_p^{v\bullet})$ for every $p \in P$ and $G = \cup_{p \in P} G_p$.

By construction, the conditions are unbranched and the defined net is acyclic, i.e. $O = (B, V, G)$ is an occurrence net. From the previous reasoning $K = (O, \rho)$ is a process of (N, m_0).

It remains to show that $<= G^+|_{V \times V} \subseteq \prec$. Denote $R = \{(v, v') \in V \times V \mid v \bullet \cap \bullet v' \neq \emptyset\}$. Observe that $G^+|_{V \times V} = R^+$ and (by construction of G) we have $(v, v') \in R \implies (\exists p \in P : x_p(v, v') > 0)$. Because $x_p(v, v') > 0$ implies $v \prec v'$ and $\prec$ is transitive, this gives $<= G^+|_{V \times V} \subseteq \prec$. $\square$

4 Testing the Flow Property

In this section we give a polynomial algorithm to test whether a labelled partial order $lpo = (V, <, l)$ with $l(V) = T$ fulfils the flow property w.r.t. (N, m_0). In the case that *lpo* fulfils the flow property, the algorithm constructs flow functions for all places.

4.1 The Algorithm

We describe the algorithm for a fixed place p. Let $lpo^0 = (V^0, <^0, l^0)$ be a 0-extension of *lpo*. Throughout this section denote $V^0 = \{v_0, v_1, \ldots, v_n\}$ with $v_i < v_j \Rightarrow i < j$. The algorithm starts with a flow function x_0 fulfilling part (A) of the flow property. Such x_0 always exists, e.g. set $x_0(v_0, v') = W((p, l(v')))$ for each $v' \in V$ and $x_0(v, v') = 0$ otherwise. In general this flow function will not fulfil part (B) of the flow property. We denote max_0 the smallest index for which x_0 does not fulfil property (B):

– $\sum_{v_{max_0} <^0 v'} x_0(v_{max_0}, v') > W((l(v_{max_0}), p))$, and
– $\forall j < max_0$: $\sum_{v_j <^0 v'} x_0(v_j, v') \leqslant W((l(v_j), p))$.

Thus, the aim is to modify x_0, such that the (B)-sum of x_0 w.r.t. the index max_0 is reduced as much as necessary to fulfil (B) w.r.t. max_0, while preserving property (A) for all indexes and property (B) for all indexes smaller than max_0. In Subsection 4.4 we will describe in detail a procedure which modifies x_0 in such a way while minimizing the (B)-sum w.r.t. max_0 in some sense. In the following we will refer to this procedure as the *main procedure* of the algorithm. Repeating the main procedure, we get the algorithm:

1. Set $i = 0$ and compute a flow function x_i fulfilling (A).
2. If x_i does not fulfil (B):
   - Compute the smallest index max_i for which x_i does not fulfil (B).
   - Repeat:
     - ∗ Apply the main procedure to modify x_i into a flow function x_{i+1} (in such a way that x_{i+1} still fulfils (A) for all indexes, still fulfils (B) for all indexes smaller than max_i, and the (B)-sum of x_{i+1} w.r.t. max_i is smaller than (or equal to) the (B)-sum of x_i w.r.t. max_i).
     - ∗ If x_{i+1} does not fulfil (B), compute the smallest index max_{i+1} for which x_{i+1} does not fulfil (B).
     - ∗ Set $i = i + 1$.

   until x_i fulfils (B) or $max_i = max_{i-1}$.

The algorithm terminates, if either x_i fulfils property (B) or $max_i = max_{i-1}$. In the first case x_i is a flow function, for which *lpo* fulfils the flow property. In the second case we will prove in Subsection 4.5 that *lpo* is not enabled w.r.t. (N, m_0).

The algorithm has to be applied for every place $p \in P$. Since $max_i \leqslant n$, the main procedure is repeated at most n times. The main procedure itself requires at most $O(n^3)$ time as shown in the Subsection 4.4. Altogether we get that the test of executability takes $O(n^4|P|)$ time.

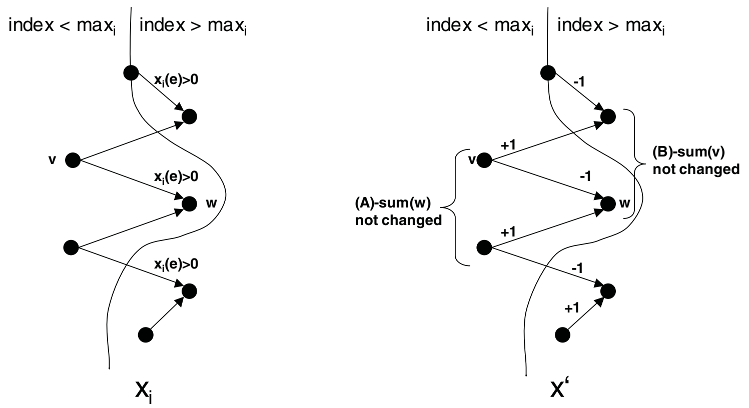

Fig. 3. The left part shows an example of a flow decreasing sequence of the first kind with $k = 3$. The right part shows the modifying operation to get a flow function x' from x_i satisfying $x'(v_{max_i}, w^1) < x_i(v_{max_i}, w^1)$, property (A) for all indexes and property (B) for all indexes smaller than max_i

4.2 Flow Decreasing Sequences

We start with a brief motivation of the main procedure for modifying x_i: Consider the following two possibilities of reducing the (B)-sum of x_i w.r.t. max_i while respecting property (A) for all indexes and property (B) for all indexes smaller than max_i.

The first possibility is to move positive values $x_i(v_{max_i}, v')$ onto edges (v_j, v_l) with $j > max_i$ (see Figure 3): Suppose a sequence of nodes $v^0 = v_{max_i}, w^1, v^1, \ldots, w^k, v^k$ such that

- $w^j \neq w^m$ and $v^j \neq v^m$ for $j \neq m$,
- $x(v^j, w^{j+1}) > 0$ and $v^j < w^j$,
- For each $0 < j < k$ there is an index $m < max_i$ with $v^j = v_m$,
- For $j = k$ there is an index $m > max_i$ with $v^k = v_m$.

Such a sequence allows to modify x_i into a new flow function x' defined as follows:

$$x'(v^j, w^j) = x_i(v^j, w^j) + 1,$$
$$x'(v^j, w^{j+1}) = x_i(v^j, w^{j+1}) - 1.$$

Obviously x' satisfies $x'(v_{max_i}, w^1) < x_i(v_{max_i}, w^1)$, property (A) for all indexes and property (B) for all indexes smaller than max_i. This modification can be applied for each such sequence as long as $x'(v^j, w^{j+1}) > 0$ for all j, thus reducing the (B)-sum of x_i w.r.t. max_i. As a consequence $x'(v^k, w^k) > x_i(v^k, w^k)$, i.e. the (B)-sum w.r.t. v^k is increased. Nevertheless property (B) remains satisfied for all indexes smaller than max_i.

The second possibility is to move positive values $x_i(v_{max_i}, v')$ onto edges (v_j, v_l) with $j < max_i$ and $\sum_{v_j < v'} x_i(v_j, v') < W((l(v_j), p))$ (see Figure 4): Suppose a sequence of nodes $v^0 = v_{max_i}, w^1, v^1, \ldots, w^k, v^k$ such that

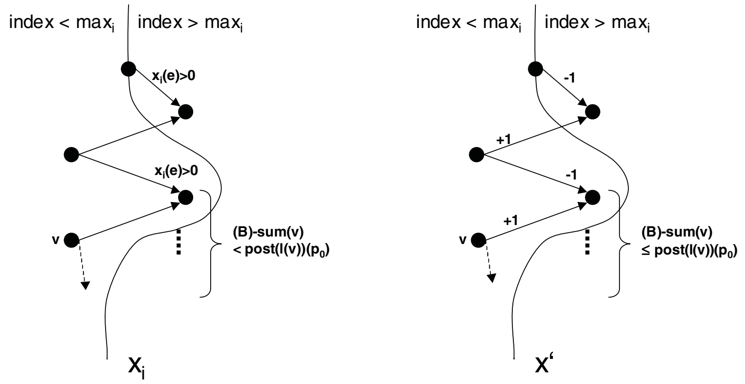

Fig. 4. The left part shows an example of a flow decreasing sequence of the second kind with $k = 2$. The right part shows the modifying operation to get a flow function x' from x_i satisfying $x'(v_{max_i}, w^1) < x_i(v_{max_i}, w^1)$, property (A) for all indexes and property (B) for all indexes smaller than max_i

- $w^j \neq w^m$ and $v^j \neq v^m$ for $j \neq m$,
- $x(v^j, w^{j+1}) > 0$ and $v^j < w^j$,
- For each $0 < j \leqslant k$ there is an index $m < max_i$ with $v^j = v_m$,
- $\sum_{v_k < v'} x_i(v_k, v') < W((l(v_k), p))$.

Such a sequence allows to modify x_i into a new flow function x' defined as follows:

$$x'(v^j, w^j) = x_i(v^j, w^j) + 1,$$
$$x'(v^j, w^{j+1}) = x_i(v^j, w^{j+1}) - 1.$$

Obviously x' satisfies $x'(v_{max_i}, w^1) < x_i(v_{max_i}, w^1)$, property (A) for all indexes and property (B) for all indexes smaller than max_i. This modification can be applied for each such sequence as long as $x'(v^j, w^{j+1}) > 0$ for all j and $\sum_{v_k < v'} x'(v_k, v') < W((l(v_k), p))$, thus reducing the (B)-sum of x_i w.r.t. max_i. As a consequence $x'(v^k, w^k) > x_i(v^k, w^k)$, i.e. the (B)-sum w.r.t. v^k is increased. Nevertheless property (B) remains satisfied for all indexes smaller than max_i.

Such sequences (of *the first or second kind*) will be called *flow decreasing sequences w.r.t. x_i)*. We want reduce the (B)-sum of the modified flow w.r.t. max_i by flow decreasing sequences in a maximal way. This can be done by transforming this problem for $(V, <, l)$ w.r.t. the flow function x_i into a *maximum flow problem* for a suitable *flow network*. The maximum flow problem is intensively studied since four decades and several algorithms running in cubic time in the number of nodes ([7]) and faster (see e.g. [3] for an overview) were developed.

4.3 The Associated Flow Network

We briefly introduce the necessary notations:

Definition 15 (Flow network). *A* flow network *is a directed graph* $(V', \to)$ *together with a capacity function* c *assigning nonnegative integers to edges in* $E_\to = \{(v, v') \mid v \to v' \vee v' \to v\}$*, satisfying: there is a node* $s \in V$*, called* source*, with no incoming edges w.r.t.* $\to$*, there is a node* t*, called* sink*, with no outgoing edges w.r.t.* $\to$*, and* $c(v, v') = 0$ *for* $v \not\to v'$*.*

A flow f *in* $(V', \to, c)$ *is a function assigning integers to edges in* $E_\to$ *in such a way that*

- *f does not exceed c:* $f(v, v') \leqslant c(v, v')$.
- $f(v, v') = -f(v', v)$.
- *For each node v except source and sink the flow into (resp. out of) v equals* 0*:* $\sum_{(v',v) \in E_\to} f(v', v) = 0$.

The value $|f|$ *of a flow f is defined as the outgoing flow of the source (or equivalently the ingoing flow of the sink)* $\sum_{s \to v'} f(s, v')$*. The* maximal flow *is the flow with maximal value among all flows.*

Given a flow f, the residual capacity c_r w.r.t. f *of* $(v, v') \in E_\to$ *is defined by* $c_r(v, v') = c(v, v') - f(v, v')$ *if* $v \to v'$ *and* $c_r(v, v') = f(v', v)$ *if* $v \not\to v'$*. The* residual network *of a flow f consists of all edges* $e \in E_\to$ *with* $c_r(e) > 0$ *together with the residual capacity. A* flow augmenting path w.r.t. a flow f *of a flow network is a simple path from source to sink in the residual network of* f*.*

One of the first algorithms solving the maximum flow problem was the flow augmenting path method by Ford and Fulkerson ([2]). They proved the following theorem giving a characterization of the maximum flow:

Theorem 2. *Let f be a flow in a flow network. If there is no flow augmenting path w.r.t. f, then f is maximal.*

The flow network $G(x_i) = (V(x_i), \to, c)$ associated to $lpo = (V, <, l)$ and x_i is defined in such a way that the flow decreasing sequences in $(V, <)$ (w.r.t. x_i) will correspond to flow augmenting paths in $(V(x_i), \to, c)$ (w.r.t. to the zero flow). The possibility of reducing the (B)-sum w.r.t. max_i through a flow decreasing sequence by a certain amount shall exactly correspond to a flow of the same amount through an associated augmenting path from source to sink. Therefore the capacity restricting the flow on edges (v^j, w^{j+1}) corresponds to the value of the flow function $x_i(v^j, w^{j+1})$. Since a node v_m with $m < max_i$ can serve as a node v^j in a flow decreasing sequence and as a node w^j in the same or another flow decreasing sequence, we split each node $v \in V$ into two nodes in the flow network: (v, out), playing the role of a node v^j, and (v, in), playing the role of a node w^j. Formally, we define $V(x_i) = (V \times \{in, out\}) \cup \{t\})$ with a node $t \notin V$, which will serve as the sink. The node (v_{max_i}, out) will have no incoming edges and serve as the *source* of the flow network. We will use a constant M as an edge capacity which can not be exceeded by the value of the maximum flow of the flow network (see Figure 5)). Since the value of the maximal flow can never exceed the sum of capacities of the edges outgoing of the source, we set $M = \sum_{v_{max_i} \to v'} x_i(v_{max_i}, v')$.

(a) For $l > max_i$ and $x_i(v_{max_i}, v_l) > 0$: $(v_{max_i}, out) \to (v_l, in)$ and $c((v_{max_i}, out), (v_l, in)) = x_i(v_{max_i}, v_l)$.

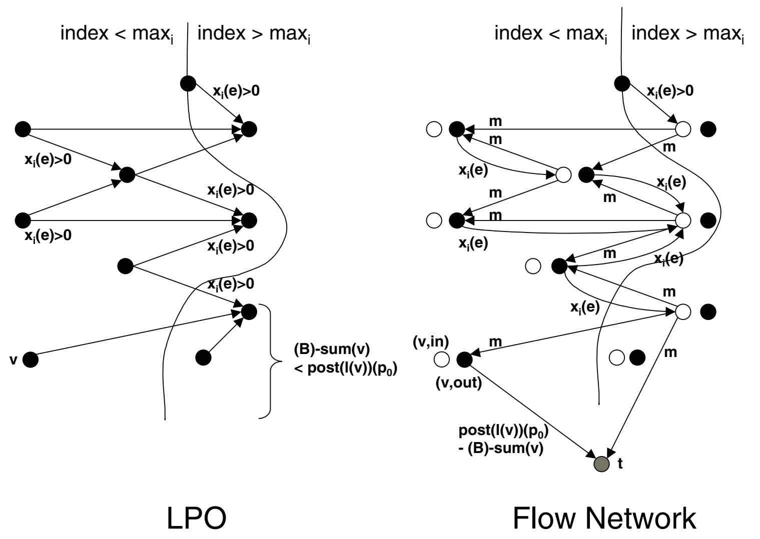

Fig. 5. The left part shows a part of a labelled partial order where it is indicated on which edges the flow function x_i has positive values. For clearness only the skeleton is regarded. The right part shows the corresponding part of the associated flow network. The "out"-nodes are filled whereas the "in"-nodes are not filled. Each edge carry its capacity

(b) For $j < max_i$ and $x_i(v_j, v_l) > 0$: $(v_j, out) \to (v_l, in)$ and $c((v_j, out), (v_l, in)) = x_i(v_j, v_l)$
(c) For $j < max_i$ and $v_j < v_l$: $(v_l, in) \to (v_j, out)$ and $c((v_l, in), (v_j, out)) = M$.
(d) For $l > max_i$ and $v_j < v_l$ for some v_j with $j > max_i$: $(v_l, in) \to t$ and $c((v_l, in), t) = M$
(e) For $j < max_i$ and $\sum_{v_j < v'} x_i(v_j, v') < W((l(v_j), p))$: $(v_j, out) \to t$ and $c((v_j, out), t) = W((l(v_j), p)) - \sum_{v_j < v'} x_i(v_j, v')$.

The following lemma states that for each flow f we can modify x_i into a flow function x_f fulfilling property (A) for all indexes and property (B) for all indexes smaller than max_i with its (B)-sum w.r.t. max_i reduced by $|f|$.

Lemma 4. *Let f be a flow in $G(x_i) = (V(x_i), \to, c)$. Then the flow function x_f, defined by modifying x_i in the following way, fulfils (A) for all indexes and (B) for all indexes smaller than max_i:*

(a) For $l > max_i$ and $x_i(v_{max_i}, v_l) > 0$: $x_f(v_{max_i}, v_l) = x_i(v_{max_i}, v_l) - f((v_{max_i}, out), (v_l, in))$.
(b) For $j < max_i$ and $x_i(v_j, v_l) > 0$: $x_f(v_j, v_l) = x_i(v_j, v_l) - f((v_j, out), (v_l, in))$.
(c) For $j < max_i$, $v_j < v_l$ and $x_i(v_j, v_l) = 0$: $x_f(v_j, v_l) = x_i(v_j, v_l) + f((v_l, in), (v_j, out))$

(d) For $l > max_i$ such that $v_j < v_l$ for some v_j with $j > max_i$:
Let J be the maximal index with $v_J < v_l$ and define $x_f(v_J, v_l) = x_i(v_J, v_l) + f((v_l, in), t)$.

Proof. Observe first that by construction all values of x_f are non-negative:
ad (a): $f((v_{max_i}, out), (v_l, in)) \leqslant c((v_{max_i}, out), (v_l, in)) = x_i(v_{max_i}, v_l)$.
ad (b): $f((v_j, out), (v_l, in)) \leqslant c((v_j, out), (v_l, in)) = x_i(v_j, v_l)$.
ad (c): $f((v_l, in), (v_j, out)) > 0$ since $c((v_j, out), (v_l, in)) = 0$.
ad (d): $f((v_l, in), t) > 0$ since $c(t, (v_l, in)) = 0$.

We first show that x_f fulfils part (A) of the flow property. For this we claim that the (A)-sums of x_i and x_f are equal w.r.t. each node v_l. For convenience assume $f(\nu, \mu) = 0$ for $(\nu, \mu) \notin E_\rightarrow$. The argumentation is based on the observation, that by definition for a node v_l with $(t, (v_l, in)) \notin E_\rightarrow$:

$$\sum_{v_j < v_l} f((v_j, out), (v_l, in)) = \sum_{(\nu, (v_l, in)) \in E_\rightarrow} f(\nu, (v_l, in)) = 0.$$

The last equality follows by the definition of flows. In this case:

$$\begin{aligned}\sum_{v_j < v_l} x_f(v_j, v_l) &= \sum_{v_j < v_l} (x_i(v_j, v_l) - f((v_j, out), (v_l, in)))\\ &= \sum_{v_j < v_l} x_i(v_j, v_l) - \sum_{v_j < v_l} f((v_j, out), (v_l, in))\\ &= \sum_{v_j < v_l} x_i(v_j, v_l).\end{aligned}$$

In case $(t, (v_l, in)) \in E_\rightarrow$ we similarly get the same result.

Finally we show that x_f fulfils part (B) of the flow property for all indexes $j < max_i$. If $(t, (v_j, out)) \notin E_\rightarrow$, we deduce as above that the (B)-sums of x_i and x_f are equal w.r.t. v_j. In the case $(t, (v_j, out)) \in E_\rightarrow$ we get:

$$\begin{aligned}\sum_{v_j < v_l} f((v_j, out), (v_l, in)) &= \left(\sum_{((v_j, out), \mu) \in E_\rightarrow} f((v_j, out), \mu) \right) - f((v_j, out), t)\\ &= -f((v_j, out), t).\end{aligned}$$

As above we deduce

$$\begin{aligned}\sum_{v_j < v_l} x_f(v_j, v_l) &= \sum_{v_j < v_l} x_i(v_j, v_l) - \sum_{v_j < v_l} f((v_j, out), (v_l, in))\\ &= \sum_{v_j < v_l} x_i(v_j, v_l) + f((v_j, out), t)\\ &\leqslant \sum_{v_j < v_l} x_i(v_j, v_l) + c((v_j, out), t) = W((l(v_j), p)).\end{aligned}$$

□

4.4 The Main Procedure

By Lemma 4, for each flow f in the associated flow network we can decrease the (B)-sum for the index max_i by $|f|$, while (A) for all indexes and (B) for all indexes smaller then max_i remain satisfied. Thus, the main procedure is defined as follows:

– Input: Flow function x_i.
– Compute associated flow network w.r.t. x_i.
– Compute maximal flow f in this flow network.
– Compute x_f.
– Output: Flow function $x_{i+1} = x_f$.

Computing the maximal flow f depends on the applied maximum flow algorithm. As already mentioned, there are maximum flow algorithms taking cubic time and faster. The other steps take at most quadratic time. Altogether the main procedure takes at most cubic time.

Let us mention that the main procedure could be optimized by terminating the maximum flow algorithm as soon as $\sum_{v_{max_i} < v'} x_i(v_{max_i}, v') - |f| \leqslant W((l(v_{max_i}), p))$.

4.5 Termination of the Algorithm

If the algorithm terminates because x_i fulfils (B), then *lpo* fulfils the flow property for the place p.

It remains to prove that if the algorithm terminates because $max_i = max_{i-1}$, then *lpo* is not enabled w.r.t. (N, m_0). From proposition 3 this implies that *lpo* is not executable in (N, m_0).

Theorem 3. *Let f be the maximal flow of the associated flow network w.r.t. x_i. Assume moreover that x_f does not fulfil (B) for the index max_i. Then there is a cut C, such that lpo is not enabled w.r.t. C:*

$$m_0(p) + \sum_{v < C, v \in V} W((l(v), p)) - \sum_{v < C} W((p, l(v))) - \sum_{v \in C} W((p, l(v))) < 0.$$

The proof is based on the following lemma which states that for each flow f and each flow decreasing sequence w.r.t. x_f there is a flow augmenting path w.r.t. f in the flow network associated to x_i.

Lemma 5. *Let f be a flow in the flow network $G(x_i) = (V(x_i), \rightarrow, c)$ and let $v^0 = v_{max_i}, w^1, v^1, \ldots, w^k, v^k$ be a flow deceasing sequence w.r.t. x_f. Then*

– *If the flow decreasing sequence is of the first kind, then $(v^0, out) = (v_{max_i}, out), (w^1, in), (v^1, out), \ldots, (w^k, in), t$ is a flow augmenting path w.r.t. f.*
– *If the flow decreasing sequence is of the second kind, then $(v^0, out) = (v_{max_i}, out), (w^1, in), (v^1, out), \ldots, (w^k, in), (v^k, out), t$ is a flow augmenting path w.r.t. f.*

Proof. Let $v_{max_i}, w^1, v^1, \ldots, w^k, v^k$ be a flow decreasing sequence of the first kind. We have to show that $(v_{max_i}, out), (w^1, in), (v^1, out), \ldots, (w^k, in), t$ is a path in the residual network $(V(x_i), \rightarrow_r, c_r)$ of f. We have

- $c_r((w^j, in), (v^j, out)) = c((w^j, in), (v^j, out)) - f((w^j, in), (v^j, out)) = M - f((w^j, in), (v^j, out)) > 0$ since $(w^j, in) \rightarrow (v^j, out)$.
- $c_r((v^j, out), (w^{j+1}, in)) = c((v^j, out), (w^{j+1}, in)) - f((v^j, out), (w^{j+1}, in)) = x_i((v^j, out), (w^{j+1}, in)) - f((v^j, out), (w^{j+1}, in)) = x_f(v^j, w^{j+1}) > 0$.
- $c_r((w^k, in), t) = c((w^k, in), t) - f((w^k, in), t) = M - f((w^k, in), t) > 0$.

Let now $v_{max_i}, w^1, v^1, \ldots, w^k, v^k$ be a flow decreasing sequence of the second kind. We show that $(v_{max_i}, out), (w^1, in), (v^1, out), \ldots, (w^k, in), (v^k, out), t$ is a path in the residual network of f.

To show $c_r((w^j, in), (v^j, out))$, $c_r((v^j, out), (w^{j+1}, in)) > 0$ works as above. Moreover, we get analogously as in the proof of lemma 4:
$c_r((v^k, out), t) = c((v^k, out), t) - f((v^k, out), t) = W((l(v^k), p)) - \sum_{v^k < v'} x_i(v^k, v')$ $- f((v^k, out), t) = W((l(v^k), p)) - \sum_{v^k < v'} x_f(v^k, v') > 0$. □

From lemma 5 and theorem 2 we get immediately that for the maximal flow f there is no flow decreasing sequence w.r.t. x_f.

Lemma 6. *Let f be the maximal flow of the network $G(x_i) = (V(x_i), \rightarrow, c)$ associated to x_i. Assume further x_f does not fulfil property (B) for the index max_i.*

Define $\mathcal{W}$ as the set consisting of v_{max_i} and all events $v \in V^0$ such that there exists a sequence $v^0 = v_{max_i}, w^1, v^1, \ldots, w^k, v^k = v$ with

(i) $w^j \neq w^m$ and $v^j \neq v^m$ for $j \neq m$,
(ii) $x_f(v^j, w^{j+1}) > 0$ and $v^j <^0 w^j$.

Define C as the set of all $w \in V$ with $w \notin \mathcal{W}$ such that there exists $v \in \mathcal{W}$ with $x_f(v, w) > 0$. Then it holds:

(a) If $v_j \in \mathcal{W}$ then $j \leqslant max_i$.
(b) $v <^0 C \Leftrightarrow v \in \mathcal{W}$.
(c) C is a co-set.
(d) $C' = \{w \in V \mid w \notin \mathcal{W} \wedge (v <^0 w \Rightarrow v \in \mathcal{W})\}$ is a cut and $C \subseteq C'$.
(e) For every $v_j \in \mathcal{W}$ with $j \neq max_i$ we have $\sum_{v_j <^0 v'} x_f(v_j, v') = W((l(v_j), p))$.

Proof. ad (a): Assume $v_j \in \mathcal{W}$ with $j > max_i$. There is a sequence $v^0 = v_{max_i}, w^1, v^1, \ldots, w^k, v^k = v_j$ fulfilling (i) and (ii). Take the smallest index m such that there is $u > max_i$ with $v^m = v_u$. By definition $v^0 = v_{max_i}, w^1, v^1, \ldots, w^m, v^m$ is a flow decreasing sequence of the first kind. This contradicts the assumption (since by Lemma 5 there is no flow decreasing sequence w.r.t. x_f).

ad (b): $(\Longrightarrow)$ Let $v <^0 C$. If $v = v_{max_i}$, then $v \in \mathcal{W}$ by definition of $\mathcal{W}$. Let $v \neq v_{max_i}$: $v <^0 C$ implies that there is $w \in C$ with $v <^0 w$. By definition of C there exists $v' \in \mathcal{W}$ with $x_f(v', w) > 0$. If $v = v'$, we get $v \in \mathcal{W}$. Let $v \neq v'$. If $v' = v_{max_i}$, then the sequence v', w, v fulfils (i) and (ii) and therefore $v \in \mathcal{W}$. Otherwise by definition of $\mathcal{W}$ there is a sequence $v^0, w^1, v^1, \ldots, w^k, v^k = v'$ fulfilling

(i) and (ii). If $v = v^j$ for some j, then $v \in \mathcal{W}$. Let $v \neq v^j$ for all j. If $w^j \neq w$ for all j, then also the sequence $v^0, w^1, v^1, \ldots, w^k, v', w, v$ fulfils (i) and (ii), i.e. $v \in \mathcal{W}$. If $w^j = w$ for some j, let m be the smallest index with $w^m = w$. Then the sequence $v^0, w^1, v^1, \ldots, w^m = w, v$ fulfils (i) and (ii), i.e. $v \in \mathcal{W}$.

($\Longleftarrow$) Let $v \in \mathcal{W}$. If $v = v_{max_i}$ then we have: Because x_f does not fulfil (B) for the index max_i, there exists a node $v_j \in V$ with $j > max_i$ such that $x_f(v, v_j) > 0$ and in particular $v <^0 v_j$. According to (a) and the definition of C we have $v_j \in C$ and $v <^0 C$ (in particular C is nonempty). If $v \neq v_{max_i}$ then there is a sequence $v^0, w^1, v^1, \ldots, w^k, v^k = v$ fulfilling (i) and (ii). Since $v^{k-1} \in \mathcal{W}$, we get $x_f(v^{k-1}, w^k) > 0$. If $w^k \notin \mathcal{W}$, this implies $w^k \in C$. From $v = v^k < w^k$ we obtain $v <^0 C$. If $w = w^k \in \mathcal{W}$, there is a sequence $v^0, w^1, v^1, \ldots, w^l, v^l = w$ fulfilling (i) and (ii). From $v <^0 w$ and $w <^0 w^l$ we obtain $v <^0 w^l$. Again if $w^l \notin \mathcal{W}$, we are done, else repeat this procedure. Obviously, this procedure terminates according to the the fact that V is finite and for every $v \in \mathcal{W}$ there is a $w \in V$ satisfying $v <^0 w$, i.e. $\mathcal{W}$ does not contain maximal elements w.r.t. partial order $<^0$.

ad (c): Assume two events $w, w' \in C$ with $w < w'$. From (b) we obtain $w \in \mathcal{W}$, which is a contradiction to $w \in C$.

ad (d): From (b) and (c) we get that $\mathcal{W}$ is a downward closed. From definition of $\mathcal{W}$ we have that it does not contain maximal elements. It is a well known fact that then the set C' is a cut. From (b) we also get $C \subset C'$.

ad (e): Assume $v_j \in \mathcal{W}$ with $j \neq max_i$ and $\sum_{v_j <^0 v'} x_f(v_j, v') \neq W((l(v_j), p))$. From (a) we get $j < max_i$. Since x_f fulfils (B) for all indexes smaller than max_i, this implies $\sum_{v_j <^0 v'} x_f(v_j, v') < W((l(v_j), p))$. Let $v^0, w^1, v^1, \ldots, w^k, v^k = v_j$ be a sequence fulfilling (i) and (ii). By the above consideration this sequence is a flow decreasing sequence of the second kind. This is a contradiction to the assumption. □

Lemma 7. *Let f be the maximal flow of the associated flow network $G(x_i)$ w.r.t. x_i and assume x_f does not fulfil (B) for the index max_i. Then C' is a cut satisfying*

$$m_0(p) + \sum_{v < C', v \in V} W((l(v), p)) - \sum_{v < C'} W((p, l(v))) - \sum_{v \in C'} W((p, l(v))) < 0.$$

Proof. We first prove the inequality for the co-set C. Since x_f fulfils (A) we can replace $W((p, l(v)))$ by $\sum_{v' <^0 v} x_f(v', v)$ for each $v <^0 C$ and $v \in C$. Because x_f fulfils statement (b) of the last lemma we can replace $W((l(v), p))$ by $\sum_{v <^0 v'} x_f(v, v')$ for each $v <^0 C$, $v \neq v_{max_i}$. Moreover $m_0(p)$ equals $\sum_{v_0 <^0 v'} x_f(v_0, v')$. Finally we can use $W((l(v_{max_i}), p)) < \sum_{v_{max_i} <^0 v'} x_f(v_{max_i}, v')$.

Altogether it is enough to show that

$$\sum_{v <^0 C} \sum_{v < v'} x_f(v, v') - \sum_{v <^0 C} \sum_{v' <^0 v} x_f(v', v) - \sum_{v \in C} \sum_{v' <^0 v} x_f(v', v) = 0.$$

We claim that in this sum each value $x_f(v, v')$ equals either 0 or is counted once positively and once negatively. The second alternative is obviously fulfiled if $v' < C$

or $v' \in C$. Observe now that for $v < C$ and $x_f(v, v') > 0$ we get by the definition of W and C that $v' <^0 C$ or $v' \in C$. That means if $x_f(v, v')$ does not fulfil the first alternative, it fulfils the second alternative.

Observe $v <^0 C \Leftrightarrow v <^0 C'$. That means, replacing C by C' in the above sum could only change the value of the third sum, namely by values $x_f(v', v)$ with $v \in C' \setminus C$. These values are equal to 0 by the definition of C. □

5 Strict Executability

In this section we briefly outline a polynomial test for the strict executability of an LPO. The flow property is a necessary and sufficient condition of the executability of a labelled partial order. We extend the flow property to get a necessary and sufficient condition for an LPO to be exactly a run of a marked p/t-net.

Given a partial order $(V, <)$, let $\sqsubset \subseteq <$ be the skeleton of $<$, i.e. the smallest subset of $<$ which fulfils: $\sqsubset^+ = <$.

Definition 16 (Strong flow property). *A labelled partial order* $(V, <, l)$ *fulfils the* strong flow property w.r.t. (N, m_0) *if there exists a family* $X = \{x_p \mid p \in P\}$ *of flows fulfilling the flow property which satisfies: the skeleton* $\sqsubset$ *of* $<$ *is a subset of the relation* $Q_X = \{(v, v') \in V \times V \mid \exists p \in P : x_p(v, v') > 0\}$.

One can easily check that the canonical flow of a run fulfils the strong flow property. Observe that the flow functions of a labelled partial order which fulfil the flow property used for construction of the process in the proof of lemma 3 are the canonical flow functions of the underlying run of the constructed process. Thus, we obtain:

Theorem 4. *A labelled partial order fulfils the strong flow property w.r.t.* (N, m_0) *if and only if it is a run of* (N, m_0).

By Definition 10, the executability of an LPO is a necessary condition for its strict executability. Now, take an LPO $lpo = (V, <, l)$ which is executable in a marked p/t net (N, m_0), i.e. which fulfils the flow property w.r.t. (N, m_0). Denote $X = \{x_p \mid p \in P\}$ the family of flows fulfilling the flow property computed by the algorithm presented in the previous section. The following algorithm decides whether lpo is strictly executable w.r.t. (N, m_0), i.e. whether it is a minimal run of (N, m_0). If the answer is positive, then X is the family of canonical flows of the underlying minimal run.

- Compute skeleton $\sqsubset$ of the partial order $(V, <)$;
- if $\sqsubset \not\subseteq Q_X$ then return "lpo is not strictly executable";
- else for each edge $e \in \sqsubset$: test whether $(V, < \setminus \{e\}, l)$ is executable; if yes, return "lpo is not strictly executable";
- return "lpo is strictly executable".

Due to lack of space, we omit a detailed proof of the correctness of the algorithm and just give some intuition: Observe that after removing a skeleton edge from a partial order $<$ one still gets a partial order. Moreover, any proper subset of a partial order $<$, which is itself a partial order, is a subset of a partial order obtained from $<$ by removing a skeleton edge. Thus, if the algorithm returns "lpo is not strictly executable", then a run was computed which is a proper subset of lpo, i.e. lpo cannot equal a minimal run. If the algorithm returns "lpo is strictly executable", then lpo fulfils the strong flow property and therefore it is a run, and all LPOs which are proper subsets of lpo are not executable, i.e. do not include a run. That means lpo does not properly include any run and therefore it is a minimal run. Obviously the algorithm runs in polynomial time.

6 Conclusion

We have presented a polynomial algorithm for testing whether an LPO is executable in a place/transition net while preserving at least the given amount of concurrency, or in other words, adding no causality (i.e. whether it is a sequentialization of a run). Further, we have formulated a polynomial test deciding whether an LPO is strictly executable, i.e. whether the given amount of concurrency is maximal, or complementary, whether the amount of causality is minimal (i.e. whether the LPO equals a minimal run). It is a question of further research to determine efficiently whether an LPO is executable preserving exactly the given amount of concurrency and causality, i.e. whether it equals a (not necessarily minimal) run. Another interesting question is generalization of so called "legal firing sequence" problem, namely to determine whether an LPO can be executed while preserving at least the given amount of causality (adding no concurrency). Finally, one could combine both approaches into a scenario based specification prescribing minimal/maxiamal level of causality and concurrency.

References

1. J. Desel and W.Reisig. Place/Transition Petri Nets. In *Lectures on Petri nets I: Basic Models*, LNCS 1491, pp. 123–174,1998.
2. L.R. Ford, Jr. and D.R. Fulkerson. Maximal Flow Through a Network. *Canadian Journal of Mathematics* 8, pp. 399–404, 1955.
3. A. Goldberg and S. Rao. Beyond the Flow Decomposition Barrier. *Journal of the ACM* 45/5, pp. 783–797, 1998.
4. U. Goltz and W. Reisig. The Non-Sequential Behaviour of Petri Nets. *Information and Control*, 57(2-3), pp. 125-147, 1983.
5. U. Goltz and W. Reisig: Processes of Place/Transition Nets. - Proc. of ICALP'83, LNCS 154, pp. 264-277, 1983.
6. J. Grabowski. On Partial Languages. *Fundamenta Informaticae* IV.2, pp. 428–498, 1981.
7. A.V. Karzanov. Determining the Maximal Flow in a Network by the Method of Preflows. *Soviet Math. Doc.* 15, pp. 434–437, 1974.
8. A. Kiehn. On the Interrelationship between Synchronized and Non-Synchronized Behavior of Petri Nets. *Journal Inf. Process. Cybern. EIK* 24, pp. 3 – 18, 1988.
9. V. Pratt. Modelling Concurrency with Partial Orders. *Int. Journal of Parallel Programming* 15, pp. 33–71, 1986.
10. W. Vogler. Modular Construction and Partial Order Semantics of Petri Nets. *LNCS* 625, 1992.
11. W. Vogler. Partial words versus processes: a short comparison, Advances in Petri Nets, LNCS 609, Springer 1992, pp. 292-303.