Unifying Petri Net Semantics with Token Flows

Gabriel Juhás[1], Robert Lorenz[2], and Jörg Desel[3]

[1] Faculty of Electrical Engineering and Information Technology
Slovak University of Technology, Bratislava, Slovakia
gabriel.juhas@stuba.sk
[2] Department of Computer Science, University of Augsburg, Germany
robert.lorenz@informatik.uni-augsburg.de
[3] Department of Applied Computer Science
Catholic University of Eichstätt-Ingolstadt, Germany
joerg.desel@ku-eichstaett.de

Abstract. In this paper we advocate a unifying technique for description of Petri net semantics. Semantics, i.e. a possible behaviour, is basically a set of node-labelled and arc-labelled directed acyclic graphs, called token flows, where the graphs are distinguished up to isomorphism. The nodes of a token flow represent occurrences of transitions of the underlying net, so they are labelled by transitions. Arcs are labelled by multisets of places. Namelly, an arc between an occurrence x of a transition a and an occurrence y of a transition b is labelled by a multiset of places, saying how many tokens produced by the occurrence x of the transition a is consumed by the occurrence y of the transition b. The variants of Petri net behaviour are given by different interpretation of arcs and different structure of token flows, resulting in different sets of labelled directed acyclic graphs accepted by the net. We show that the most prominent semantics of Petri nets, namely processes of Goltz and Reisig, partial languages of Petri nets introduced by Grabowski, rewriting terms of Meseguer and Montanari, step sequences as well as classical occurrence (firing) sequences correspond to different subsets of token flows. Finally, we discuss several results achieved using token flows during the last four years, including polynomial test for the acceptance of a partial word by a Petri net, synthesis of Petri nets from partial languages and token flow unfolding.

1 Introduction

Let us begin with a short story: An alien from a foreign planet comes with his UFO and observes the following situation. He sees a man pressing a button of his mobile phone, then the man is starting to call and after that sitting down. Actually, the man is totally surprised because he had never seen a UFO and an alien, so he immediately calls his wife and sits down to realize what he just saw. The alien writes a report to his planet: ”On Earth, if you want to sit down, you first have to press a button on a mobile phone and then to start to talk to the phone.” As we all know, this is not exactly true, because you can sit down independently (concurrently) from pressing a button and starting to call. But it is still partially true, because in order to call you have to press a button of your phone (or to do something equivalent).

In the case of a sequential machine with a single processor, it is not necessary to deal with the problem of independency of events or actions. One can at most observe non-determinism, e.g. observe sitting down and calling in any order. But concurrency substantially differs from non-determinism. Concurrency is not only the possibility to occur in any order, but to be independent of each other. A typical example making this difference clear, is the occurrence of two events (e.g. two persons wanting to call) sharing one resource (only one phone available). These two events can occur in any order but they obviously cannot happen in parallel. One could say that concurrency includes possible occurrence in any order and simultaneous occurrence. The introducing example also shows that concurrency is not only simultaneous occurrence, which is transitive: a man can sit down concurrently to pressing a button and concurrently to starting to call. However, pressing a button and starting to call are not concurrent, but causally dependent. Observe that the action *calling* in the short story causally depends on the action *pressing a button*, but we can make more calls and pressing buttons many times. If we say that *calling* is an action and *pressing a button* is another action, we speak about causal dependencies between *occurences* of actions rather than about causal dependencies between actions alone.

The study of concurrency as a phenomenon of systems behavior became much attention in recent years, because of an increasing number of distributed systems, multiprocessors systems and communication networks, which are concurrent in their nature. There are many ways in the literature to describe non-sequential behaviour, most of them are based on directed acyclic graphs (DAGs). Usually, nodes of DAGs represent the occurrences of actions, i.e. they are labelled by the set of actions and the labelled DAGs (LDAGs) are distinguished up to isomorphism. Such LDAGs are reffered in the literature as abstract [16]. Very often LDAGs with the transitive arcs are used, i.e. the partial orders. Such structures are referred as partially ordered multisets, shortly pomsets, and can formally be seen as isomorphism classes of labelled partial orders [18]. Pomsets are also called partial words [9], emphasizing their close relation to words or sequences; the total order of elements in a sequence is replaced by a partial order.

Petri nets are one of the most prominent formalisms for both understanding the concurrency phenomenon on theoretical and conceptual level and for modelling of real concurrent systems in many application areas. There are many reasons for that, among others the combination of graphical notation and sound mathematical description, see e.g. [5] for a more detailed discussion.

A place/transition Petri net (shorty a Petri net), consists of a set of transitions (actions), which can occur (fire) and a set of places (buffers of tokens), which can carry a number of tokens. Distribution of tokens in places constitutes a state of a Petri net, called a marking. Formally, a marking is given by a multiset of places, i.e. by a function attaching to each place the number of tokens contained in the place. An occurrence of a transition in a marking removes prescribed fixed numbers of tokens from places, i.e. consume a multiset of places, and adds prescribed fixed numbers of tokens to places, i.e. produce a multiset of places, resulting in a new marking. A transition is enabled to occur in a marking, if there is enouhg number of tokens to consume by an occurrence of the transition. In the paper, occurrences of transitions will be referred as events. We also consider a fixed initial marking and a set of legal final markings.

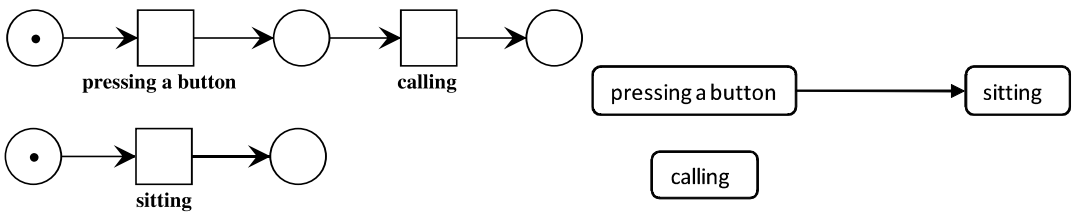

Fig. 1. A Petri net modelling the introductory story (left). To describe the independency relation we need at least two steps sequences: the sequence with step "pressing a button and sitting" followed by the step "calling" and the sequence with the step "pressing a button" followed by the step "sitting and calling". The underlying pomset of the behaviour (right).

There are many different ways how to define behaviour of Petri nets. The simplest way is to take occurrence sequences, i.e. sequences of occuring transitions.

Another possibility is to extend the sequences of occuring transitions to sequences of steps of transitions. Steps are multisets of transitions. A step is enabled to occur in a marking if there is enough tokens to consume by the simultaneous occurrences of transitions in the multiset. A situation described in the introductory example, where independence relation of transition occurrences is not transitive, cannot be decribed by a single step sequence, see Figure 1.

Therefore, pomsets seems to be a better choice to formalize non-sequential semantics (see e.g. [18,9]. The natural question arises: which pomsets do express behaviour of a Petri net? The answer has close relationships to the step semantics. In [9,12] it is suggested to take pomsets satisfying: For each co-set of events (i.e. for each set of unordered events) there holds: The step of events in the co-set is enabled to occur in the marking reached from the initial marking by occurrence of all events smaller than an event from the co-set.

Another possibility to express behaviour of Petri nets, is to take processes of [7,8], which are a special kind of acyclic nets, called occurrence nets, together with a labelling which associates the places (called conditions) and transitions (called events) of the occurrence nets to the places and transitions of the original nets preserving the number of consumed and produces tokens, see Figure 2. Processes can be understood as (unbranched) unfoldings of the original nets: every event in the process represents an occurrence of its label in the original net. Abstracting from conditions of process nets, LDAGs on events are defined. These LDAGs express the direct causality between events. Adding transitive arcs to these LDAGs, we get pomsets called runs, which express (not necessarily direct) causality between events. In contrast to enabled pomsets, events ordered by an arc in a run cannot be independent. A special role plays those runs, which are minimal w.r.t. extension: they express the minimal causality between events. An important result relating enabled pomsets and runs was proven in [12,19]: Every enabled pomset includes a run and every run is an enabled pomset. Therefore, minimal enabled pomsets equal minimal runs. In contrast to sequential semantics and step semantics, processes distinguish between the history of tokens. An example is shown in Figure 2. The process nets distinguish a token in place p_3 produced by the occurrence of transition t_1 from a token in place p_3 produced by the occurrence of transition t_2. As a consequence, one occurrence sequence, e.g. $t_1t_2t_3$, can be an extension of two

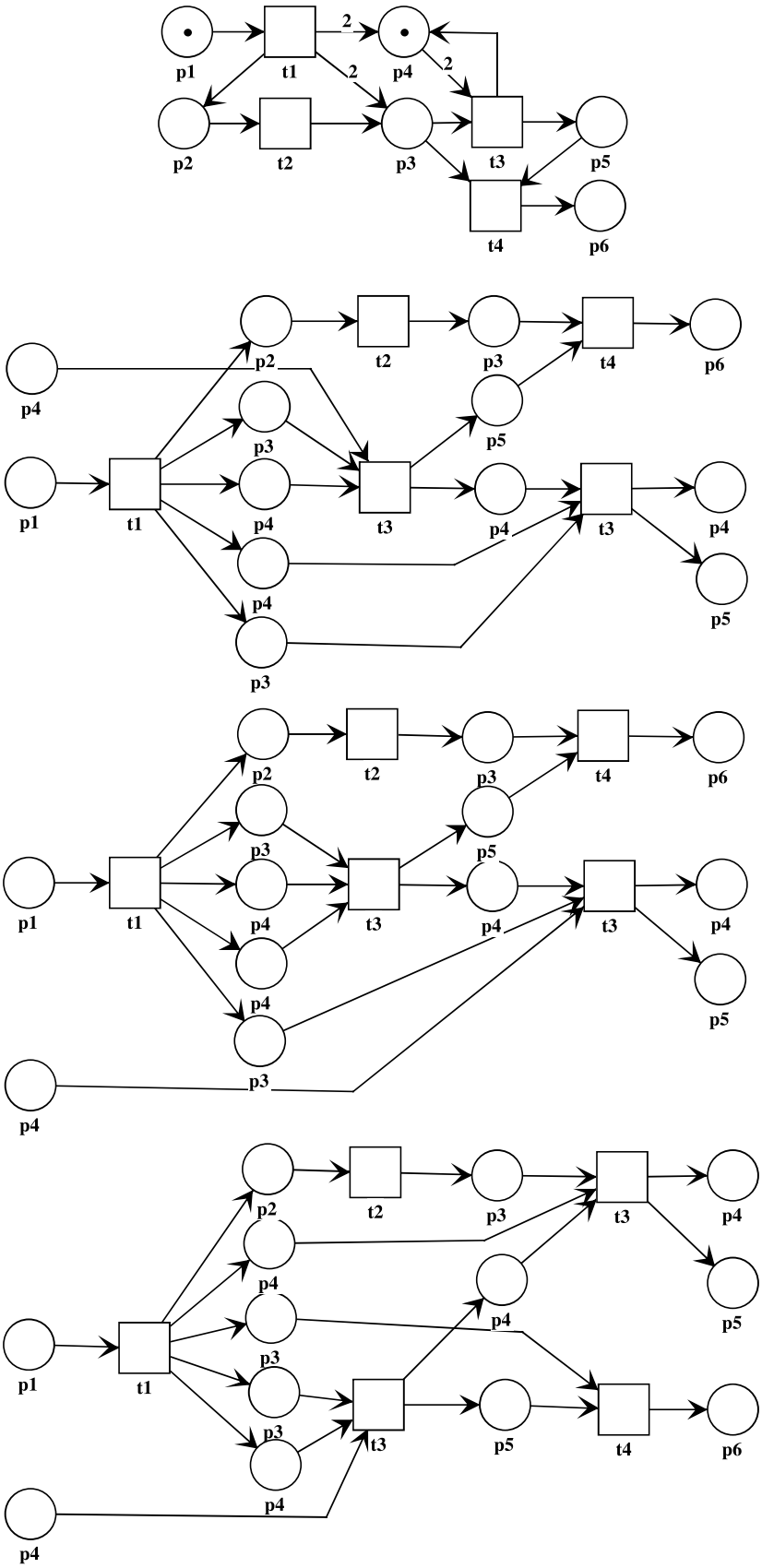

Fig. 2. A Petri net (above) with three processes (below)

different processes. The process semantics defined in [7] is also called individual token semantics. Notice that in the case of the process semantics of elementary nets (with at most one token in a place), any occurrence sequence and any step sequence uniquely determine a process. In [3] the collective token semantics, which does not distinguish between the history of tokens, is introduced. It is defined using an equivalence relation between processes. The equivalence relates processes differing only in permuting (swapping) unordered conditions representing tokens in the same place of the original net. For example, the processes in Figure 2 are equivalent w.r.t. swapping equivalence. For swapping equivalence classes, called commutative processes, there holds that any occurrence sequence and any step sequence uniquely determine a commutative process.

In [15] behaviour is described using rewrite terms generated from elementary terms using concurrent and sequential composition. In this algebraic approach any transition t is an elementary rewrite term, allowing to replace the marking $consume(t)$ by the marking $produce(t)$. Any marking consisting of a single place p is an elementary term rewriting p by p itself. Rewrite terms are constructed inductively from elementary terms using operator ; for sequential and operator $\|$ for concurrent composition. Each term has associated an initial marking and final marking. Two terms can be composed sequentially only if the final marking of the first term coincides with the initial marking of the second one. For concurrent composition of two terms, the initial marking of the resulting term is obtained by multiset addition of the initial markings of the composed terms, and likewise for the final marking. The behavior of a net is given by equivalence classes of rewrite terms defined by a set of equations. In [4] it is shown that the equivalence class of rewrite terms as defined in [15] corresponds to the swapping equivalence class of processes. Obviously, one can attach pomsets to rewrite terms. Each process term α defines a partially ordered set of events representing transition occurrences in an obvious way: an event e_1 is smaller than an event e_2 if the rewrite term α contains a sub-term $\alpha_1; \alpha_2$ such that e_1 occurs in α_1 and e_2 occurs in α_2. The pomsets of rewrite terms have a special structure. It is proven in [6], that a pomset is generated by concurrent and sequential composition from single element pomsets if and only if it does not contain the shape of so called N-form. As a consequence, we get the characterization of pomsets which are associated to rewrite terms of a Petri net: an enabed pomset is associated with a rewrite term of a net if and only if it is N-free.

Most of the discussed results showing the correspondence between different semantics are quite complicated, despite the clear intuition. The differences in the technique used to describe the behaviour often cause, that even straightforward relationships have technically difficult proofs. The formal description of the intuition chosen for single semantics has some limitations too. Consider for example enabled pomsets and processes. The definition of enabled pomsets is inherently exponential: to test the enabledness using co-sets and steps in innefective. The main advantages of the processes, as claimed in the literature, is that they describe behaviour in the same modelling language, i.e. using occurrence nets. But this is also their main disadvantage, because to model each individual token by a single condition make the formal manipulation with processes difficult. In fact, there is no need to remember each single token (it is suitable for elementary nets, where at most one token for each place can be present in a marking, but not so efficient for place/transition Petri nets), but it is enough to remember how

many tokens produced by an occurrence are consumed by another occurrence, i.e. how many tokens flow on the arcs connecting occurrences of transitions, abstracting from the individuality of conditions.

The natural question arise: Is there a possibility to express all semantics discussed above using a unifying but yet simple formal description? We give a positive answer to this question, presenting a framework for description of different variants of semantics of Petri nets. The semantics in the framework is basically a set of node-labelled and arc-labelled directed acyclic graphs, called token flows, where the graphs are distinguished up to isomorphism, defined originally in [10]. The nodes of a token flow represent occurrences of transitions of the underlying net, so they are labelled by transitions. Arcs are labelled by multisets of places. Namelly, an arc between an occurrence x of a transition a and an occurrence y of a transition b is labelled by a multiset of places, saying how many tokens produced by the occurrence x of the transition a is consumed by the occurrence y of the transition b. A token flow of a Petri net have to fulfill so called token flow property:

- Ingoing token flow (i.e. the sum of multisets over ingoing arcs) of any occurrence of a transition a equals the multiset of places consumed in the net by firing transition a.
- Outgoing token flow (i.e. the sum of multisets over outgoing arcs) of any occurrence of a transition a equals the multiset of places produced in the net by firing transition a.

To keep the information which occurrences consume tokens directly from the initial marking and which occurrences produce unconsumed tokens in the final marking, we add a special single entry node and a special single exit node, which are labelled by the initial and a final marking. respectively. The outgoing token flow of the entry equals the initial marking and the ingoing token flow of the exit equals a final marking. An important question arises, when expressing the behaviour by a set of LDAGs, i.e. by an LDAG language: What is the interpretation of arcs in an LDAG? The answer is not unique. Using Petri nets, there are two basic interpretations:

1. An occurrence y of an action b *directly causaly depends* on an occurrence x of an action a: By the occurrence y of a transition b, transition b consumes some tokens produced by the occurrence x of a. This interpretaton is used by processes of Petri nets [7,8]. Direct causal interpretation of arcs is resulting in the requirement that there is a non-zero token flow between the occurrences, i.e. the multiset label of any arc does not equal empty multiset. The transitive closure of direct causality gives causal dependency.
2. An occurrence y of an action b *follows* an occurrence x of an action a: The occurrence y of a transition b is either causaly dependent on the occurrence x or the occurrences y and x are independent. This interpretation is used in the occurrence sequences, step sequences and enabled pomsets of Petri nets [9]. We call this interpretation occurrence interpretation. By occurrence interpretation, also arcs with the zero token flow are allowed.

The variants of Petri net behaviour are given by different interpretation of arcs and different structure of token flows, resulting in different sets of labelled directed acyclic

graphs accepted by the net. We show that the most prominent semantics of Petri nets correspond to different subsets of token flows. Namely:

- Processes of Goltz and Reisig [7,8] correspond to *direct causal token flows*, i.e. to the token flows with direct causal interpretation of arcs, where all arcs have non-zero token flows.
- Token flows in which LDAGs are pomsets and at least the skeleton arcs (non-transitive arcs) have non-zero token flows are called the *causal token flows* of Petri nets and represent the causal semantics. They are obtained from direct causal token flows by adding all transitive arcs.
- Enabled pomsets, i.e. partial words introduced by Grabowski [9], correspond to *pomset token flows*, i.e. to the token flows, where LDAGs are pomsets with occurrence interpretation of arcs (arcs may have the zero token flow).
- Rewriting terms of Meseguer and Montanari [15] correspond to *N-free token flows*, i.e. to the pomset token flows where the underlying pomsets are N-free.

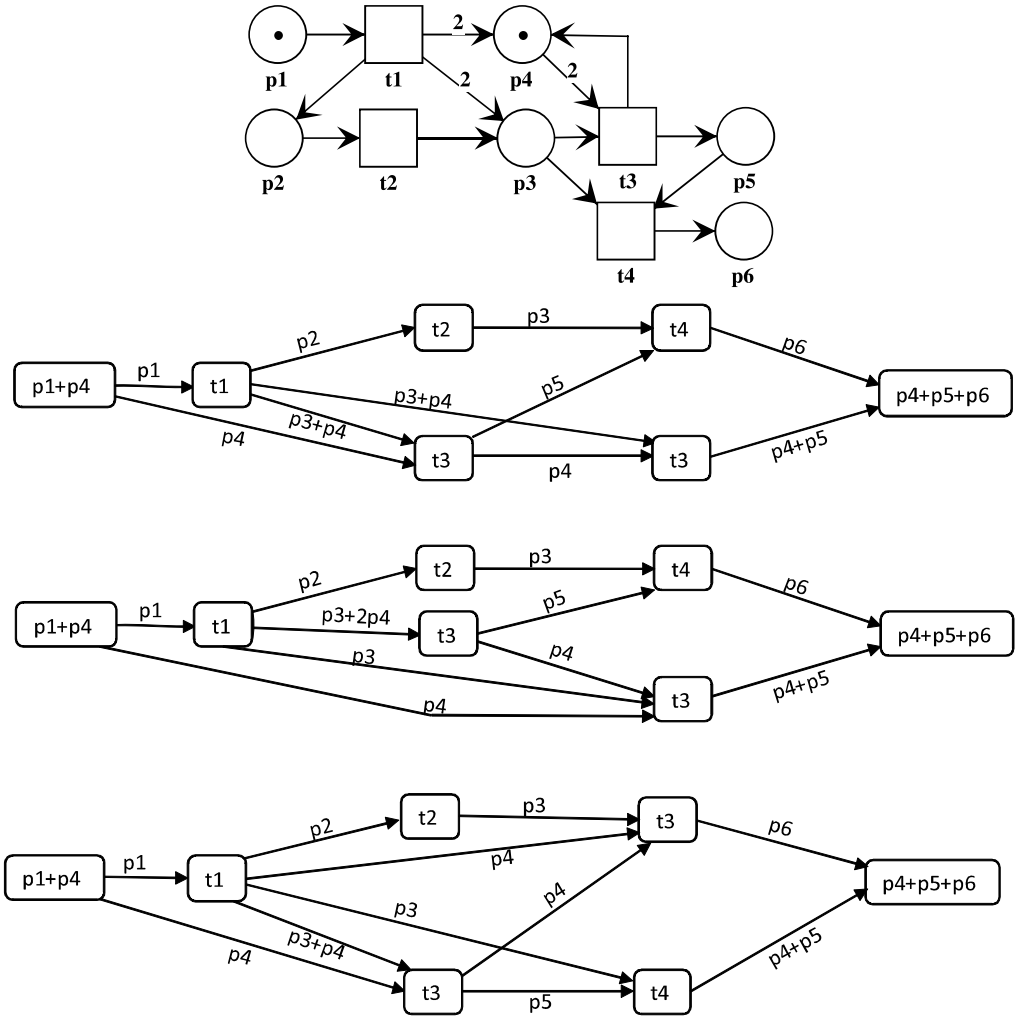

Fig. 3. Petri net from the Figureřefprocex (above) and the dicausal token flows corresponding to the processes in Figure 2

- Step sequences correspond to step ordered multiset token flows, shortly *somset token flows*, i.e. to the pomset token flows, where the relation given by unordered pairs of nodes is transitive.
- Occurrence sequences correspond to totally ordered token flows, shortly *tomset token flows*, i.e. to the pomset token flows where the relation given by unordered pairs of nodes is empty.

For better illustration, dicausal token flows corresponding to processes from Figure 2 are included in Figure 3.

An important role plays the equivalence given by symmetric and transitive closure of the extension relation of LDAGs, called *extension equivalence*: equivalence classes given by exchange equivalence on sequences, swapping equivalence on processes, and equivalence on rewrite terms correspond to restrictions of exchange equivalence classes to tomset token flows, direct causal token flows and N-free token flows, respectively.

Finally, we discuss the results achieved using token flows during the last four years, including the first polynomial test for the acceptance of a partial word by a Petri net, synthesis of Petri nets from partial languages and token flow unfolding.

2 Token Flows

We use $\mathbb{N}$ to denote the nonnegative integers. Given a finite set A, $|A|$ denotes the cardinality of A. Given a function f from A to B and $C \subseteq A$ we write $f|_C$ to denote the restriction of f to C. We write $A \setminus B$ to denote the usual set difference of sets A and B. The set of all subsets of a set A is denoted by 2^A. The set of all multisets over a set A, i.e. the set of functions from A to $\mathbb{N}$ is denoted by $\mathbb{N}^A$. We use $\emptyset$ to denote the empty multiset, i.e. $\forall a \in A : \emptyset(a) = 0$. The sum of multisets, the comparison of multisets and the difference of multisets are given as usual: given $m, m' \in \mathbb{N}^A$, $(m + m')(a) = m(a) + m'(a)$ for each $a \in A$, $m \geq m' \Leftrightarrow m(a) \geq m'(a)$ for each $a \in A$, and whenever $m \geq m'$ then $(m - m')(a) = m(a) - m'(a)$ for each $a \in A$.

We write $\sum_{a \in A} m(a)a$ to denote the multiset m over A. Given a function l from a set V to a set X, and a subset $S \subseteq V$, we define the multiset $\sum_{s \in S} l(s) \subseteq \mathbb{N}^X$ by $(\sum_{s \in S} l(s))(x) = |\{v \in V \mid v \in S \wedge l(v) = x\}|$. Given a binary relation $R \subseteq A \times A$ over a set A, R^+ denotes the transitive closure of R and R^* the reflexive and transitive closure of R.

A *directed graph* is a pair $(V, \rightarrow)$, where V is a finite *set of nodes* and $\rightarrow \subseteq V \times V$ is a binary relation over V called the *set of arcs*. Given a binary relation $\rightarrow$ we write $v \rightarrow v'$ to denote $(v, v') \in \rightarrow$ and $v \not\rightarrow v'$ to denote $(v, v') \notin \rightarrow$. We define ${}^\bullet v = \{v' \mid v' \rightarrow v\}$ and $v^\bullet \{v' \mid v \rightarrow v'\}$. A node $v \in V$ is called entry (also initial node, source, start or input), if ${}^\bullet v = \emptyset$. A node $v \in V$ is called exit (also final node, sink, end or output), if $v^\bullet = \emptyset$. Skeleton of a directed graph $(V, \rightarrow)$ is the directed graph $(V, \rightarrow')$ with $\rightarrow' = \{(v, v') \mid \nexists v'' : v \rightarrow^+ v'' \rightarrow^+ v'\}$ containing no transitiove arcs of $\rightarrow$.

A *partial order* is a directed graph $po = (V, <)$, where $<$ is irreflexive and transitive on V. Given a partial order $(V, <)$, for a set $S \subseteq V$ and a node $v \in V \wedge v \notin S$ 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: $\forall x, y \in S : \; x \, co \, y$. A *cut* (also called a slice) is a maximal co-set. A partial order $to = (V, <)$ satisfying $\forall (v, v') \in V \times V : v \neq v' \Rightarrow (v < v' \vee v' < v)$ is a *total order*. A partial order $so = (V, <)$, where co is transitive, is a *step order*. Given a step order $so = (V, <)$, we write $S < S'$ for cuts $S, S' \in V$ whenever there exist $v \in S$ such that $v < S'$. Because relation co is symmetric and reflexive, in a step order it is an equivalence relation. A partial order $po = (V, <)$ satisfying $(v < v' \wedge w < w' \wedge v < w') \Rightarrow \neg(v \, co \, w \wedge v' \, co \, w \wedge v' \, co \, w')$ is an *N-free partial order*.

A *directed acyclic graph* is a directed graph $dag = (V, \rightarrow)$, where the $po_{dag} = (V, \rightarrow^+)$ is a partial order. Given directed acyclic graphs $(V, \rightarrow)$ and $(V, \rightarrow')$ we say that $(V, \rightarrow)$ is an extension of $(V, \rightarrow')$ iff $\rightarrow' \subseteq \rightarrow$.

A *labelled directed acyclic graph*, shortly an LDAG, is a triple $ldag = (V, \rightarrow, l)$, where $(V, \rightarrow)$ is a directed acyclic graph and l is a labelling function from V to a set of labels. Two LDAGs $(V_1, \rightarrow_1, l_1), (V_2, <_2, l_2)$ are isomorphic iff there exists a bijection γ from V_1 to V_2 between nodes which preserve the arcs and the labelling function, i.e. $\forall v_1, v_2 \in V_1 : v_1 \rightarrow_1 v_2 \iff \gamma(v_1) \rightarrow_2 \gamma(v_2) \wedge l_1(v_1) = l_2(\gamma(v_1))$. We are not interested in the identity of nodes of an LDAG, so we distinguish LDAGs up to isomorphism. Without lose of generality, we will use any LDAG from an isomorphism class of LDAGs to denote the whole class.

A *pomset* is an LDAG $lpo = (V, <, l)$, where $(V, <)$ is a partial order. A *tomset* is an LDAG $lto = (V, <, l)$, where $(V, <)$ is a total order. A *somset* is an LDAG $lso = (V, <, l)$, where $(V, <)$ is a step order. An *N-free pomset* is an LDAG $(V, <, l)$, where $(V, <)$ is an N-free partial order.

An LDAG language is a set of (isomorphism classes of) LDAGs. Given an LDAG language L, and a subset $L' \subseteq L$, L' is called a sublanguage of L, and by L_{min} we denote its minimal sublanguage $L_{min} = \{(V, \rightarrow, l) \in L \mid \nexists (V, \rightarrow', l) \in L : \rightarrow' \subset \rightarrow\}$.

We use special LDAGs with a single entry and a single exit. An $ldag = (V, \rightarrow, l)$ with a single entry and a single exit is called single-entry and single-exit LDAG, shortly a SESE LDAG, $entry(ldag)$ denotes its entry and $exit(ldag)$ denotes its exit.

We also remove nodes and delete arcs from an LDAG. Let $ldag = (V, \rightarrow, l)$ be an LDAG. Let $X \subseteq V$. We define $remove(ldag, X) = (V', \rightarrow', l')$, where $V' = V \setminus X$, $\rightarrow' = \rightarrow \cap (V' \times V')$ and $\forall v \in V' : l'(v) = l(v)$. Observe, that removing nodes from an LDAG one gets an LDAG. Moreover, removing nodes from a pomset one gets a pomset. Let $\rightharpoonup \subseteq \rightarrow$. We define $delete(ldag, \rightharpoonup) = (V, \rightharpoonup, l)$, where $\rightharpoonup = \{v \rightarrow v' \mid v \not\rightharpoonup v'\}$.

Now we are prepared to define token flow functions of LDAGs over a finite set, and ingoing and outgoing token flows.

Definition 1 (Token flow function). *Let $ldag = (V, \rightarrow, l)$ be an LDAG. Let P be a finite set. A function $flow$ from the set of arcs $\rightarrow$ to $\mathbb{N}^P$ is called a* token flow function *of $ldag$ over P. The function $flow$ defines two functions attaching multisets over P to nodes:*

- *the function in_{flow} from V to $\mathbb{N}^P$, given by $in_{flow}(v) = \sum_{v' \in {}^\bullet v} flow(v', v)$, called the* ingoing token flow *of v,*
- *the function out_{flow} from V to $\mathbb{N}^P$, given by $out_{flow}(v) = \sum_{v' \in v^\bullet} flow(v, v')$, called the* outgoing token flow *of v.*

Let us define the first central notion of the paper - a token flow over a finite set: it is an LDAG with a single entry, a single exit, and with arcs labelled by multisets over a finite set.

Definition 2 (Token flow). *Let P be a finite set. A* token flow *over P is a pair $flowdag = (ldag, flow)$, where $ldag = (V, \rightarrow, l)$ is a SESE LDAG, and $flow$ is a token flow function of $ldag$ over P. A set of token flows over P is called P-token flow language, or shortly* token flow language.

3 Token Flows of Petri Nets

Definition 3 (Petri Net). *A place/transition Petri net (shortly a* Petri net*) is a 6-tuple $PN = (P, T, consume, produce, initial, final)$ where P is a finite set of places, T is a finite set of transitions, $T \cap (P \cup \mathbb{N}^P) = \emptyset$, $consume$ and $produce$ are functions from T to $\mathbb{N}^P$, such that $\forall t \in T : consume(t) \neq \emptyset \wedge produce(t) \neq \emptyset$, multiset $initial \in \mathbb{N}^P$ is an initial marking and $final \subseteq N^P$ is a set of legal final markings.*

In the rest of the paper we suppose that a $PN = (P, T, consume, produce, initial, final)$ is given. A multiset $m \in \mathbb{N}^P$ is called a marking of PN. Sequential behaviour of the PN is given by occurrences (firings) of transitions: A transition $t \in T$ is enabled to occur in a marking m of PN iff $m \geq consume(t)$. An occurrence of enabled transition t in a marking m leads to the follower marking $m' = m - comsume(t) + produce(t)$. We write $m \stackrel{t}{\longmapsto} m'$ to denote that t is enabled to occur in m and that its occurrence leads to m'. Sequential behaviour can easily be extended to the simplest way to dercsribe concurrent occurrences of transitions - to the occurrences of steps, which are multisets of transitions: Given a step $s \in \mathbb{N}^T$, denote by $consume(s)$ the multiset of places given by $\forall p \in P : consume(s)(p) = \sum_{t \in T} s(t) consume(t)(p)$. By $produce(s)$ denote the multiset of places given by $\forall p \in P : produce(s)(p) = \sum_{t \in T} s(t) produce(t)(p)$. A step $s \in \mathbb{N}^T$ is enabled to occur in a marking m of PN iff $m \geq consume(s)$. An occurrence of enabled step s in a marking m leads to the follower marking $m' = m - comsume(s) + produce(s)$. We write $m \stackrel{s}{\longmapsto} m'$ to denote that s is enabled to occur in m and that its occurrence leads to m'.

Le us notice, that the above definition differs from the usual definition of place Petri nets, however the difference is only technnical. Usually, a place/transition Petri net is given as a bipartite directed graph with weighted arcs, with nodes formed by places and transition, places and arcs labelled by nonnegative integers. The labelling of places gives the marking. From technical reasons in our definition we additionaly require that no transition equals a multiset of places. In a usual definition, instead of the functions $consume, produce$ the relationship between places and transitions is given using a set of arcs $F \subseteq ((P \times T) \cup (T \times P))$ (also called flow relation) and a weight function W from F to $\mathbb{N}$. Using our definition, F and W can be easily reconstructed: $F = \{(p, t) \in P \times T \mid consume(t)(p) \neq 0\} \cup \{(t, p) \in T \times P \mid produce(t)(p) \neq 0\}$, $\forall (p, t) \in F : W(p, t) = consume(t)(p), \forall (t, p) \in F : W(t, p) = produce(t)(p)$.

In a usual definition, the set of final marking is not defined. The intended meaning of the set of final markings is to allow acceptance of only a subset of LDAGs generated

by processes. Obviously, taking the set of all multisets as legal final marking, one gets that all of the processes are accepted.

Now we are prepared to define the second central notion of the paper: token flows of PN, as token flows over P such that the entry is labelled by the initial marking and the outgoing token flow of the entry equals the initial marking, the exit is labelled by a final marking and the outgoing token flow of the exit equals the final marking, and for all other nodes the ingoing token flow equals the $consume$ value of their label and outgoing token flow equals the $produce$ value of their label.

Definition 4 (Token Flow of a Petri Net). *Let $PN = (P, T, consume, produce, initial, final)$ be a Petri net and let $flowdag = (ldag, flow)$ be a token flow over P with $ldag = (V, \rightarrow, l)$. Then $flowdag$ is called* token flow of PN*, and we say that $flow$ fulfils the token flow property iff:*

1. $\forall v \in V : v \notin \{entry(ldag), exit(ldag)\} \Rightarrow (l(v) \in T \wedge consume(l(v)) = in_{flow}(v) \wedge produce(l(v)) = out_{flow}(v))$
2. $l(entry(ldag)) = out_{flow}(entry(ldag)) = initial$,
3. $l(exit(ldag)) = in_{flow}(exit(ldag)) \subseteq final$,

The set of all token flows of PN is denoted by $L^{tf}_{all}(PN)$ and called the token flow language of PN.

Observe that given a SESE LDAG $ldag = (V, \rightarrow, l)$ with the entry labelled by the initial marking, the exit labelled by any final marking and remaining nodes labelled by transitions of PN, the token flow functions fulfilling the token flow property are simply nonnegative integer solutions of the system (1 - 3) of linear equations from the previous definition, with $in_{flow}(v)$ replaced by $\sum_{v' \in {}^\bullet v} flow(v', v)$, $out_{flow}(v)$ replaced by $\sum_{v' \in v^\bullet} flow(v, v')$ for any $v \in V$, and unknown variables $flow(v, v')(p)$ for each $p \in P$ and each $v \rightarrow v'$. We will call such a system the *token flow system of the $ldag$*.

Using the interpretation of arcs in the token flows and the structure of the LDAGs, one can define all prominent semantics of PN. Basiacally, a semantics denoted by sem is determinded by a subset $L^{tf}_{sem}(PN) \subseteq L^{all}tf(PN)$ of token flows of PN. Each semantics can be recognignized on four levels. The first level is given by a language obtained from token flows by forgetting the flow function. The second level is formed by the minimal sublanguage of this language. The third level is given by a language obained from the first level by forgetting the entry and exit. The fourth level is given by the minimal sublangage of the third language.

Definition 5 (Languages of a Petri Net). *Let $L^{tf}_{sem}(PN) \subseteq L^{tf}_{all}(PN)$. We derive following four LDAG languages from $L^{tf}_{sem}(PN)$:*

1. $L^{io}_{sem}(PN) = \{ldag \mid (ldag, flow) \in L^{tf}_{sem}(PN)\}$,
2. $L^{io}_{sem}(PN)_{min}$,
3. $L^{nio}_{sem}(PN) = \{remove(ldag, \{entry(ldag), exit(ldag)\}) \mid ldag \in L^{io}(PN)\}$,
4. $L^{nio}_{sem}(PN)_{min}$.

Given an $ldag \in L^{io}_{all}(PN)$, its final marking, i.e. the label of the final node, can be easily determined.

Proposition 1. *Let* $ldag \in L^{io}_{all}(PN)$ *with* $ldag = (V, \rightarrow, l)$. *Then* $l(exit(ldag)) = initial + \sum_{v \in V \setminus \{entry(ldag), exit(ldag)\}} (produce(l(v)) - consume(l(v)))$.

If $L^{io}_{sem}(PN)$ is a pomset language, then for each LDAG $ldag' \in L^{nio}_{sem}(PN)$, there exists one and only one SESE LDAGs $ldag \in L^{io}_{sem}(PN)$, such that $ldag' = remove$ $(ldag, \{entry(ldag), exit(ldag)\}$. As a consequence for pomset languages we get that $L^{nio}_{sem}(PN)_{min}$ is the image of the restriction of $remove(ldag, \{entry(ldag),\ exit$ $(ldag)\})$ to $L^{io}_{sem}(PN)_{min}$.

3.1 Token Flow Semantics of Petri Nets

As the first semantics we define the direct causal token flows. The arcs in a direct causal token flow represent direct causality, i.e. the fact, that the source of the arc produced at least one token consumed by the target of the arc.

Definition 6 (Direct Causal Token Flow). *Let* $flowdag = (ldag, flow)$ *with* $ldag = (V, \rightarrow, l)$ *be a token flow of* PN *satisfying* $\forall (v, v') \in \rightarrow\colon\ flow(v, v') \neq \emptyset$. *Then* $flowdag$ *is called direct causal token flow of* PN, *shortly* dicausal token flow of PN. *The set of all dicausal token flows of* PN *is denoted by* $L^{tf}_{dicausal}(PN)$ *and called* dicausal token flow language of PN.

Observe, that the inequations $flow(v, v') \neq \emptyset$ can be rewrited to the integer inequation $\sum_{p \in P} flow(v, v')(p) \neq 0$. Adding the inequations to the token flow system of $ldag$ we get the system of linear inequetions, called *dicausal token flow system of* $ldag$. Obviously, $ldag \in L^{io}_{dicausal}(PN)$ iff the dicausal token flow system of $ldag$ has a nonnegative integer solution.

Dicausal token flows of PN contain complete information about causal dependency of transition occurrences, including the information which occurrences consume tokens from the initial marking and which occurrences produce tokens in the final marking. The difference between elements of $L^{io}_{dicausal}(PN)$ and $L^{nio}_{dicausal}(PN)$ is that in the elements from $L^{nio}_{sem}(PN)$ the information which occurrences consume tokens from the initial marking and which occurrences produce tokens in the final marking is forgotten.

The second semantics are the causal token flows. The arcs in a causal pomset represent causality between the source and the target, not necessarily the direct one.

Definition 7 (Causal Token Flow). *Let* $flowdag = (ldag, flow)$ *be a token flow of* PN *such that* $ldag = (V, <, l)$ *is a pomset. Let* $(V, \rightarrow)$ *be the skeleton of* $(V, <)$. *If* $\forall (v, v') \in \rightarrow\colon\ flow(v, v') \neq \emptyset$ *then* $flowdag$ *is called* causal token flow of PN. *The set of all causal token flows of* PN *is denoted by* $L^{tf}_{causal}(PN)$ *and called* causal token flow language of PN.

The next semantics are the pomset token flows. In a pomset token flow, the arcs represent the fact, that the source and the target occurred sequentially. This in fact means, that either these occurrences are independent or the target is causaly dependent on the source.

Definition 8 (Pomset Token Flow). *Let* $flowdag = (ldag, flow)$ *be a token flow of* PN *such that* $ldag$ *is a pomset. Then* $flowdag$ *is called* pomset token flow of PN. *The set of all pomset token flows of* PN *is denoted by* $L^{tf}_{pomset}(PN)$ *and called* pomset token flow language of PN.

The next semantics are token flows, where the underlying LDAGs are N-free pomsets.

Definition 9 (N-Free Token Flow). *Let $flowdag = (ldag, flow)$ be a token flow of PN such that $ldag$ is an N-free pomset. Then $flowdag$ is called* N-free token flow of PN. *The set of all N-free token flows of PN is denoted by $L^{tf}_{Nfree}(PN)$ and called* N-free token flow language of PN.

As the last two semantics we define somset and tomset token flows.

Definition 10 (Somset Token Flow). *Let $flowdag = (ldag, flow)$ be a token flow of PN such that $ldag$ is a somset. Then $flowdag$ is called* somset token flow of PN. *The set of all somset token flows of PN is denoted by $L^{tf}_{somset}(PN)$ and called* somset token flow language of PN.

Definition 11 (Tomset Token Flow). *Let $flowdag = (ldag, flow)$ be a token flow of PN such that $ldag$ is a tomset. Then $flowdag$ is called* tomset token flow of PN. *The set of all tomset token flows of PN is denoted by $L^{tf}_{tomset}(PN)$ and called* tomset token flow language of PN.

3.2 Relationship between Token Flow Semantics of Petri Nets

Directly from the above defintions we can see the following relationships between the presented token flow semantics of PN. Let $(ldag, flow)$ be a token flow of PN with $ldag = (V, \rightarrow, l)$. We define $positive(ldag, flow) = delete(ldag, \{v \rightarrow v' \mid flow(v, v') = \emptyset\})$. Consider that $x \in \{io, nio\}$, i.e. x can be replaced by either io or nio, and $sem \in \{causal, pomset, Nfree, somset, tomset\}$:

$$L^x_{causal}(PN) = \{(V, \rightarrow^+, l) \mid (V, \rightarrow, l) \in L^x_{dicausal}(PN)\}$$

$$L^{io}_{dicausal}(PN) = \{positive(ldag, flow) \mid (ldag, flow) \in L^{tf}_{sem}(PN)\}$$

$$L^x_{tomset}(PN) \subseteq L^x_{somset}(PN) \subseteq L^x_{Nfree}(PN) \subseteq L^x_{pomset}(PN)$$

$$L^x_{causal}(PN) \subseteq L^x_{pomset}(PN)$$

Another important relationship between these semantics is the relationship w.r.t. extension. Taking two sets X, Y of pomsets, we denote by $X \ni Y$ that for each pomset $(V, <, l)$ from X there exists a pomset $(V, <', l)$ from Y such that $(V, <)$ is an extension of $(V, <')$. We observe the following:

$$L^x_{tomset}(PN) \ni L^x_{somset}(PN) \ni L^x_{Nfree}(PN) \ni L^x_{pomset}(PN) \ni L^x_{causal}(PN)$$

As a consequence:

$$L^x_{causal}(PN)_{min} = L^x_{pomset}(PN)_{min}$$

Observe, that the total order of a tomset token flow can be an extension of DAGs of several dicausal token flows with different DAGs. On the other hand, there can be several tomset token flows with different total orders, which are extensions of the DAG of a dicausal token flow. Therefore we introduce an equivalence on $L^x_{all}(PN)$ as the symmetric and transitive closure of the relation "being an extension". The equivalence is called the extension equivalence on $L^x_{all}(PN)$.

Definition 12 (Extension Equivalence). *Let $x \in \{io, nio\}$. Let $(V, <, l), (V, <', l) \in L^x_{all}(PN)$. Define $(V, \rightarrow, l) \propto (V, \rightarrow', l)$ if $(V, \rightarrow)$ is an extension of $(V, \rightarrow')$. The symmetric and transitive closure $\equiv$ of $\propto$ is called* extension equivalence *on $L^x_{all}(PN)$.*

3.3 Direct Causal Token Flows and Processes

In this subsection we discuss the relationship between dicausal token flows and processes of [7,8].

Definition 13 (Occurrence Net). *An* occurrence net *is a directed acyclic graph $O = (B \cup E, G)$, with two partitions of nodes denoted by B and E (called conditions and events) s. t. $(B \cup 2^B) \cap E = \emptyset$, and flow relation $G \subseteq (B \times E) \cup (E \times B)$, s. t. $|\bullet b|, |b \bullet| \leq 1$ for every $b \in B$ and no node from E is an entry or an exit.*

The set of conditions of an occurrence net $O = (B\cup, G)$ which are entries and exits are denoted by $Min(O)$ $Max(O)$, respectively.

Definition 14 (Process). *Let $PN = (P, T, consume, produce, initial, final)$ be a Petri net. A* process *of PN is a pair $K = (O, \rho)$, where $O = (B \cup E, G)$ 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) $\forall e \in E, \forall p \in P : |\{b \in \bullet e \mid \rho(b) = p\}| = consume(\rho(e))(p)$ *and*
$\forall e \in E, \forall p \in P : |\{b \in e\bullet \mid \rho(b) = p\}| = produce(\rho(e)), (p)$.
(iii) $\forall p \in P : |\{b \in Min(O) \mid \rho(b) = p\}| = initial(p)$
(iv) $\exists fin \in final$ *such that* $\forall p \in P : |\{b \in Max(O) \mid \rho(b) = p\}| = fin(p)$.

Definition 15 (Canonical LDAG, Canonical Token Flow). *Let $K = (O, \rho)$ be a process of a Petri net PN. Define*

$$entryarc(K) = (Min(O), e) \mid \exists b \in Min(O) : (b, e) \in G,$$
$$exitarc(K) = (e, Max(O)) \mid \exists b \in Max(O) : (e, b) \in G.$$

The canonical LDAG of process K *is the LDAG $ldag_K = (V, \to, l)$, where*

$$V = E \cup \{Min(O), Max(O)\},$$
$$\to = G^2|_{E\times E} \cup entryarc(K) \cup exitarc(K),$$
$$l|_E = \rho|_E,\ l(Min(O)) = \textstyle\sum_{b\in Min(O)} \rho(b) \text{ and } l(Max(O)) = \sum_{b\in Max(O)} \rho(b).$$

The canonical token flow function $flow_K$ of process K *is the token flow function of $ldag_K$ over P given by $flow_K(v, v') = \sum_{b\in(v^\bullet \cup \bullet v')} \rho(b)$ for each $v \to v'$. The* canonical token flow of process K *is the pair $(ldag_K, flow_K)$. The language of canonical token flows of all processes of PN is denoted by $CL(PN)$.*

We have proven the following result in [10].

Theorem 1. *Let PN be a Petri net. Then $CL(PN) = L^{tk}_{dicausal}(PN)$.*

In [3] so called swapping equivalence on processes is defined.

Definition 16 (Swapping). *Let PN be a Petri net. Let $K = (O, \rho)$, be a process of PN with $O = (B \cup E, G)$. Let $b_1, b_2 \in B$, b_1 co b_2 w.r.t. G^+ and $\rho(b_1) = \rho(b_2)$. Define $G_1 = \{(b_1, e) \mid (b_2, e) \in G\}$ and $G_2 = \{(b_2, e) \mid (b_1, e) \in G\}$. Define $G' = G_1 \cup G_2 \cup (G \cap (E \times B)) \cup (G \cap ((B \setminus \{b_1, b_2\}) \times E))$. G' is obtained from G by interchanging arcs from b_1 and b_2. Finally, define $swap(K, b_1, b_2) = ((B, E, G'), \rho)$.*

Definition 17 (Swapping Relation). *Let $K_1 = ((B \cup E, G), \rho)$ and K_2 be processes of PN. Let us define $K_1 \equiv_1 K_2$ if there are conditions $b_1, b_2 \in B$ satisfying $b_1 \, co \, b_2$ w.r.t. G^+, $\rho(b_1) = \rho(b_2)$ and K_2 is (isomorphic to) $swap(K_1, b_1, b_2)$.*

It is easy to see that $\equiv_1$ is symmetric. Thus, $\equiv_1^*$ is an equivalence relation on processes of PN.

Definition 18 (Swapping Equivalence). *The equivalence relation $\equiv_1^*$ on processes of PN is called* swapping equivalence. *The equivalence classes of processes w.r.t. the swapping equivalence are called* commutative processes *of PN.*

We extend the swapping equivalence to canonical LDAGs: Given two processes K_1, K_2 of a Petri net PN, we define $ldag_{K_1} \equiv_1^* ldag_{K_2}$ whenever $K_1 \equiv_1^* K_2$. Based on the results in [10] we state that the extension equivalence restricted to $L^{io}_{dicausal}$ and swapping equivalence coincide:

Theorem 2. *For each $ldag_1, ldag_2 \in L^{io}_{dicausal}$: $ldag_1 \equiv_1^* ldag_2 \Leftrightarrow ldag_1 \equiv ldag_2$.*

3.4 Pomset Token Flows and Enabled Pomsets

In this subsection we recall the definition of enabled pomsets, also known as partial words [9,12,19] and discuss their relationship to pomset token flows.

Definition 19 (Enabled Pomset). *Let $PN = (P, T, consume, produce, initial, final)$ be a Petri net. A pomset $lpo = (V, <, l)$ with $l : V \to T$ is* enabled to occur in PN *if the following statements hold:*

(a) For each co-set S of $(V, <)$:

$$initial + \sum_{v \in V \wedge v < S} (produce(l(v)) - consume(l(v))) \geq \sum_{v \in S} consume(l(v)).$$

(b) $m = initial + \sum_{v \in V} (produce(l(v)) - consume(l(v))) \in final$.

We say that occurrence of lpo leads from $initial$ to m and m is the final marking of the lpo. The enabled pomsets are also called partial words of PN *and the language of all enabled pomsets is called the* partial language of PN *and denoted by $PL(PN)$.*

Actually, the definition of enabledness can be reformulated considering only slices of labelled partial orders (for the proof see e.g. [19]). In [10] we have proven the following result.

Theorem 3. *Let PN be a Petri net. Then $PL(PN) = L^{nio}_{pomset}(PN)$.*

3.5 N-Free Token Flows and Rewrite Terms

In this subsection we establish the relationship between N-free token flows and rewriting semantics originally introduced in [15]. In this subsection we write $t : m \to m'$ to denote that $t \in T$, $consume(t) = m$ and $produce(t) = m'$.

Definition 20 (Rewrite Term Semantics). *Let $PN = (P, T, consume, produce, initial, final)$ be a Petri net. The set of general rewrite terms $\mathcal{GT}(PN)$ of PN is defined inductively by the following production rules:*

$$\frac{m \in \mathbb{N}^P}{m : m \to m \in \mathcal{GT}(PN)}$$

$$\frac{t \in T}{t : consume(t) \to produce(t) \in \mathcal{GT}(PN)}$$

$$\frac{\alpha_1 : m_1 \to m_1' \in \mathcal{GT}(PN) \wedge \alpha_2 : m_2 \to m_2' \in \mathcal{GT}(PN)}{(\alpha_1 \parallel \alpha_2) : m_1 + m_2 \to m_1' + m_2' \in \mathcal{GT}(PN)}$$

$$\frac{\alpha_1 : m \to m' \in \mathcal{GT}(PN) \wedge \alpha_2 : m' \to m'' \in \mathcal{GT}(PN)}{(\alpha_1 ; \alpha_2) : m \to m'' \in \mathcal{GT}(PN)}$$

These rules define binary operations, called concurrent composition ($\parallel$) *and* sequential composition ($;$) *of rewrite terms. The set of* rewrite terms of PN *denoted by $\mathcal{T}(PN)$ is the subset of $\mathcal{GT}(PN)$ given by $\mathcal{T}(PN) = \{\alpha : initial \to m \in \mathcal{GT}(PN) \mid m \in final\}$.*

Given a rewrite term $\alpha : m \to m'$, we shortly say that α is a term and we denote by $pre(\alpha) = m$ the initial marking and by $post(\alpha) = m'$ the final marking of α.

Definition 21 (Pomset of a Rewrite Term). *Define inductively the pomset lpo_α of a term α:*

- *Given a marking m, $lpo_m = (\emptyset, \emptyset, \emptyset)$.*
- *Given a transition $t \in T$, $lpo_t = (\{v\}, \emptyset, l)$, where $l(v) = t$.*
- *Given terms α_1 and α_2 with $lpo_{\alpha_1} = (V_1, <_1, l_1)$ and $lpo_{\alpha_2} = (V_2, <_2, l_2)$, $lpo_{\alpha_1 \| \alpha_2} = (V_1 \cup V_2, <_1 \cup <_2, l_1 \cup l_2)$, where V_1 and V_2 are assumed to be disjoint (what can be achieved by appropriate renaming of nodes).*
- *Given terms α_1 and α_2 with $lpo_{\alpha_1} = (V_1, <_1, l_1)$ and $lpo_{\alpha_2} = (V_2, <_2, l_2)$, $lpo_{\alpha_1 ; \alpha_2} = (V_1 \cup V_2, <_1 \cup <_2 \cup \{(a, b) \mid a \in V_1,\ b \in V_2\}, l_1 \cup l_2)$, where V_1 and V_2 are assumed to be disjoint (what can be achieved by appropriate renaming of nodes).*

The language of pomsets associated to all rewrite terms of PN is denoted by $TL(PN)$. The elements of $TL(PN)$ are called term pomsets of PN.

The previous definition is sound in the sense, that the structures attached to terms are pomsets. Pomsets of rewrite terms of Petri nets coincide with so called finite series-parallel pomsets, i.e. with pomsets generated from single element pomsets by concurrent composition (disjoint union side by side) and sequential composition. A finite pomset is series-parallel iff it is N-free (for a proof see e.g. [6]). As a consequence we get the following result based on [10]:

Theorem 4. *Let PN be a Petri net. Then $TL(PN) = L^{nio}_{Nfree}(PN)$.*

Rewrite terms are identified by an equivalence relation $\sim$ which preserves the operations $\parallel$ and $;$ (i.e. by a congruence w.r.t. the operations $\parallel$ and $;$), given by the following axioms: Let $m, m' \in \mathbb{N}^P$ and $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ be rewrite terms.

(1) $(\alpha_1 \parallel \alpha_2) \sim (\alpha_2 \parallel \alpha_1)$.
(2) $((\alpha_1; \alpha_2); \alpha_3) \sim (\alpha_1; (\alpha_2; \alpha_3))$, whenever these terms are defined.
(3) $((\alpha_1 \parallel \alpha_2) \parallel \alpha_3) \sim (\alpha_1 \parallel (\alpha_2 \parallel \alpha_3))$.
(4) $((\alpha_1 \parallel \alpha_2); (\alpha_3 \parallel \alpha_4)) \sim ((\alpha_1; \alpha_3) \parallel (\alpha_2; \alpha_4))$, whenever these terms are defined.
(5) $(\alpha_1; post(\alpha_1)) \sim \alpha_1 \sim (pre(\alpha_1); \alpha_1)$.
(6) $m + m' \sim (m \parallel m')$
(7) $\alpha_1 + \emptyset \sim \alpha_1$ for the empty multiset $\emptyset$.

Observe that for any two equivalent terms $\alpha_1 \sim \alpha_2$, we have $pre(\alpha_1) = pre(\alpha_2)$ and $post(\alpha_1) = post(\alpha_2)$.

We extend the equivalence $\sim$ to the set $TL(PN)$: Given two terms α_1, α_2 of a Petri net PN, we define $lpo_{\alpha_1} \sim lpo_{\alpha_2}$ whenever $\alpha_1 \sim \alpha_2$. Based on the results in [10] we state that the extension equivalence restricted to $TL(PN)$ and $\sim$-equivalence coincide:

Theorem 5. *For each $ldag_1, ldag_2 \in TL(PN) : ldag_1 \sim ldag_2 \Leftrightarrow ldag_1 \equiv ldag_2$.*

3.6 Somset Token Flows and Step Sequences

We briefly mention the relationship between step sequences and somset token flows.

Definition 22 (Step Sequence). *Let PN be a Petri net. A finite sequence of steps of PN $\sigma = s_1 \dots s_n$ $(n \in \mathbb{N})$ is called a* step sequence of PN *if there exists a sequence of markings $m_1, \dots, m_n$ such that $initial \stackrel{s_1}{\longmapsto} m_1 \stackrel{s_2}{\longmapsto} \dots \stackrel{s_n}{\longmapsto} m_n$ and m_n is a legal final marking of PN.*

Definition 23. *Let PN be a Petri net. Let $\sigma = s_1 \dots s_n$ be a step sequence of PN. Then the somset $lso_\sigma = (V, \prec, l)$ with $l : V \to T$ and with cuts $S_1, \dots S_n$ satisfying $|S_i| = s_i$ and $i < j \Rightarrow S_i \prec S_j$ for every $i, j \in \{1, \dots n\}$ is associated to σ. The language of somsets associated to all step sequences of PN is denoted by $SL(PN)$.*

Theorem 6. *Let PN be a Petri net. Then $SL(PN) = L^{nio}_{somset}$.*

3.7 Tomset Token Flows and Occurrence Sequences

Finally, we discuss the relationship between occurrence sequences and tomset token flows.

Definition 24 (Occurrence sequence). *Let PN be a Petri net. A finite sequence of transitions of PN $\sigma = t_1 \dots t_n$ $(n \in \mathbb{N})$ is called* occurrence sequence of PN *if there exists a sequence of markings $m_1, \dots, m_n$ such that $initial \stackrel{t_1}{\longmapsto} m_1 \stackrel{t_2}{\longmapsto} \dots \stackrel{t_n}{\longmapsto} m_n$ and m_n is a legal final marking of PN.*

Definition 25. *Let PN be a Petri net. Let $\sigma = l(v_1) \dots l(v_n)$ is an occurrence sequence of PN. Then the tomset $lto = (\{v_1, \dots, v_n\}, \prec, l)$ satisfying $i < j \Rightarrow v_i \prec v_j$ for every $i, j \in \{1, \dots n\}$ is associated to occurrence sequence σ. The language of tomsets associated to all occurrence sequences of PN is denoted by $OL(PN)$.*

Theorem 7. *Let PN be a Petri net. Then $OL(PN) = L^{nio}_{tomset}$.*

In [3] the exchange equivalence on occurrence sequences is defined.

Definition 26 (Exchange Relation). *Let PN be a Petri net.*
Let $\sigma_1 = t_1 \dots t_{i-1} t_i t_{i+1} t_{i+2} \dots t_n, \sigma_2 = t_1 \dots t_{i-1} t_{i+1} t_i t_{i+2} \dots t_n$ be occurrence sequences of PN. Then $\sigma_1 \equiv_0 \sigma_2$ iff $\sigma = \{t_1\} \dots \{t_{i-1}\}\{t_i, t_{i+1}\}\{t_{i+2}\} \dots \{t_n\}$ is a step sequence of PN.

It is easy to see that $\equiv_0$ is symmetric and therefore $\equiv_0^*$ is an equivalence relation.

Definition 27 (Exchange Equivalence). *The equivalence relation $\equiv_0^*$ on occurrence sequences of PN is called* exchange equivalence.

Based on the results in [10] we state that the extension equivalence restricted to $OL(PN)$ and exchange equivalence coincide:

Theorem 8. *For each $lto_1, lto_2 \in OL(PN) : lto_1 \equiv_0^* lto_2 \Leftrightarrow lto_1 \equiv lto_2$.*

4 Results and Related Works

As we mentioned in Introduction, motivation for introducing token flows was to have not only a unifying framework for different flavours of Petri net semantics, but to have also a simple formalism and effective technique to solve problems. In this section we briefly discuss the results achieved using token flows during the last four years, including the first polynomial test for the acceptance of a partial word by a Petri net, the first algorithm for synthesis of Petri nets from partial languages and token flow unfolding.

Testing Pomsets of Petri Nets: The important question arise: Given a pomset lpo an a Petri net PN, is lpo enabled in PN? The definition of enabledness of pomsets is inherently exponential, since a pomset can have exponentially many cuts in the number of nodes. That means, the definition is not appropriate to develop a test for partial words.

Using the fact that partial language $PL(PN)$ equals L^{nio}_{pomset} of PN, which is obtained by forgetting token flow function and the entry and the exit in token flows from L^{tk}_{pomset}, the problem is reduced to answer the question : Given a SESE pomset, can we label its arcs by a token flow function to get a token flow of PN? The answer is positive, if and only if the token flow system of the pomset is solvable in nonnegative integers. Unfortunately, the solvability of a system of linear equations in nonnegative integers is in general NP-complete [20]. That means, to use a general algorithm for solving linear equations in nonnegative integers is not appropriate to develop a test for partial words.

In [11,14] we present algorithms to test a partial word in polynomial time, i.e we answer the question whether a pomset lpo belongs to $L^x_{pomset}(PN)$, where $x \in \{io, nio\}$, in a polynomial time. In [11,14] we have shown that decision whether a pomset is a minimal causal pomset, i.e. whether a pomset lpo belongs to $L^x_{causal}(PN)_m in$, where $x \in \{io, nio\}$, can be obtained in a polynomial time.

Synthesis of Petri Nets from Pomset Languages: In papers [13,2] token flows are used to synthetize a Petri net PN (with all markings beeing final) from a pomset language PL

(closed w.r.t. extension and prefixes) in such a way that either $PL = L^{nio}_{pomset}(PN)$ or $PL \subseteq L^{nio}_{pomset}(PN)$ and there is no Petri net PN' satisfying $PL \subseteq L^{nio}_{pomset}(PN') \subset L^{nio}_{pomset}(PN)$). Obviously, the labels of the pomsets give transitions T. The synthesis reduces to finding places, the values of $consume$ and $produce$ functions as well as the initial marking in such a way, that still all pomsets are accepted. In the synthesis procedure, the pomsets are extended by adding a single entry and a single exit. The entry is labelled in all pomsets by the same unique symbol, not used as a label of other nodes. The exit in each pomset is labelled by a different symbol, not used in the labels of other nodes. The main idea is to consider simple token flow functions, which attach a nonnegative integer to each arc in each pomset. If such a simple token flow function fulfils that equally labelled nodes in all pomsets have equal ingoing token flows and equal outgoing token flows, then it is called a token flow region and defines a place p. In the token flow region, we can speak about ingoing token flows and outgoing token flows of labels: the ingoing token flow of a label $t \in T$ defines the $consume(t)(p)$, the outgoing token flow of a label $t \in T$ defines $produce(t)(p)$, and the ingoing token flow of the entry label defines the initial marking of the place p. Adding a place determined by a token flow region still all pomsets will be accepted by the net. Adding places given by all regions, we get the seeked Petri net PN. Similarly to token flow systems of an LDAG, the token flow regions are nonnegative integer solutions of a system of linear equations, where the single equations just states that the equally labelled nodes in pomsets have equal ingoing and outgoing token flow. If the number of pomsets is finite, then the number of nodes is finite and the system have finite number of equations. The number of solutions, and therefore places, can still be infinite. Fortunatelly, it is enouhg to take places derived from the Hilbert basis of the system, which is finite. Namely, the net with the places derived from the Hilbert basis accept the same pomsets as the net PN.

Token flow unfolding: The idea of token flow unfolding presented in [1] is a straightforward extension of token flows. Instead of attaching a token flow function to pomsets, obtaining causal token flows, the idea is to attach a token flow function to prime event structures, to get token flow event structures, which are actually unions of causal token flows.

Token flow Hasse diagrams: Another idea to extend token flows can be found in the paper [17] in this volume. Instead of considering pomsets, authors consider Hasse diagrams, which are actually skeletons of LDAGs. It is shown in [17], that the token flow function of a pomset can be reconstructed from the extended token flow function of its skeleton, called interlaced flow. The interlaced flow attaches four multisets of tokens to each arc $v \to v'$ of the skeleton: the first multiset says how many tokens produced by v are consumed by v', the second says how many tokens produced by v are consumed in the future of v', the third counts how many tokens produced in the past of v and consumed by v'; and the last multiset says how many tokens produced in the past of v and consumed in the future of v'.

References

1. 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)
2. Bergenthum, D.J., Lorenz, R., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. Fundamenta Informaticae 88(4), 437–468 (2008)
3. Best, E., Devillers, R.: Sequential and Concurrent Behaviour in Petri Net Theory. Theoretical Computer Science 55(1), 87–136 (1987)
4. Degano, E., Meseguer, J., Montanari, U.: Axiomatizing the Algebra of Net Computations and Processes. Acta Informatica 33(7), 641–667 (1996)
5. Desel, J., Juhás, G.: What is a Petri Net? In: Ehrig, H., Juhás, G., Padberg, J., Rozenberg, G. (eds.) APN 2001. LNCS, vol. 2128, pp. 1–25. Springer, Heidelberg (2001)
6. Gischer, J.L.: The equational theory of pomsets. Theoretical Computer Science 61(2-3), 199–224 (1988)
7. Goltz, U., Reisig, W.: The Non-Sequential Behaviour of Petri Nets. Information and Control 57(2-3), 125–147 (1983)
8. Goltz, U., Reisig, W.: Processes of Place/Transition Nets. In: Díaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 264–277. Springer, Heidelberg (1983)
9. Grabowski, J.: On Partial Languages. Fundamenta Informaticae 4(2), 428–498 (1981)
10. Juhás, G.: Are these events independent? It depends! Habilitation thesis, Katholic University Eichstätt-Ingolstadt (2005)
11. 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)
12. Kiehn, A.: On the Interrelationship between Synchronized and Non-Synchronized Behavior of Petri Nets. Journal Inf. Process. Cybern. EIK 24(1-2), 3–18 (1988)
13. Lorenz, R., Juhás, G.: Toward Synthesis of Petri Nets from Scenarios. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 302–321. Springer, Heidelberg (2006)
14. Lorenz, R., Juhás, G., Bergenthum, R., Desel, J., Mauser, S.: Executability of scenarios in Petri nets. Theoretical Computer Science 410(12-13), 1190–1216 (2009)
15. Meseguer, J., Montanari, U.: Petri nets are monoids. Information and Computation 88(2), 105–155 (1990)
16. Priese, L.: Semi-rational sets of dags. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 385–396. Springer, Heidelberg (2005)
17. Oliveira, M.: Hasse Diagram Generators and Petri Nets. In: Petri Nets 2009. LNCS. Springer, Heidelberg (to appear, 2009)
18. Pratt, V.: Modelling Concurrency with Partial Orders. Int. Journal of Parallel Programming 15(1), 33–71 (1986)
19. Vogler, W.: Partial words versus processes: a short comparison. In: Rozenberg, G. (ed.) APN 1992. LNCS, vol. 609, pp. 292–303. Springer, Heidelberg (1992)
20. Schrijver, A.: Theory of linear and integer programming. Wiley, Chichester (1986)