# Models from Scenarios

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

[1] Department of Computer Science
University of Augsburg, Germany
robert.lorenz@informatik.uni-augsburg.de
[2] Department of Software Engineering
Distance University of Hagen, Germany
joerg.desel@fernuni-hagen.de
[3] Faculty of Electrical Engineering and Information Technology
Slovak University of Technology, Bratislava, Slovakia
gabriel.juhas@stuba.sk

**Abstract.** Synthesis of Petri nets from behavioral descriptions has important applications in the design of systems in different application areas. In this paper we present a survey on the technique of region based synthesis of Petri nets from languages. Each word in a given language specifies one run of the searched Petri net, i.e. represents one observable scenario of the system.

We concentrate on recent developments for languages of different kinds of causal structures (such as partial orders and stratified order structures). Causal structures represent causal relationships between events of one run. Expressible causal relationships are for example direct and indirect causal dependency, concurrency and synchronicity of events.

Concerning infinite languages, several possibilities of a finite representation are discussed. As the goal of synthesis, place/transition nets and inhibitor nets as well as several restrictions of these net classes are used. The presented framework integrates all classical results on sequential languages.

**Keywords:** Synthesis, Region Theory, Petri Net, Causal Semantics, Partial Language, Partial Order, Stratified Order Structure.

## 1 Introduction

Synthesis of Petri nets from behavioral descriptions has been a successful line of research since the 1990s. There is a rich body of nontrivial theoretical results and there are important applications in industry, in particular in hardware design [9,19], in control of manufacturing systems [33] and recently also in process mining [32,31,4,17] and workflow design [12,6].

The synthesis problem is the problem to construct, for a given behavioral specification, a Petri net such that the behavior of this net coincides with the specified behavior (if such a net exists). There are many different methods which are presented in literature to solve this problem. They differ mainly in the Petri net class and the model for

⋆ Supported by the German Research Council, project SYNOPS 2008 - 2012.

the behavioral specification considered. All these methods are based on one common theoretical concept, the notion of a *region* of the given behavioral specification.

In this paper, we present an overview of region-based synthesis methods, which regard languages as behavioral specifications, where each word in a given language specifies one run of the searched Petri net. Classical results consider sequential languages representing sequential runs of Petri nets. Recent developments examine languages of different kinds of causal structures (such as partial orders and stratified order structures) representing non-sequential runs. Such causal structures are able to represent different causal relationships between events of one run, such as for example direct and indirect causal dependency, concurrency and synchronicity.

In the following we describe the general approach of region based synthesis from languages. Denote the set of runs of a Petri net $N$ by $L(N)$. It depends on the Petri net class and the considered net semantics, which kind of runs are considered in $L(N)$. Formally the synthesis problem w.r.t. different Petri net classes and different language types is:

**Given:** A prefix-closed language $L$ over a finite alphabet of transition names $T$.
**Searched:** A Petri net $N$ with set of transitions $T$ and $L(N) = L$.

This means, we search for an exact solution of the problem. Such an exact a solution may not exist, i.e. not each language $L$ is a *net language*.

The classical idea of region-based synthesis is as follows: First consider the net $N$ having an empty set of places but all transitions occurring as labels in $L$. This net generates each execution in $L$ (i.e. $L \subseteq L(N)$), because there are no places restricting transition occurrences. But it generates much more executions. Since we are interested in an exact solution, we restrict $L(N)$ by adding places.

There are places $p$, which restrict the set of executions too much in the sense that $L \setminus L(N) \neq \emptyset$, if $p$ together with adjacent weighted arcs is added to $N$. Such places are called *non-feasible (w.r.t. $L$)*. We only add so called *feasible* places $p$ satisfying $L \subseteq L(N)$, if $p$ is added to $N$ (Figure 1). The idea of region-based synthesis is to add *all* feasible places to $N$. The resulting net $N_{sat}$ is called the *saturated feasible net*. $N_{sat}$ has by construction the following very nice property:

**(min)** $L(N_{sat})$ is the smallest net language satisfying $L \subseteq L(N_{sat})$.

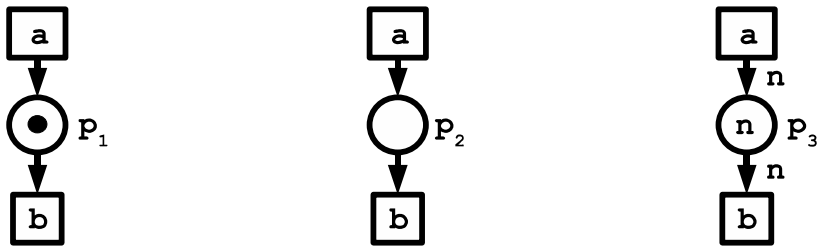



**Fig. 1.** The place $p_1$ is feasible, the place $p_2$ is not feasible w.r.t. the language $L = \{a, b, ab, ba, abb, bab\}$ ($b$ is no execution of the net shown in the middle). The place $p_3$ is feasible w.r.t. $L$ for each integer $n \in \mathbb{N}$

This is clear, since $L(N_{sat})$ could only be further restricted by adding non-feasible places. The property $(min)$ directly implies that there is an exact solution of the synthesis problem if and only if $N_{sat}$ is such an exact solution. Moreover, if there is no exact solution, $N_{sat}$ is the best approximation to such a solution "from above".

Unfortunately, this result is only of theoretical value, since the set of feasible places is in general *infinite* (Figure 1). Therefore, for a practical solution, a finite subset of the set of all feasible places is defined, such that the net $N_{fin}$ defined by this finite subset fulfills $L(N_{fin}) = L(N_{sat})$. Such a net $N_{fin}$ is called *finite representation* of $N_{sat}$. In order to construct such a finite representation, in an intermediate step a feasible place is defined through a so called *region* of the given language $L$, where the set of all regions equals the set of non-negative integral solutions of an appropriate linear system of the form $\mathbf{A}_L \cdot \mathbf{x} \leq \mathbf{b}_L$.

The described approach is common to all known region-based synthesis methods (see Figure 2), where different notions of regions and of finite representations $N_{fin}$ are used. There are two types of definitions of regions and two types of definitions of finite representations, whose four combinations cover all known region-based synthesis methods. All these combinations can be applied to almost each Petri net class and each language type (leading to different nets $N_{fin}$ having the same behavior).

Summarizing, the form of the synthesis problem and the solution method can be varied along the following lines: *Petri net class*, *language type*, *region type* and *finite representation type*. This paper presents a common framework for all these variations based on a combination of and extending the publications [27], [26] and [7].

The organisation of the paper is as follows: In the first part we develop a basic framework considering the synthesis of place/transition nets from finite and from simple infinite languages of labelled partial orders, using both region types and both finite representation types. For the finite specification of infinite languages a simple term based notation is used. In the second part we extend and generalize the basic framework along several lines:

- We consider the synthesis of inhibitor nets from finite and from simple infinite languages of labelled stratified order structures.
- We discuss synthesis from languages of non-transitive order structures.

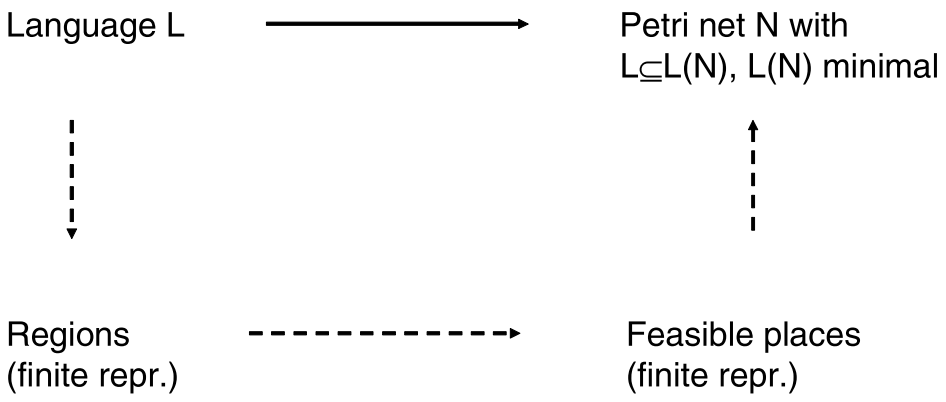



**Fig. 2.** The approach of region-based synthesis

- We examine the synthesis of nets of restricted net classes.
- We suggest several possibilities for a finite representation of more general infinite languages.

We do not consider labelled Petri nets (this is another line of research), and nets with invisible (internal) transitions or high level Petri nets (these are future topics of research).

This paper only gives a technical overview. Case studies and issues from practise are out of scope of this paper.

## 2 Basic Framework

In this section we present all main concepts by means of the synthesis of place/transition Petri nets from languages of labelled partial orders.

### 2.1 Mathematical Preliminiaries

In this subsection we present necessary notions and definitions including labelled partial orders, place/transition Petri nets and runs of place/transition Petri nets.

**Basic Notions.** By $\mathbb{N}_0$ we denote the set of *nonnegative integers*, by $\mathbb{N}$ the set of *positive integers*.

Given a function $f$ from $X$ to $Y$ and a subset $Z$ of $X$ we write $f|_Z$ to denote the *restriction* of $f$ to the set $Z$.

Given a finite set $X$, the symbol $|X|$ denotes the *cardinality* of $X$. The set of all subsets of $X$ is denoted by $\mathcal{P}(X)$.

The set of all *multisets* over a set $X$ is the set $\mathbb{N}^X$ of all functions $f : X \to \mathbb{N}$. Addition $+$ on multisets is defined by $(m + m')(x) = m(x) + m'(x)$. The relation $\leq$ between multiset is defined through $m \leq m' \iff \exists m''(m+m'' = m')$. We define $x \in m$ if $m(x) > 0$. A multiset is *finite*, if $\sum_{x \in X} m(x)$ is finite. A set $A \subseteq X$ is identified with the multiset $m$ satisfying $m(x) = 1 \iff x \in A \wedge m(x) = 0 \iff x \notin A$. The support of a multiset $m$ ist the set $set(m) = \{x \mid x \in m\}$. If $X$ is finite, a multiset $m$ we also write in the form of an $|X|$-tuple $(m(x)x)_{x \in X}$. For example, the finite multiset $m$ over $\{a, b, c\}$ defined by $m(a) = 1$ and $m(b) = 2$ we denote by $(1a, 2b, 0c)$. A multiset $m$ satisfying $m(a) > 0$ for exactly one element $a$ we call *singleton multiset* and denote it by $m(a)a$. The multiset $m$ satisfying $\forall x \in X : \ m(x) = 0$ we call *empty multiset* and denote it by $\epsilon$.

Let $X, T$ be sets and $l : X \to T$ be a labelling function assigning to each $x \in X$ a label $l(x)$ from $T$. Such a labelling function can be lifted to subsets $Y \subseteq X$ in the following way: $l(Y)$ is the multiset over $T$ given by $l(Y)(t) = |l^{-1}(t) \cap Y|$.

Given a binary relation $R \subseteq X \times Y$ and a binary relation $S \subseteq Y \times Z$ for sets $X, Y, Z$, then their composition is defined by $R \circ S = \{(x, z) \mid \exists y((x, y) \in R \wedge (y, z) \in S)\} \subseteq X \times Z$. For a binary relation $R \subseteq X \times X$ over a set $X$, we denote $R^1 = R$ and $R^n = R \circ R^{n-1}$ for $n \geq 2$. The symbol $R^+$ denotes the *transitive closure* $\bigcup_{n \in \mathbb{N}} R^n$ of $R$ and the symbol $R^*$ denotes the *reflexive transitive closure* $R^+ \cup \{(x, x) \mid x \in X\}$ of $R$. We also write $aRb$ to denote $(a, b) \in R$.

Let $A$ be a finite set of characters. A *(classical) language over* $A$ is a (possibly infinite) set of finite sequences of characters from $A$. For a language $L$ and $w \in L$, $|w|_a$ denotes the number of $a$'s occurring in $w$ (for example $|aba|_a = 2$). A *(concurrent) step over* $A$ is a multiset over $A$. A *step language over* $A$ is a (possibly infinite) set of finite sequences of steps over $A$. For a sequence of steps $w = \alpha_1 \dots \alpha_m$, $|w|_a = \sum_{i=1}^{m} \alpha_i(a)$ denotes the number of $a$'s occurring in $w$ (for example $|(1a, 0b)(0a, 2b)|_b = 2$).

**Partial Orders.** A *directed graph* is a pair $G = (V, \rightarrow)$, where $V$ is a finite *set of nodes* and $\rightarrow \subseteq V \times V$ is a binary relation over V, called the *set of edges* (all graphs considered in this paper are finite). The set of nodes of a directed graph $G$ is also denoted by $V(G)$. The *preset* of a node $v \in V$ is the set ${}^\bullet v = \{u \mid u \rightarrow v\}$. The *postset* of a node $v \in V$ is the set $v^\bullet = \{u \mid v \rightarrow u\}$. The *preset* of a subset $W \subseteq V$ is the set ${}^\bullet W = \bigcup_{w \in W} {}^\bullet w$. The *postset* of a subset $W \subseteq V$ is the set $W^\bullet = \bigcup_{w \in W} w^\bullet$. A *path* is a sequence of (not necessarily distinct) nodes $v_1 \dots v_n$ $(n > 1)$ such that $v_i \rightarrow v_{i+1}$ for $i = 1, \dots, n-1$. A path $v_1 \dots v_n$ is a *cycle*, if $v_1 = v_n$. A directed graph is called *acyclic*, if it has no cycles. The set of maximal nodes of an acyclic directed graph $G = (V, \rightarrow)$ is the set $Max(G) = \{v \mid v^\bullet = \emptyset\}$, the set of its minimal nodes is the set $Min(G) = \{v \mid {}^\bullet v = \emptyset\}$. An acyclic directed graph $(V, \rightarrow')$ is an *extension* of an acyclic directed graph $(V, \rightarrow)$ if $\rightarrow \subseteq \rightarrow'$. An acyclic directed graph $(V', \rightarrow)$ is a *prefix* of an acyclic directed graph $(V, \rightarrow)$ if $V' \subseteq V$ and $(v' \in V') \wedge (v \rightarrow v') \Rightarrow (v \in V')$. An acyclic directed graph $(V', \rightarrow)$ is a *sub-graph* of an acyclic directed graph $(V, \rightarrow)$ if $V' = U \setminus W$ for prefixes $(U, \rightarrow)$ and $(W, \rightarrow)$. Then $(W, \rightarrow)$ is called *prefix of the sub-graph* $(V', \rightarrow)$.

A *partial order* over a set $V$ is a binary relation $< \subseteq V \times V$ which is irreflexive $(\forall v \in V : v \not< v)$ and transitive $(<=<^+)$. We associate a finite partial order $<$ over $V$ with the directed graph $(V, <)$.

Two nodes $v, v' \in V$ 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* is a subset $C \subseteq V$ fulfilling $\forall x, y \in C : x \, co_< \, y$. A *cut* is a maximal co-set w.r.t. set inclusion. For a co-set $C$ of a partial order $(V, <)$ and a node $v \in V \setminus C$ we write $v < C$, if $v < s$ for an element $s \in C$ and $v \, co_< \, C$, if $v \, co_< \, s$ for all elements $s \in C$. The sets $Max(po)$ and $Min(po)$ are cuts.

The *skeleton* of a finite partial order $po = (V, <)$ is the minimal relation $\prec \subseteq <$ satisfying $\prec^+ = <$.

Graphically, nodes of partial orders are drawn as small squares and the relatrion by (drawn-through) arrows between nodes. Figure 3 shows an example partial order $po$. The nodes $v_1$ and $v_2$ as well as $v_3$ and $v_2$ are independent. It holds $Max(po) = \{v_2, v_3\}$ and $Min(po) = \{v_1, v_2\}$.

**Place/Transition Petri Nets.** A *net* is a 3-tuple $N = (P, T, F)$, where $P$ is a finite set of *places*, $T$ is a finite set of *transitions* disjoint from $P$ and $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*. A *marking* of a net assigns to each place $p \in P$ a number $m(p) \in \mathbb{N}_0$, i.e. a marking is a multiset over $P$. A *marked net* is a net $N = (P, T, F)$ together with an *initial marking* $m_0$. Graphically, places are drawn as circles, transitions as squares and the flow relation as arrows between places and transitions. A marking $m$ is illustrated by drawing $m(p)$ tokens inside place $p$.

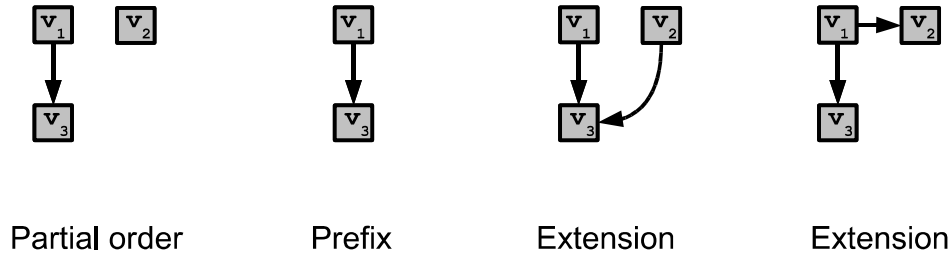



**Fig. 3.** Example of a partial order and a prefix and two extensions of this partial order

**Definition 1 (Place/Transition Petri Net).** *A* place/transition Petri net *(*PT-net*) is a 4-tuple* $N = (P, T, F, W)$*, where* $(P, T, F)$ *is a net and* $W : (P \times T) \cup (T \times P) \to \mathbb{N}_0$ *is a* weight function *satisfying* $W(x, y) > 0 \Leftrightarrow (x, y) \in F$.

Graphically, the number $W(x, y)$ is assigned to an arrow from $x$ to $y$, if $W(x, y) > 1$ (that means, $W(x, y) = 1$ for arrows $(x, y)$ without assigned weight). Figure 4 shows a marked PT-net with $P = \{p_1, p_2, p_3\}$, $T = \{a, b\}$, $m_0(p_1) = m_0(p_2) = 1$, $m_0(p_3) = 0$ and $W(p_1, a) = W(a, p_2) = W(p_2, b) = W(b, p_3) = 1$.

We introduce the following multisets of places:

- ${}^\bullet t(p) = W(p, t)$ and $t^\bullet(p) = W(t, p)$ for transitions $t$.
- ${}^\bullet \tau(p) = \sum_{t \in T} \tau(t)\, {}^\bullet t(p)$ and $\tau^\bullet(p) = \sum_{t \in T} \tau(t) t^\bullet(p)$ for multisets of transitions $\tau$.

The definition of executions of PT-nets depends on the *occurrence rule* of transitions, stating in which markings a transition (or a multiset of transitions) can occur and how these markings are changed by its occurrence.

**Definition 2 (Occurrence Rule).** *A transition* $t \in T$ can occur *in a marking* $m$*, if* $m \geq {}^\bullet t$*. A multiset of transitions* $\tau$ can occur *in* $m$*, if* $m \geq {}^\bullet \tau$.

*If a transition* $t$ *occurs in a marking* $m$*, the resulting marking* $m'$ *is defined by* $m' = m - {}^\bullet t + t^\bullet$*. If a multiset of transitions* $\tau$ *occurs in* $m$*, then the resulting marking* $m'$ *is defined by* $m' = m - {}^\bullet \tau + \tau^\bullet$*. We write* $m \xrightarrow{t} m'$ *(*$m \xrightarrow{\tau} m'$*) to denote that* $t$ *(*$\tau$*) can occur in* $m$ *and that its occurrence leads to* $m'$.

The number $W(p, t)$ represents the number of tokens *consumed from* $p$ by an occurrence of $t$ and the number $W(t, p)$ represents the number of tokens *produced in* $p$ by an occurrence of $t$.

The occurrence of a multiset of transitions $\tau$ in a marking $m$ means, that all transitions in $\tau$ occur in parallel.

The notion of *execution* depends on the chosen net semantics. In the following definition we consider sequential semantics and step semantics. Causal semantics is defined in the next subsection.

**Definition 3 (Execution).** *A* sequential execution in $m$ *of a PT-net is a finite sequence of transitions* $\sigma = t_1 \dots t_n$ *such that there are markings* $m_1, \dots, m_n$ *satisfying* $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} m_n$.

*A* step execution in $m$ *of a PT-net is a finite sequence of multisets of transitions* $\sigma = \tau_1 \dots \tau_n$ *such that there are markings* $m_1, \dots, m_n$ *satisfying* $m \xrightarrow{\tau_1} m_1 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_n} m_n$.

*We write* $m \xrightarrow{\sigma} m_n$ *to denote the occurrence of such executions* $\sigma$.

Each sequential execution is also a step execution. The markings which can be reached from the initial marking via sequential executions (resp. step executions) are called *reachable*.

The PT-net shown in Figure 4 has the sequential executions $a, b, ab, ba, abb, bab$ and the additional step execution $(1a, 1b)(0a, 1b), (1a, 0b)(0a, 2b)$ in the initial marking.

If $\tau$ is a multiset of transitions which can occur in a marking $m$ and $\tau = t_1 + \dots + t_n$ for transitions $t_1, \dots, t_n$, then $t_1 \dots t_n$ is a sequential execution in $m$, i.e. the transitions in $\tau$ can occur in $m$ in arbitrary sequential order.

Finally, we recall process semantics of PT-nets.

**Definition 4 (Occurrence Net).** *An* occurrence net *is a net* $O = (B, E, G)$ *satisfying:*

- $B$ *and* $E$ *are finite and disjoint sets.*
- $G \subseteq (B \times E) \cup (E \times B)$.
- $(B \cup E, G)$ *is a directed acyclic graph.*
- $\forall b \in B(|{}^\bullet b| \leq 1 \wedge |b^\bullet| \leq 1)$.

*The elements of* $B$ *are called* conditions *and the elements of* $E$ *are called* events. *The relation* $G$ *is called* flow relation.

Since an occurrence net can be identified with an acyclic directed graph, we use notations introduced for acyclic directed graphs also for occurrence nets. A *slice* of an occurrence net is a cut consisting solely of conditions.

In a process, the events of an occurrence net are interpreted as transition occurrences of a PT-net. Conditions represent tokens in places.

**Definition 5 (Process).** *Let* $N = (P, T, F, W, m_0)$ *be a marked PT-net. A* process *of* $N$ *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*

- $\rho(B) \subseteq P$ *and* $\rho(E) \subseteq T$.
- $\forall e \in E : \ \rho({}^\bullet e) = {}^\bullet \rho(e) \wedge \rho(e^\bullet) = \rho(e)^\bullet$.
- $\rho(Min(O)) = m_0$.

In a process of a PT-net, two transition occurrences are *directly causally dependent* if one transition occurrence $e'$ consumes tokens which are produced by the other transition occurrence $e$. Such a situation is called *token flow* between transition occurrences and can be directly observed in a process via $e^\bullet \cap {}^\bullet e' \neq \emptyset$. Figure 4 shows a process of a PT-net, where names of conditions are omitted. The names of events are shown inside, the labels of events and conditions outside of the graphical object. In this process, $v_1$ and $v_3$ are directly causally dependent.

For each slice $C$ of a process, $\rho(C)$ is a reachable marking of the net. On the other hand, for each reachable marking $m$ there is a slice $C$ in some process such that $\rho(C) = m$.

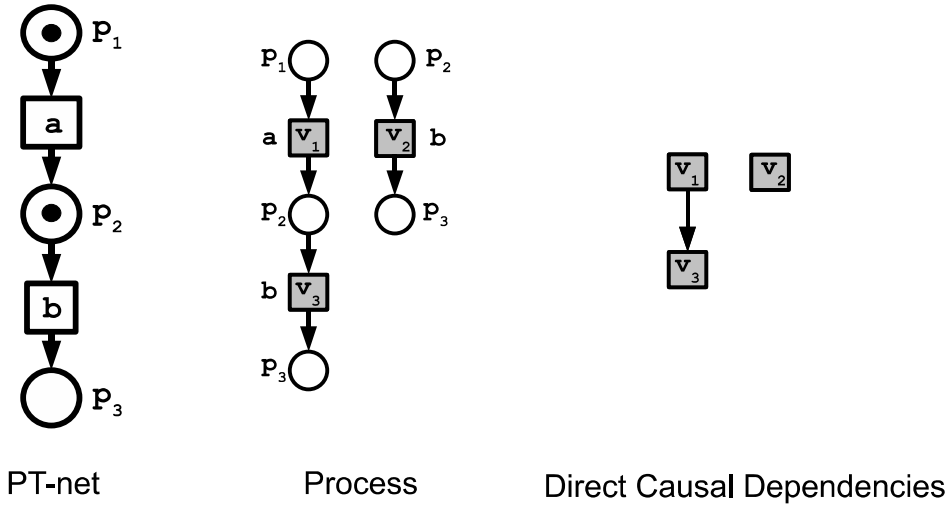



**Fig. 4.** Example of a PT-net and one of its processes

## 2.2 Causal Semantics

We use partial orders labelled by transition names to represent single (non-sequential) runs of PT-nets. The nodes of a partial order represent transition occurrences and its arrows an "earlier than"-relation between transition ocurrences in the sense that one transition occurrence can be observed earlier than another transition occurrence. If there are no arrows between two transition occurrences, then these transition occurrences are independend and are called *concurrent*. Concurrent transition occurrences can be observed in arbitrary sequential order and in parallel. This interpretation of arrows is called *occurrence interpretation*.

**Definition 6 (Labelled Partial Order).** *A* labelled partial order *(*LPO*)* over $T$ *is a 3-tuple* $(V, <, l)$*, where* $(V, <)$ *is a partial order and* $l : V \to T$ *is a labelling function on* $V$*.*

We only consider LPOs up to isomorphism, i.e. only the labelling of events is of interest, but not the event names. Formally, two LPOs $(V, <, l)$ and $(V', <', l')$ are *isomorphic*, if there is a renaming function $I : V \to V'$ satisfying $l(v) = l'(I(v))$ and $v < w \Leftrightarrow I(v) <' I(w)$.

A *linear order* is an LPO $(V, <, l)$ where $<$ is a total order, i.e. there is no independence between transition occurrences: $\forall u, v \in V : \; u < v \vee v < u$. Linear orders represent sequential executions of Petri nets in the obvious way. For example, the LPO $lpo_4$ shown in Figure 5 is linear and represents the sequential execution $abb$.

A *stepwise linear LPO* is an LPO $(V, <, l)$ where the relation $co_<$ is transitive. The maximal sets of independent transition occurrences are called *steps*. The steps of a stepwise linear LPOs are linearly ordered. Thus, stepwise linear LPOs represent step executions of Petri nets. For example, the LPO $lpo_1$ shown in Figure 5 is not stepwise linear, while the LPOs $lpo_2$ (representing the step execution $(1a, 1b)(0a, 1b)$) and $lpo_3$ (representing the step execution $(1a, 0b)(0a, 2b)$) are stepwise linear.

The *set of step-linearizations* of an LPO is the set of stepwise linear LPOs which are extensions of this LPO. For example, the LPOs $lpo_2$ and $lpo_3$ shown in Figure 5 are step linearizations of $lpo_1$.

**Definition 7 (LPO-run).** *Let $N = (P, T, F, W, m_0)$ be a PT-net. An LPO $(V, <, l)$ is a* LPO-run *of $N$ if there is a process $K = (O, \rho)$, $O = (B, E, G)$, of $N$ such that $(V, <)$ is an extension of $(E, \{(e, f) \mid e^\bullet \cap {}^\bullet f \neq \emptyset\})$ and $l = \rho|_E$.*

*An LPO-run $lpo$ of $N$ is said to be* minimal*, if there exists no other LPO-run $lpo'$ of $N$ such that $lpo$ is an extension of $lpo'$.*

Note that $(E, \{(e, f) \mid e^\bullet \cap {}^\bullet f \neq \emptyset\})$ is an acyclic directed graph representing all direct causal dependencies between transition occurrences of a process of the net. This means, along the "earlier than"-relations between transition occurrences of an LPO-run token flow is allowed, but not required. Figure 5 shows a PT-net together with some of its LPO-runs. Note that the LPO-run $lpo_1$ exactly represents all direct causal dependencies between transition occurrences of a process of the net (which is shown in Figure 4). Moreover, $lpo_1$ is minimal, since a second occurrence of $b$ must be preceeded by an occurrence of $a$.

From the definition follows that extensions of LPO-runs also are LPO-runs. This means, the set of all LPO-runs can be deduced from the set of minimal LPO-runs.

There are two alternative but equivalent definitions of LPO-runs in literature:

- An LPO $lpo = (V, <, l)$ is an LPO-run of a PT-net $N$ if and only if each step-linearization of $lpo$ is a step execution of $N$. This means, LPO-runs are consistent with the step semantics of PT-nets.
- An LPO $lpo = (V, <, l)$ is an LPO-run if and only if for each cut $C$ of $lpo$ and each place $p$ there holds:

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

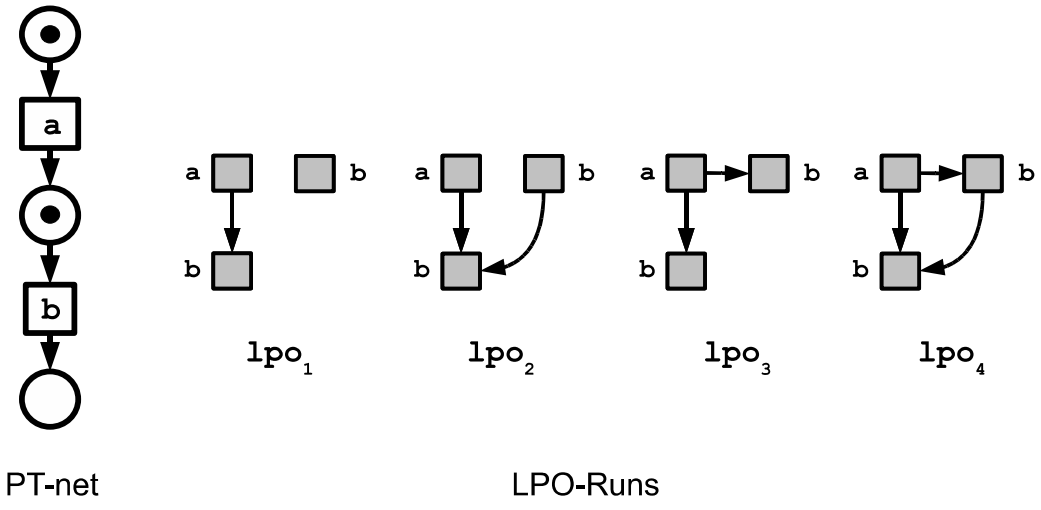



**Fig. 5.** A PT-net with four of its LPO-runs. The LPOs $lpo_2$, $lpo_3$ and $lpo_4$ are step linearizations of $lpo_1$. The LPO-run $lpo_1$ is minimal. The LPO $lpo_4$ is linear.

This means, after the occurrence of each prefix of $lpo$ there are enough tokens for the occurrence of the multiset of transitions occurrences directly following the prefix.

In figures we often omit transitive arrows of LPOs for a clearer presentation.

### 2.3 Regions of Finite Languages

The formal problem statement, which we consider from now, is:

**Given:** A prefix-closed and extension-closed finite language $L$ of LPOs over a finite alphabet of transition names $T$.
**Searched:** A PT-net $N$ with set of transitions $T$ such that all LPOs in $L$ are LPO-runs of $N$ and $N$ has a minimal number of additional LPO-runs.

As explained in the introduction, for the computation of places of $N$ so-called regions are defined. In this subsection we define two different types of PT-net regions of finite languages of LPOs as non-negative integral solutions of appropriate linear systems of the form $\mathbf{A}_L \cdot \mathbf{x} \leq \mathbf{b}_L$. For these definitions and in examples we only consider those LPOs from $L$, which are not extensions or prefixes of other LPOs from $L$. If a place is feasible w.r.t. these LPOs, then this place is feasible w.r.t. $L$, since the set of LPO-runs of a PT-net is prefix- and extension-closed. Throughout the rest of this subsection we use the language shown in Figure 6 as a running example. It is enough to consider the LPOs $lpo_1$ and $lpo_2$, since the other LPOs are prefixes or extensions of $lpo_1$.

**Transition-Regions.** A *(PT-net) transition-region* $\mathbf{r}$ directly defines the parameters of a place $p_\mathbf{r}$ of PT-nets, i.e. it determines the numbers $m_0(p_\mathbf{r})$ and $W(p_\mathbf{r}, t)$ and $W(t, p_\mathbf{r})$ for each $t \in T$. If $T = \{t_1, \ldots, t_n\}$, then $\mathbf{r}$ is given as a $(2n + 1)$-tuple

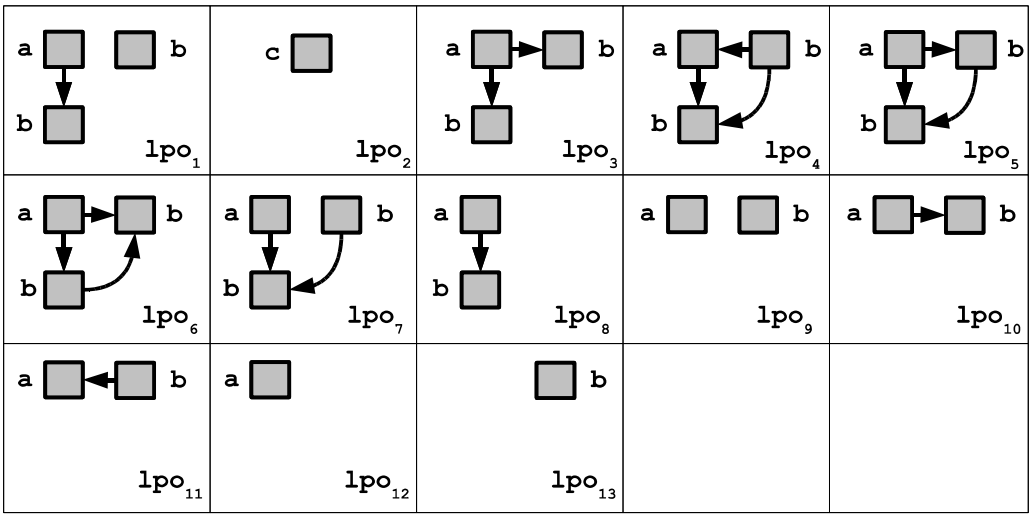



**Fig. 6.** Running example language

$\mathbf{r} = (r_0, \ldots, r_{2n})$ of non-negative integers. Its components define these numbers via $m_0(p_\mathbf{r}) = r_0$, $W(p_\mathbf{r}, t_i) = r_i$ and $W(t_i, p_\mathbf{r}) = r_{n+i}$ for $i \in \{1, \ldots, n\}$. In the running example, denote $t_1 = a$, $t_2 = b$ and $t_3 = c$.

Since a region $\mathbf{r}$ is intended to define a *feasible* place $p_\mathbf{r}$, it is required to satisfy a property $(f)_L$ ensuring that $p_\mathbf{r}$ is feasible w.r.t. $L$. Remember that $p_\mathbf{r}$ is feasible w.r.t. $L$ if the net resulting from adding $p_\mathbf{r}$ still generates at least $L$. For this, the property $(f)_L$ formalizes that for each cut of events there are enough tokens in $p_\mathbf{r}$ for the occurrence of the corresponding step of transitions after the occurrence of the prefix preceeding the cut (which can be the empty prefix). For example, in the running example the transition step $(1a, 1b)$ must be able to occur after the empty prefix, i.e. in the initial marking (see Figure 7). This means, $p_\mathbf{r}$ has to satisfy $m_0(p_\mathbf{r}) \geq W(p_\mathbf{r}, a) + W(p_\mathbf{r}, b)$, i.e. $r_0 \geq r_1 + r_2$.

The definition of $(f)_L$ for a finite language $L$ of LPOs and PT-nets is as follows: For each $lpo = (V, <, l) \in L$ and for each cut $C$ of $lpo$ we require

$$r_0 + \sum_{i=1}^{n} l(V')(t_i)(r_{n+i} - r_i) - \sum_{i=1}^{n} l(C)(t_i) r_i \geq 0,$$

where $V' = \{v \in V \mid v < C\}$. This is the case if and only if (for each $lpo \in L$ and for each cut $C$ of $lpo$) $\mathbf{a}_{lpo,C} \cdot \mathbf{r} \leq 0$ for $\mathbf{a}_{lpo,C} = (a_{C,0}, \ldots, a_{C,2n})$ defined by:

$$\mathbf{a}_{C,j} = \begin{cases} -1 & \text{if } j = 0, \\ l(V' \cup C)(t_j) & \text{if } j \in \{1, \ldots, n\}, \\ -l(V')(t_{j-n}) & \text{if } j \in \{n+1, \ldots, 2n\}. \end{cases}$$

For the cut $C$ corrsponding to the transition step $(1a, 1b)$ in the running example we require $r_0 - r_1 - r_2 \geq 0$. This is the case if and only if $\mathbf{a}_{lpo1,C} \cdot \mathbf{r} \leq 0$ for

$$\mathbf{a}_{lpo1,C} = (-1, 1, 1, 0, 0, 0, 0).$$

**Definition 8 (Transition-Region).** *A tuple* $\mathbf{r}$ *as above is called a* transition-region *if it satisfies* $(f)_L$.

**Theorem 1 ([27]).** *A tuple* $\mathbf{r}$ *satisfies* $(f)_L$ *if and only if* $p_\mathbf{r}$ *is feasible w.r.t.* $L$.

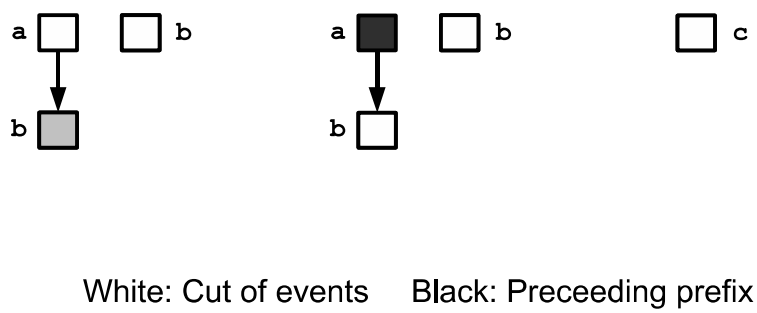



**Fig. 7.** All cuts of the LPOs $lpo_1$ and $lpo_2$ together with their preceeding prefixes

Let $\mathbf{A}_L$ be the matrix consisting of all rows $\mathbf{a}_{lpo,C}$ for LPOs $lpo \in L$ and cuts $C$ of $lpo$. Since $L$ is assumed to be finite, $\mathbf{A}_L$ is finite. Thus, the set of all regions can be computed as the set of all integral solutions of the homogenous linear inequation system $\mathbf{A}_L \cdot \mathbf{x} \leq \mathbf{0}$. In the running example, $\mathbf{A}_L$ looks as follows (compare Figure 7):

$$\begin{pmatrix} -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 2 & 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Solutions are for example $\mathbf{r} = (1,1,0,1,0,0,0)$ with corresponding place $p_1$, $\mathbf{r} = (1,0,1,1,1,0,0)$ with corresponding place $p_2$ and $\mathbf{r} = (0,0,0,0,0,1,0)$ with corresponding place $p_3$ in Figure 8.

**Theorem 2 ([27]).** *If $L$ is finite then there is a finite matrix $\mathbf{A}_L$ such that the set of transition-regions is the set of solutions of the linear inequation system $\mathbf{A}_L \cdot \mathbf{x} \leq \mathbf{0}$.*

**Token Flow Regions.** A *token flow-region* $\mathbf{r}$ defines a place $p_{\mathbf{r}}$ indirectly by determining the *token flow* w.r.t. this place between transition occurrences in LPOs from $L$, i.e. by directly determining the number of tokens produced by a transition occurrence which are consumed by a subsequent transition occurrence in an LPO specified in $L$.

Such numbers are assigned to the arrows between transition occurrences of LPOs. Moreover, for each transition occurrence the number of tokens consumed from the initial marking and the number of tokens which are produced but not further consumed by other transition occurrences are considered. Finally, there may be tokens in the initial marking which are not consumed by any transition occurrence of an LPO.

If $W = \bigcup_{(V,<,l)\in L} V$ is the set of nodes of LPOs in $L$ and $E = \bigcup_{(V,<,l)\in L} <$ is the set of arrows of LPOs in $L$, then a *(PT-net) token flow-region* $\mathbf{r}$ is given as a tuple $\mathbf{r} = (r_i)_{i\in W\times\{in,out\}\cup E\cup L}$ of non-negative integers. Its components define

- the number of tokens an event $v \in W$ consumes from the initial marking by $r_{v,in}$,
- the number of tokens produced by an event $v$ and not consumed by a subsequent event by $r_{v,out}$,

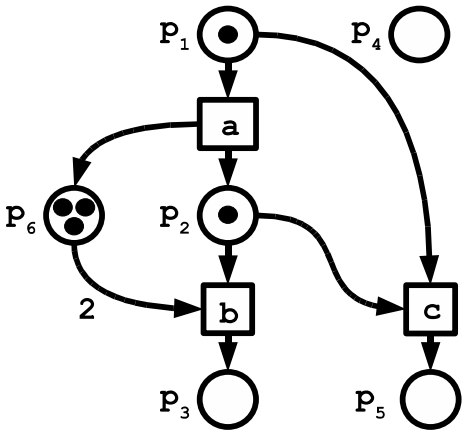



**Fig. 8.** Some solution places for the example language

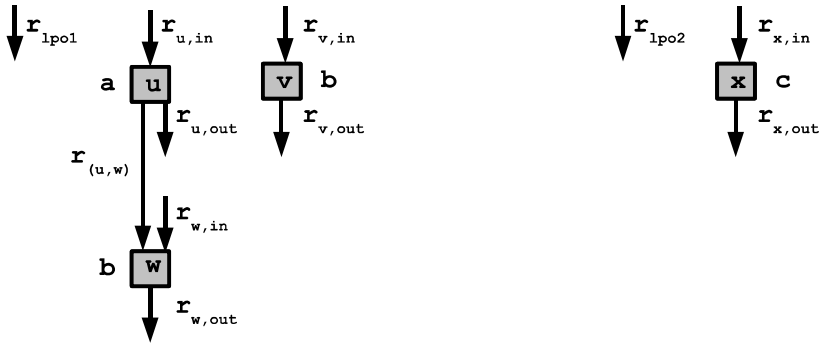



**Fig. 9.** Illustration of token flow regions for the running example

- the number of tokens produced by an event $v$ and consumed by an event $w$ for $e = (v, w) \in E$ by $r_e$,
- the number of tokens in the initial marking, which are not consumed by any subsequent event of an LPO $lpo \in L$ by $r_{lpo}$.

Figure 9 illustrates all these numbers for the running example. For example, $r_{u,in}$ represents the number of tokens, this occurrence of transition $a$ consumes from the initial marking of the represented place. The number $r_{(u,w)}$ represented the number of tokens, which are produced by transition $a$ in the represented place and then consumed by the following transition occurrence of $b$. The number $r_{u,out}$ represents the number of tokens, this occurrence of transition $a$ produces in the the represented place and which are not consumed by further transitions. For the running example, we denote

$$\mathbf{r} = (r_{lpo1}, r_{lpo2}, r_{u,in}, r_{v,in}, r_{w,in}, r_{x,in}, r_{u,out}, r_{v,out}, r_{w,out}, r_{x,out}, r_{(u,w)}).$$

A token-flow region $\mathbf{r}$ defines a PT-net-place $p_{\mathbf{r}}$ as follows:

- $m_0(p_{\mathbf{r}}) = r_{lpo} + \sum_{v \in V} r_{v,in}$ for some LPO $lpo = (V, <, l) \in L$ – the sum is called *initial token flow* of $lpo$. The initial token flow of $lpo_1$ equals $r_{lpo1} + r_{u,in} + r_{v,in} + r_{w,in}$.
- $W(p_{\mathbf{r}}, t) = r_{v,in} + \sum_{e=(u,v) \in E} r_e$ for some LPO $lpo = (V, <, l) \in L$ and $v \in V$ with $l(v) = t$ – the sum is called *intoken flow* of $v$. The intoken flow event $w$ is $r_{(u,w)} + r_{w,in}$.
- $W(t, p_{\mathbf{r}}) = r_{v,out} + \sum_{e=(v,u) \in E} r_e$ for some LPO $lpo = (V, <, l) \in L$ and $v \in V$ with $l(v) = t$ – the sum is called *outtoken flow* of $v$. The outtoken flow event $u$ computes $r_{(u,w)} + r_{u,out}$.

This construction is still dependent on the choice of $lpo = (V, <, l) \in L$ and $v \in V$, thus $p_{\mathbf{r}}$ is not uniquely defined. Therefore, we require $\mathbf{r}$ to fulfill a property $(wd)_L$ which makes $p_{\mathbf{r}}$ defined independently from the choice of $lpo = (V, <, l) \in L$ and $v \in V$. The property $(wd)_L$ states the following:

– *The initial token flows of different LPOs are equal*:

$$r_{lpo} + \sum_{v \in V} r_{v,in} = r_{lpo'} + \sum_{v' \in V'} r_{v',in}$$

for LPOs $lpo = (V, <, l), lpo' = (V', <', l')$ from $L$. This property can be expressed as a linear equation system $\mathbf{A}_{L,a} \cdot \mathbf{r} = \mathbf{0}$ as follows:
Let $L = \{lpo_1, lpo_2, \ldots, lpo_n\}$ and $lpo_k = (V_k, <_k, l_k)$. Then, for each $k \geq 2$, the matrix $\mathbf{A}_{L,a}$ has a row $\mathbf{a}_k = (a_{k,i})_{i \in W \times \{in,out\} \cup E \cup L}$ defined by

$$\mathbf{a}_{k,i} = \begin{cases} 1 & \text{if } i = (v, in) \text{ for } v \in V_k, \\ 1 & \text{if } i = lpo_k, \\ -1 & \text{if } i = (v', in) \text{ for } v' \in V_{k-1} \\ -1 & \text{if } i = lpo_{k-1}, \\ 0 & \text{else.} \end{cases}$$

For example, the two LPOs of the running example shown in Figure 9 should have the same initial token flow, i.e. we require

$$r_{lpo1} + r_{u,in} + r_{v,in} + r_{w,in} = r_{lpo2} + r_{x,in}.$$

This is satisfied if and only if $\mathbf{a}_2 \cdot \mathbf{r} = \mathbf{0}$ for

$$\mathbf{a}_2 = (-1, 1, -1, -1, -1, 1, 0, 0, 0, 0, 0).$$

– *The intoken flows of equally labelled events are equal*:

$$r_{v,in} + \sum_{e=(u,v) \in <} r_e = r_{v',in} + \sum_{e=(u,v') \in <'} r_e$$

for $v \in V$, $v' \in V'$, $(V, <, l), (V', <', l') \in L$ and $l(v) = l'(v')$. This property can be expressed as a linear equation system $\mathbf{A}_{L,b} \cdot \mathbf{r} = \mathbf{0}$ as follows: Let $W_t = \{v \in W \mid l(v) = t\} = \{v_1^t, v_2^t, \ldots, v_n^t\}$ be the set of all $t$-labeled events for $t \in T$ . Then, for each $t$ and each $k \geq 2$, the matrix $\mathbf{A}_{L,b}$ has a row $\mathbf{b}_k^t = (b_{k,i}^t)_{i \in W \times \{in,out\} \cup E \cup L}$ defined by

$$\mathbf{b}_{k,i}^t = \begin{cases} 1 & \text{if } i = (v_k^t, in) \vee i = (u, v_k^t), \\ -1 & \text{if } i = (v_{k-1}^t, in) \vee i = (u, v_{k-1}^t) \\ 0 & \text{else.} \end{cases}$$

For example, the two occurrences $v$ and $w$ of transition $b$ shown in Figure 9 should have the same intoken flow, i.e. we require

$$r_{v,in} = r_{w,in} + r_{(u,w)}.$$

If we denote $v_1^b = v$ and $v_2^b = w$, this is satisfied if and only if $\mathbf{b}_2^b \cdot \mathbf{r} = \mathbf{0}$ for

$$\mathbf{b}_2^b = (0, 0, 0, -1, 1, 0, 0, 0, 0, 0, 1).$$

– *The outtoken flows of equally labeled events are equal*:

$$r_{v,out} + \sum_{e=(v,u)\in <} r_e = r_{v',out} + \sum_{e=(v',u)\in <'} r_e$$

for $v \in V$, $v' \in V'$, $(V, <, l), (V', <', l') \in L$ and $l(v) = l'(v')$. This property can be expressed as a linear equation system $\mathbf{A}_{L,c} \cdot \mathbf{r} = \mathbf{0}$ as follows: For each $t$ and each $k \geq 2$ the matrix $\mathbf{A}_{L,c}$ has a row $\mathbf{c}^t_k = (c^t_{k,i})_{i\in W\times\{in,out\}\cup E\cup L}$ defined by

$$\mathbf{c}^t_{k,i} = \begin{cases} 1 & \text{if } i = (v^t_k, out) \vee i = (v^t_k, u), \\ -1 & \text{if } i = (v^t_{k-1}, out) \vee i = (v^t_{k-1}, u) \\ 0 & \text{else.} \end{cases}$$

For example, the two occurrences $v$ and $w$ of transition $b$ shown in Figure 9 should have the same outtoken flow, i.e. we require

$$r_{v,out} = r_{w,out}.$$

If we denote $v^b_1 = v$ and $v^b_2 = w$, this is satisfied if and only if $\mathbf{c}^b_2 \cdot \mathbf{r} = \mathbf{0}$ for

$$\mathbf{c}^b_2 = (0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0).$$

It can be shown that the place $p_{\mathbf{r}}$ is feasible w.r.t. $L$ by construction (it is possible to construct a process of the synthesized net directly from token flows).

**Definition 9 (Token Flow-Region).** *A tuple* $\mathbf{r}$ *as above is called a* token flow-region *if it satisfies* $(wd)_L$.

For the property $(wd)_L$ the following theorem holds for PT-net places [5]:

**Theorem 3.** *A tuple* $\mathbf{r}$ *satisfies* $(wd)_L$ *if and only if* $p_{\mathbf{r}}$ *is feasible w.r.t.* $L$.

Let $\mathbf{A}_L$ be the matrix consisting of all rows from the matrices $\mathbf{A}_{L,a}$, $\mathbf{A}_{L,b}$ and $\mathbf{A}_{L,c}$. Since $L$ is assumed to be finite, $\mathbf{A}_L$ is finite. Thus, the set of all token flow-regions can be computed as the set of all integral solutions of the homogenous linear equation system $\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$. If we denote $v^b_1 = v$ and $v^b_2 = w$, $\mathbf{A}_L$ looks as follows for the running example (the matrices $\mathbf{A}_{L,a}$, $\mathbf{A}_{L,b}$ and $\mathbf{A}_{L,c}$ each consist exactly of the one row already shown):

$$\begin{pmatrix} -1 & 1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \end{pmatrix}$$

Figure 10 shows a solution. This solution represents the place $p_2$ from Figure 8.

**Theorem 4 ([5,27]).** *If* $L$ *is finite then there is a finite matrix* $\mathbf{A}_L$ *such that the set of token flow-regions is the set of solutions of the linear equation system* $\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$.

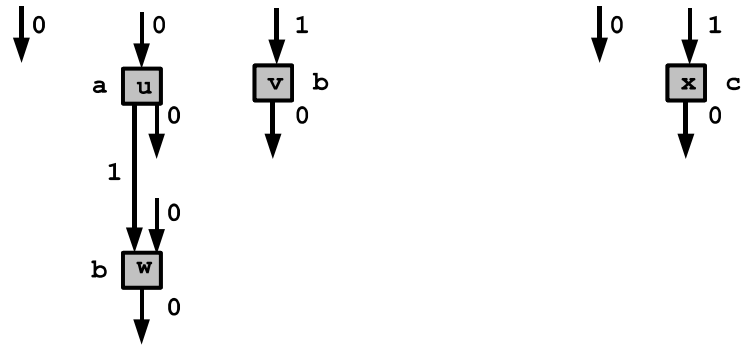



**Fig. 10.** Illustration of the token flow region $\mathbf{r} = (0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1)$

## 2.4 Finite Representations

As shown in the last paragraph, the set of regions can be computed as the set of non-negative integer solutions of a homogenuous equation ot inequation system. Such systems have an infinite set of solutions. In this subsection we present two possibilities to finitely represent this set of solutions.

**Separating Representation.** One idea to derive a finite representation is to separate behavior specified in $L$ from behavior not specified in $L$ by a finite set of regions. For this, one defines a finite set of executions $L^c$ with $L \cap L^c = \emptyset$ satisfying that $L(N) \cap L^c = \emptyset \implies L(N) = L$ for each net $N$. Then for each $w \in L^c$ one tries to find a region $\mathbf{r}(w)$ such that $w$ is not an execution of the net having the place $p_{\mathbf{r}(w)}$, i.e. a region which *separates $L$ from $w$*. The elements of $L^c$ are called *wrong continuations*. If such a region exists, then the corresponding place is added to the net $N_{sep}$ called *separating representation* of $N_{sat}$.

There is an exact solution of the synthesis problem if and only if for each $w \in L^c$ there is such a region $\mathbf{r}(w)$. In case $L$ is a net language (of the considered net class), it holds $L(N_{sep}) = L(N_{sat}) = L$, i.e $N_{sep}$ is a possible solution.

If $L$ is not a net language, $L^c$ does not have in general the property $L(N_{sat}) = L(N_{sep})$. One common approach is to define a wrong continuation $w$ as an LPO extending an LPO of $L$ by one event. If there is no place separating $L$ from such a wrong continuation $w$, this does not mean that there are no places separating $L$ from further "continuations" of $w$. In order to achieve $L(N_{sat}) = L(N_{sep})$, also all continuations of wrong continuations which cannot be separated from $L$ must be considered. In general, there is no *finite* set $L^c$ with $L \cap L^c = \emptyset$ satisfying $L(N_{sat}) = L(N_{sep})$ [10]. Thus, the separating representation is not necessarily the best approximation to a solution of the synthesis problem generating $L$.

In the following we construct a finite set $L^c$. Remember that $lpo \in L$ is a run of a net $N$ if and only if each step linearization of $lpo$ is a step execution of $N$. Denote by $L^{step}$ the set of step linearizations of LPOs in $L$. In order to define wrong continuations we extend elements from $L^{step}$ by one event as follows:

**Definition 10 (Wrong Continuation of LPO-languages).** *Let* $\sigma = \alpha_1 \dots \alpha_{n-1}\alpha_n \in L^{step}$ *and* $t \in T$ *such that* $w_{\sigma,t} = \alpha_1 \dots \alpha_{n-1}(\alpha_n + t) \notin L^{step}$*, where* $\alpha_n$ *is allowed to be the empty step. Then* $w_{\sigma,t}$ *is called* wrong continuation of $L$.

*We call* $\alpha_1 \dots \alpha_{n-1}$ *the* prefix *and* $\alpha_n + t$ *the* follower step of the wrong continuation.

The following table lists $L^{step}$ together all wrong continuations of the running example (multisets are denoted as sums of singleton multisets):

| | | | | | |
|---|---|---|---|---|---|
| $(a+b)b\epsilon$ | $(a+b)ba$<br>$(a+b)bb$<br>$(a+b)bc$ | $abb\epsilon$ | $abba$<br>$abbb$<br>$abbc$ | $bab\epsilon$ | $baba$<br>$babb$<br>$babc$ |
| $a(2b)\epsilon$ | $a(2b)a$<br>$a(2b)b$<br>$a(2b)c$ | $(a+b)\epsilon$ | $(a+b)a$<br>$(a+b)c$ | $ab\epsilon$ | $aba$<br>$abc$ |
| $ba\epsilon$ | $baa$<br>$bac$ | $(a+b)b$ | $(a+b)(b+a)$<br>$(a+b)(2b)$<br>$(a+b)(b+c)$ | $abb\epsilon$ | $ab(b+a)$<br>$ab(2b)$<br>$ab(b+c)$ |
| $bab$ | $ba(b+a)$<br>$ba(2b)$<br>$ba(b+c)$ | $a\epsilon$ | $aa$<br>$ac$ | $ab$ | $a(b+a)$<br>$a(b+c)$ |

| | | | | | |
|---|---|---|---|---|---|
| $a(2b)$ | $a(2b+a)$<br>$a(3b)$<br>$a(2b+c)$ | $b\epsilon$ | $bb$<br>$bc$ | $ba$ | $b(2a)$<br>$b(a+b)$<br>$b(a+c)$ |
| $c\epsilon$ | $ca$<br>$cb$<br>$cc$ | $\epsilon\epsilon$ | | $\epsilon a$ | $\epsilon(2a)$<br>$\epsilon(a+c)$ |
| $\epsilon b$ | $\epsilon(2b)$<br>$\epsilon(b+c)$ | $\epsilon c$ | $\epsilon(c+a)$<br>$\epsilon(c+b)$<br>$\epsilon(2c)$ | $\epsilon(a+b)$ | $\epsilon(2a+b)$<br>$\epsilon(a+2b)$<br>$\epsilon(a+b+c)$ |

To prohibit a wrong continuation, one needs to find a feasible place $p$ such that after occurrence of its prefix there are not enough tokens in $p$ to fire its follower step. A a prefix of wrong continuations corresponds to a prefix $(V', <, l)$ of an LPO $lpo = (V, <, l) \in L$, which is stepwise linearized by $\alpha_1 \dots \alpha_{n-1}$. A follower step of such a prefix can be constructed by taking a subset $S$ of its direct successors $\{v \in V \setminus V' \mid u < v \implies u \in V'\}$ and add a labelled event $z$ parallel to this subset. That means, wrong continuations can be represented on the level of LPOs, where wrong continuations having the same follower step and whose prefixes stepwise linearize the same LPO-prefix need not be distinguished. Figure 11 shows some representations of wrong continuations of the running example.

Since the follower marking after the occurrence of $(V', <, l)$ only depends on the number of occurrences of each transition in $V'$, but not on their ordering, it is enough to represent $(V', <, l)$ by the multiset $l(V')$. Altogether, the set of all wrong continuations

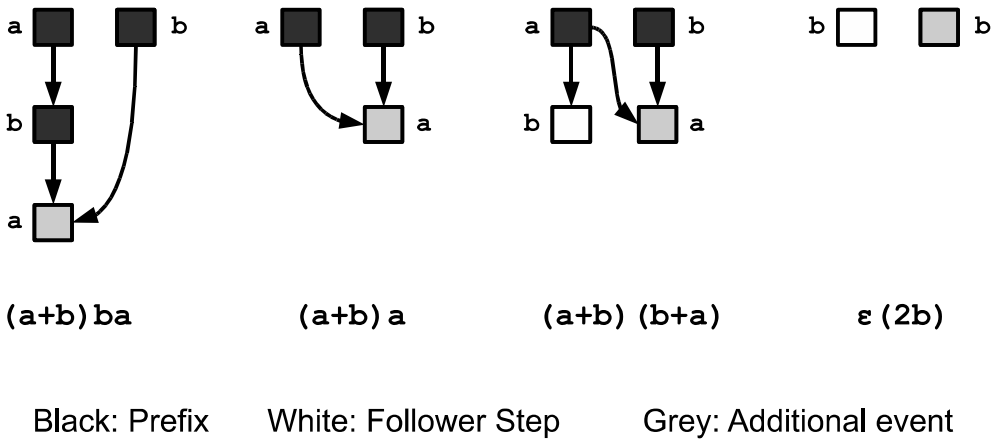



**Fig. 11.** Some wrong continuations on the level of LPOs

can be constructed as the set of all pairs of multisets $(l(V'), l(S \cup \{z\}))$, where $(V', <, l)$ is a prefix of some LPO in $L$, $S$ is a subset of direct successors of $(V', <, l)$ and $z$ is an additional labelled event. For example, the wrong continuations shown in Figure 11 are represented in the form $(a + 2b, a), (a + b, a), (a + b, a + b), (\epsilon, 2c)$ (from left to right).

With these notations and the notations from the previous subsection we directly deduce the following statements for transition regions and token flow regions.

If $\mathbf{r}$ is a transition region, $T = \{t_1, \ldots, t_n\}$, $C = S \cup \{z\}$ and $l(z) = t$ then $w_{\sigma,t}$ is not a step execution w.r.t. $p_{\mathbf{r}}$ if and only if

$$r_0 + \sum_{i=1}^{n} l(V')(t_i)(r_{n+i} - r_i) - \sum_{i=1}^{n} l(C)(t_i) r_i < 0.$$

This is the case if and only if $\mathbf{d}(w_{\sigma,t}) \cdot \mathbf{r} < 0$ for $\mathbf{d}(w_{\sigma,t}) = (d_0, \ldots, d_{2n})$ defined by:

$$d_j = \begin{cases} 1 & \text{if } j = 0, \\ -l(V' \cup C)(t_j) & \text{if } j \in \{1, \ldots, n\}, \\ l(V')(t_{j-n}) & \text{if } j \in \{n+1, \ldots, 2n\}. \end{cases}$$

For example, for the left most wrong continuation in Figure 11 we require $r_0 + ((r_4 - r_1) + 2(r_5 - r_2)) - (r_1) < 0$ (remember $t_1 = a$, $t_2 = b$ and $t_3 = c$). This is the case if and only if $\mathbf{d}(w_{\sigma,t}) \cdot \mathbf{r} < 0$ for

$$\mathbf{d}(w_{\sigma,t}) = (1, -2, -2, 0, 1, 2, 0).$$

The region

$$\mathbf{r} = (1, 1, 0, 1, 0, 0, 0)$$

(corresponding to place $p_1$ in Figure 8) is a solution which prohibits this wrong continuation.

If $\mathbf{r}$ is a token flow region, then the number of tokens in the place $p_{\mathbf{r}}$ after the occurrence of a prefix $(V', <, l)$ of some LPO $lpo = (V, <, l))$ equals the initial token flow of $lpo$ minus the sum of intoken flows of events in $V'$ plus the sum of outtoken flows of

events in $V'$. This sum needs to be smaller than the sum of intoken flows of events in $C$. Formally, if $v_t$ an arbitrary event with label $t$ then $w_{\sigma,t}$ is not a step execution w.r.t. $p_{\mathbf{r}}$ if and only if

$$r_{lpo} + \sum_{u \in V \setminus (V' \cup S)} r_{u,in} + \sum_{v \in V'} r_{v,out} + \sum_{v \in V',\, u \in V \setminus (V' \cup S)} r_{v,u} - (r_{(v_t,in)} + \sum_{u < v_t} r_{(u,v_t)}) < 0.$$

This is the case if and only if $\mathbf{d}(w_{\sigma,t}) \cdot \mathbf{r} < 0$ for $\mathbf{d}(w_{ext(t)}) = (d_i)_{i \in W \times \{in,out\} \cup E \cup L}$ defined by:

$$d_i = \begin{cases} 1 & \text{if } i = lpo, \\ 1 & \text{if } i = (u, in) \wedge u \in V \setminus (V' \cup S) \wedge u \neq v_t, \\ 1 & \text{if } i = (v, out) \wedge v \in V', \\ 1 & \text{if } i = (v, u) \wedge v \in V' \wedge u \in V \setminus (V' \cup S) \wedge u \neq v_t, \\ -1 & \text{if } i = (v_t, in) \wedge v_t \notin V \setminus (V' \cup S), \\ -1 & \text{if } i = (u, v_t) \wedge v_t \notin V \setminus (V' \cup S), \\ 0 & \text{else.} \end{cases}$$

For example, for the right most wrong continuation in Figure 11 we require $r_{lpo1} + r_{u,in} + r_{w,in} - r_{v,in} < 0$ (we use the notations from Figure 9 and $V' = \emptyset$ and $S = \{v\}$) This is the case if and only if $\mathbf{d}(w_{\sigma,t}) \cdot \mathbf{r} < 0$ for

$$\mathbf{d}(w_{\sigma,t}) = (1, 0, 1, -1, 1, 0, 0, 0, 0, 0, 0, 0).$$

The region

$$\mathbf{r} = (0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1)$$

(illustrated in Figure 10 and corresponding to place $p_2$ in Figure 8) is a solution which prohibits this wrong continuation.

Summarizing, a region separating $L$ from some wrong continuation $w$ is computed as a solution of an adequate homogenuous linear inequation system. Such a system consists of the equations/inequations given by $\mathbf{A}_L$ defining regions and an additional row $\mathbf{d}(w)$ which is defined in such a way that $\mathbf{d}(w) \cdot \mathbf{r} < 0$ if and only if $w$ is not an execution of the net having the place $p_{\mathbf{r}}$. There are effective algorithms to compute a non-negative integer solution of the resulting system. One example is the simplex algorithm, which allows to compute a solution of such a system which addionally minimizes or maximizes a given linear target function. Such a target function may be used to compute simple places. This is of high importance in practise where the derived system model should be as clear and compact as possible. For example, the places $p_2$ and $p_6$ in Figure 8 both prohibit the most right wrong continuation from Figure 11, but place $p_2$ is much more simple and intuitive.

**Definition 11 (Linear Target Function of LPO-languages).** *A* linear target function (of an LPO-language) *is a function of the form* $T(\mathbf{r}) = \mathbf{m} \cdot \mathbf{r}$ *for regions* $\mathbf{r}$*, where* $\mathbf{m}$ *is the vector defining* $T$.

It is possbile to define $\mathbf{m}$ in such a way that the sum of initial marking and arc weight on incoming and outgoing edges w.r.t. $p_{\mathbf{r}}$ is minimized when $T$ is minimized.

If $\mathbf{r}$ is a transition region and $T = \{t_1, \ldots, t_n\}$, then $\mathbf{m}$ is defined through

$$\mathbf{m} = (1, \ldots, 1),$$

because $\mathbf{m} \cdot \mathbf{r} = \sum_{i=0}^{2n} m_i r_i = m_0(p_{\mathbf{r}}) + \sum_{i=1}^{n} (W(p_{\mathbf{r}}, t_i) + W(t_i, p_{\mathbf{r}}))$.

If $\mathbf{r}$ is a token flow region, then the initial marking of $p_{\mathbf{r}}$ can be computed as the initial token flow of some LPO in $L$ and the arc weights on incoming and outgoing edges w.r.t. $p_{\mathbf{r}}$ can be computed as intoken and outtoken flows of some events $v_1, \ldots, v_n$ satisfying $l(v_j) = t_j$ (we omit a formal definition here). In the running example $\mathbf{m}$ can be defined through

$$\mathbf{m} = (0, 1, 1, 1, 0, 2, 1, 1, 0, 1, 1),$$

because $m_0(p_{\mathbf{r}}) = r_{lpo2} + r_{x,in}$, $W(p_{\mathbf{r}}, a) = r_{u,in}$, $W(p_{\mathbf{r}}, b) = r_{v,in}$, $W(p_{\mathbf{r}}, c) = r_{x,in}$, $W(a, p_{\mathbf{r}}) = r_{u,out} + r_{u,w}$, $W(b, p_{\mathbf{r}}) = r_{v,out}$ and $W(c, p_{\mathbf{r}}) = r_{x,out}$.

**Definition 12 (Minimal Places).** *Given a target function $T$, a feasible place is called* minimal w.r.t. a wrong continuation*, if it minimizes $T$ among all feasible places prohibiting the wrong continuation.*

Figure 12 illustrates that a wrong continuation may be prohibited by several different feasible places. The target function $T$ has different values for these places. The smaller the value of $T$ is, the more simple is the place.

On the other side, Figure 12 shows that a feasible place may prohibit several wrong continuations. Sometimes, a place prohibits all wrong continuations another place prohibits. Such places are more expressive in the sense that, that less places of this kind are needed to prohibit all wrong continuations.

**Definition 13 (Expressive Places).** *A feasible place $p$ is called* more expressive than *second feasible place $p'$, if the set of wrong continuations prohibited by $p'$ is contained in the set of wrong continuations prohibited by $p$.*

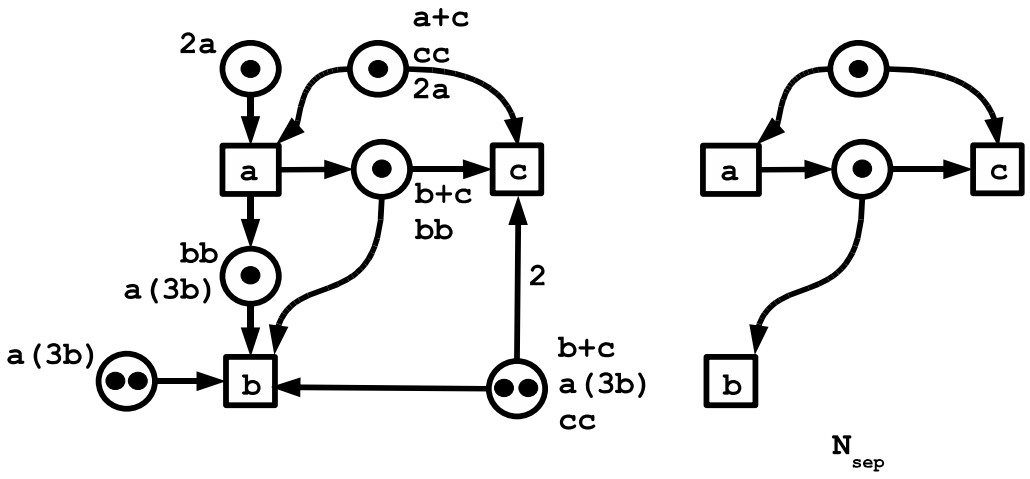



**Fig. 12.** Left side: Some feasible places together with some of the wrong continuations they prohibit. Right side: A solution

A place which is more expressive than another place is not minimal w.r.t. some wrong continuations. This means, it is necessary to find a trade-off between simple (minimal) and expressive places.

Since feasible places in general prohibit more than one wrong contintuation, $N_{sep}$ is constructed as follows:

1. Compute a sequence of all wrong continuations.
2. For each wrong continuation $w$ (considered in the given order):
   a. If $w$ is prohibited by a previously computed place, skip $w$.
   b. If $w$ is not prohibited by a previously computed place, then compute and add a place prohibiting $w$ (if possible.

The above considerations imply that it depends on the considered order of wrong continuations, which places are computed. On the right side of Figure 12 a possible solution is shown. It is advantageous to choose an order such that more expressive places are computed first. There are several methods for constructing an appropriate order of wrong continuations.

In general, the number of wrong continuations is exponential in the number of nodes in $L$ (first step of the algorithm). Since feasible places often prohibit more than one wrong continuation, the number of computed places is usually much smaller. For step 2a. we need to test for every previously computed place, whether it solves the inequation system. The simplex algorithm for solving step 2b. needs worst case exponential time (there are other worts case polynomial algorithms, but probabilistic and experimental results show that the Simplex algorithm has a significantly faster average runtime).

**Basis Representation.** For systems of the form $\mathbf{A}_L \cdot \mathbf{r} = 0$ or $\mathbf{A}_L \cdot \mathbf{r} \leq 0$ there is a so called *basis representation* of the set of all non-negative solutions. This means there are non-negative *basis-solutions* $\mathbf{y}_1, \ldots, \mathbf{y}_n$ such that each solution $\mathbf{x}$ is a non-negative linear combination of $\mathbf{y}_1, \ldots, \mathbf{y}_n$, i.e.

$$\mathbf{x} = \sum_{i=1}^{n} \lambda_i \mathbf{y}_i$$

for real numbers $\lambda_1, \ldots, \lambda_n \geq 0$. In the case that all values in $\mathbf{A}_L$ are integral (this is the case here) also the values of $\mathbf{y}_1, \ldots, \mathbf{y}_n$ can be chosen integral. If $p_i$ is the place defined by $\mathbf{y}_i$ and $N_{basis}$ is the net containing exactly the places $p_1, \ldots, p_n$, then $L(N_{basis}) = L(N_{sat})$ [5]. This means $N_{basis}$ is also the best approximation to a solution of the synthesis problem generating $L$ but $N_{basis}$ is moreover finite. $N_{basis}$ is called *basis representation* of $N_{sat}$.

In the worst case $n$ can be exponential in the number of rows of $\mathbf{A}_L$, but in practice it is often small. There are effective algorithms to compute *minimal* basis solutions $\mathbf{y}_1, \ldots, \mathbf{y}_n$, where a solution $\mathbf{y} = (y_1, \ldots, y_k)$ is minimal, if there is no other solution $\mathbf{z} = (z_1, \ldots, z_k)$ satisfying $(\forall i : \ z_i \leq y_i) \wedge (\exists j : \ z_j < y_j)$. Figure 13 shows $N_{basis}$ for the running example language. Observe that there are basis places which do not restrict the behavior of the net (filled with grey color). They are special cases of so called implicit places.

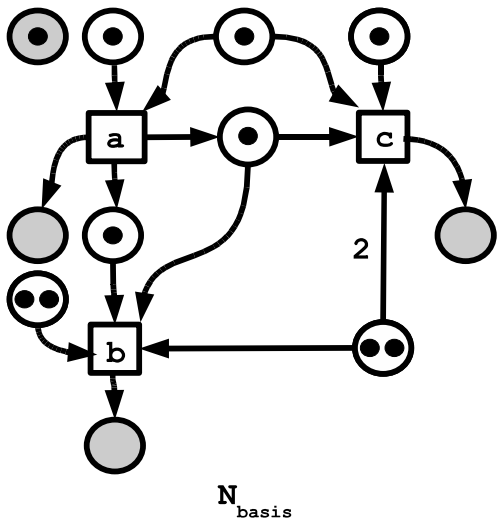



**Fig. 13.** The basis representation of the running example

**Definition 14 (Implicit Place).** *A place of a PT-net is called* implicit*, it the modified net without this place has the same set of executions.*

The are polynomial methods to detect some of the implicit places of a net which can be applied in a postprocessing phase to the synthesized net. An easy example are places whiich are dominated by one another place (w.r.t. the behavioral restriction).

**Definition 15 (Dominating Place).** *A place $p'$* dominates *another place $p$, if for some $\lambda > 0$:*

- $\lambda m_0(p) \geq \lambda m_0(p')$,
- $\lambda W(t,p) \geq \lambda W(t,p')$ *and* $\lambda W(p,t) \leq \lambda W(p',t)$ *for all transitions $t$.*

Dependent on the initial marking and the arc weights, a maximal number for $\lambda$ can be determined. Then places can be compared pairwise. Advanced methods are able to detect places which are dominated by positive linear combinations of other places. Such places are implicit, too.

Since from the construction it is not clear whether $N_{basis}$ is an exact solution of the synthesis problem, it is finally necessary to test whether $L(N_{basis}) = L$ or not. One possibility of such an equality test is to test whether no $w \in L^c$ ($L^c$ was defined in the last paragraph) is an execution of $N_{basis}$. There are polynomial algoritms testing whether an LPO is an execution of a PT-net [25], but $L^c$ in general contains exponential many LPOs in the number of nodes in $L$.

**Discussion.** Experiments in the first phase of the project SYNOPS showed that the so called separation representation produces Petri nets which are simpler and more compact, especially having less places [5]. It turned out that in the presence of much concurrency the use of the basis representation of token flow regions is most efficient (including an equality test), whereas in the case of low concurrency and much nondeterminism the use of the separation representation of transition regions ist advantageous (concerning runtime).

### 2.5 Regions of Infinite Languages

In this subsection we generalize the presented framework to inifinite LPO-languages. In order to finitely represent such languages we introduce a term based representation extending regular expressions. A term is built from a given finite set of LPOs through operators for iteration, parallel composition, sequential composition and union. The parallel composition operation represents concurrent LPOs. By the iteration operation infinite sets of LPOs can be constructed. For a term $\alpha$ we denote by $L(\alpha)$ the LPO-language represented by $\alpha$.

The formal problem statement, which we consider from now, is:

**Given:** A term $\alpha$ over a finite alphabet of transition names $T$.
**Searched:** A PT-net $N$ with set of transitions $T$ such that all LPOs in $L(\alpha)$ are LPO-runs of $N$ and $N$ has a minimal number of additional LPO-runs.

As in the finite case, we will define transition regions and token flow regions of $L(\alpha)$ as non-negative integral solutions of appropriate linear systems of the form $\mathbf{A}_\alpha \cdot \mathbf{x} \leq \mathbf{b}_\alpha$. Throughout the rest of this subsection we use the language shown in Figure 14 as a running example, where zero, one and two iterations of the action $b$ are shown and prefixes and extensions are omitted.

**LPO-terms.** Let $\mathcal{A}$ be a finite set of LPOs. For $A \in \mathcal{A}$ we write $A = (V_A, <_A, l_A)$. We denote by $\lambda = (\emptyset, \emptyset, \emptyset)$ the empty LPO. LPOs consisting only of one single event we denote by the label of this event.

**Definition 16 (LPO-term).** *The set of* LPO-terms *over a finite set of LPOs $\mathcal{A}$ is inductively defined as follows:*

- *The elements $A \in \mathcal{A}$ and $\lambda$ are LPO-terms.*
- *Let $\alpha_1$ and $\alpha_2$ be LPO-terms. Then*
  - $\alpha_1; \alpha_2$ *(sequential composition),*
  - $\alpha_1 + \alpha_2$ *(union),*
  - $(\alpha_1)^*$ *(iteration),*
  - $\alpha_1 \parallel \alpha_2$ *(parallel composition)*

  *are LPO-terms.*

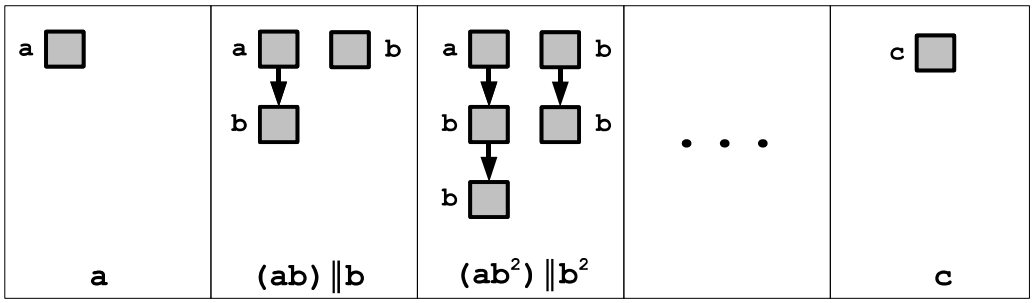



**Fig. 14.** Infinite example language, represented by a term

In the running example shown in Figure 14 $\mathcal{A} = \{a, b, c\}$ consists of three single event LPOs. If each LPO in $\mathcal{A}$ is single event LPO, an LPO-term defines a so called series rational sp-language [22,23].

We assign to an arbitrary LPO-term $\alpha$ a possibly infinite prefix and extension closed LPO-language $L(\alpha)$. The language $L(\alpha$ is defined as the prefix and extension closure of an appropriate LPO-language $K(\alpha)$. In order to construct $K(\alpha)$, we define the *sequential composition of LPOs* $A, B \in \mathcal{A}$ by

$$AB = (V_A \cup V_B, <_A \cup <_B \cup (V_A \times V_B), l_A \cup l_B),$$

the *parallel composition of LPOs* $A, B \in \mathcal{A}$ by

$$A \parallel B = (V_A \cup V_B, <_A \cup <_B, l_A \cup l_B),$$

and the *$n$-th iteration of an LPO* $A \in \mathcal{A}$ by

$$A^n = A^{n-1}A$$

for $n \in \mathbb{N}^+$ (we can assume that $A, B$ have disjoint sets of nodes).

**Definition 17 (LPO-language of an LPO-term).** *We define inductively:*

- $K(\lambda) = \{\lambda\}$ *and* $K(A) = \{A\}$ *for* $A \in \mathcal{A}$*,*
- *Let* $\alpha_1$ *and* $\alpha_2$ *be LPO-terms. Then:*
  - $K(\alpha_1 + \alpha_2) = K(\alpha_1) \cup K(\alpha_2)$*,*
  - $K(\alpha_1; \alpha_2) = \{A_1 A_2 \mid A_1 \in K(\alpha_1),\ A_2 \in K(\alpha_2)\}$*,*
  - $K((\alpha_1)^*) = \{A_1 \dots A_n \mid A_1, \dots, A_n \in K(\alpha_1)\} \cup \{\lambda\}$*,*
  - $K(\alpha_1 \parallel \alpha_2) = \{A_1 \parallel A_2 \mid A_1 \in K(\alpha_1),\ A_2 \in K(\alpha_2)\}$*.*

Some of the the LPOs in $K(\alpha)$ from the example LPO-term shown in Figure 14 are $c, a, ab, abb, \dots, a \parallel b, a \parallel b^2, \dots, (ab) \parallel b, (ab) \parallel b^2, \dots$.

In the second part of this paper, LPO-languages which cannot be generated by LPO-terms are discussed. This means, LPO-languages generated by LPO-terms (over finite sets of LPOs) form a certain class of LPO-languages.

**Regions of LPO-Terms.** We now describe a technique to represent the infinite set $K(\alpha)$ by two finite sets of LPOs $R(\alpha)$ and $I(\alpha)$. Regions will be defined w.r.t. these finite sets.

An LPO $A$ can occur arbitrarily often consecutively in a certain marking $m$ if and only if it consumes in every place at most as many tokens as it produces in this place (then an occurrence of $A$ does not reduce the number of tokens in this place). Consequently, if $A$ can occur iterated in $m$, then another LPO $B$ can occur after the occurrence of $A^n$ for each $n \in \mathbb{N}$ if and only if it can occur in $m$, since an occurrence of $A$ does not reduce the number of tokens in a place. This principle can be used to define the two finite sets $R(\alpha)$ and $I(\alpha)$.

For an LPO-term $\alpha$ the set $R(\alpha)$ is the set of, roughly speaking, all LPOs containing each iterated part of $\alpha$ at most once. The running example term $((a; b^*) \parallel b^*) + c$ contains to iterated $b$-labelled events, thus $R(((a; b^*) \parallel b^*) + c) = \{a, ab, a \parallel b, (ab) \parallel b, c\}$.

We call in $R(\alpha)$ the representation set. The first idea for the definition of regions is to ensure that the place defined by a region is feasible w.r.t. the representation set, i.e. we require that a transition region satisfies the property $(f)_{R(\alpha)}$ and a token flow regions satsfies the property $(wd)_{R(\alpha)}$ as defined for finite LPO-languages.

It remains to ensure that certain LPOs can occur iterated w.r.t. the place defined by a region. The set $I(\alpha)$ consist of, roughly speaking, all LPOs associated to iterated subterms of $\alpha$. The running example term $((a; b^*) \parallel b^*) + c$ contains to iterated $b$-labelled events, thus $I(((a; b^*) \parallel b^*) + c) = \{b\}$. We call $I(\alpha)$ the iteration set. We will introduce an additional property $(i)_\alpha$ of regions which ensures that the LPOs in $I(\alpha)$ produce at least as many tokens as they consume in the place defined by a region. This implies that the place defined by a region is feasible w.r.t. $K(\alpha)$.

**Definition 18 (Representation/Iteration set).** *The* representation set $R(\alpha)$ *and the* iteration set $I(\alpha)$ *of an LPO-term* $\alpha$ *are defined inductively as follows (*$\alpha_1$ *and* $\alpha_2$ *are LPO-terms):*

| | |
|---|---|
| $R(\lambda) = \{\lambda\}$ | $I(\lambda) = \emptyset$ |
| $R(A) = \{A\}$ *for* $A \in \mathcal{A}$ | $I(A) = \emptyset$ *for* $A \in \mathcal{A}$ |
| $R(\alpha_1 + \alpha_2) = R(\alpha_1) \cup R(\alpha_2)$ | $I(\alpha_1 + \alpha_2) = I(\alpha_1) \cup I(\alpha_2)$ |
| $R(\alpha_1; \alpha_2) = \{A_1 A_2 \mid A_1 \in R(\alpha_1),\ A_2 \in R(\alpha_2)\}$ | $I(\alpha_1; \alpha_2) = I(\alpha_1) \cup I(\alpha_2)$ |
| $R((\alpha_1)^*) = R(\alpha_1) \cup \{\lambda\}$ | $I((\alpha_1)^*) = I(\alpha_1) \cup R(\alpha_1)$ |
| $R(\alpha_1 \parallel \alpha_2) = \{A_1 \parallel A_2 \mid A_1 \in R(\alpha_1),\ A_2 \in R(\alpha_2)\}$ | $I(\alpha_1 \parallel \alpha_2) = I(\alpha_1) \cup I(\alpha_2)$ |

This approach generalizes the ideas in [10] where the authors define regions by two finite sets representing a regular expression.

The requirement, that every LPO in $I(\alpha)$ produces at least as many tokens as it consumes in a place $p$, corresponds to the requirement, that the sum of tokens produced by all events of an LPO in $I(\alpha)$ exceeds the sum of tokens consumed by all events. For $lpo = (V, <, l)$ and some place $p$ we define

$$Prod(lpo, p) := \sum_{t \in l(V)} l(V)(t)(W(t, p) - W(p, t)).$$

A region $\mathbf{r}$, i.e. property $(i)_\alpha$, is defined in such a way that $Prod(lpo, p_\mathbf{r}) \geq 0$ for each LPO $lpo \in I(\alpha)$. For the running example LPO-term we require $W(b, p_\mathbf{r}) - W(p_\mathbf{r}, b) \geq 0$.

The definition of $(i)_\alpha$ for transition regions of LPO-terms $\alpha$ and PT-nets is as follows, where a (PT-net) transition-region $\mathbf{r} = (r_0, \ldots, r_{2n})$ directly defines the parameters of a place $p_\mathbf{r}$ of PT-nets via $m_0(p_\mathbf{r}) = r_0$, $W(p_\mathbf{r}, t_i) = r_i$ and $W(t_i, p_\mathbf{r}) = r_{n+i}$, if $T = \{t_1, \ldots, t_n\}$ is the set of transition names occurring in LPOs from $K(\alpha)$. For each $lpo = (V, <, l) \in I(\alpha)$ we require

$$\sum_{i=1}^{n} l(V)(t_i)(r_{n+i} - r_i) \geq 0.$$

This is the case if and only if (for each $lpo \in I(\alpha)$) $\mathbf{i}_{lpo} \cdot \mathbf{r} \leq 0$ for $\mathbf{i}_{lpo} = (i_0, \ldots, i_{2n})$ defined by:

$$\mathbf{i}_j = \begin{cases} 0 & \text{if } j = 0, \\ l(V)(t_j) & \text{if } j \in \{1, \ldots, n\}, \\ -l(V)(t_{j-n}) & \text{if } j \in \{n+1, \ldots, 2n\}. \end{cases}$$

In the running example, denote $t_1 = a$, $t_2 = b$ and $t_3 = c$. For the LPO $b \in I(\alpha)$ in the running example we require $r_5 - r_2 \geq 0$. This is the case if and only if $\mathbf{i}_b \cdot \mathbf{r} \leq 0$ for

$$\mathbf{i}_b = (0, 0, 1, 0, 0, -1, 0).$$

**Definition 19 (Transition-Region of LPO-term).** *We call a tuple* $\mathbf{r}$ *a* transition-region *of an LPO-term* $\alpha$ *if it satisfies* $(f)_{R(\alpha)}$ *and* $(i)_\alpha$.

With these definitions and notions the following theorem holds:

**Theorem 5.** *A tuple* $\mathbf{r}$ *satisfies* $(f)_{R(\alpha)}$ *and* $(i)_\alpha$ *if and only if* $p_{\mathbf{r}}$ *is feasible w.r.t.* $K(\alpha)$.

Let $\mathbf{A}_\alpha$ be the matrix consisting of all rows of $\mathbf{A}_{R(\alpha)}$ and all rows $\mathbf{i}_{lpo}$ for LPOs $lpo \in I(\alpha)$. Then the set of all regions can be computed as the set of all integral solutions of the homogenous linear inequation system $\mathbf{A}_\alpha \cdot \mathbf{x} \leq \mathbf{0}$. In the running example, $\mathbf{A}_\alpha$ looks as follows (the set $R(\alpha)$ can be represented by the finite example language from the subsection on finite languages):

$$\begin{pmatrix} -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 2 & 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Solutions are for example $\mathbf{r} = (1, 1, 0, 1, 0, 0, 0)$ with corresponding place $p_1$, $\mathbf{r} = (1, 0, 1, 1, 1, 1, 0)$ and $\mathbf{r} = (0, 0, 0, 0, 0, 1, 0)$ with corresponding place $p_3$ of the PT-net shown in Figure 15.

**Theorem 6.** *If* $\alpha$ *is an LPO-term then there is a finite matrix* $\mathbf{A}_\alpha$ *such that the set of transition-regions is the set of solutions of the linear inequation system* $\mathbf{A}_\alpha \cdot \mathbf{x} \leq \mathbf{0}$.

The previous theorems are proven in [7] for token flow regions and easily carry over to transition regions.

If

$$W = \bigcup_{(V,<,l) \in R(\alpha)} V$$

is the set of nodes of LPOs in $R(\alpha)$ and

$$E = \bigcup_{(V,<,l) \in R(\alpha)} <$$

is the set of arrows of LPOs in $R(\alpha)$, then a *(PT-net) token flow-region* $\mathbf{r}$ *of an LPO-term* $\alpha$ is given as a tuple $\mathbf{r} = (r_i)_{i \in W \times \{in,out\} \cup E \cup L}$ of non-negative integers. Its components are interpreted and define places as for finite languages.

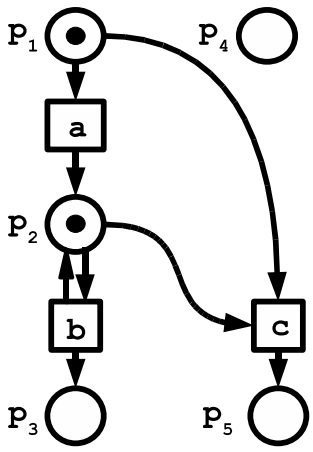



**Fig. 15.** Some solution places for the example LPO-term

The definition of $(i)_\alpha$ for token flow regions of LPO-terms $\alpha$ and PT-nets is as follows: For each $lpo \in I(\alpha)$ we require

$$\sum_{v \in V_{lpo}} (r_{v,out} + \sum_{e=(v,u)\in <} r_e - r_{v,in} - \sum_{e=(u,v)\in <} r_e) \geq 0,$$

where $(V_{lpo}, <, l)$ is a sub-LPO of some LPO $(V, <, l) \in R(\alpha)$ which is isomorphic to $lpo$. This is the case if and only if $\mathbf{i}_{lpo} \cdot \mathbf{r} \leq 0$ for $\mathbf{i}_{lpo} = (i_j)_{j \in W \times \{in,out\} \cup E \cup L}$ defined by:

$$\mathbf{i}_j = \begin{cases} 1 & \text{if } j = (v, in) \text{ for } v \in V_{lpo}, \\ 1 & \text{if } j = (u, v) \text{ for } v \in V_{lpo},\ u \in V \setminus V_{lpo}, \\ -1 & \text{if } i = (v, out) \text{ for } v \in V_{lpo}, \\ -1 & \text{if } j = (v, u) \text{ for } v \in V_{lpo},\ u \in V \setminus V_{lpo}, \\ 0 & \text{else.} \end{cases}$$

In the running example, it is enough to consider the LPOs $lpo1 = (ab) \parallel b$ and $lpo2 = c$ from $R(\alpha)$, since the other LPOs are prefixes of $lpo1$. Thus, $R(\alpha)$ can be treatened in the same way as the finite example language from the section on finite languages. As before, we denote

$$\mathbf{r} = (r_{lpo1}, r_{lpo2}, r_{u,in}, r_{v,in}, r_{w,in}, r_{x,in}, r_{u,out}, r_{v,out}, r_{w,out}, r_{x,out}, r_{(u,w)}),$$

where the events $u, v, w$ and $x$ are shown in Figure 9. If we represented the iterated LPO $b \in I((a; b^*) \parallel b^*)$ by the event $v$, then we require

$$r_{v,in} - r_{v,out} \leq 0.$$

This is satisfied if and only if $\mathbf{i}_b \cdot \mathbf{r} = \mathbf{0}$ for

$$\mathbf{i}_b = (0, 0, 0, 1, 0, 0, 0, -1, 0, 0, 0).$$

**Definition 20 (Token Flow-Region of LPO-term).** *A tuple* $\mathbf{r}$ *as above is called a* token flow-region *of an LPO-term* $\alpha$ *if it satisfies* $(wd)_{R(\alpha)}$ *and* $(i)_\alpha$.

With these definitions and notions the following theorem holds:

**Theorem 7.** *A tuple* $\mathbf{r}$ *satisfies* $(wd)_{R(\alpha)}$ *and* $(i)_\alpha$ *if and only if* $p_{\mathbf{r}}$ *is feasible w.r.t.* $K(\alpha)$.

Let $\mathbf{A}_L$ be the matrix consisting of all rows from the matrices $\mathbf{A}_{L,a}$, $\mathbf{A}_{L,b}$ and $\mathbf{A}_{L,c}$. Since $L$ is assumed to be finite, $\mathbf{A}_L$ is finite. Thus, the set of all token flow-regions can be computed as the set of all integral solutions of the homogenous linear equation system $\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$. If we denote $v_1^b = v$ and $v_2^b = w$, $\mathbf{A}_L$ looks as follows for the running example (the matrices $\mathbf{A}_{L,a}$, $\mathbf{A}_{L,b}$ and $\mathbf{A}_{L,c}$ each consist exactly of the one row already shown):

Let $\mathbf{A}_{I(\alpha)}$ be the matrix consisting of all rows $\mathbf{i}_{lpo}$ for LPOs $lpo \in I(\alpha)$. The set of all regions can be computed as the set of all integral solutions of the homogenous linear inequation system $\mathbf{A}_{R(\alpha)} \cdot \mathbf{x} = \mathbf{0}$ and $\mathbf{A}_{I(\alpha)} \cdot \mathbf{x} \leq \mathbf{0}$. In the running example, the matrix of this system looks as follows:

$$\begin{pmatrix} -1 & 1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{pmatrix}$$

Figure 16 shows a solution. This solution represents the place $p_2$ from Figure 15.

**Theorem 8 ([7]).** *If* $\alpha$ *is an LPO-term then there is a finite matrix* $\mathbf{A}_\alpha$ *such that the set of token flow-regions is the set of solutions of the linear inequation system* $\mathbf{A}_\alpha \cdot \mathbf{x} \leq \mathbf{0}$.

**Finite Representation.** As in the case of finite languages, the set of regions of an LPO-term is defined as the set of positive integral solutions of a homogenuous linear inequation system. Thus, it has a finite basis representation. Figure 17 shows the basis representation for the running example term.

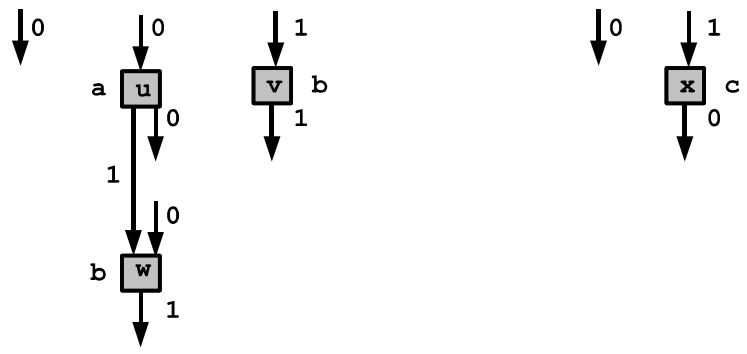



**Fig. 16.** Illustration of the token flow region $\mathbf{r} = (0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1)$

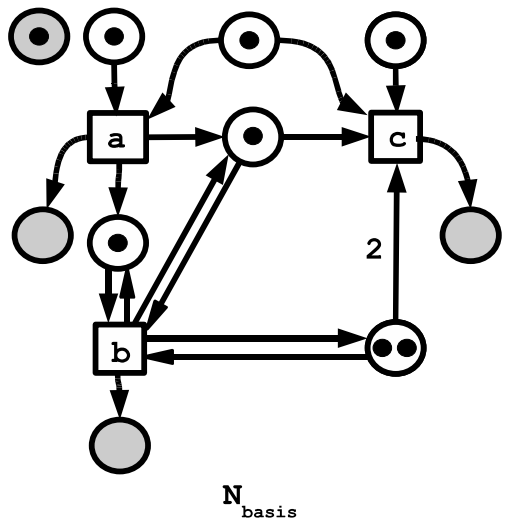



**Fig. 17.** The basis representation of the running example. The iterated transition $b$ does not decrease the number of tokens in places

In order to compute a finite separation representation we need to finitely represent the infinite set of wrong continuations $K(\alpha)^c$. The next definition proposes such a finite representation:

**Definition 21 (Wrong Continuation of LPO-term).** *Let $\alpha$ be an LPO-term and let $\sigma = \beta_1 \dots \beta_{n-1}\beta_n \in R(\alpha)^{step}$ and $t \in T$ such that $w_{\sigma,t} = \beta_1 \dots \beta_{n-1}(\beta_n + t) \notin K(\alpha)^{step}$, where $\beta_n$ is allowed to be the empty step. Then $w_{\sigma,t}$ is called* wrong continuation of $\alpha$.

*We call $\beta_1 \dots \beta_{n-1}$ the* prefix *and $\beta_n + t$ the* follower step of the wrong continuation.

Note that this finite set of wrong continuations is usually a proper subset of $R(\alpha)^c$ because of the requirement $w_{\sigma,t} \notin K(\alpha)^{step}$. For the running example term $\alpha = ((a; b^*) \parallel b^*) + c$, $R(\alpha)^c$ is listed in the subsection on finite languages, where (for example) $bb \in R(\alpha)^c$ is not a wrong continuation of $\alpha$.

For each wrong continuation $w$ of $\alpha$ we search for a place $p_{\mathbf{r}}$ not only prohibiting $w$, but also an infinite set of wrong continuations $I(w) \subseteq K(\alpha)^c$. The idea is that, if the prefix of $w$ contains the prefix of a sub-LPO (of some LPO in $R(\alpha)$) which can be iterated, then the follower step must be prohibited by $p_{\mathbf{r}}$ also after all finite iterations of this sub-LPO. In other words, $I(w)$ contains all wrong continuations from $K(\alpha)^c$ which can be constructed from $w$ by inserting iterations of sub-LPOs into the prefix of $w$. For example, $I(b(2a)) = \{b^n(2a) \mid n \in \mathbb{N}\}$. The left side of Figure 18 shows example places.

A place $p_{\mathbf{r}}$ prohibits each wrong continuations in $I(w)$ (for some wrong continuation $w$), if and only if the following two properties are satisfied:

- $p_{\mathbf{r}}$ prohibits $w$.
- If the prefix of $w$ contains the prefix of a sub-LPO which can be iterated, then the occurrence of this sub-LPO does not increase the number of tokens in $p_{\mathbf{r}}$.

The first property can be encoded as a homogenuous linear inequation as in the case of finite languages. Together with the defining property of regions of LPO-terms, the

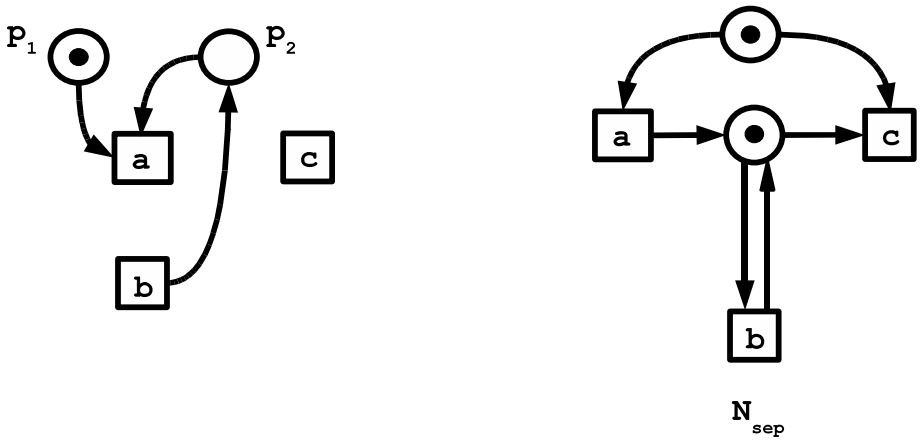



**Fig. 18.** Left Side: Place $p_2$ prohibits $b(2a)$ but not $bb(2a)$; Place $p_1$ prohibits each wrong continuation of the form $b^n(2a)$ for $n \in \mathbb{N}$. Right side: The separation representation of the example LPO-term.

second property implies that after the occurrence of such a sub-LPO the number of tokens in $p_{\mathbf{r}}$ is the same as before the occurrence. This can be ensured by requiring

$$Prod(lpo, p_{\mathbf{r}}) = 0$$

for each sub-LPO $lpo$ whose prefix is prefix of the prefix of $w$ and which can be iterated. The corresponding defining linear inequation system for transition-regions and token flow-regions can be constructed in an analoguous way as for the property $Prod(lpo, p_{\mathbf{r}}) \geq 0$ of regions of LPO-terms. For the running example LPO-term and the wrong continuation $b(2a)$ we have $I(b(2a)) = \{b\}$ and require $Prod(b, p_{\mathbf{r}}) = W(b, p_{\mathbf{r}}) - W(p_{\mathbf{r}}, b) = 0$.

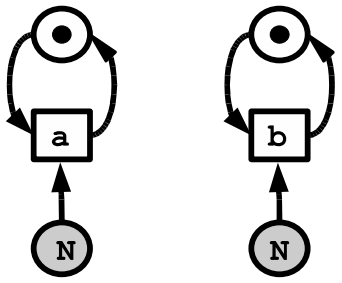
It remains to determine the term of an *iterated sub-LPO*. Of course, each sub-LPO which is isomorphic to an LPO in $I(\alpha)$ is a candidat. But the follower marking after the occurrence of some sub-LPO $(V', <, l)$ only depends on the multiset $L(V')$, i.e. on the number of occurrences of each transition in the sub-LPO.

**Definition 22 (Iterated sub-LPO).** *We say that a sub-LPO $(V', <', l')$ of an LPO in $R(\alpha)$* can be iterated, *if $l'(V') = l(V)$ for an LPO $(V, <, l) \in I(\alpha)$.*

The right side of Figure 18 shows the separation representation for the example LPO-term. Note that the wrong continuations of the form $b^n c$ cannot be prohibited (compare also Figure 17 showing the basis representation). The reason is that $b^n$ as well as $c$ can occur in the initial state and $b^n$ does not decrease the number of tokens in places. Note that it is possible to consider more general Petri net classes, as for example inhibitor nets, which allow to prohibit $b^n c$.

## 2.6 Speeding Up Synthesis for Finite Languages

If a finite language of LPOs contains finite iterations of sub-LPOs, then the technique from the previous subsection can be used to reduce the runtime for computing the separation representation. In the following we will illustrate this by an example.




$\mathtt{L = \{a^N \parallel b^N\}}$

**Fig. 19.** PT-net synthesized from $L = \{a^N \parallel b^N\}$

Consider the familiy of LPOs $(a^n \parallel b^n)$ $(n \in \mathbb{N})$. The number of wrong continuations of the language $L = \{(a^n \parallel b^n)\}$ (for some fixed $n$) grows quadratically with $n$. For a fixed $N$ the wrong continuations are of the form $(a^j \parallel b^i)(2a)$, $(a^j \parallel b^i)(2b)$, $(a^j \parallel b^i)(2a + b)$ and $(a^j \parallel b^i)(2b + a)$ for $0 \leq i, j \leq N - 1$, $(a^N \parallel b^i)a$, $(a^N \parallel b^i)(2b)$ and $(a^N \parallel b^i)(2b + a)$ for $0 \leq i \leq N - 1$, $(a^j \parallel b^N)b$, $(a^j \parallel b^N)(2a)$ and $(a^j \parallel b^N)(2a + b)$ for $0 \leq j \leq N - 1$, and $(a^N \parallel b^N)a$ and $(a^N \parallel b^N)b$.

Instead of considering all these wrong continuations we can equivalently

- first compute the wrong continuations of the LPO-term $(a^* \parallel b^*)$, which are only those of $a \parallel b$ (iterations only occur once in the representation set. The corresponding separation representation prohibits all wrong continuations of $L = \{(a^N \parallel b^N)\}$ except $(a^N \parallel b^i)a$ for $0 \leq i \leq N - 1$, $(a^j \parallel b^N)b$ for $0 \leq j \leq N - 1$, and $(a^N \parallel b^N)a$ and $(a^N \parallel b^N)b$.
- second add the remaining wrong continuations, which are not yet prohibited.

Figure 19 shows the PT-net synthesized with this technique. The white places allow arbitrary iterations and prohibit the first kind of wrong continuations. The grey places restrict the length of LPO-runs and prohibit the second kind of wronf continuations.

The technique significantly reduces the number of wrong continuations we need to consider.

## 2.7 Concluding Remarks

The presented synthesis theory for LPO-languages was developed within the last years. In earlier years, the synthesis problem already was solved for finite and regular languages over single action names. As already mentioned, such languages are a special case of LPO-languages as considered in this paper.

The technique of computing the separation representation of the set of PT-net transition regions as presented in this paper coincides with the earlier developed technique for this special case [10,2,3]. In these early publications a parametric definition of Petri nets, which can be instantiated with different concrete net classes like elementary nets or inhibitor nets, was considered. In the second part of this paper we extend the presented framework to several of these other net classes.

In [16] PT-net transition-regions for trace languages (step languages) are introduced. The authors do not consider the subclasses of finite or regular step languages and therefore only define an infinite separating representation (involving infinite many constraints)

Token flow regions and the basis representation were developed later in the context of LPO-languages.

There are many other publications considering (step) transition systems instead of languages as behavioral model (e.g., [14,15,28,29,8,30,11]). These approaches are restricted to transition-regions and separating representations.

# 3 Extensions and Generilizations

In this second part of the paper we extend and generalize the framework presented in the first part in the following directions:

- We consider the synthesis of inhibitor nets from finite and from simple infinite languages of labelled stratified order structures.
- We discuss synthesis from languages of non-transitive order structures.
- We examine the synthesis of nets of restricted net classes.
- We suggest several possibilities for a finite representation of more general infinite languages.

For a clear presentation we do not combine these generalizations to common definitions of regions and finite representations. Instead we consider each generilization separately and give some hints on how to combine different concepts in each subsection.

## 3.1 Inhibitor Nets

As examples in the first part of this paper illutrated, not always the given LPO-language can be exactly represented by a PT-net. In such cases, it is possible to consider more general Petri net classes in order find a better representation. One of the most general Petri net classes are inhibitor nets, which are as powerful as Turing machines for languages over single action names. In this subsection we extend the concept of regions to inhibitor nets.

**Stratified Order Structures.** Partial orders are used in the first part of this paper to represent causal dependencies between transition occurrences of PT-nets. When considering inhibitor nets, we need finer causal structures, so called *relational structures* [18]. A *relational structure* (*rel-structure*) is a triple $\mathcal{S} = (V, \prec, \sqsubset)$, where $V$ is a finite set (of *nodes*) and $\prec \subseteq V \times V$ and $\sqsubset \subseteq V \times V$ are binary relations on $V$. The notions of preset and postset are only used w.r.t. the relation $\prec$. A rel-structure $\mathcal{S}$ is called *acyclic* if

$$(\prec \cup \sqsubset)^* \circ \prec \circ (\prec \cup \sqsubset)^*$$

is irreflexive. Similar to the notion of the transitive closure of a binary relation the *transitive closure* $\mathcal{S}^+$ of a rel-structure $\mathcal{S} = (V, \prec, \sqsubset)$ is defined by

$$\mathcal{S}^+ = (V, \prec_{\mathcal{S}^+}, \sqsubset_{\mathcal{S}^+}) = (V, (\prec \cup \sqsubset)^* \circ \prec \circ (\prec \cup \sqsubset)^*, (\prec \cup \sqsubset)^* \setminus id_V).$$

An *extension* of an acyclic rel-structure $(V, \prec, \sqsubset)$ is an acyclic rel-structure $(V, \prec', \sqsubset')$ satisfying $\prec \subseteq \prec' \wedge \sqsubset \subseteq \sqsubset'$. A *prefix* of an acyclic rel-structure $(V, \prec, \sqsubset)$ is an acyclic rel-structure $(V', \prec, \sqsubset)$ with $V' \subseteq V$ satisfying $(v' \in V') \wedge (v(\sqsubset \cup \prec)v') \Rightarrow (v \in V')$.

A rel-structure $\mathcal{S} = (V, \prec, \sqsubset)$ is called *stratified order structure* (*so-structure*) if the following conditions are satisfied for all $u, v, w \in V$:

- $u \not\sqsubset u$.
- $u \prec v \implies u \sqsubset v$.
- $u \sqsubset v \sqsubset w \wedge u \neq w \implies u \sqsubset w$ (weak transitivity).
- $u \sqsubset v \prec w \vee u \prec v \sqsubset w \implies u \prec w$ (strong mixed transitivity).

$(V, \prec)$ is a partial order. Thus a partial order can always be interpreted as an so-structure with $\sqsubset = \prec$. The *transitive closure* $\mathcal{S}^+$ of a rel-structure $\mathcal{S}$ is an so-structure if and only if $\mathcal{S}$ is acyclic (for this and further results see [18]). Later on, we will interpret the relation $\prec$ as an "earlier than"-relation between events and the relation $\sqsubset$ as an "not later than"-relation between events.

Two nodes $v, v' \in V$ of an so-structure $(V, \prec, \sqsubset)$ are called *independent* if $v \not\sqsubset v'$ and $v' \not\sqsubset v$. Co-sets, cuts, minimal and maximal events are defined w.r.t. the partial order $\prec$.

The *skeleton* of an so-structure $(V, \prec, \sqsubset)$ is the rel-structure $(V, \prec', \sqsubset')$ with $\prec' \subseteq \prec$ minimal, $\sqsubset' \subseteq \sqsubset$ minimal and $(V, \prec', \sqsubset')^+ = (V, \prec, \sqsubset)$.

Graphically, "earlier than"-relation is drawn by drawn-through arrows and the "not later than"-relation by dotted arrows between events. For a clear illustration, often transitiv arrows are not drawn. Figure 20 shows an example so-structure, where the nodes $v_2$ and $v_3$ are in "not later than"-relation and the nodes $v_1$ and $v_2$ as well as $v_1$ and $v_3$ are in "earlier than"-relation.

**Semantics of Inhibitor Nets.** Inhibitor nets are PT-nets extended by read arcs testing places for absence of tokens.

**Definition 23 (Inhibitor Net).** *An* inhibitor net *(*PTI-net*)* $N = (P, T, F, W, I)$ *consists of a PT-net* $(P, T, F, W)$ *and a mapping* $I : P \times T \to \mathbb{N}_0 \cup \{\infty\}$ *called* inhibitor function.

*We denote* $Und(N) = (P, T, F, W)$ *the PT-net* underlying $N$.

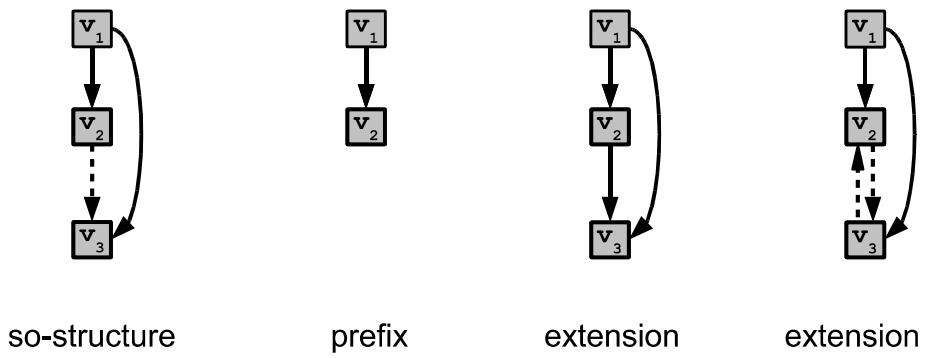



**Fig. 20.** An so-structure with a prefix and two extensions

The inhibitor function specifies upper bounds for the number of tokens allowed in a place for the occurrence of a transition.

Graphically, the number $I(p,t)$ is assigned to an arrow from $p$ to $t$ which has a circle as arrowhead, where in the case $I(p,t) = \infty$ no arrow is drawn and in the case $I(p,t) = 0$ only the arrow is drawn. Figure 22 shows a marked PTI-net with $I(p_5, b) = 0$ and $I = \infty$ in all other cases.

We introduce the following multiset of places:

- $^{-}t(p) = I(p,t)$ for transitions $t$.
- $^{-}\tau(p) = min(\{\, ^{-}t(p) \mid t \in \tau\})$ for multisets of transitions $\tau$.

There are two different semantics of inhibitor nets. We only consider the *a-priori* semantics here, because it leads to more general causal semantics. For the so called *a-posteriori* semantics of inhibitor nets, causal semantics is based on LPOs as for PT-nets.

**Definition 24 (A-Priori Occurrence Rule).** *A multiset of transitions* $\tau$ can occur *in* $m$, *if* $m \geq {}^{\bullet}\tau$ *and* $m \leq {}^{-}\tau$.

*The notion of* $m \xrightarrow{\tau} m'$ *is defined as for PT-nets.*

The notions of sequential executions and step executions are deduced from the occurrence rule as for PT-nets.

The PTI-net shown in Figure 22 has the sequential executions $a, c, ac, ab, abc$ and the additional step execution $(1a, 0b, 1c), (1a, 0b, 0c)(0a, 1b, 1c)$ in the initial marking.

Finally, we recall process semantics of PTI-nets. The problem of defining processes for PTI-nets is that the absence of tokens in a place – this is tested by inhibitor arcs – cannot be directly represented in an occurrence net. This is solved by introducing local extra conditions and read arcs – also called *activator arcs* – connected to these conditions. These extra conditions are introduced "on demand" to directly represent dependencies of events caused by the presence of an inhibitor arc in the net. The conditions are artificial conditions without a reference to inhibitor weights or places of the net. They only focus on the dependencies that result from inhibitor tests. Thus, activator arcs represent local information regarding the lack of tokens in a place.

The process definition is based on the usual notion of occurrence nets extended by activator arcs. These occurrence nets are (labeled) acyclic nets with non-branching conditions whose underlying causal relationship between events is described by so-structures (similar to partial orders describing causal relationships between events of PT-net processes). In the following definition $B$ represents the finite set of *conditions*, $E$ the finite set of *events*, $R$ the *flow relation* and $Act$ the set of *activator arcs* of the occurrence net.

**Definition 25 (Activator Occurrence Net [21]).** *An* activator occurrence net *(ao-net) is a five-tuple* $AON = (B, E, R, Act)$ *satisfying:*

- $B$ *and* $E$ *are finite disjoint sets.*
- $R \subseteq (B \times E) \cup (E \times B)$ *and* $Act \subseteq B \times E$.
- *The rel-structure* $\mathcal{S}(AON) = (E, \prec_{loc}, \sqsubset_{loc}, l|_E) = (E, (R \circ R)|_{E\times E} \cup (R \circ Act), (Act^{-1} \circ R) \setminus id_E, l|_E)$ *is acyclic.*
- $\forall b \in B(|\, {}^{\bullet}b| \leq 1 \wedge |b^{\bullet}\,| \leq 1$.

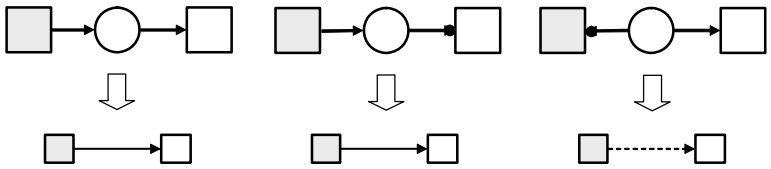


**Fig. 21.** Generation of the orders $\prec_{loc}$ (drawn-through arrow) and $\sqsubset_{loc}$ (dotted arrow) in $ao$-nets. Activator arcs have a filled cirlce as arrowhead.

*For $x \in E$ and $X \subseteq E$ we denote $x^+ = \{y \mid (y,x) \in Act\}$ and $X^+ = \bigcup_{x \in X} x^+$.*

Note that this definition is a conservative extension of standard occurrence nets by read arcs. The relations $\prec_{loc}$ and $\sqsubset_{loc}$ represent the local information about causal relationships between events. Figure 21 shows their construction rule.

Since an activator occurrence net can be identified with an acyclic directed graph, we can use notations introduced for acyclic directed graphs. For the definition of processes we need the notions of weak configurations and weak slices for $ao$-nets (there is also the notion of strong slices which we do not need in this paper). A set of events $D \subseteq E$ is called a *weak configuration* of $AON$, if $e \in D$ and $f(\prec_{loc} \cup \sqsubset_{loc})^+ e$ implies $f \in D$. A *weak slice* of $AON$ is a maximal (w.r.t. set inclusion) set of conditions $S \subseteq B$ which are $R \circ (\prec_{loc} \cup \sqsubset_{loc})^* \circ R$-independent. $WSL(AON)$ denotes the set of all weak slices.

Each weak slice is of the form $S_C = (C^\bullet \cup Min(AON)) \setminus {}^\bullet C$ for a weak configuration $C$ [21]. For a weak slice $S$ there is always a finite sequence of steps of $\prec_{loc}^+$-independent events $\tau_1 \ldots \tau_n$ with $S \xrightarrow{\tau_1} S_1 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_n} Max(AON)$. This means weak slices represent reachable markings which allow to complete a given process.

Now we are prepared to define processes of PTI-nets as in [20]. The mentioned artificial conditions in such processes are labeled by the special symbol $\lightning$. They are introduced in two kinds of situations:

- A transition $t \in T$ tests a place in the pre- or post-multiset of another transition $w \in T$ for absence of tokens, i.e. $I(p,t) \neq \infty$ and ${}^\bullet w(p) + w^\bullet(p) \neq 0$ for some $p \in P$. Then occurrences $f$ of $w$ and $e$ of $t$ in a process must eventually be ordered via a $\lightning$-condition intended either to ensure that tokens are consumed earlier than the test occurs (${}^\bullet w(p) \neq 0$) or to ensure that tokens are produced not later than the test occurs ($w^\bullet(p) \neq 0$). Such situations are abbreviated by $w \multimap t$.
- A transiton $z$ testing some place for absense of tokens occurs concurrently to transitions $t$ consuming and $w$ producing tokens in this place, i.e. $I(p,z) \neq \infty$ and $p \in {}^\bullet t \cap w^\bullet$ for some $p \in P$. Then occurrences $f$ of $w$ and $e$ of $t$ in a process must eventually be ordered via a $\lightning$-condition intended to ensure that tokens are consumed not later than produced in order to restrict the maximal number of tokens in the place according to the inhibitor weight.

In both situations the two occurrences $f$ and $e$ are adjacent to a common $\lightning$-condition representing the described causal dependency of $f$ and $e$. This means there exists a $\lightning$-labeled condition $b$ such that $(b,e) \in Act$ and $b \in ({}^\bullet f \cup f^\bullet)$. This is abbreviated by $f \multimap\!\bullet\, e$.

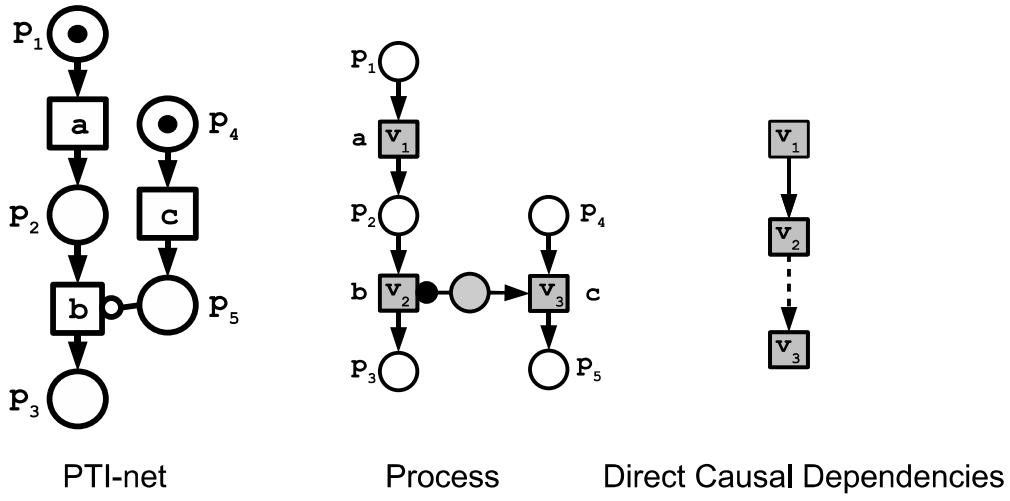



**Fig. 22.** A PTI-net and one of its processes

**Definition 26 (Complete activator process [20]).** *Let $N = (P, T, F, W, I, m_0)$ be a PTI-net. A* complete activator process *(ca-process) of $N$ is an ao-net $AON = (B \uplus \widetilde{B}, E, R, Act)$ together with a labelling function $l : B \uplus \widetilde{B} \cup E \to P \cup T \cup \{\curlywedge\}$ satisfying:*

**(Cond1)** $l(B) \subseteq P$, $l(E) \subseteq T$ *and* $l(\widetilde{B}) = \{\curlywedge\}$.
**(Cond2)** $\widetilde{B} = \{b \mid \exists e \in E((b,e) \in Act)\}$.
**(Cond3)** $m_0 = l(Min(AON) \cap B)$.
**(Cond4)** *For all* $e \in E$: ${}^\bullet l(e) = l({}^\bullet e \cap B)$ *and* $l(e)^\bullet = l(e^\bullet \cap B)$.
**(Cond5)** *For all* $b \in \widetilde{B}$, *there are unique* $g, h \in E$ *such that one of the following properties hold:*
- ${}^\bullet b \cup b^\bullet = \{g\}$, $(b,h) \in Act$ *and* $l(g) \multimap l(h)$.
- $b^\bullet = \{g\}$, $(b,h) \in Act$ *and additionally* ${}^\bullet l(h) \cap l(g)^\bullet \cap {}^- z \neq \emptyset$ *for some* $z \in T$.

**(Cond6)** *For all* $e, f \in E$: *if* $f \multimap e$ *then there is exactly one* $c \in \widetilde{B}$ *such that* $f \multimap e$ *through* $c$.
**(Cond7)** *For all* $e \in E$ *and* $S \in WSL(AON)$: *if* ${}^\bullet e \cup \{b \in \widetilde{B} \mid (b,e) \in Act\} \subseteq S$ *then* $l(S \cap B) \leq {}^- l(e)$.

Figure 22 shows a process of a PTI-net, where names of conditions are omitted. The names of events are shown inside, the labels of events and conditions outside of the graphical object. There is one condition from $\widetilde{B}$, which is filled by grey color. Observe that transition $b$ tests the post-set $c^\bullet$ of $c$ for absence of tokens, i.e. $c \multimap b$. This is reflected in the process by $v_3 \multimap v_2$. The right side of the Figure shows the acyclic rel-structure underlying the process.

The requirements (Cond1), (Cond3), (Cond4) represent common features of processes well-known from PT-nets. They ensure that ca-processes constitute a conservative extension of standard PT-net processes. This means, the set of processes of $Und(N)$ can be derived from the set of ca-processes by omitting the $\curlywedge$-labeled

conditions (omitting the $\lambda$-conditions from a ca-process $AON$ leads to the so called *underlying process* $Und(AON)$ of $AON$). If $N$ has no inhibitor arcs (thus $N = Und(N)$), ca-processes coincide with standard processes.

The properties (Cond2) and (Cond5) together with the rule (Cond6) – describing when $\lambda$-conditions have to be inserted – constitute the structure of the $\lambda$-conditions. The requirement (Cond7) expresses that in the weak slices of $AON$ the inhibitor constraints of the PTI-net have to be properly reflected. That means, for events enabled in a certain weak slice of $AON$ the respective transitions are also enabled in the respective marking in the PTI-net $N$.

If AON is a process of a PTI-net $N$ then the underlying causal relationsships between events $\mathcal{S}(AON)^+$ form an so-structure, because $\mathcal{S}(AON)$ is an acyclic rel-structure. This means, we can represent single (non-sequential) runs of PTI-nets by so-structures labelled by transition names. Such labelled so-structure extend LPOs by adding a second causal relation between transition occurrences which is interpreted as a "not later than"-relation. This relation represents the situation that two transition occurrences can be observed simultaneously and as a sequence in one order (but not the other order). In this model of runs, we can distinguish concurrency from *synchronicity*. Synchronicity means that transiton occurrences can be observed only simultaneously, but not as a sequence in any order. If two transition occurrences are in symmetric "not later than"-relation, then they are synchronous.

**Definition 27 (Labelled SO-Structure).** *A* labelled so-structure *(*LSO*) over $T$ is a $4$-tuple $(V, \prec, \sqsubset, l)$, where $(V, \prec, \sqsubset)$ is an so-structure and $l : V \to T$ is a labelling function on $V$.*

We only consider LSOs up to isomorphism, i.e. only the labelling of events is of interest, but not the event names. Formally, two LSOs $(V, \prec, \sqsubset, l)$ and $(V', \prec', \sqsubset', l')$ are *isomorphic*, if there is a renaming function $I : V \to V'$ satisfying $l(v) = l'(I(v))$, $v \prec w \Leftrightarrow I(v) \prec' I(w)$ and $v \sqsubset w \Leftrightarrow I(v) \sqsubset' I(w)$.

As a special kind of LSOs we consider *linear LSOs*. A linear LSO is an LSO satisfying $co_\prec = \sqsubset \setminus \prec$, This means, the relation $\sqsubset \setminus \prec$ is symmetric and defines synchronous transition occurrences. The maximal sets of synchronous transition occurrences are called *synchronous steps*. The synchronous steps of a linear LSO are linearly ordered w.r.t. the "earlier than"-relation. Linear LSOs cannot be extended and represent (synchronous) step executions of PTI-nets.

The *set of step linearizations* of an LSO is the set of linear LSOs which are extensions of this LSO. For example, $lso_1$ shown in Figure 23 is not linear, since two events are in asymetric "not later than"-relation. The LSOs $lso_2$ and $lso_3$ are step-linearizations of $lso_1$, where

- $lso_2$ represents the step execution $(1a, 0b, 0c)(0a, 1b, 0c)(0a, 0b, 1c)$,
- $lso_3$ represents the step execution $(1a, 0b, 0c)(0a, 1b, 1c)$.

**Definition 28 (LSO-run).** *An LSO $(V, \prec, \sqsubset, l)$ is an* LSO-run *of a PTI-net $N$ if there is a ca-process $AON$ of $N$ such that $(V, \prec, \sqsubset, l)$ is an extension of $\mathcal{S}(AON)$.*

*An LSO-run $lso$ is said to be* minimal*, if there exists no other LSO-run $lso'$ of $N$ such that $lso$ is an extension of $lso'$.*

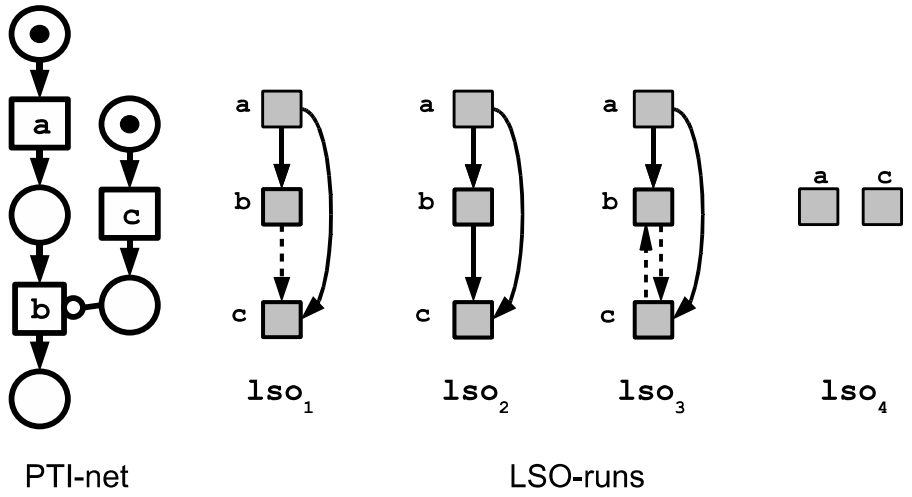



**Fig. 23.** A PTI-net with four of its LSO-runs. The LSOs $lso_2$ and $lso_3$ are step-linearizations of $lso_1$. The LSO-runs $lso_1$ and $lso_4$ are minimal.

Figure 23 shows a PTI-net together with some of its LSO-runs. Note that the LSO-run $lso_1$ exactly represents all causal dependencies between transition occurrences of a process of the net (which is shown in Figure 22). Moreover, $lso_1$ is minimal, since $b$ may not occur simultaneously with $a$ and may not occur after $c$.

From the definition follows that extensions of LSO-runs also are LSO-runs. This means, the set of all LSO-runs can be deduced from the set of minimal LSO-runs.

We show in [20] that an LSO $lso = (V, \prec, \sqsubset, l)$ is an LSO-run of a PTI-net $N$ if and only if each step-linearization of $lso$ is a step execution of $N$. Equivalently, $lso$ is an LSO-run if and only if for each cut $C$ of $lso$ and each place $p$ of $N = (P, T, F, W, I, m_0)$ there holds:

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

and

$$m_0(p) + \sum_{v<C} (W(l(v), p) - W(p, l(v))) \leq I(p, t)$$

for each $t \in l(C)$.

We often omit transitive arrows of LSOs for a clearer presentation.

**Regions of PTI-Nets.** The formal problem statement, which we consider from now, is:

**Given:** A prefix-closed and extension-closed finite language $L$ of LSOs over a finite alphabet of transition names $T$.

**Searched:** A PTI-net $N$ with set of transitions $T$ such that all LSOs in $L$ are LSO-runs of $N$ and $N$ has a minimal number of additional LPO-runs.

As for PT-nets and LPOs, we define transition regions and token flow regions of PTI-nets as as non-negative integral solutions of appropriate linear systems of the form

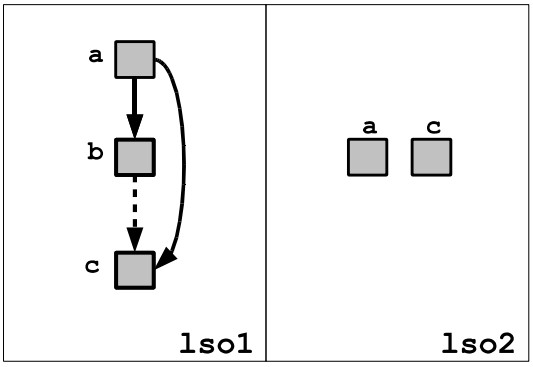



**Fig. 24.** Running example language (prefixes and extensions are not shown)

$\mathbf{A}_L \cdot \mathbf{x} \leq \mathbf{b}_L$. As in the case of LPOs, it is enough to consider only those LSOs from $L$, which are not extensions or prefixes of other LSOs from $L$. Throughout the rest of this subsection we use the language represented by the LSOs shown in Figure 24 as a running example.

A *(PTI-net) transition-region* $\mathbf{r}$ directly defines the parameters of a place $p_\mathbf{r}$ of PTI-nets, i.e. it determines the numbers $m_0(p_\mathbf{r})$, $W(p_\mathbf{r}, t)$ and $W(t, p_\mathbf{r})$ for each $t \in T$ (PT-net part), and $I(p_\mathbf{r}, t)$ for each $t \in T$. If $T = \{t_1, \ldots, t_n\}$, then $\mathbf{r}$ is given as a $(3n+1)$-tuple $\mathbf{r} = (r_0, \ldots, r_{3n})$ of non-negative integers. Its components define these numbers via $m_0(p_\mathbf{r}) = r_0$, $W(p_\mathbf{r}, t_i) = r_i$, $W(t_i, p_\mathbf{r}) = r_{n+i}$ and $I(p_\mathbf{r}, t_i) = r_{n+2i}$ for $i \in \{1, \ldots, n\}$. In the running example, denote $t_1 = a$, $t_2 = b$ and $t_3 = c$.

Since a region $\mathbf{r}$ is intended to define a *feasible* place $p_\mathbf{r}$, it is required to satisfy a property $(f)_L$ ensuring that $p_\mathbf{r}$ is feasible w.r.t. $L$. Remember that $p_\mathbf{r}$ is feasible w.r.t. $L$ if the net resulting from adding $p_\mathbf{r}$ still generates at least $L$. For this, the property $(f)_L$ formalizes that

- (as in the PT-net case) for each cut of events there are at least as much tokens in $p_\mathbf{r}$ as consumed by the occurrence of the corresponding step of transitions after the occurrence of the prefix preceeding the cut (PT-net constraint).
- additionally for each cut of events there are at most as much tokens in $p_\mathbf{r}$ as required by inhibitor tests of transitions in the corresponding step of transitions after the occurrence of the prefix preceeding the cut (inhibitor constraint).

In the running example transition step $(1b, 1c)$ of $lso1$ must be able to occur after one occurrence of $a$. This means, $p_\mathbf{r}$ has to satisfy

- $m_0(p_\mathbf{r}) - W(p_\mathbf{r}, a) + W(a, p_\mathbf{r}) \geq W(p_\mathbf{r}, b) + W(p_\mathbf{r}, c)$,
- $m_0(p_\mathbf{r}) - W(p_\mathbf{r}, a) + W(a, p_\mathbf{r}) \leq I(p_\mathbf{r}, b)$,
- $m_0(p_\mathbf{r}) - W(p_\mathbf{r}, a) + W(a, p_\mathbf{r}) \leq I(p_\mathbf{r}, c)$,

i.e. $r_0 - r_1 + r_4 \geq r_2 + r_3$, $r_0 - r_1 + r_4 \leq r_8$ and $r_0 - r_1 + r_4 \leq r_9$.

The property $(f)_L$ for a finite language $L$ of LSOs and PTI-nets contains all PT-net constraints and additionally the following *inhibitor constraint*: For each $lso = (V, <, \sqsubset, l) \in L$, for each cut $C$ of $lso$ and for each $t = t_k \in l(C)$:

$$r_0 + \sum_{i=1}^{n} l(V')(t_i)(r_{n+i} - r_i) - r_{2n+k} \leq 0,$$

where $V' = \{v \in V \mid v \prec C\}$. This is the case if and only if $\mathbf{b}_{lso,C,t} \cdot \mathbf{r} \leq 0$ for the vector $\mathbf{b}_{lso,C,t} = (b_{C,t,0}, \ldots, b_{C,t,3n})$ defined by

$$\mathbf{b}_{C,t,j} = \begin{cases} 1 & \text{if } j = 0, \\ -l(V')(t_j) & \text{if } j \in \{1, \ldots, n\}, \\ l(V')(t_{j-n}) & \text{if } n+1 \leq j \leq 2n, \\ -1 & \text{if } j = 2n + k, \\ 0 & \text{else} \end{cases}$$

For the cut $C$ corrsponding to the transition step $(1b, 1c)$ of $lso1$ in the running example we require $r_0 - r_1 + r_4 \leq r_8$ and $r_0 - r_1 + r_4 \leq r_9$. This is the case if and only if $\mathbf{b}_{lso1,C,b} \cdot \mathbf{r} \leq 0$ and $\mathbf{b}_{lso1,C,c} \cdot \mathbf{r} \leq 0$ for

$$\mathbf{b}_{lso1,C,b} = (1, -1, 0, 0, 1, 0, 0, 0, -1, 0),$$

$$\mathbf{b}_{lso1,C,c} = (1, -1, 0, 0, 1, 0, 0, 0, 0, -1).$$

**Definition 29 (Transition-Region).** *A tuple* $\mathbf{r}$ *as above is called a* transition-region *if it satisfies* $(f)_L$.

For the defined property $(f)_L$ the following theorem holds [27]:

**Theorem 9.** *A tuple* $\mathbf{r}$ *satisfies* $(f)_L$ *if and only if* $p_{\mathbf{r}}$ *is feasible w.r.t.* $L$.

Let $\mathbf{A}_L$ be the matrix consisting of all rows $\mathbf{a}_{lso,C}$ (PT-net constraints) and $\mathbf{b}_{lso,C,t}$ for LSOs $lso = (V, <, \sqsubset, l) \in L$, cuts $C$ of $lso$ and transitions $t \in l(C)$. Since $L$ is assumed to be finite, $\mathbf{A}_L$ is finite. Thus, the set of all regions can be computed as the set of all integral solutions of the homogenous linear inequation system $\mathbf{A}_L \cdot \mathbf{x} \leq \mathbf{0}$.

Note that we never compute the inhibitor weight $\infty$ representing the case that there is no inhibitor restriction. This is not necessary in the case of a finite LSO-language, since each feasible place is bounded (for each feasible place $p$ there is an upper $b \in \mathbb{N}$ such that $m(p) \leq b$ for all reachable markings $m$). In this case, an inhibitor weight exceeding the bound for the number of tokens in a place is equivalent to $\infty$, i.e. does not restrict the behavior. After the computation of a feasible place, it can be simplified by replacing such useless inhibitor weights by the value $\infty$.

All places of the PTI-net from Figure 22 are solutions for the running example language. For example, the region $r_2 = (0, 0, 1, 0, 1, 0, 0, 1, 1, 1)$ defines $p_2$ and the region $r_5 = (0, 0, 0, 1, 0, 0, 0, 1, 0, 1)$ defines $p_5$ (these regions still contain the useless inhibitor value $1$ which can be replaced by $\infty$, since the corresponding places are $1$-bounded).

**Theorem 10 ([27]).** *If* $L$ *is finite then there is a finite matrix* $\mathbf{A}_L$ *such that the set of transition-regions is the set of solutions of the linear inequation system* $\mathbf{A}_L \cdot \mathbf{x} \leq \mathbf{0}$.

A *token flow-region* $\mathbf{r}$ defines a place $p_{\mathbf{r}}$ indirectly by determining the *token flow* w.r.t. this place between transition occurrences in LSOs from $L$, i.e. by directly determining the number of tokens produced by a transition occurrence which are consumed by a subsequent transition occurrence in an LSO specified in $L$. Such numbers are assigned to the "earlier than"-arrows between transition occurrences of LSOs. As for LPOs, for each transition occurrence the number of tokens consumed from the initial marking, the number of tokens which are produced but not further consumed by other transition occurrences and the number of tokens in the initial marking which are not consumed by any transition occurrence of an LSO are considered.

Since inhibitor values cannot be represented by token flows, we define them directly in the same way as for transition regions. If $W = \bigcup_{(V,\prec,\sqsubset,l)\in L} V$ is the set of nodes of LSOs in $L$ and $E = \bigcup_{(V,\prec,\sqsubset,l)\in L} \prec$ is the set of "earlier than"-arrows of LSOs in $L$, then a *(PTI-net) token flow-region* $\mathbf{r}$ is given as a tuple $\mathbf{r} = (r_i)_{i\in W\times\{in,out\}\cup E\cup L\cup T}$ of non-negative integers. The components $r_i$ with $i \in W \times \{in, out\} \cup E \cup L$ are interpreted as in the PT-net case and $I(p_{\mathbf{r}}, t) = r_t$ for $t \in T$. Initial marking and the weight function w.r.t. $p_{\mathbf{r}}$ are defined as in the PT-net case.

Since a region $\mathbf{r}$ is intended to define a *feasible* place $p_{\mathbf{r}}$, it is required to satisfy a property $(wd)_L$ ensuring that $p_{\mathbf{r}}$ is feasible w.r.t. $L$. The property $(wd)_L$ for PTI-nets and LSOs requires addionally to the PT-net constraints that the marking reached after the occurrence of some prefix of an LSO in $L$ does not exceed the inhibitor constraints of transition occurrences subsequent to this prefix, i.e. for each $lso = (V, \prec, \sqsubset, l) \in L$, for each cut $C$ of $lso$ and for each $t \in l(C)$:

$$r_{lso} + \sum_{v\in V\setminus V'} r_{v,in} + \sum_{v'\in V',\, v\in V\setminus V'} r_{(v',v)} + \sum_{v'\in V'} r_{v',out} - r_t \leq 0,$$

where $V' = \{v \in V \mid v \prec C\}$. This is the case it and only if $\mathbf{A}_{L,d} \cdot \mathbf{r} \leq \mathbf{0}$, where for each $t$ and each cut $C$ of an LSO $lso \in L$ with $t \in l(C)$ and $V' = \{v \in V \mid v \prec C\}$ the matrix $\mathbf{A}_{L,d}$ has a row $\mathbf{d}_{C,t} = (d_{C,t,i})_{i\in W\times\{in,out\}\cup E\cup L\cup T}$ defined by

$$d_{n,t,i} = \begin{cases} 1 & \text{if } i = lso, \\ 1 & \text{if } i = (v, in) \wedge v \in V \setminus V', \\ 1 & \text{if } i = (v', v) \wedge v' \in V' \wedge v \in V \setminus V', \\ 1 & \text{if } i = (v', out) \wedge v' \in V', \\ -1 & \text{if } i = t \\ 0 & \text{else.} \end{cases}$$

**Definition 30 (Token Flow-Region).** *A tuple* $\mathbf{r}$ *as above is called a* token flow-region *if it satisfies* $(wd)_L$.

For the property $(wd)_L$ the following theorem holds for PTI-net places:

**Theorem 11 ([5]).** *A tuple* $\mathbf{r}$ *satisfies* $(wd)_L$ *if and only if* $p_{\mathbf{r}}$ *is feasible w.r.t.* $L$.

Let $\mathbf{A}_L$ be the matrix consisting of all rows from the matrices $\mathbf{A}_{L,a}$, $\mathbf{A}_{L,b}$, $\mathbf{A}_{L,c}$ (PT-net constraints) and $\mathbf{A}_{L,d}$ (inhibitor constraint). Since $L$ is assumed to be finite, $\mathbf{A}_L$ is finite. Thus, the set of all token flow-regions can be computed as the set of all integral solutions of the homogenous linear equation system $\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$.

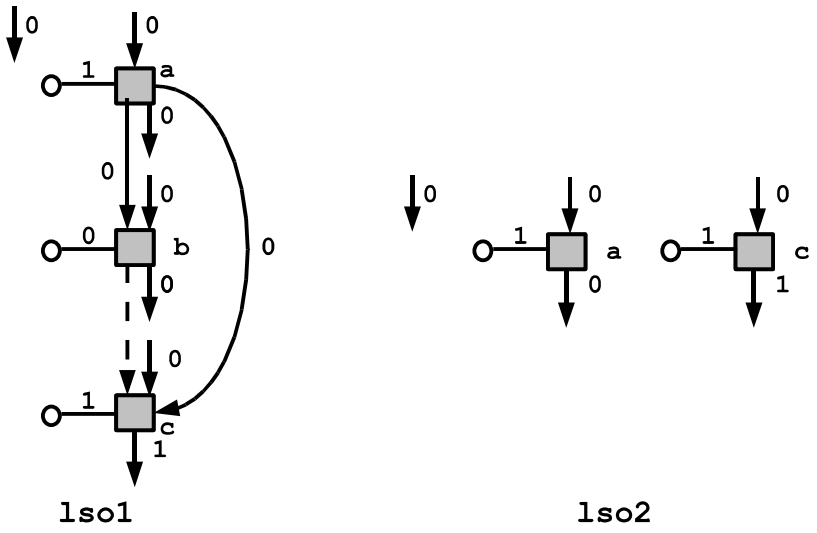



**Fig. 25.** Illustration of a PTI-net token flow region

All places of the PTI-net from Figure 22 are solutions for the running example language. Figure 10 shows a token flow region representing the place $p_5$ from Figure 22. Inhibitor weights are annotated to additional arcs having a circle as arrowhead. Again, useless inhibitor weights exceeding a place bound can be replaced by the value $\infty$. Observe that the token flow leaving the prefix consisting of the occurrence of $a$ of $lso1$ equals 0 such that the inhibitor constraint for the subsequent occurrence of $b$ is fulfilled.

**Theorem 12 ([5,27]).** *If $L$ is finite then there is a finite matrix $\mathbf{A}_L$ such that the set of token flow-regions is the set of solutions of the linear equation system $\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$.*

**Finite Representations of PTI-Net Regions.** As in the case of PT-nets and LPO-languages, the set of PTI-net regions of an LSO-language is defined as the set of positive integral solutions of a homogenuous linear inequation system. Thus, it has a finite basis representation. Many places of this basis represenation are relatively complex, since there is an inhibitor arc connection to every transition. As argued in the last paragraph, these inhibitor weight may be useless. In this case a place can be simplified by replacing the useless inhibitor weight by the value $\infty$. As in the case of PT-nets, many basis places are implicit and can be omitted and in some cases there are easy strategies to compute them. Figure 27 shows some implicit places for the running example in grey color. A detailed examination is a topic of future research.

In order to compute a finite separation representation we need to compute the set $L^c$ of wrong continuations of an LSO-language $L$. As for LPO-languages we denote by $L^{step}$ the set of step linearizations of LSOs in $L$. We need to consider two kinds of wrong continuations: Wrong continuations of an analoguous form as in the LPO-case and wrong continuations representing situations where steps of transitions cannot be sequentialized in any order.

**Definition 31 (Wrong Continuation of an LSO-language).** *A* wrong flow continuation *of an LSO-language $L$ is a sequence of transition steps of the form $w_{\sigma,t}$*

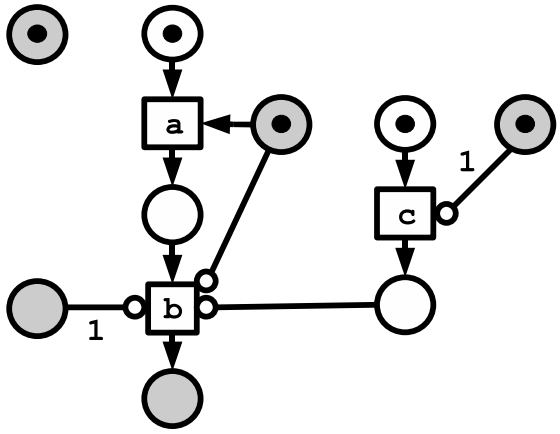



**Fig. 26.** Some basis solution places for the running example. Iimplicit places are filled with grey color.

$= \alpha_1 \ldots \alpha_{n-1}(\alpha_n + t) \notin L^{step}$ *for* $\sigma = \alpha_1 \ldots \alpha_{n-1}\alpha_n \in L^{step}$ *and* $t \in T$*, where we call* $\alpha_1 \ldots \alpha_{n-1}$ *the* prefix *and* $\alpha_n + t$ *the* follower step of the wrong continuation.

*A* wrong inhibitor continuation *of an LSO-language* $L$ *is a sequence of transition steps of the form* $w_{\sigma,\beta_1,\beta_2} = \alpha_1 \ldots \alpha_{n-1}\beta_1\beta_2 \notin L^{step}$ *for* $\beta_1 + \beta_2 \leq \alpha_n$ *and* $\sigma = \alpha_1 \ldots \alpha_{n-1}\alpha_n \in L^{step}$*, where we call* $\alpha_1 \ldots \alpha_{n-1}\beta_1$ *the* prefix *and* $\beta_2$ *the* follower step of the wrong continuation.

Some of the wrong flow continuations of the running example are $(2a), b, 2c$ (all having an empty prefix) and one wrong inhibitor continuation of the running example is $acb$ (since $a(b + c) \in L^{step}$), where multisets are denoted as sums of singleton multisets.

As in the case of LPO-languages, we represent a wrong flow continuation by a pair of multisets $(l(V'), l(S \cup \{z\}))$, where $(V', \prec, \sqsubset, l)$ is a prefix of an LSO in $L$ representing the prefix of the wrong flow continuation, $S$ is a subset of direct successors of $(V', \prec, \sqsubset, l)$ and $z$ is an additional labelled event, where $S$ and $z$ together represent the follower step of the wrong flow continuation.

A wrong inhibitor continuation $\alpha_1 \ldots \alpha_{n-1}\beta_1\beta_2$ we represent by a pair of multisets $(l(V' \cup B_1), l(B_2))$, where $(V', \prec, \sqsubset, l)$ is a prefix of an LSO in $L$ representing $\alpha_1 \ldots \alpha_{n-1}$, $B_1$ is a subset of direct successors of this prefix representing $\beta_1$ and $B_2$ is a subset of direct successors of this prefix representing $\beta_2$.

To prohibit a wrong flow continuation, one needs to find a feasible place $p$ such that one of the following constraints is fulfilled:

- *PT-net constraint*: After occurrence of its prefix there are not as much tokens in $p$ as its follower step consumes.
- *Inhibitor constraint*: After occurrence of its prefix there are more tokens in $p$ as allowed by the inhibitor constraint of the additional event in the follower step. Note that this constraint only can be fulfilled, if the label of the additional event does not occur twice in the follower step.

To prohibit a wrong inhibitor continuation $\alpha_1 \ldots \alpha_{n-1}\beta_1\beta_2$, one needs to find a feasible place $p$ such that the following inhibitor constraint is fulfilled: After occurrence of its prefix there are more tokens in $p$ as allowed by the inhibitor constraints of the events

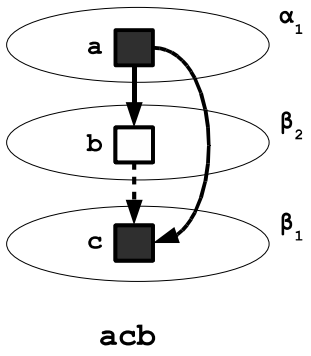



**Fig. 27.** Illustration of the wrong inhibitor continuation $acb$

in the follower step. Note that, since $\alpha_1 \dots \alpha_{n-1}\alpha_n \in L^{step}$ and $\beta_1 + \beta_2 \leq \alpha_n$, after occurrence of $\alpha_1 \dots \alpha_{n-1}\beta_1$ there are always at least as many tokens in a place $p$ as $\beta_2$ consumes.

The PT-net constraint can be expressed as a linear inequality as in the case of LPO-languages and PT-nets. In the following we show, that also the inhibitor constraints of a wrong continuation can represented by a linear inequality.

If $\mathbf{r}$ is a transition region, $T = \{t_1, \dots, t_n\}$ and $l(z) = t_k$ then $p_{\mathbf{r}}$ satisfies the inhibitor constraint for a wrong flow continuation if and only if

$$r_0 + \sum_{i=1}^{n} l(V')(t_i)(r_{n+i} - r_i) - r_{2n+k} > 0.$$

This is the case if and only if $\mathbf{d}(w_{\sigma,t_k}) \cdot \mathbf{r} < 0$ for $\mathbf{d}(w_{\sigma,t_k}) = (d_0, \dots, d_{3n})$ defined by:

$$d_j = \begin{cases} -1 & \text{if } j = 0, \\ l(V')(t_j) & \text{if } j \in \{1, \dots, n\}, \\ -l(V')(t_{j-n}) & \text{if } n+1 \leq j \leq 2n, \\ 1 & \text{if } j = 2n+k, \\ 0 & \text{else} \end{cases}$$

Similarly, if $\mathbf{r}$ is a transition region and $T = \{t_1, \dots, t_n\}$ then $p_{\mathbf{r}}$ satisfies the inhibitor constraint for a wrong inhibitor continuation if and only if, for each $k$ with $t_k \in \beta_2$, $\mathbf{d}(w_{\sigma,t_k}) \cdot \mathbf{r} < 0$ for $\mathbf{d}(w_{\sigma,t_k}) = (d_0, \dots, d_{3n})$ defined as for wrong flow continuations. For example, for the wrong inhibitor continuation $acb$ we require $-r_0 + ((r_1 - r_4) + 2(r_3 - r_6)) + (r_8) < 0$ (remember $t_1 = a$, $t_2 = b$ and $t_3 = c$). This is the case if and only if $\mathbf{d}(w_{\sigma,t}) \cdot \mathbf{r} < 0$ for

$$\mathbf{d}(w_{\sigma,t}) = (-1, 1, 0, 1, -1, 0, -1, 0, 1, 0).$$

The region

$$\mathbf{r} = (0, 0, 0, 0, 0, 0, 1, 1, 0, 1)$$

(corresponding to place $p_5$ in Figure 22 after replacing useless inhibitor weights by $\infty$) is a solution which prohibits this wrong continuation.

If $\mathbf{r}$ is a token flow region and $l(z) = t$ then $p_{\mathbf{r}}$ satisfies the inhibitor constraint if and only if

$$r_{lso} + \sum_{u \notin V'} r_{u,in} + \sum_{v \in V'} r_{v,out} + \sum_{v \in V',\, u \notin V'} r_{v,u} - r_t > 0.$$

This is the case if and only if $\mathbf{d}(w_{\sigma,t}) \cdot \mathbf{r} < 0$ for $\mathbf{d}(w_{\sigma,t}) = (d_i)_{i \in W \times \{in,out\} \cup E \cup L \cup T}$ defined by:

$$d_i = \begin{cases} -1 & \text{if } i = lso, \\ -1 & \text{if } i = (u, in) \wedge u \notin V', \\ -1 & \text{if } i = (v, out) \wedge v \in V', \\ -1 & \text{if } i = (v, u) \wedge v \in V' \wedge u \notin V', \\ 1 & \text{if } i = t, \\ 0 & \text{else.} \end{cases}$$

Similarly, if $\mathbf{r}$ is a token flow region then $p_{\mathbf{r}}$ satisfies the inhibitor constraint if and only if, for $t \in \beta_2$, $\mathbf{d}(w_{\sigma,t}) \cdot \mathbf{r} < 0$ for $\mathbf{d}(w_{\sigma,t}) = (d_i)_{i \in W \times \{in,out\} \cup E \cup L \cup T}$ defined as for wrong flow continuations. The token flow region shown in Figure 25 prohibits the wrong continuation $acb$.

One possible strategy for computing a token flow regions prohibiting a wrong flow continuation is:

- First try to find a solution using the PT-net constraint.
- If there is no solution using the PT-net constraint, then try to find a solution using the inhibitor constraint.

In order to compute a token flow region prohibiting a wrong inhibitor continuation, there is only to possibility to use the inhibitor constraint. For example, the wrong continuation $acb$ cannot be prohibited by a place satisfying the PT-net constraint, since $a(b + c) \in L^{step}$.

There are techniques for computing simple places, as for example:

- If the PT-net constraint is used, all inhibitor weights can be chosen as the value $\infty$. A target function can be used to minimize initial marking and weight on flow arrows as in the case of LPO-languages.
- If the inhibitor constraint is used, through an appropriate target function the inhibitor weight associated to forbidden transitions can be chosen minimal (all other inhibitor weights can be chosen as the value $\infty$).
- The number of wrong inhibitor continuations $\alpha_1 \ldots \alpha_{n-1}\beta_1\beta_2$ can be reduced by considering only singleton multisets $\beta_1$ and $\beta_2$ (since bigger multisets always contain such a singleton).

**Infinite LSO-Languages.** It is possible to extend the presented framework to infinite LSO-languages along the same lines as in the case of LPO-languages. The idea is to use LPO-terms extended by a term-based representation of ”not later than”-relations.

To represent "not later than"-relations between events we introduce a new composition operator $<$ for *weak sequential composition*. *Synchronous composition* between events will be expressed by a new operator $<>$.

Since a term like $(a; b) <> (c; d)$ cannot be interpreted in a meaningful way, we do not apply $<>$ to arbitrary terms, but only to single action names. On the other hand, $<$ can be applied to arbitrary terms: Writing $\alpha < \beta$ for terms $\alpha, \beta$ means that all events in $\alpha$ are in "not later than"-relation to all events in $\beta$.

We do not introduce an operator for the iteration of the operator $<$ for the same reason we do not consider iteration of the operator $\|$: Such iteration operators allow to specify runs with arbitrary large multisets of synchronous resp. independent transition occurrences in the same state of the system (a state of the system can be identified with each prefix of a specified run). Such a behavior cannot be produced through Petri net models.

Let A be a finite set of LSOs. For $A \in \mathcal{A}$ we write $A = (V_A, \prec_A, \sqsubset_A, l_A)$. We denote by $\lambda = (\emptyset, \emptyset, \emptyset, \emptyset)$ the empty LSO. LSOs consisting only of one single event we denote by the label of this event.

**Definition 32 (LSO-term).** *The set of* LSO-terms *over a finite set of LSOs $\mathcal{A}$ is inductively defined as follows:*

- *Each singleton LSO from $\mathcal{A}$ is a* synchronous step.
- *If $s_1$ and $s_2$ are synchronous steps, then $s_1 <> s_2$ is a* synchronous step.
- *The elements $A \in \mathcal{A}$, all synchronous steps and $\lambda$ are* LSO-terms.
- *Let $\alpha_1$ and $\alpha_2$ be* LSO-terms. *Then*
  - $\alpha_1; \alpha_2$ *(sequential composition),*
  - $\alpha_1 < \alpha_2$ *(weak sequential composition),*
  - $\alpha_1 + \alpha_2$ *(union),*
  - $(\alpha_1)^*$ *(iteration),*
  - $\alpha_1 \parallel \alpha_2$ *(parallel composition)*

  *are* LSO-terms.

In the following we consider the running example LSO-term $c^*; (a < b)$. Figure 28 illustrates the LSO-language generated by this term.

We assign to an arbitrary LSO-term $\alpha$ a possibly infinite prefix and extension closed LSO-language $L(\alpha)$. The language $L(\alpha)$ is defined as the prefix and extension closure of an appropriate LSO-language $K(\alpha)$. In order to construct $K(\alpha)$, we define the weak sequential composition of two LSOs $A$ and $B$ by

$$A < B = (V_A \cup V_B, <_A \cup <_B \cup (V_A \setminus Max(A) \times V_B) \cup (V_A \times V_B \setminus Min(B)),$$
$$\sqsubset_A \cup \sqsubset_B \cup (V_A \times V_B), l_A \cup l_B).$$

Sequential composition, parallel composition and iteration of LSOs is defined w.r.t. $\prec$ as for LPOs. We identify synchronous steps with LSOs as follows: If $s_A = (V_A, \prec_A, \sqsubset_A, l_A)$ and $s_B = (V_B, \prec_B, \sqsubset_B, l_B)$ are synchronous steps then

$$s_A <> s_B = (V_A \cup V_B, \emptyset, \sqsubset_A \cup \sqsubset_B \cup (V_A \times V_B) \cup (V_B \times V_A), l_A \cup l_B).$$

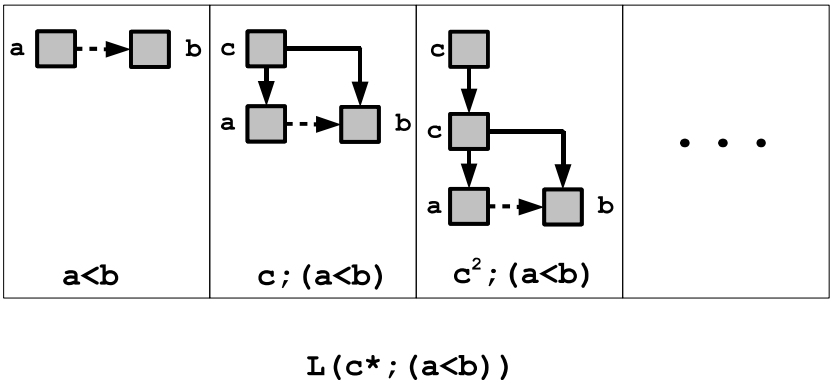



**Fig. 28.** Infinite LSO-language, represented by an LSO-term

**Definition 33 (LSO-language of an LSO-term).** *We define inductively:*

- $K(\lambda) = \{\lambda\}$, $K(A) = \{A\}$ *for* $A \in \mathcal{A}$ *and* $K(s) = s$ *for synchronous steps* $s$.
- *Let* $\alpha_1$ *and* $\alpha_2$ *be LSO-terms. Then:*
  - $K(\alpha_1 + \alpha_2) = K(\alpha_1) \cup K(\alpha_2)$,
  - $K(\alpha_1; \alpha_2) = \{A_1 A_2 \mid A_1 \in K(\alpha_1),\ A_2 \in K(\alpha_2)\}$,
  - $K(\alpha_1 < \alpha_2) = \{A_1 < A_2 \mid A_1 \in K(\alpha_1),\ A_2 \in K(\alpha_2)\}$,
  - $K((\alpha_1)^*) = \{A_1 \ldots A_n \mid A_1, \ldots, A_n \in K(\alpha_1)\} \cup \{\lambda\}$,
  - $K(\alpha_1 \parallel \alpha_2) = \{A_1 \parallel A_2 \mid A_1 \in K(\alpha_1),\ A_2 \in K(\alpha_2)\}$.

Some of the LSOs in $K(c^*; (a < b))$ are $a < b$, $c; (a < b)$ and $c^2; (a < b)$.

Since the additional operations do not introduce side effects to iterations, $K(\alpha)$ can be finitely represented by a representation set $R(\alpha)$ and an iteration set $I(\alpha)$ in an analoguous way as in the case of LPO-terms:

**Definition 34 (Representation/Iteration Set of LSO-terms).** *The* representation set $R(\alpha)$ *and the* iteration set $I(\alpha)$ *of an LSO-term* $\alpha$ *are defined inductively as follows (*$\alpha_1$ *and* $\alpha_2$ *are LSO-terms,* $\mathcal{S}$ *denotes the set of synchronous steps):*

| | |
|---|---|
| $R(\lambda) = \{\lambda\}$ | $I(\lambda) = \emptyset$ |
| $R(A) = \{A\}$ *for* $A \in \mathcal{A}$ | $I(A) = \emptyset$ *for* $A \in \mathcal{A}$ |
| $R(s) = \{s\}$ *for* $s \in \mathcal{S}$ | $I(s) = \emptyset$ *for* $s \in \mathcal{S}$ |
| $R(\alpha_1 + \alpha_2) = R(\alpha_1) \cup R(\alpha_2)$ | $I(\alpha_1 + \alpha_2) = I(\alpha_1) \cup I(\alpha_2)$ |
| $R(\alpha_1; \alpha_2) = \{AB \mid A \in R(\alpha_1), B \in R(\alpha_2)\}$ | $I(\alpha_1; \alpha_2) = I(\alpha_1) \cup I(\alpha_2)$ |
| $R(\alpha_1 < \alpha_2) = \{A < B \mid A \in R(\alpha_1), B \in R(\alpha_2)\}$ | $I(\alpha_1 < \alpha_2) = I(\alpha_1) \cup I(\alpha_2)$ |
| $R((\alpha_1)^*) = R(\alpha_1) \cup \{\lambda\}$ | $I((\alpha_1)^*) = I(\alpha_1) \cup R(\alpha_1)$ |
| $R(\alpha_1 \parallel \alpha_2) = \{A \parallel B \mid A \in R(\alpha_1), B \in R(\alpha_2)\}$ | $I(\alpha_1 \parallel \alpha_2) = I(\alpha_1) \cup I(\alpha_2)$ |

In the running example, $R(c^*; (a < b)) = \{a < b, c; (a < b)\})$ and $I(c^*; (a < b)) = \{c\}$.

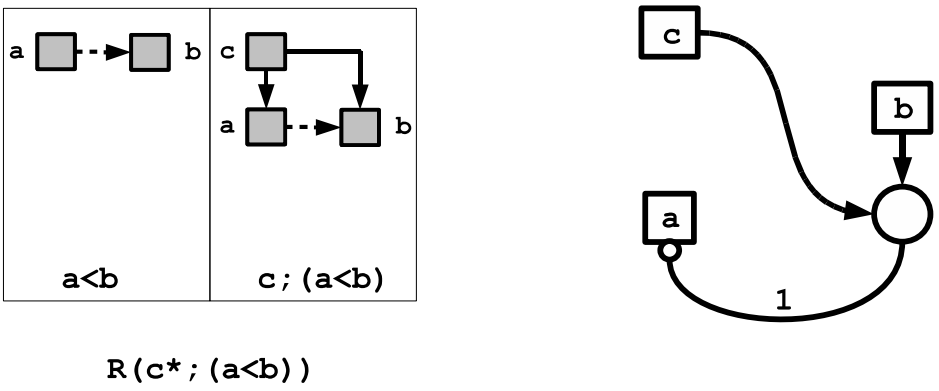



**Fig. 29.** A non-feasible place

We start to characterize the set of feasible places of $L(\alpha)$ analoguously as for LPO-terms by defining regions $\mathbf{r}$ of LSO-terms as regions w.r.t. the finite set $R(\alpha)$ satisfying the additional property

$$Prod(lso, p_{\mathbf{r}}) := \sum_{t \in l(V)} l(V)(t)(W(t, p_{\mathbf{r}}) - W(p_{\mathbf{r}}, t)) \geq 0.$$

for all LSOs $lso = (V, \prec, \sqsubset, l) \in I(\alpha)$. In the running example this means $(W(c, p_{\mathbf{r}}) - W(p_{\mathbf{r}}, c)) \geq 0$.

But the set of places corresponding to such regions still contains places which are not feasible. For example, the place shown in Figure 29 is feasible w.r.t. $R(c^*; (a < b))$, but prohibits $c^2; (a < b)$, because iterations of $c$ add tokens to a place with an inhibitor restriction w.r.t. $a$. The general situation is a place $p$ with the following properties:

- There is an LSO $lso \in I(\alpha)$ with $Prod(lso, p) > 0$ (in the example $lso = c$).
- There is an event $v$ of an LSO in $(V, \prec, \sqsubset, l) \in R(\alpha)$ such that:
  - $I(p, l(v)) < \infty$ (in the example $I(p, a) < \infty$).
  - The prefix of a sub-LSO isomorphic to $lso$ is prefix of the prefix of a cut containing $v$ (the empty prefix in the example).

If these properties are satisfied, $p$ is not feasible because the iteration of $lso$ arbitrarily increases the number of tokens in $p$ such that in the end the occurrence of $l(v)$ is prohibited by $I(p, l(v)) < \infty$. Obviously, we can construct a feasible place from $p$ by changing $I(p, l(v))$ into the value $\infty$ in each such situation. If we do this for all such places $p$, we get the set of all places which are feasible w.r.t. $L(\alpha)$. Some of these places can be simplyfied by replacing useless inhibitor weights by the value $\infty$ as in the finite case, if the place is bounded and the inhibitor weight exceeds the place bound. Bounded places $p$ can be easily found, since they are characterized by $Prod(lso, p) = 0$ for all LSOs $lso \in I(\alpha)$.

Since the set of regions again can be defined as the set of solutions of a linear homogenuous inequation system, it is possible to generate a basis representation. Basis places which are not feasible can be turned into feasible places and simplified as described above.

It is also possible to compute a finite separation representation using the following definition of wrong continuations which combines ideas from wrong continuations of LPO-terms and finite LSO-languages.

**Definition 35 (Wrong Continuation of LSO-term).** *Let $\alpha$ be an LPO-term and let $\sigma = \beta_1 \dots \beta_{n-1}\beta_n \in R(\alpha)^{step}$ and $t \in T$ such that $w_{\sigma,t} = \beta_1 \dots \beta_{n-1}(\beta_n + t) \notin K(\alpha)^{step}$, where $\beta_n$ is allowed to be the empty step. Then $w_{\sigma,t}$ is called* wrong flow continuation of $\alpha$. *We call $\beta_1 \dots \beta_{n-1}$ the* prefix *and $\beta_n + t$ the* follower step of the wrong flow continuation.

*A* wrong inhibitor continuation *of an LSO-term $\alpha$ is a sequence of transition steps of the form $w_{\sigma,\beta_1,\beta_2} = \alpha_1 \dots \alpha_{n-1}\beta_1\beta_2 \notin R(\alpha)^{step}$ for $\beta_1 + \beta_2 \leq \alpha_n$ and $\sigma = \alpha_1 \dots \alpha_{n-1}\alpha_n \in R(\alpha)^{step}$, where we call $\alpha_1 \dots \alpha_{n-1}\beta_1$ the* prefix *and $\beta_2$ the* follower step of the wrong inhibitor continuation.

In the example, some wrong flow continuations are $(a + b)a, c(a + b)b, 2c$ and some wrong inhibitor continuations are $ba, cba$. As for finite LSO-languages, wrong flow continuations can be forbidden using a PT-net constraint or an inhibitor constraint, wrong inhibitor continuations can be only forbidden using an inhibitor constraint. If the PT-net constraint is used, all inhibitor weights can be chosen to be $\infty$ and for some iterated LSOs $lso$ we require $Prod(lso, p_{\mathbf{r}}) = 0$ as in the case of LPO-terms. If the inhibitor constraint w.r.t. an event $v$ is used, we also require $Prod(lso, p_{\mathbf{r}}) = 0$ for some iterated LSOs $lso$, namely if the prefix of a sub-LSO isomorphic to $lso$ is prefix of the prefix of a cut containing an event with label $l(v)$ (as argued above).

Each LPO-term is a special case of an LSO-term. This means, if the synthesis problem of LPO-term has no exact PT-net solution, then can try find an exact PTI-net solution using the extended techniques from this paragraph. For example, consider the LPO-term $((a; b^*) \parallel b^*) + c$ from the subsection on LPO-terms. As discribed, it is not possible to prohibit wrong continuations of the form $b^n c$ by a PT-net place. As illustrated in Figure 30 it is possible to prohibit these wrong continuations by a PTI-net place.

## 3.2 Non-transitive Causal Structures

It is possible to specify the set of runs of a Petri net by non-transitive causal structures like labelled acyclic directed graphs (for PT-nets) or labelled acyclic rel-structures (for PTI-nets). In the following we only consider labelled acyclic directed graphs in more detail. All definitions and results can be extended and generalized to labelled acyclic rel-structures along the same lines as before for LPOs and LSOs.

Acyclic directed graphs labelled by transition names can be used to represent the direct causal dependencies between transition occurrences underlying processes. This means, we require a token flow between transition occurrences along each specified arrow.

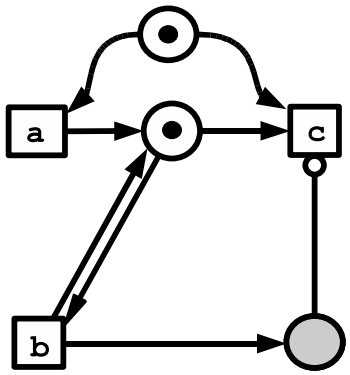



**Fig. 30.** The grey PTI-net place prohibits executions of the form $b^n c$

**Definition 36 (Labelled Acyclic Directed Graph).** *A* labelled acyclic directed graph *(*LDAG*)* over $T$ *is a 3-tuple* $(V, \rightarrow, l)$*, where* $(V, \rightarrow)$ *is an acyclic directed graph and* $l : V \rightarrow T$ *is a labelling function on* $V$*.*

As LPOs, we only consider LDAGs up to isomorphism.

**Definition 37 (LDAG-run).** *An LDAG* $(V, \rightarrow, l)$ *is a* LDAG-run *of a PT-net* $N = (P, T, F, W, m_0)$ *if there is a process* $K = (O, \rho)$*,* $O = (B, E, G)$*, of* $N$ *such that* $(V, \rightarrow) = (E, \{(e, f) \mid e^\bullet \cap {}^\bullet f \neq \emptyset\})$ *and* $l = \rho|_E$*.*

*An LDAG-run* $ldag$ *of* $N$ *is said to be* minimal*, if there exists no other LDAG-run* $ldag'$ *of* $N$ *such that* $ldag$ *is an extension of* $ldag'$*.*

From the definition follows that extensions of LDAG-runs in general are no LDAG-runs (see Figure 31).

The formal synthesis problem statement, which we consider here, is:

**Given:** A language $L$ of LDAGs over a finite alphabet of transition names $T$.
**Searched:** A PT-net $N$ with set of transitions $T$ such that all LDAGs in $L$ are LDAG-runs of $N$ and $N$ has a minimal number of additional minimal LDAG-runs.

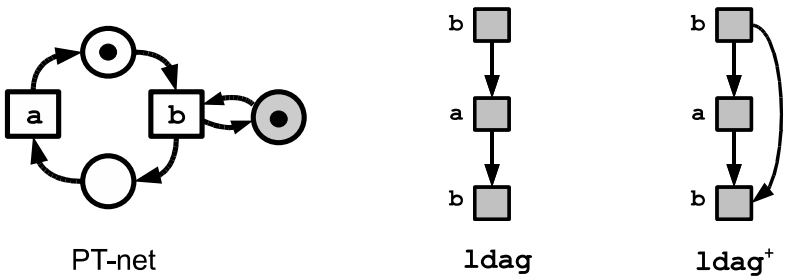



**Fig. 31.** For the PT-net without the grey place, $ldag$ is an LDAG-run, but $ldag^+$ is not an LDAG-run. For the PT-net with the grey place, $ldag^+$ is an LDAG-run, but $ldag$ is not.

On the one side, it is not clear how to define transition regions for LDAG-languages, since direct causal dependencies can be expressed by the components of transition regions.

On the other side, a token flow region contains components directly representing all direct causal dependencies within a specified run. This means, we can define token flow regions of LDAG-languages in an analoguous way as for LPO-languages. If between two transition occurrences there is no arrow specified, then no token flow is possible between these transition occurrences (compare $ldag$ and the PT-net without the grey place in Figure 31). If between two transition occurrences there is an arrow specified, then always a token flow region with positive token flow along this edge can be found (compare $ldag^+$ and the PT-net with the grey place in Figure 31).

The basis representation of the set of token flow regions of an LDAG-language is defined as for LPO-languages. By construction, the basis representation generates all specified LDAG-runs.

In order to define a separating representation of an LDAG-language $L$, observe that if an LDAG $ldag$ is an LDAG-run of a PT-net, then its transitive closure $ldag^+$ is an LPO-run of the PT-net. This means conversely, if $ldag^+$ is not an LPO-run, then $ldag$ is not an LDAG-run. Thus, each wrong continuation of $L^+ = \{ldag^+ \mid ldag \in L\}$ is also a wrong continuation of $L$. The computation of the separating representation is analoguous as in the case of LPOs by representing wrong continuations on the level of LDAGs. It remains to ensure that for each specified arrow there is a place such that there is token flow along this arrow w.r.t. this place, if this is not yet the case. Such places can be computed by token flow regions through requiring a positive token flow on such an arrow edge through an appropriate homogenuous linear inequation. For example, consider $ldag^+$ from Figure 31: The white places of the shown PT-net separate all wrong continuations $(2b), a, bb, b(2a), ba(2b), baa, babb, baba$, while the grey place ensures the direct causal dependency between the two occurrences of transition $b$ specified by the transitive edge in $ldag^+$.

In order to define inifinite LDAG-languages, LDAG-terms can be defined analoguosly as LPO-terms. The only difference concerns the definition of sequential composition $AB$ of LDAGs $A = (V_A, \rightarrow_A, l_A)$ and $B = (V_B, \rightarrow_B, l_B)$ used for the generation of the language of an LDAG-term. Since LDAGs represent only direct causal dependencies between transition occurrences, the sequential composition only introduces such direct causal dependencies between maximal events of the first and minimal events of the second LDAG:

$$AB = (V_A \cup V_B, <_A \cup <_B \cup (Max(A) \times Min(B), l_A \cup l_B).$$

### 3.3 Restricted Net Classes

In this subsection we discuss the synthesis of nets from several restricted net classes. In principle, each restriction which can be encoded as a finite set of linear inequations over the components of regions can be integrated into the defintion of regions. This way, regions can represent places of restricted net classes. In the following paragraphs we briefly suggest several such restrictions.

Some of the restrictions leed to non-homogenuous linear inequations of the form $\mathbf{B} \cdot \mathbf{x} \leq \mathbf{b}$ with $\mathbf{b} \neq \mathbf{0}$. In these cases, the set of solutions has no basis representation, i.e. the separation representation must be computed.

**Bounds for Place Markings.** For finite languages, each prefix of a specified causal structure represents a reachable marking. The number of tokens in a place reached after occurrence of such a prefix can be expressed as a linear sum of components of transition regions and token flow regions. Thus a bound restricting the number of tokens in all places can be introduced by a set of non-homogenuous linear inequations (for each prefix a linear inequation must be added).

For infinite languages, all prefixes of the finite representation set need to be considered. Additionally, iterated parts may not increase the number of tokens in a place.

Using these ideas, places of bounded PT-nets and bounded PTI-nets can be computed.

**Bounds for Flow Weights.** Also flow weigths can be can be expressed as a linear sum of components of transition regions and token flow regions:

- Transition regions have components directly representing flow weights.
- For token flow regions, the intoken flows and outtoken flows represent flow weights.

This means, a bound restricting all flow weights can be introduced by a set of non-homogenuous linear inequations.

Using the bound $1$ for markings and flow weights, it is possible to compute places of elementary nets.

**Bounds for Inhibitor Weights.** Analoguously to flow weights, inhibitor weights can be bounded, because they are directly represented by components of transition regions and token flow regions.

Using the bound $0$ for inhibitor weights, it is possible to compute places of simple inhibitor nets.

**Final Markings.** A final marking is a marking reached after occurrence of a complete specified causal structure. It is possible to introduce combinations of the following useful restrictions for final markings by sets of linear inequations:

- All final markings of some subset of specified causal structures are equal (homogenuous linear inequations).
- The final marking of some causal structure is bounded (non-homogenuous linear inequations).
- The final marking of some causal structure equals a fixed number.

For infinite languages, in the second and third case, all final markigs of the finite representation set need to be considered. Additionally, iterated parts may not increase the number of tokens in a place.

**Initial Markings.** It is possible to introduce combinations of the following useful restrictions for initial markings by sets of linear inequations:

- The initial marking of all places is bounded (non-homogenuous linear inequations).
- The initial marking of all places is a fixed number.

It is possible to compute intermediate places of sound workflow nets by requiring that all initial and final markings equal the number $0$.

### 3.4 More on Infinite Languages

LPO-terms (and LSO-terms) only represent a small class of LPO-languages. In the following we show several examples of languages which cannot be represented by LPO-terms and discuss possibilities to generalize LPO-terms in order to cover some of these examples.

Figure 33 shows a simple example of a PT-net-language which cannot be represented by an LPO-term. The reason for this is that by the iteration operator it is not possible to append LPOs to a part of a previous LPO, but only to the whole LPO.

Therefore *generalized LPO-terms*, allowing to iteratively append LPOs only to parts of previous LPOs, are introduced in [7]. These parts, which we call *interfaces*, are given by prefixes containing maximal events of the LPO. In the example, the LPO $a; (b \parallel c)$ is iterated only w.r.t. its prefix $a; b$. This can be expressed by operators $;_X$ and $^{*X}$ for sequential composition and iteration w.r.t. an interface $X$.

Figure 33 shows a simple safe Petri net, whose LPO-language even cannot be represented by such generalized LPO-terms. This is because for no choice of an iterated part there is a prefix of the iterated part, such that all subsequent events causally depend on all events from this prefix. Instead, different subsequent events depend on different prefixes.

One possibility to further generalize interfaces is to specify direct causal dependencies between events of the previous LPO and events of the subsequent LPO by pairs of multisets of action names.

The PT-net-language in the last example then can be represented by the term $((a \parallel c);_X (d \parallel b))^{*Y}$ with

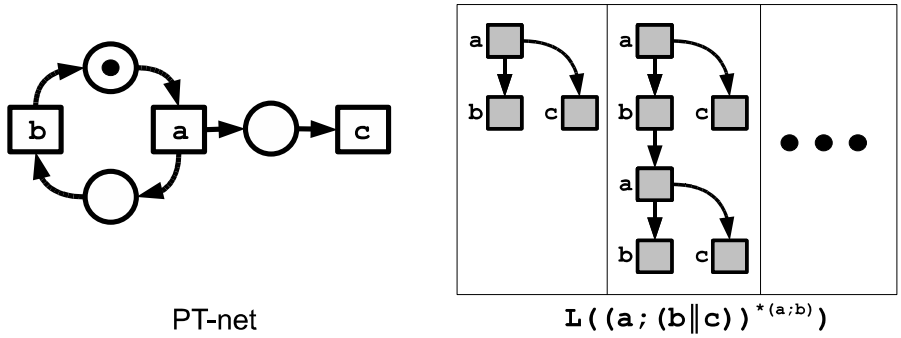



**Fig. 32.** A PT-net whose set of LPO-runs cannot be represented by an LPO-term

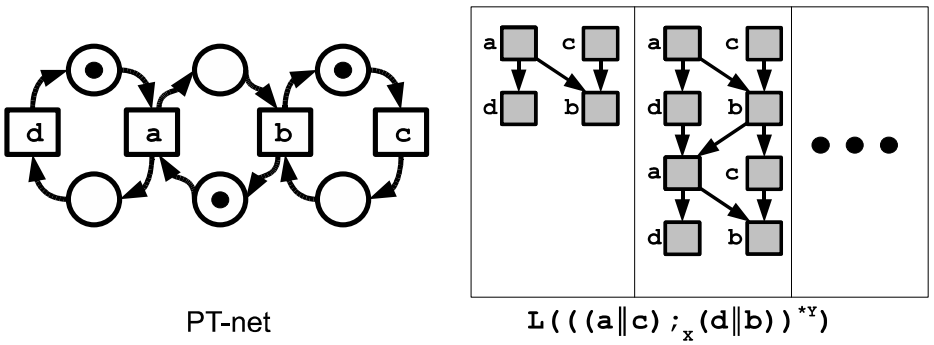



**Fig. 33.** A PT-net whose set of runs cannot be represented by a generalized LPO-term.

- an interface set $X = \{(a, b + d), (a + c, b)\}$ defining which events of the second LPO are directly causally dependent on which events of the previous LPO.
- and an interface set $Y = \{(d + b, a), (b, a + c)\}$ defining which events of the subsequent occurrence of the iterated LPO are directly causally dependent on which events of the previous occurrence of the iterated LPO.

In general an interface $X = \{(A_1, B_1), \ldots, (A_k, B_k)\}$ is interpreted as follows: If an LPO $(V, <, l) = lpo_1 ;_X lpo_2$ is constructed from a sequential composition of two LPOs $lpo_1$ and $lpo_2$ w.r.t. this interface, then it satisfies the following properties:

- If $v$ is a maximal event of $lpo_1$ with $l(v) \in \bigcup_i A_i$ and $W'$ is the set of its direct successor events in $lpo_2$, then $l(W') \leq B_i$ for some $B_i$ with $l(v) \in A_i$.
- If $w$ is a minimal event of $lpo_2$ with $l(w) \in \bigcup_i B_i$ and $V'$ is the set of its direct predecessor events in $lpo_1$, then $l(V') = A_i$ for some maximal multiset $A_i$ among $A_1, \ldots, A_n$ with $l(w) \in B_i$.

Consider the interface $X$ in the previous example:

- An occurrence of $a$ can be directly followed by at most one occurrence of $b$ and one occurrence of $d$, since $(a, b + d) \in X$ and $b < b + d$.
- An occurrence of $b$ exactly directly causally depends on one occurrence of $a$ and one occurrence of $c$, since $(a + c, b) \in X$ and $a < a + c$.

Analoguous properties must hold for iterations w.r.t. interfaces.

Figure 34 shows that in presence of several equally labelled events there are several possibilities to sequentially compose LPOs w.r.t. to a given interface.

Note that the interfaces $X = \{(a, b + d), (a + c, b)\}$ and $X' = \{(a, b), (a, d), (c, b)\}$ have a different interpretation. According to $X'$, an occurrence of $a$ is not allowed to have two direct successors. Instead, alternatively action $b$ or action $d$ can occur directly after $a$. Moreover, an occurrence of $b$ directly causally depends on an occurrence of $a$ or an occurrence of $c$, but not on both occurrences.

This approach is very flexible and intuitive, since only direct causal dependencies between actions need to be specified and also multiple direct causal dependencies can be considered (for example, the interface $\{(2b, a)\}$ specufies that $a$ directly causally

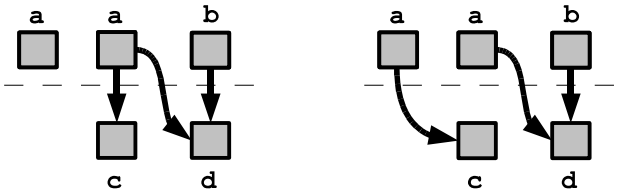



**Fig. 34.** Two instantiations of an interface

depends on two occurrences of $b$). It allows to construct arbitrary LPOs from single action names.

Moreover, it can also be applied to LSOs, as illustrated by Figure 35.

On the other side, as Figure 34 illustrates, the set of possible instantiations of an interface may be complex and difficult to predict from the syntax. Another formulation, which is more clear and restrictive, is to specify interfaces directly through LPOs.

For all mentioned generilizations through interfaces, the notions of transition region and token flow region can be adapted in such a way that they are characterized as solutions of an appropriate homogenuous linear inequations system and such that it is possible to compute a basis representation and a separation representation.

As for LPO-terms and LSO-terms, a finite representation set and a finite iteration set represent the language generated by term with interfaces.

The representation set is defined as for LPO-terms and LSO-terms, where sequential composition is defined w.r.t. a given interface. We define

$$R(\alpha_1 ;_X \alpha_2) = \{A ;_x B \mid A \in R(\alpha_1), B \in R(\alpha_2),\ x \in R(A, B, X)\},$$

where $R(A, B, X)$ is the set of instantiations of $X$ w.r.t. $A$ and $B$ (as described above and illustrated in Figure 34 - we omit a formal definition here) and

$$A ;_x B = (V_A \cup V_B, <_A \cup <_B \cup x, l_A \cup l_B)$$

in the case LPO-terms with interfaces (this definition can be extended to LSO-terms with interfaces in a straightforward way).

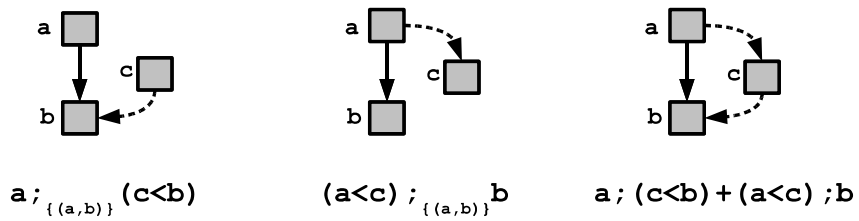



**Fig. 35.** Some LSOs which cannot be constructed from sinlge action names by LSO-terms

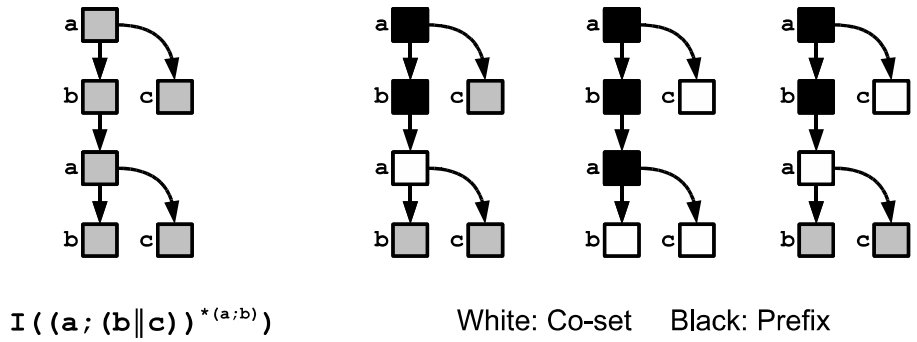



**Fig. 36.** A double iteration with some co-sets of events, which are considered for the definition of regions

In order to reflect interface connections, the iteration set is not defined from single iterations, but double iterations:

$$I(\alpha^{*X}) = \{A;_x B \mid A, B \in R(\alpha),\ x \in R(A, B, X)\} \cup I(\alpha).$$

For each double iteration $A;_x B$ we require that $B$ can occur solely consuming tokens produced by $A$. This means that each co-set of $A;_x B$ containing events from $B$ can occur after the occurrence of its prefix in $A;_x B$. This can be encoded by linear inequations in the usual way, where "initial markings" of double iterations are chosen consistent with markings reached after prefixes of corresponding iterated parts in $R(\alpha)$. Figure 36 illustrates some of the considered co-sets for the example shown in Figure 32.

## 4 Conclusion

In this paper we gave a survey on region based synthesis of Petri nets from languages. We considered place/transition nets (PT-nets), inhibitor nets (PTI-nets) and several restrictions of these net classes on the one side, and languages of labelled partial orders, labelled stratified orders, labelled acyclic graphs and labelled relational structures on the other side. The presented framework includes synthesis from finite languages and several classes of infinite languages finitely represented in term based notations and integrates all classical results on sequential languages.

Most of the results are combinations and reformulations of results from [27], [26] and [7]. There are some easy new adaptions of techniques for token flow regions to transition regions as the synthesis of PT-nets from transition regions of LPO-terms. New developments, which are not yet published, are:

- Computation of the separation representation of regions of LPO-terms.
- Computation of the separation representation of regions of finite LSO-languages.
- Definition of LSO-terms and synthesis from LSO-terms.
- Synthesis from non-transitive causal structures.
- Synthesis of nets from terms with general interfaces.

There is tool support for several of the presented techniques:

- The graphical Petri net editor VIPtool [13] supports business process modelling and synthesis and has also verification and simulation capabilities.
- The command line tool Synops [24] supports the term-based construction of partial languages and the synthesis of Petri nets from partial languages.

## References


1. Seventh International Conference on Application of Concurrency to System Design (ACSD 2007), July 10-13, Bratislava, Slovak Republic. IEEE Computer Society (2007)
2. Badouel, E., Darondeau, P.: On the Synthesis of General Petri Nets. Technical Report 3025, Inria (1996)
3. Badouel, E., Darondeau, P.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
4. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process Mining Based on Regions of Languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007)
5. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. Fundam. Inform. 88(4), 437–468 (2008)
6. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Construction of process models from example runs. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 243–259. Springer, Heidelberg (2009)
7. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Synthesis of Petri Nets from Term Based Representations of Infinite Partial Languages. Fundam. Inform. 95(1), 187–217 (2009)
8. Busi, N., Pinna, G.M.: Synthesis of Nets with Inhibitor Arcs. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 151–165. Springer, Heidelberg (1997)
9. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Hardware and Petri Nets: Application to Asynchronous Circuit Design. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 1–15. Springer, Heidelberg (2000)
10. Darondeau, P.: Deriving Unbounded Petri Nets from Formal Languages. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 533–548. Springer, Heidelberg (1998)
11. Darondeau, P.: Unbounded Petri Net Synthesis. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 413–438. Springer, Heidelberg (2004)
12. Desel, J.: From Human Knowledge to Process Models. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (eds.) UNISCON. LNBIP, vol. 5, pp. 84–95. Springer, Heidelberg (2008)
13. Desel, J.: VipTool-Homepage (2010), `http://www.fernuni-hagen.de/se/viptool.html`
14. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-Structures. Part I: Basic Notions and the Representation Problem / Part II: State Spaces of Concurrent Systems. Acta Inf. 27(4), 315–368 (1989)
15. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-Structures. Part II: State Spaces of Concurrent Systems. Acta Inf. 27(4), 343–368 (1989)
16. Hoogers, P., Kleijn, H., Thiagarajan, P.: A Trace Semantics for Petri Nets. Information and Computation 117(1), 98–114 (1995)
17. van der Werf, C.H.J., van Dongen, B., Serebrenik, A.: Process Discovery using Integer Linear Programming. Fundamenta Informaticae 94(3), 387–412 (2009)
18. Janicki, R., Koutny, M.: Semantics of Inhibitor Nets. Inf. Comput. 123(1), 1–16 (1995)

19. Josephs, M.B., Furey, D.P.: A Programming Approach to the Design of Asynchronous Logic Blocks. In: Cortadella, J., Yakovlev, A., Rozenberg, G. (eds.) Concurrency and Hardware Design. LNCS, vol. 2549, pp. 34–60. Springer, Heidelberg (2002)
20. Juhas, G., Lorenz, R., Mauser, S.: Complete Process Semantics of Petri Nets. Fundamenta Informaticae 87(3-4), 331–365 (2008)
21. Kleijn, H.C.M., Koutny, M.: Process Semantics of General Inhibitor Nets. Inf. Comput. 190(1), 18–69 (2004)
22. Lodaya, K., Weil, P.: Series-Parallel Posets: Algebra, Automata and Languages. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 555–565. Springer, Heidelberg (1998)
23. Lodaya, K., Weil, P.: Series-Parallel Languages and the Bounded-Width Property. Theor. Comput. Sci. 237(1-2), 347–380 (2000)
24. Lorenz, R.: Synops-Homepage (2010), `http://www.informatik.uni-augsburg.de/lehrstuehle/inf/projekte/synops/`
25. Lorenz, R., Juhás, G., Bergenthum, R., Desel, J., Mauser, S.: Executability of Scenarios in Petri Nets. Theor. Comput. Sci. 410(12-13), 1190–1216 (2009)
26. Lorenz, R., Mauser, S., Bergenthum, R.: Theory of Regions for the Synthesis of Inhibitor Nets from Scenarios. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 342–361. Springer, Heidelberg (2007)
27. Lorenz, R., Mauser, S., Juhás, G.: How to Synthesize Nets from Languages: A Survey. In: Henderson, S.G., Biller, B., Hsieh, M.-H., Shortle, J., Tew, J.D., Barton, R.R. (eds.) Winter Simulation Conference, pp. 637–647. WSC (2007)
28. Mukund, M.: Petri Nets and Step Transition Systems. Int. J. Found. Comput. Sci. 3(4), 443–478 (1992)
29. Pietkiewicz-Koutny, M.: The Synthesis Problem for Elementary Net Systems with Inhibitor Arcs. Fundam. Inform. 40(2-3), 251–283 (1999)
30. Pietkiewicz-Koutny, M.: Synthesising Elementary Net Systems with Inhibitor Arcs from Step Transition Systems. Fundam. Inform. 50(2), 175–203 (2002)
31. van der Aalst, W.M.P., Günther, C.W.: Finding Structure in Unstructured Processes: The Case for Process Mining. In: ACSD [1], pp. 3–12
32. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. Data Knowl. Eng. 47(2), 237–267 (2003)
33. Zhou, M., Cesare, F.D.: Petri Net Synthesis for Discrete Event Control of Manufacturing Systems. Kluwer (1993)