Modeling quantitative aspects of concurrent systems using weighted Petri net transducers

Robert Lorenz

Angaben zur Veröffentlichung / Publication details:

Lorenz, Robert. 2015. "Modeling quantitative aspects of concurrent systems using weighted Petri net transducers." In *Application and theory of petri nets and concurrency: 36th International Conference, PETRI NETS 2015, Brussels, Belgium, June 21-26, 2015*, edited by Raymond Devillers and Antti Valmari, 9115:49–76. Cham: Springer International. https://doi.org/10.1007/978-3-319-19488-2_3.

Modeling Quantitative Aspects of Concurrent Systems Using Weighted Petri Net Transducers

Robert Lorenz(✉)

Department of Computer Science, University of Augsburg, Augsburg, Germany
robert.lorenz@informatik.uni-augsburg.de

Abstract. In this paper we present a basic framework for weighted Petri net transducers (PNTs) for the weighted translation of partial languages (consisting of partial words) as a natural generalisation of weighted finite state transducers (FSTs). Weights may represent cost, time consumption, reward, reliability or probability of a transition execution, i.e. PNTs may serve as a general model to consider such quantitative aspects of process calculi represented by arbitrary partial words.

Concerning weights, we use the algebraic structure of concurrent semirings which is based on bisemirings and induces a natural order on its elements. Using the operations of this algebra, the weight of general partial words can be defined in a natural way and turns out to be compositional.

As desirable, complex PNTs can be composed from simple PNTs through composition operations like union, product, closure, parallel product and also language composition, lifting standard composition operations on FSTs. Composed PNTs yield a compositional computation of weights, except for the case of language composition.

For the quick construction of PNTs and evaluation of PNT-algorithms we developed the tool $\text{PNT}_{\varepsilon}^{\circ\circ\text{L}}$. $\text{PNT}_{\varepsilon}^{\circ\circ\text{L}}$ is a python library based on the framework SNAKES allowing for the modular construction of PNTs through composition operations, the visualization of PNTs, and the simulation of constructed PNTs. We present basic simulation algorithms and use PNTool to show illustrating examples.

Keywords: Petri net · Petri net transducer · Weighted transducer · Labelled partial order · Weighted labelled partial order · Partial language · Semiring · Bisemiring · Concurrent semiring · Cleanness

1 Introduction

In [25] we presented a basic framework for weighted Petri Net Transducers (PNTs). A PNT is essentially a *place/transition net (PT-net)* having transitions equipped with input symbols, output symbols and weights. An labelled partial order (LPO)[1] over the set of input symbols (input-LPO) is translated

[1] Also called *partial words* [14] or *pomsets* [29].

into an LPO over the set of output symbols (output-LPO) via weighted LPO-runs (partially ordered runs) of the net, where weights are coming from an algebraic bisemiring structure. These weights may represent cost, time consumption, reward, reliability or probability of a transition execution. The underlying bisemiring structure provides binary operations of addition, sequential multiplication and parallel multiplication of weights. The sequential multiplication is used for determining the weight of transitions occurring sequentially and the parallel multiplication for determining the weight of transitions occurring in parallel. Each translation of an input-LPO into an output-LPO is assigned a weight which is obtained by the sum of the weights of the LPO-runs of the net relating the input-LPO to the output-LPO. Thus, PNTs define (in a natural way) a weighted translation between partial languages, consisting of general LPOs, over different alphabets and may serve as a general model to consider quantitative aspects of process calculi represented by such LPOs.

We use a special bisemiring structure called *concurrent semirings* [16][2] to represent weights. Concurrent semirings are a bisemiring structure with some additional laws interrelating its operations. They where already used by Gischer [13], who showed that the set of all extension closed sets of LPOs can be equipped with algebraic operations yielding a concurrent semiring. In particular, concurrent semirings have an idempotent addition inducing a natural order on the set of weights. This feature allows to define the weight of a general LPO in a natural way as the supremum of all weights of its sequential parallel extensions (w.r.t. this order). As a fundamental result we showed in [25] that concurrent semirings are the least restrictive idempotent bisemiring structure such that LPOs with less dependencies have bigger weights. Moreover, this weight definition is compositional, i.e. the weight of (sequential or parallel) composed LPOs equals the corresponding bisemiring composition of the weights of its components.

In practical applications, it is important to be able to create complex transducers through composition of simple ones. To this end in [25] we introduced *cleanness* of PNTs and composition operations of union, product, closure, parallel product and language composition on clean PNTs. Cleanness ensures that runs always terminate properly and is shown to be preserved by the above operations. Moreover, we showed that the presented composition operations are compatible with suitable notions of equivalent PNTs.

The presented framework mainly aims at an application in the field of semantic dialogue modelling as described in [40]. In [22,23] we applied PNTs to small case studies in this area, in particular we proposed the translation between utterances (represented by words) and meanings (represented by general LPOs) using PNTs. Other application areas of PNTs are, for example, the specification of man-machine-dialogues or the coordination of intelligent machines (for more details see the related work section).

[2] In [16] concurrent semirings are applied in a trace model of programme semantics. Another axiomatic approach to partial order semantics using algebraic structures extending semirings by an additional operation of concurrent composition is [5] using the notion of trioids.

In order to be able to apply PNTs to practical relevant problems, it is necessary to develop efficient algorithms for the composition of PNTs, the analysis and optimization of PNTs, the computation of weights of weighted LPOs and the translation of partial words. As a basis for the implementation and evaluation of such algorithms, we are developing the tool PNT$_\varepsilon^{\circ\circ\mathrm{L}}$. PNT$_\varepsilon^{\circ\circ\mathrm{L}}$ is a python library whose basic functionalities were developed in the bachelor thesis [32] and presented in [18,24]. Actually, it supports the modular construction of PNTs through composition operations, an export of PNTs in all standard picture formats, in Ti*k*Z-format and in an XML-format based on the standard PNML format. All figures in this paper showing PNTs were generated with PNT$_\varepsilon^{\circ\circ\mathrm{L}}$.

In [31] algorithms for the computation of weights of weighted LPOs and for the translation of partial words were developed. At present we are integrating these algorithms in PNT$_\varepsilon^{\circ\circ\mathrm{L}}$ in an optimized form.

The paper gives an overview of our research on PNTS. It summarizes and integrates the main results and developments from [18,24,25] extended by several examples and remarks and by central algorithms from [31]. It is organised as follows: In section 2 we recall basic definitions, including LPOs, Petri nets and concurrent semirings. In section 3 we introduce weighted LPOs and present fundamental relationships between the weight of LPOs and the algebraic weight structure of concurrent semirings. Then (section 4) we give syntax, semantics, equivalence and composition operations of PNTs. In section 5 we propose algorithms for the computation of LPO weights and the translation of partial word by PNTs and in section 6 we give a brief descrition of PNT$_\varepsilon^{\circ\circ\mathrm{L}}$. Finally, we summarize related word in section 7 and give an detailed outlook on future work in section 8.

2 Basic Definitions and Notations

In this section we recall basic definitions and mathematical notations.

2.1 Mathematical Preliminaries

By $\mathbb{N}_0$ we denote the set of *non-negative integers*, by $\mathbb{N}$ the set of *positive integers*. Given a finite set X, the symbol $|X|$ denotes the *cardinality* of X.

The set of all *multisets* over a set X is the set ${\mathbb{N}_0}^X$ of all functions $m : X \to \mathbb{N}_0$. Addition $+$ on multisets is defined by $(m + m')(x) = m(x) + m'(x)$. The relation $\leq$ between multisets is defined through $m \leq m' \iff \exists m''(m+m'' = m')$. We write $x \in m$ if $m(x) > 0$. A set $A \subseteq X$ is identified with the multiset m_A satisfying $m_A(x) = 1 \iff x \in A \wedge m_A(x) = 0 \iff x \notin A$. A multiset m satisfying $m(a) > 0$ for exactly one element a we call *singleton multiset* and denote it by $m(a)a$.

Given a binary relation $R \subseteq X \times Y$ and a binary relation $S \subseteq Y \times Z$ for sets X, Y, Z, their composition is defined by $R \circ S = \{(x, z) \mid \exists y \in Y((x, y) \in R \wedge (y, z) \in S)\} \subseteq X \times Z$. For $X' \subseteq X$ and $Y' \subseteq Y$ the restriction of R onto $X' \times Y'$ is denoted by $R|_{X' \times Y'}$. 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.

Let A be a finite set of symbols. A *(linear) word* over A is a finite sequence of symbols from A. For a word w its length $|w|$ is defined as the number of its symbols. The symbol ε denotes the *empty word* satisfying $|\varepsilon| = 0$. The empty word is the neutral element w.r.t. concatenation of words: $w\varepsilon = \varepsilon w = w$. By A^* we denote the set of all words over A, including the empty word. A *language over* A is a (possibly infinite) subset of A^*.

A *step over* A is a multiset over A. A *step sequence* or *step-wise linear word* over A is an element of $(\mathbb{N}_0^A)^*$ and a *step language over* A is a (possibly infinite) subset of $(\mathbb{N}_0^A)^*$.

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*. 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 $v^\bullet = \{u \mid v \rightarrow u\}$. A *path* is a sequence of (not necessarily distinct) nodes $v_1 \ldots v_n$ $(n > 1)$ such that $v_i \rightarrow v_{i+1}$ for $i = 1, \ldots, n-1$. A path $v_1 \ldots v_n$ is a *cycle*, if $v_1 = v_n$. A directed graph is called *acyclic*, if it has no cycles. An acyclic directed graph $G' = (V, \rightarrow')$ is an *extension* of an acyclic directed graph $G = (V, \rightarrow)$ if $\rightarrow \subseteq \rightarrow'$. In this case we write $G' \leq G$. 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 *irreflexive partial order* over a set V is a binary relation $< \subseteq V \times V$ which satisfies $\forall v \in V(v \not< v)$ (irreflexivity) and $<=<^+$ (transitivity). We identify a finite irreflexive partial order $<$ over V with the directed graph $(V, <)$. Two nodes $v, v' \in V$ of an irreflexive partial order $po = (V, <)$ 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. The set of minimal nodes of an irreflexive partial order is $\min(po) = \{v \mid {}^\bullet v = \emptyset\}$ and the set of maximal nodes $\max(po) = \{v \mid v^\bullet = \emptyset\}$. We denote by $po|_W = (W, < |_{W\times W})$ the restriction of po to a subset $W \subset V$.

A *reflexive partial order* over V is a binary relation $\leq \subseteq V \times V$ which satisfies $\forall v \in V(v \leq v)$ (reflexivity) and $\forall v \in V(v \leq w \wedge w \leq v \implies v = w)$ (antisymmetry) and which is transitive.

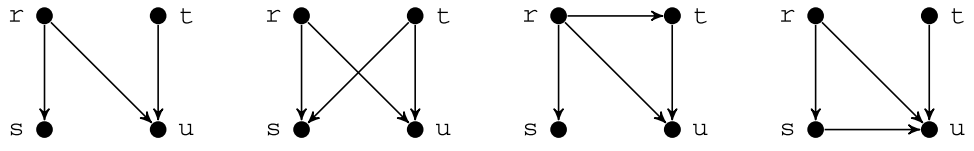
A *semiring* is a quintuple $\mathscr{S} = (S, \oplus, \otimes, \overline{0}, \overline{1})$, where $(S, \oplus, \overline{0})$ is a commutative monoid, $(S, \otimes, \overline{1})$ is a monoid, $\otimes$ (the *S-multiplication*) distributes over $\oplus$ (the *S-addition*) from both sides of $\otimes$ and the zero $\overline{0}$ is absorbing w.r.t. $\otimes$ $(\overline{0}\otimes x = x\otimes\overline{0} = \overline{0})$. If $\otimes$ is commutative, then the semiring is called *commutative*.

2.2 Labelled Partial Orders

We use irreflexive partial orders labelled by action names to represent single non-sequential runs of concurrent systems. The nodes of such a labelled partial order represent events and its arrows an "earlier than"-relation between them in the sense that one event can be observed earlier than another event. If there are no arrows between two events, then these events are independent and are called *concurrent*. Concurrent events can be observed in arbitrary sequential order and simultaneously.

Formally, a *labelled partial order (LPO) over a set* X is a 3-tuple $(V, <, l)$, where $(V, <)$ is an irreflexive partial order and $l : V \to X$ is a labelling function on V. In particular, LPOs are directed graphs, thus all notions introduced w.r.t. directed graphs may also be used for LPOs. LPOs over X are also called *partial words over* X.

In most cases, 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 bijective renaming function $I : V \to V'$ satisfying $l(v) = l'(I(v))$ and $v < w \Leftrightarrow I(v) <' I(w)$. If an LPO *lpo* is of the form $(\{v\}, \emptyset, l)$, then it is called a *singleton LPO* and denoted by $lpo = l(v)$. A set of pairwise non-isomorphic LPOs we call a *partial language*. If L is a partial language, then an LPO $lpo \in L$ is called *minimal (in L)*, if there is no extension of *lpo* in L. In figures, in general we do not show the names of the nodes of an LPO, but only their labels and we often omit transitive arrows of LPOs for a clearer presentation.



lpo_1 lpo_2 lpo_3 lpo_4

Fig. 1. An N-form (lpo_1) together with its minimal sequential parallel extensions (lpo_2, lpo_3, lpo_4).

A *step-wise linear LPO* is an LPO $(V, <, l)$ where the relation $co_<$ is transitive. The maximal sets of independent events of such an LPO are called *steps*. The steps of a step-wise linear LPO are linearly ordered. Thus, step-wise linear LPOs can be identified with step sequences. A *step-linearisation* of an LPO *lpo* is a step-wise linear LPO which is an extension of *lpo*.

The set of *sequential parallel* LPOs (*sp*-LPOs) is the smallest set of LPOs containing all singleton LPOs (over a set X) and being closed under the sequential and parallel product of LPOs. The *sequential product* of two LPOs $lpo_1 = (V_1, <_1, l_1)$ and $lpo_2 = (V_2, <_2, l_2)$ is defined by $lpo_1 ; lpo_2 = (V_1 \cup V_2, <_1 \cup <_2 \cup (V_1 \times V_2), l_1 \cup l_2)$, where V_1 and V_2 are assumed to be disjoint. Their *parallel product* is defined by $lpo_1 \parallel lpo_2 = (V_1 \cup V_2, <_1 \cup <_2, l_1 \cup l_2)$, where again V_1 and V_2 are assumed to be disjoint.

Each sp-LPO (over X) is defined by an *sp-term (over X)*. Such terms are defined as follows:

- Each $x \in X$ is an sp-term.
- If s, t are sp-terms, then also $s ; t$ and $s \parallel t$ are sp-terms.

For an LPO *lpo* we denote by $SP(lpo)$ the set of all sequential parallel extensions of *lpo* and by $SP_{min}(lpo)$ the set of all minimal sequential parallel extensions of *lpo* in $SP(lpo)$.

Given an LPO $lpo = (V, <, l)$, an *N-form* of lpo is a sub-LPO consisting of four nodes $u, v, x, y \in V$ satisfying $u < x$, $u < y$, $v < y$ and $u \mathrel{co_<} v$, $v \mathrel{co_<} x$, $x \mathrel{co_<} y$. An LPO is *N-free*, if it does not contain an N-form. There is the following important relationship between sp-LPOs and N-free LPOs:

Theorem 1. *An LPO is N-free if and only if it is sequential parallel.*

A proof can be found in [13]. This proof is constructive: Given an N-free LPO, it shows a method to construct an sp-term defining the LPO. We will describe and use this method later on.

Figure 1 shows four LPOs: LPO lpo_1 consists of an N-form and the LPOs $lpo_2 = (r \parallel t)\,;(s \parallel u)$, $lpo_3 = r\,;(s \parallel (t\,; u))$ and $lpo_4 = ((r\,; s) \parallel t)\,; u$ are its minimal sequential parallel extensions.

The sequential and parallel product of LPOs is extended to sets of LPOs A, B in the obvious way: $A \parallel B = \{a \parallel b \mid a \in A, b \in B\}$ and $A\,; B = \{a\,; b \mid a \in A, b \in B\}$. Moreover, we define the closure of a set of LPOs A by $A^* = \{a_1\,; \ldots\,; a_n \mid n \in \mathbb{N}, a_i \in A\} \cup \{\varepsilon\}$, where ε denotes the empty LPO having an empty set of nodes.

2.3 Continuous Concurrent Semirings

A binary operation $\oplus$ on a set S defines a binary relation on S via $a \leq_\oplus b :\Leftrightarrow a \oplus b = b$. If $\oplus$ is idempotent, associative and commutative, then this relation is reflexive, transitive and antisymmetric, hence a reflexive partial order. Moreover, if S is equipped with the partial order $\leq_\oplus$, then $\forall a, b \in S :\ a \oplus b = \sup\{a, b\}$, where the supremum is taken w.r.t. $\leq_\oplus$.

If $(S, \oplus, \overline{0})$ is a monoid, and if $T \subseteq S$ is an arbitrary subset, then $\bigoplus T := \bigoplus_{t \in T}\ t := \sup(T)$, where the supremum of the empty set is understood to be the neutral element of the monoid. A semiring $(S, \oplus, \otimes, \overline{0}, \overline{1})$ is called *idempotent*, if $\oplus$ is idempotent. An idempotent semiring is called *continuous* [7], if, for any subset $T \subseteq S$, the supremum is well-defined in S (that means the semiring is *complete*), and $\otimes$ distributes over the supremum from both sides: $\forall s \in S :$ $s \otimes \bigoplus T = \bigoplus_{t \in T}\ s \otimes t$ and $(\bigoplus T) \otimes s = \bigoplus_{t \in T}\ t \otimes s$.

A *bisemiring* is a six-tuple $\mathscr{S} = (S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$, where $(S, \oplus, \otimes, \overline{0}, \overline{1})$ is a semiring and $(S, \oplus, \boxtimes, \overline{0}, \overline{1})$ is a commutative semiring.[3] The binary operation $\boxtimes$ on the set S is called *S-parallel multiplication*. If $\otimes$ distributes over $\boxtimes$ from both sides, the bisemiring is called *distributive*, if $\oplus$ is idempotent, the bisemiring is called *idempotent*, and if both semirings $(S, \oplus, \otimes, \overline{0}, \overline{1})$ and $(S, \oplus, \boxtimes, \overline{0}, \overline{1})$ are continuous, the bisemiring is called *continuous*.

According to [16], a *concurrent semiring* is an idempotent bisemiring $(S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$ satisfying

$$\forall a, b, c, d \in S :\quad (a \boxtimes b) \otimes (c \boxtimes d) \leq_\oplus (a \otimes c) \boxtimes (b \otimes d). \qquad \textbf{(CS)}$$

[3] In particular, both multiplications share the same unit. A similar algebraic structure without requiring commutativity of the second semiring is defined in [6], where it is called *Q-Algebra* and coined for application in quality management. In [20] a slightly different notion of bisemirings is used where parallel multiplication may miss a unit.

Concurrent semirings will be used to define the weight of a run of a Petri net transducer. $\otimes$ will be used to model the composition of weights of a sequence of runs and $\boxtimes$ models the composition of weights of concurrent runs. Therefore, $\boxtimes$ is required to be commutative. The unit $\overline{1}$ can be thought of as the weight of the empty run (the analogue of the empty word). It is shared by $\otimes$ and $\boxtimes$, since the sequential or concurrent execution of a run r and the empty run does not change r. Using $\otimes$ and $\boxtimes$, the weight of a sequential parallel run can be defined then in the standard way (for details see the next section).

As explained above, idempotence of $\oplus$ induces a natural order on the set of weights. We will define the weight of a general run in a natural way as the supremum of all weights of its sequential parallel extensions w.r.t. this order. Condition (**CS**) will ensure that runs with fewer dependencies have bigger weights.

Example 1. If $\mathscr{S} = (S, \oplus, \otimes, \overline{0}, \overline{1})$ is an idempotent semiring such that $\leq_\oplus$ is a total order and $\overline{1}$ is maximal w.r.t. to that order, then we have $\mathscr{S} = (S, \max, \otimes, \overline{0}, \overline{1})$, and $(S, \max, \otimes, \min, \overline{0}, \overline{1})$ is a concurrent semiring extending $\mathscr{S}$.

If $\mathscr{S} = (S, \oplus, \otimes, \overline{0}, \overline{1})$ is an idempotent and commutative semiring, then the *doubled semiring* $(S, \oplus, \otimes, \otimes, \overline{0}, \overline{1})$ is a concurrent semiring extending $\mathscr{S}$.

Example 2. Based on the well-known *Viterbi semiring* $([0,1], \max, \cdot, 0, 1)$ representing probabilities of actions, the structure $\mathscr{V} := ([0,1], \max, \cdot, \min, 0, 1)$ yields a continuous concurrent semiring.

The structure $\mathscr{T} := ([0,\infty], \min, +, \max, \infty, 0)$ is a continuous concurrent semiring. It is based on the well-known *tropical semiring* $([0,\infty], \min, +, \infty, 0)$ representing execution times of actions.

Note that $\mathscr{V}$ and $\mathscr{T}$ are isomorphic, e.g. an isomorphism is given by $t = -\log(v)$. Both concurrent semirings extend a semiring as in the first construction of example 1.

An example of a concurrent semiring, which is not of the above kind, is $\mathscr{A} := (\{-\infty\} \cup [0,\infty[, \max, +, \boxtimes, -\infty, 0)$, where $a \boxtimes b := a + b + \min(a, b)$. It is based on the *arctic semiring* or *max-plus-algebra*.

2.4 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.

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 *flow weight function* satisfying $W(x, y) > 0 \Leftrightarrow (x, y) \in F$. For (transition) steps τ over T we introduce the two multisets of places ${}^\bullet\tau(p) = \sum_{t \in T} \tau(t) W(p, t)$ and $\tau^\bullet(p) = \sum_{t \in T} \tau(t) W(t, p)$. A transition step τ *can occur* in m, if $m \geq {}^\bullet\tau$. If a transition step τ occurs in m, then the resulting marking m' is defined by $m' = m - {}^\bullet\tau + \tau^\bullet$. We write $m \xrightarrow{\tau} m'$ to denote that τ can occur in m and

that its occurrence leads to m'. 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 $m \xrightarrow{\tau_1} m_1 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_n} m_n$. The markings which can be reached from the initial marking m_0 via step executions are called *reachable*.

We use LPOs over T to represent single non-sequential runs of PT-nets, i.e. the events of an LPO represent transition occurrences. An LPO $lpo = (V, <, l)$ over T is an *LPO-run* of a marked PT-net $N = (P, T, F, W, m_0)$ if each step-linearisation of lpo is a step execution of N in m_0. If an LPO-run $lpo = (V, <, l)$ occurs in a marking m, the resulting marking m' is defined by $m' = m - \Sigma_{v \in V} {}^\bullet l(v) + \Sigma_{v \in V} l(v)^\bullet$. We write $m \xrightarrow{lpo} m'$ to denote the occurrence of an LPO-run lpo. 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'.

3 Weighted LPOs

For the representation of runs of weighted Petri net transducers (PNTs), we consider weighted LPOs (WLPOs) which are LPOs with additional node weights [25]. We assume that the set of possible weights is equipped with the algebraic structure of a *concurrent semiring*. Then the total weight of a WLPO is computed from the node weights using binary operations of this algebraic structure. As a central property we showed in [25], that the use of a concurrent semiring ensures that total weights of runs can be computed in a compositional way and that runs with fewer dependencies have bigger weights (w.r.t. the order induced by the idempotent addition operation).

A *weighted LPO* (WLPO) over an alphabet $\mathscr{A}$ and a bisemiring $\mathscr{S} = (S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$ is a quadruple $(V, <, l, \nu)$ such that $(V, <, l)$ is an LPO over $\mathscr{A}$ and $\nu : V \to S$ is an additional *weight function*. We use all notions introduced for LPOs also for WLPOs. Figure 2 shows examples of WLPOs, where labels $l(v) = t$ and weights s are annotated to a node v in the form t/s.

The total weight of sp-WLPOs can be defined through applying $\otimes$ to the sequential product and $\boxtimes$ to the parallel product of sub-WLPOs.

Definition 1 (Weight of sp-WLPOs [25]). *We define the weight* $\omega(wlpo)$ *of an sp-WLPO* $wlpo = (V, <, l, \nu)$ *over a bisemiring inductively as follows:*

- *If* $V = \{v\}$, *then* $\omega(wlpo) = \nu(v)$.
- *If* $wlpo = wlpo_1 \,;\, wlpo_2$, *then* $\omega(wlpo) = \omega(wlpo_1) \otimes \omega(wlpo_2)$.
- *If* $wlpo = wlpo_1 \parallel wlpo_2$, *then* $\omega(wlpo) = \omega(wlpo_1) \boxtimes \omega(wlpo_2)$.

This is the standard technique to define weights of sp-LPOs [20] with weights coming from a bisemiring. In particular, the given weight of sp-WLPOs is well-defined, since the set of sp-WLPOs as well as the sub-structure $(S, \otimes, \boxtimes)$ of a bisemiring $(S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$ form an sp-algebra admitting an sp-algebra homomorphism from the set of sp-WLPOs into the bisemiring.

In [25] we proposed the following weight definition for a general WLPO based on the weigths of its sequential parallel extensions.

Definition 2 (Sequential-Parallel Weight of WLPOs [25]). *Let* $wlpo = (V, <, l, \omega)$ *be a WLPO. Then its* sp-weight *is defined by*

$$\omega_{sp}(wlpo) = \bigoplus_{wlpo' \in SP(wlpo)} \omega(wlpo').$$

As the considered bisemiring of weights is idempotent, the sp-weight of a WLPO-run equals the maximal weight of its sequential parallel extensions.

As a fundamental result we showed in [25] that condition (**CS**) of concurrent semirings is the minimal requirement on idempotent bisemirings such that less restrictive weighted LPOs yield bigger weights.

Theorem 2 ([25]). *Let* $\mathscr{A}$ *be an alphabet and* $\mathscr{S} = (S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$ *be an idempotent bisemiring. Then the following assertions are equivalent:*

(A) *If* u_1, u_2 *are sp-WLPOs over* $\mathscr{A}$ *and* $\mathscr{S}$ *and if* u_1 *is an extension of* u_2*, then* $\omega(u_1) \leq_\oplus \omega(u_2)$*.*
(B) $\mathscr{S}$ *is a concurrent semiring.*

Obviously, one gets a similar result, if inequation **CS** is reversed.

Corollary 1. *Let* $\mathscr{A}$ *be an alphabet and* $\mathscr{S} = (S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$ *be an idempotent bisemiring. Then the following assertions are equivalent:*

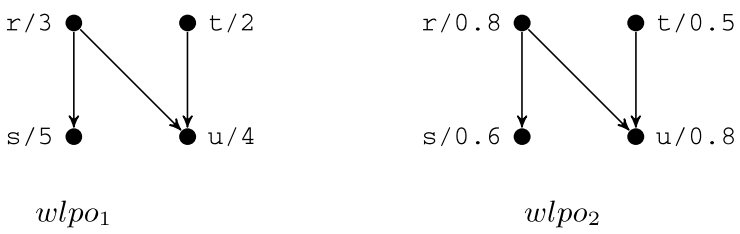
(A) *If* u_1, u_2 *are sp-WLPOs over* $\mathscr{A}$ *and* $\mathscr{S}$ *and if* u_1 *is an extension of* u_2*, then* $\omega(u_1) \geq_\oplus \omega(u_2)$*.*
(B) $\mathscr{S}$ *satisfies*

$$\forall a, b, c, d \in S: \quad (a \boxtimes b) \otimes (c \boxtimes d) \geq_\oplus (a \otimes c) \boxtimes (b \otimes d). \qquad \textbf{(CS')}$$

Moreover, the use of concurrent semirings ensures that the sp-weight of WLPOs can be computed in a modular way using bisemiring-operations [25].

Theorem 3 ([25]). *Let* $\mathscr{A}$ *be an alphabet and* $\mathscr{S} = (S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$ *be a concurrent semiring. Then the following assertions hold for weighted LPOs* $wlpo_1, wlpo_2$ *over* $\mathscr{A}$ *and* $\mathscr{S}$*:*

(C) $\omega_{sp}(wlpo_1 \,;\, wlpo_2) = \omega_{sp}(wlpo_1) \otimes \omega_{sp}(wlpo_2)$*.*
(D) $\omega_{sp}(wlpo_1 \parallel wlpo_2) = \omega_{sp}(wlpo_1) \boxtimes \omega_{sp}(wlpo_2)$*.*



$wlpo_1$ $wlpo_2$

Fig. 2. WLPOs over the concurrent semirings $\mathscr{T}$ ($wlpo_1$) and $\mathscr{V}$ ($wlpo_2$).

Example 3. Consider the concurrent semiring $\mathscr{T}$ defined in subsection 2.3. It can be used to compute the minimal execution time of a run given by an arbitrary WLPO *wlpo* of a concurrent system.

The WLPO $wlpo_1$ shown in figure 2 is a WLPO over $\mathscr{T}$. The weights of its minimal sp-extensions (see figure 1) are:

- $\omega((r \parallel t)\,;(s \parallel u)) = \max(3,2) + \max(5,4) = 8$,
- $\omega(((r\,;s) \parallel t)\,;u) = \max(3+5,2) + 4 = 12$,
- $\omega(r\,;(s \parallel (t\,;u))) = 3 + \max(5, 2+4) = 9$.

Thus, the minimal execution time is $\min(8, 12, 9) = 8$. Note, however, that there is a more efficient method to compute the minimal execution time of a general LPO computing the maximal weigth of a line of the LPO.

Example 4. Consider the concurrent semiring $\mathscr{V}$ defined in subsection 2.3. The decision for min as parallel multiplication can be interpreted as follows: If $wlpo = wlpo_1 \parallel wlpo_2$, then $wlpo_1$ and $wlpo_2$ are both necessary but independent parts of the run *wlpo* of a concurrent system and the probability of *wlpo* cannot be better than the probability of one of its parts. In [39] we give a justification for that choice of min in the context of semantic dialogue modelling.

The WLPO $wlpo_2$ shown in figure 2 is a WLPO over $\mathscr{V}$. The weights of its minimal sp-extensions (see figure 1) are:

- $\omega((r \parallel t)\,;(s \parallel u)) = \min(0.8, 0.5) \cdot \min(0.6, 0.8) = 0.3$,
- $\omega(((r\,;s) \parallel t)\,;u) = \min(0.8 \cdot 0.6, 0.5) \cdot 0.8 = 0.384$,
- $\omega(r\,;(s \parallel (t\,;u))) = 0.8 \cdot \min(0.6, 0.8 \cdot 0.5) = 0.32$.

Thus, the weight (probability) of $wlpo_2$ is $\max(0.3, 0.384, 0.32) = 0.384$.

Remark 1. Concurrent semirings are probably not the most abstract algebraic structure yielding the previous results.

There may be abstractions of the underlying structure of idempotent bisemirings in two directions:

- It is possible to define natural partial orders on non-idempotent semirings via $a < b :\Leftrightarrow \exists c :\ a \oplus c = b$ [7].
- It is possible to consider structures having no shared unit for $\otimes$ and $\boxtimes$. In this context, in [16] the following axioms are used to define so called *concurrent semigroups*:
  - (i) $a \otimes b \leq a \boxtimes b$.
  - (ii) $(a \boxtimes b) \otimes c \leq (a \otimes c) \boxtimes b$.
  - (iii) $c \otimes (a \boxtimes b) \leq (c \otimes a) \boxtimes b$.
  - (iv) $(a \boxtimes b) \otimes (c \boxtimes d) \leq (a \otimes c) \boxtimes (b \otimes d)$ (this condition equals condition (**CS**) of concurrent semirings).

  It is easy to see that axioms (i) to (iii) follow from axiom (iv), if $\otimes$ and $\boxtimes$ share a unit. If this is not the case, axioms (i) to (iv) are irreducible as proven in [16]. The axioms (i) to (iv) seem to be suitable to derive similar results as for concurrent semirings.

We decided to use concurrent semirings because they appear in a natural way in the context of sets of extension closed sp-LPOs as shown by Gischer [13]. However, the mentioned more abstract algebraic structures may make accessible additional practical problems using PNTs. This is a topic of further research.

4 Petri Net Transducers

A PNT is a Petri net which, for every transition occurrence, may read a symbol x from an input alphabet Σ and may print a symbol y from an output alphabet Δ. Additionally, a weight s from a bisemiring is assigned to each transition. If no input symbol should be read or no output symbol should be printed, we use the empty word symbol ε as annotation. We use the basic Petri net class of PT-nets to define PNTs. In graphics an input symbol x, an output symbol y and a weight s of a transition t are annotated to t in the form $x{:}y/s$, and annotations of the form $\varepsilon{:}\varepsilon/\overline{1}$ are not shown.

Definition 3 (Petri Net Transducer [25]). *A* Petri net transducer (PNT) *over a bisemiring* $\mathcal{S} = (S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$ *is a tuple* $N = (P, T, F, W, p_I, p_F, \Sigma, \sigma, \Delta, \delta, \omega)$, *where*

- (P, T, F, W) *is a marked PT-net (called the* underlying PT-net*),*
- $p_I \in P$ *is the* source place *satisfying* ${}^\bullet p_I = \emptyset$,
- $p_F \in P$ *is the* sink place *satisfying* $p_F^\bullet = \emptyset$,
- Σ *is a set of* input symbols,
- $\sigma : T \to \Sigma \cup \{\varepsilon\}$ *is the* input mapping,
- Δ *is a set of* output symbols,
- $\delta : T \to \Delta \cup \{\varepsilon\}$ *is the* output mapping.
- $\omega : T \to S$ *is the* weight function.

We call the marking $m_0 = p_I$ *the* initial marking *and* $m_F = p_F$ *the* final marking. *A PNT is called* clean, *if the final marking is the only reachable marking* m *with* $m(p_F) > 0$.

A WLPO $wlpo = (V, <, l, \nu)$ *over* T *is a* weighted LPO-run *of* N, *if the underlying LPO* $lpo = (V, <, l)$ *is an LPO-run of* N *with* $m_0 \xrightarrow{lpo} m_F$ *and if* $\nu(v) = \omega(l(v))$ *for each* $v \in V$. *We denote by* $WLPO(N)$ *the set of all weighted LPO-runs of* N.

The cleanness property is similar to cleanness of Boxes [4] or soundness of workflow nets [35] and ensures that PNT semantics are closed under (sequential) product and closure. The final marking can be reached only from a finite set of reachable markings [15].

Considering non-sequential semantics of Petri nets, a PNT can be used to translate a partial language into another partial language, where so called input words are related to so called output words. Input and output words are defined as LPOs $lpo_\varepsilon = (V, <, l_\varepsilon)$ with a labelling function $l_\varepsilon : V \to \mathscr{A} \cup \{\varepsilon\}$ for some input or output alphabet $\mathscr{A}$. Such LPOs we call *ε-LPOs*. For each such ε-LPO

we construct the *corresponding* ε*-free LPO* $lpo = (W, <|_{W\times W}, l_\varepsilon|_W)$ by deleting ε-labelled nodes together with their adjacent edges via $W = V \setminus l_\varepsilon^{-1}(\varepsilon)$. Since partial orders are transitive, this does not change the order between the remaining nodes.

Definition 4 (Input and Output Labels of Runs [25]). *Let* $N = (P, T, F, W, p_I, p_F, \Sigma, \sigma, \Delta, \delta, \omega)$ *be a PNT and let* $wlpo = (V, <, l, \nu) \in WLPO(N)$.

The input *of* $wlpo$ *is the* ε*-free LPO* $wlpo_\Sigma$ *corresponding to the* ε*-LPO* $(V, <, \sigma \circ l)$.

The output *of* $wlpo$ *is the* ε*-free LPO* $wlpo_\Delta$ *corresponding to the* ε*-LPO* $(V, <, \delta \circ l)$.

For LPOs u *over* Σ *and* v *over* Δ, *we denote by* $WLPO(N, u)$ *the subset of all WLPOs* $wlpo$ *from* $WLPO(N)$ *with input* $wlpo_\Sigma = u$, *and by* $WLPO(N, u, v)$ *the subset of all WLPOs from* $WLPO(N, u)$ *with output* $wlpo_\Delta = v$.

The input language $L_I(N)$ *of* N *is the set of all inputs of weighted LPO-runs. Its elements are also called* input words. *The* output language $L_O(N)$ *of* N *is the set of all outputs of weighted LPO-runs. Its elements are also called* output words.

The language $L(N)$ *of* N *is the set of all pairs of LPOs* (u, v) *over* $\Sigma \times \Delta$ *with* $WLPO(N, u, v) \neq \emptyset$.

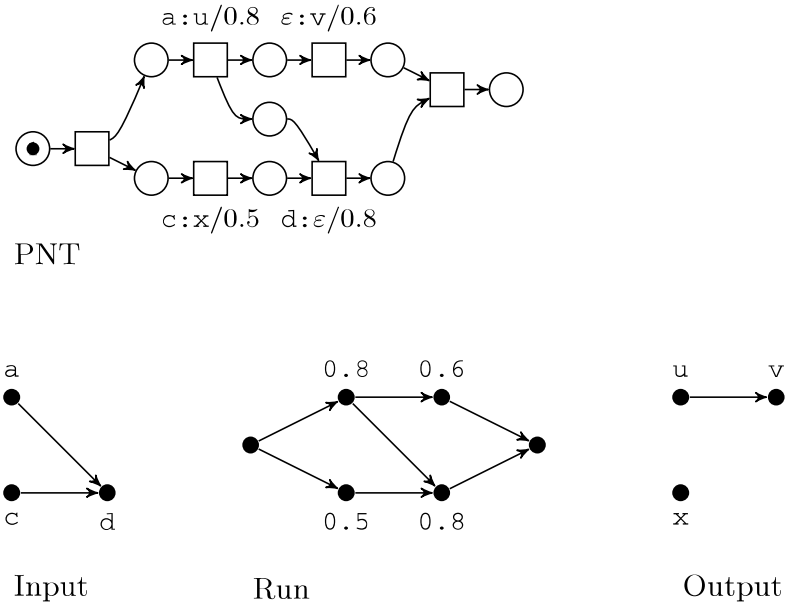


Fig. 3. A PNT together with an LPO-run and associated input and output.

The input and output language of a PNT N are extension closed, since $WLPO(N)$ is extension closed. The *output weight* of a PNT assigned to all pairs of LPOs u over Σ and v over Δ is based on weights of its WLPO-runs.

Definition 5 (Output Weight of PNTs [25]). *Let* $N = (P, T, F, W, p_I, p_F, \Sigma, \sigma, \Delta, \delta, \omega)$ *be a PNT over a concurrent semiring* $\mathcal{S} = (S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$, u *be*

an LPO over Σ and v be an LPO over Δ. The output weight $N(u,v)$ *is defined by*

$$N(u,v) = \bigoplus_{wlpo \in WLPO(N,u,v)} \omega_{sp}(wlpo),$$

when this sum is well-defined in S (note that the sum may be infinite). We set $N(u,v) = \overline{0}$ if $WLPO(N,u,v) = \emptyset$.

The output weight equals the supremum of all weights of corresponding runs, since $\oplus$ is idempotent. If the concurrent semiring is continuous, the supremum always exists in S [7]. From the considerations in the previous section we immediately deduce that it is enough to consider minimal weighted sp-runs in the defining sum of the output weight using condition (**CS**) of concurrent semirings.

Corollary 2 ([25]). *Let $N = (P, T, F, W, p_I, p_F, \Sigma, \sigma, \Delta, \delta, \omega)$ be a PNT over a concurrent semiring $\mathscr{S} = (S, \oplus, \otimes, \boxtimes, \overline{0}, \overline{1})$, u be an LPO over Σ and v be an LPO over Δ. Then*

$$N(u,v) = \bigoplus_{wlpo \in WLPO_{min}(N,u,v), wlpo' \in SP_{min}(wlpo)} \omega(wlpo'),$$

when this sum is well-defined in S, where $WLPO_{min}(\cdot)$ is the subset of all minimal WLPOs in $WLPO(\cdot)$.

Figure 3 shows an example of a PNT together with an LPO-run and associated input and output. The figure illustrates the translation of partial words in the presence of ε-inputs and -outputs. According to example 4 the output weight of the shown input-output-pair equals 0.384.

In practical applications, it is important to be able to create complex transducers through composition of simple ones. For this purpose we introduced in [25] composition operations of union, product, closure and parallel product for clean PNTs. Cleanness ensures that runs always terminate properly and is preserved by the above operations.

For each operation, there are a functional definition defining the output weight of the composed PNT based on the output weights its components and bisemiring-operations and an effective (and more or less straightforward) construction of the composed PNT. In the following, we recall the functional definitions and illustrate the constructions (explicitly given in [25]) in figure 4, where for a compact presentation input symbols, output symbols and weights of transitions are omitted if possible.

The *sum (or union)* $N_1 \oplus N_2$ of two PNTs N_1 and N_2 over $\mathscr{S}$ with the same input alphabet Σ and output alphabet Δ is defined as a PNT over $\mathscr{S}$ in such a way that for each pair of LPOs u over Σ and v over Δ:

$$(N_1 \oplus N_2)(u,v) = N_1(u,v) \oplus N_2(u,v).$$

The *product (concatenation)* $N_1 \otimes N_2$ of two PNTs N_1 and N_2 over $\mathscr{S}$ with the same input alphabet Σ and output alphabet Δ is defined as a PNT over $\mathscr{S}$

in such a way that for each pair of LPOs u over Σ and v over Δ:

$$(N_1 \otimes N_2)(u, v) = \bigoplus_{u=u_1 \,;\, u_2,\, v=v_1 \,;\, v_2} N_1(u_1, v_1) \otimes N_2(u_2, v_2).$$

The product of $n > 0$ instances of a PNT N we denote by N^n. By convention $N^0 = \mathscr{I}$, where $\mathscr{I}$ is the PNT satisfying $\mathscr{I}(u, v) = \overline{1}$ if u and v are both the empty LPO $(\emptyset, \emptyset, \emptyset)$ and $\mathscr{I}(u, v) = \overline{0}$ otherwise.

The *closure* N^* of a PNT N over $\mathscr{S}$ with input alphabet Σ and output alphabet Δ is defined as a PNT over $\mathscr{S}$ in such a way that for each pair of LPOs u over Σ and v over Δ:

$$N^*(u, v) = \bigoplus_{n=0}^{\infty} N^n(u, v).$$

The *parallel product* $N_1 \boxtimes N_2$ of two PNTs N_1 and N_2 over $\mathscr{S}$ with the same input alphabet Σ and output alphabet Δ is defined as a PNT over $\mathscr{S}$ in such a way that for each pair of LPOs u over Σ and v over Δ:

$$(N_1 \boxtimes N_2)(u, v) = \bigoplus_{u=u_1 \| u_2,\, v=v_1 \| v_2} N_1(u_1, v_1) \boxtimes N_2(u_2, v_2).$$

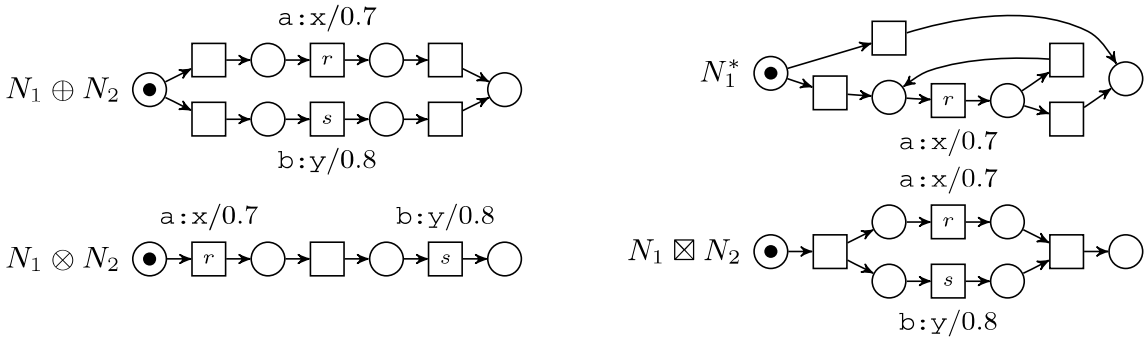


Fig. 4. Illustration of the union, (sequential) product, closure and parallel product of PNTs $N_1 = N(a, x, .7)$ and $N_2 = N(b, y, .8)$ over $\mathscr{V}$.

Moreover, we proposed the following construction of the central transducer composition operation of language composition [25]:

Let N_1 be a PNT over $\mathscr{S}$ with input alphabet Σ_1 and output alphabet Δ_1 and a N_2 be a PNT over $\mathscr{S}$ with input alphabet $\Sigma_2 = \Delta_1$ and output alphabet Δ_2. The composed PNT $N_1[\otimes]N_2$ is constructed as the parallel product of N_1 and N_2, where each transition t_1 from N_1 is merged with each transition t_2 from N_2 satisfying $\delta(t_1) = \sigma(t_2)$ to a transition t with input symbol $\sigma(t) = \sigma(t_1)$ and output symbol $\delta(t) = \delta(t_2)$, weight $\omega(t) = \omega(t_1) \otimes \omega(t_2)$ and connections ${}^\bullet t = {}^\bullet t_1 + {}^\bullet t_2$ and $t^\bullet = t_1^\bullet + t_2^\bullet$. Moreover, all transitions of N_1 having empty output symbol, as well as all transitions of N_2 having empty input symbol are kept with unchanged input symbols, output symbols, weights and connections. All other transitions of N_1 and N_2 are omitted. Figure 5 illustrates the construction.

We proved in [25] that this construction perserves cleaness and yields the following properties concerning compositionality w.r.t. weights:

Theorem 4 ([25]). *The PNT $N_1[\otimes]N_2$ satisfies the following properties:*

(i) If $\mathscr{S}$ is the doubled semiring, then

$$(N_1[\otimes]N_2)(u,w) = \bigoplus_v N_1(u,v) \otimes N_2(v,w),$$

where the sum runs over all LPOs over $\Sigma_2 = \Delta_1$ representing outputs of weighted LPO-runs of N_1 and inputs of weighted LPO-runs of N_2.

(ii) If $(N_1[\otimes]N_2)(u,w) = \bigoplus_v N_1(u,v)$ op $N_2(v,w)$, where the sum runs over all LPOs v over $\Sigma_2 = \Delta_1$ representing outputs of weighted LPO-runs of N_1 and inputs of weighted LPO-runs of N_2 and op *is a semiring operation, then* op $= \otimes$ *and $\mathscr{S}$ is the doubled semiring.*

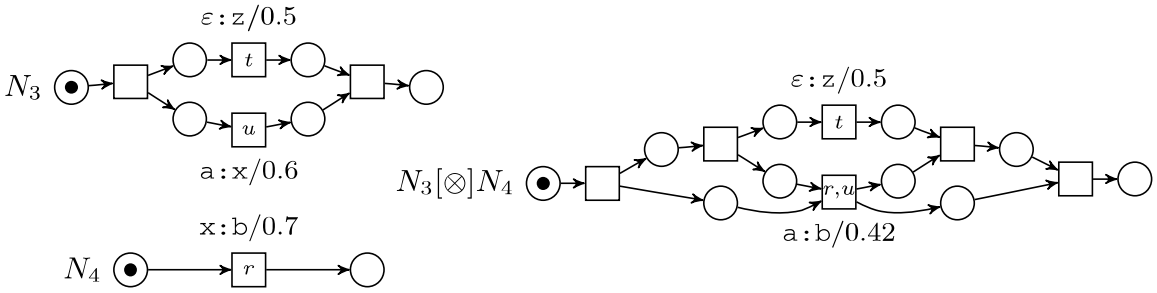


Fig. 5. Language composition for $N_3 = N(x,b,.6) \boxtimes N(\varepsilon,z,.5)$ and $N_4 = N(a,x,.7)$ over $\mathscr{V}$.

Concerning PNT semantics, only the input output behaviour is relevant. Since transitions also may have empty input and/or empty output, there are always (infinitely) many PNTs having the same semantics. For practical application, such PNTs are equivalent.

Definition 6. *Equivalent PNTs [25]] Let N_1, N_2 be two PNTs.*

(a) N_1 and N_2 are called structure equivalent, *if $L(N_1) = L(N_2)$.*

(b) If N_1 and N_2 are structure equivalent, then they are called output equivalent, *if $N_1(u,v) = N_2(u,v)$ for all $(u,v) \in L(N_1) = L(N_2)$.*

Two structure equivalent PNTs perform the same translation between input and output words, but the weights of these translations may be different. Two output equivalent PNTs perform the same weighted translation between input and output words, but the distribution of weights within WLPO-runs may be different.

Example 5. Consider a fixed concurrent semiring serving as the set of weights. We denote by $N(a,b,w)$ the clean PNT consisting of no other places than the source and sink place and exactly one transition with input symbol a, output symbol b and weight w connecting the source with the sink place.

The following PNTs are structure equivalent: $N_1 = N(a, b, w)$, $N_2 = N(a, \epsilon, u) \otimes N(\epsilon, b, v)$ and $N_3 = N(a, \epsilon, x) \boxtimes N(\epsilon, b, y)$. They are output equivalent if $w = u \otimes v = x \boxtimes y$. Moreover, the following PNTs are output equivalent: $N_2 = N(a, \epsilon, u) \otimes N(\epsilon, b, v)$, $N_5 = N(a, \epsilon, v) \otimes N(\epsilon, b, u)$ and $N_6 = N(a, \epsilon, u \otimes v) \otimes N(\epsilon, b, \overline{1})$.

An important application of equivalence in practise is the transformation of a PNT into an equivalent and simpler PNT allowing for more efficient algorithms. A central technique to do this is to replace parts of a complex composed PNT by equivalent parts. This technique requires that equivalence is consistent with composition operations.

To this end, we showed in [25], that the composition operations of union, sequential product, closure and parallel product preserve output equivalence of PNTs and that language composition perserves structure equivalence, but in general does not preserve output equivalence.

An important topic of further research is the deveopment of techniques and rules for the transformstion of a PNT into an equivalent PNT, as for example the removal of ε-transitions, the merging of transitions or the pushing of weights along paths.

5 Algorithms

In order to be able to apply PNTs to practical relevant problems, it is necessary to develop efficient algorithms for the composition of PNTs, the analysis and optimization of PNTs, the computation of weights of weighted LPOs and the translation of partial words. In [18,24] we presented the tool PNT$^{\text{OOL}}_{\varepsilon}$, which supports the modular construction of PNTs through composition operations and an export of PNTs in all standard picture formats, in Ti*k*Z-format and in an XML-format based on the standard PNML format (see the following section).

In this section we briefly present algorithms for the computation of the weight of general WLPOs, of the input and output language and of the output weight of input-output-pairs of PNTs. Basic versions of these algorithms were developed in the master thesis [31]. At present we are integrating these algorithms in PNT$^{\text{OOL}}_{\varepsilon}$.

5.1 Computing the weight of WLPOs

For the computation of the weight of some WLPO we can distinguish between WLPOs which are sequential parallel and WLPOs which are not sequential parallel.

In the first case of sp-WLPOs, according to definition 1, the weight can be computed directly from the sp-term defining the underlying LPO using the binary operations of the concurrent semiring of weights. That means, given a WLPO *wlpo* we need to do the following:

1. Decide, whether *wlpo* is sequential parallel (or, equivalently, N-free).

2. Compute the sp-term defining the LPO underlying *wlpo*, if the answer to the first step is yes.

From the proof of theorem 1 in [13] we can deduce a method, given an N-free LPO, to construct the sp-term defining the LPO in a top down way. The proof shows that if this method fails at some point, then the LPO must contain an N-form. Thus, we can use this method for both steps.

The method is as follows, where we denote the sp-term defining an sp-LPO $lpo = (V, <, l)$ by $sp(lpo)$:

(I) If $V = \{v\}$, then $sp(lpo) = l(v)$.
(II) If the undirected graph underlying *lpo* is not connected, then it can be decomposed into its connected components $lpo_1, \ldots, lpo_n$ and we get $sp(lpo) = sp(lpo_1) \parallel \cdots \parallel sp(lpo_n)$.
Proceed with step (I) applied to $lpo_1, \ldots, lpo_n$.
(III) If $\min(lpo) = \{v\}$ then $sp(lpo) = l(v); sp(lpo|_W)$ with $W = V \setminus \{v\}$.
Proceed with step (I) applied to $lpo|_W$.
(IV) Denote $D = \{v \mid \forall v' \in \min(lpo)(v' < v)\}$. If *lpo* is N-free, then $D \neq \emptyset$ and $lpo = lpo|_{V \setminus D}; lpo|_D$, i.e. $sp(lpo) = sp(lpo|_{V \setminus D}); sp(lpo|_D)$.
Proceed with step (I) applied to $lpo|_{V \setminus D}$ *and* $lpo|_D$.

If the construction of step (IV) fails ($D = \emptyset$ or $lpo \neq lpo|_{V \setminus D}; lpo|_D$), then an N-form can be found in linear time.

It is easy to see by symmetrie, that the above construction can be completed by the following criteria for the N-freeness of an LPO:

(III)' If $\max(lpo) = \{v\}$ then $sp(lpo) = sp(lpo|_W); l(v)$ with $W = V \setminus \{v\}$.
(IV)' Denote $E = \{v \mid \forall v' \in \max(lpo)(v < v')\}$. If *lpo* is N-free, then $E \neq \emptyset$ and $lpo = lpo|_E; lpo|_{V \setminus E}$.

Using these facts, it is possible to adapt the described method also for the computation of the weight of WLPOs which are not N-free. In this case, according to definition 2, it is necessary to compute the maximum of the weights of all minimal sequential parallel extensions. That means we need to compute the sp-terms defining all minimal sequential parallel extensions. This can be done by extending each N-form, which is found in step (IV), in a minimal way. There are three possibilities of minimal extensions of N-forms, all shown in figure 1. We get the following algorithm for the computation of all sp-terms defining minimal sequential parallel extensions of a general LPO:

(I) If $V = \{v\}$, then $sp(lpo) = l(v)$.
(II) If the undirected graph underlying *lpo* is not connected, then it can be decomposed into its connected components $lpo_1, \ldots, lpo_n$ and we get $sp(lpo) = sp(lpo_1) \parallel \cdots \parallel sp(lpo_n)$.
Proceed with step (I) applied to $lpo_1, \ldots, lpo_n$.
(III) (a) If $\min(lpo) = \{v\}$ then $sp(lpo) = l(v); sp(lpo|_W)$ with $W = V \setminus \{v\}$.
Proceed with step (I) applied to $lpo|_W$.

(b) If $\max(lpo) = \{v\}$ then $sp(lpo) = sp(lpo|_W); l(v)$ with $W = V \setminus \{v\}$. *Proceed with step (I) applied to* $lpo|_W$.

(IV) Denote $D = \{v \mid \forall v' \in \min(lpo)(v' < v)\}$ and $E = \{v \mid \forall v' \in \max(lpo)(v < v')\}$.

(a) If $D = \emptyset$ or $lpo \neq lpo|_{V\setminus D}; lpo|_D$, and $E = \emptyset$ or $lpo \neq lpo|_E; lpo|_{V\setminus E}$: Find an N-form and *proceed with step (III) for each minimal extension of lpo w.r.t. this N-form.*

(b) If $lpo = lpo|_{V\setminus D}; lpo|_D$: $sp(lpo) = sp(lpo|_{V\setminus D}); sp(lpo|_D)$. *Proceed with step (I) applied to* $lpo|_{V\setminus D}$ *and* $lpo|_D$.

(c) If $lpo \neq lpo|_E; lpo|_{V\setminus E}$: $sp(lpo) = sp(lpo|_E); sp(lpo|_{V\setminus E})$. *Proceed with step (I) applied to* $lpo|_{V\setminus E}$ *and* $lpo|_E$.

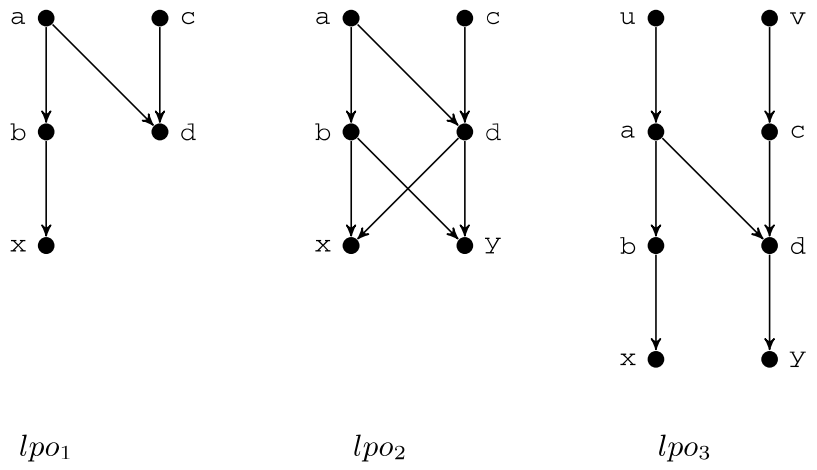
This is a recursive procedure splitting into three paths for each found N-form. That means, its running time is exponential in the number of N-forms contained in the considered LPO. Moreover, a minimal extension of an LPO w.r.t. an N-form may produce additional N-forms. As an example, see LPO lpo_1 from figure 6: If the N-form defined by $\{a, b, c, d\}$ is extended by the edge $b < d$, then the new N-form defined by $\{b, x, c, d\}$ is produced.

On the other side, if a PNT is composed using operation for union, sequential product, closure and/or parallel product, then the above algorithms needs to be applied only to LPO-runs of its PNT-components, since the computation of weights is compositional.

Furthermore, there a several possibilities to optimize the step (IV) concerning the definition of D and E and the choice of the next N-form. For example, we can use the following constructions in order to find possibilities for a sequential composition in a more effective way::

If $D \neq \emptyset$ and $lpo \neq lpo|_{V\setminus D}; lpo|_D$, then there is an N-form with nodes in D and in $V \setminus D$. If we delete all nodes from D belonging to an N-form which is not completely contained in D, we may get a non-empty subset $D' \subset D$ satisfying $lpo = lpo|_{V\setminus D'}; lpo|_{D'}$. As an example, see LPO lpo_2 from figure 6: If we delete the node d belonging to the N-form defined by $\{a, b, c, d\}$ from the set $D = \{d, x, y\}$, we get the set $D' = \{x, y\}$ satisfying $lpo_2 = lpo_2|_{V\setminus D'}; lpo_2|_{D'}$. An analoguous construction holds for E.

If $D = \emptyset$ or $lpo \neq lpo|_{V\setminus D}; lpo|_D$, in general many N-forms can be found. If we minimally extend one of these N-forms, also some other N-forms may be minimally extended because of transitivity. In step (IV)(a) we should choose such an N-form, whose extension also extends a maximal number of other N-forms. As an example, see LPO lpo_3 from figure 6: The N-form defined by $\{a, b, c, d\}$ causes many other N-forms due to transitivity, as for example $\{a, x, c, d\}$, $\{a, b, c, y\}$ or $\{a, b, x, y\}$ (and so on). If we extend the N-form $\{a, x, c, d\}$ by the edge $x < d$, then also the other mentioned N-forms are extended. In this example, the extension $x < d$ of the N-form $\{a, x, c, d\}$ extends a maximal number of other N-forms. A similar argumentation holds for the extension $c < b$ of the N-form $\{a, b, c, d\}$ and the extension $a < v$ of the N-form $\{a, b, v, d\}$. Again, an analoguous construction holds for E.



lpo_1 lpo_2 lpo_3

Fig. 6. Example-LPOs illustrating the computation of minimal series parallel extensions.

An exact and formal definition of such optimizations and their implementation, as well as experimental results are topics of further research. Moreover, it is necessary to have a closer look at the language composition of PNTs. On the one side, the weight of LPO-runs is not compositional w.r.t. to language composition, but on the other side, the structure of LPO-runs of a composed PNT can be determined from LPO-runs of its components. This may be used to construct sp-extensions of LPO-runs of a composed PNT from sp-extensions of LPO-runs of its components.

5.2 Computing the output weight of input-output-pairs

In order to compute the language of a PNT (the set of input-output-pairs), it is necessary to compute all of its LPO-runs. The most effective algorithm doing this for PT-nets is the token flow unfolding algorithm [3].

For PT-nets (underlying a PNT) having a finite set of LPO-runs, the token flow unfolding is finite and contains all LPO-runs of the net.

If a PT-net has infinitely many LPO-runs, then there are two possibilities. The first possibility is, that the PT-net is bounded. In this case the set of reachable markings is finite and it is possible to compute the so called complete finite prefix of the token flow unfolding. This prefix contains for each reachable marking at least one LPO-run leading to this marking. Since it is possible to compute all these markings, we may restrict the finite prefix to those LPO-runs leading to the final marking and we may use the algorithm to test, whether the PT-net is clean.

Note that the behavior of a bounded PNT may contain cycles (leading from a reachable marking back to the same marking). If all cycles produce empty input (empty output), then the input language (output language) is finite, otherwise infinite. It is possible during the computation of the finite prefix, to save additionally all sub-LPOs of LPO-runs which form a cycle. This information makes it possible later on to decide, whether a given input (output) belongs to the input language (output language) of the PNT.

The second possibility is, that the PT-net is unbounded. Then there is at least one unbounded place. If an unbounded place contributes to a reachable marking, from which the final marking can be reached, then the PT-net is not clean. If an unbounded place does not contribute to a reachable marking, from which the final marking can be reached, then it belongs to a useless part of the PNT. Therefore we do not want to consider PNTs having unbounded places.[4] The token flow unfolding algorithm can easily be extended in such a way that unbounded places are recognized.

In concrete applications it may not be of interest to compute the whole behavior of a PNT, but to test, whether a PNT has a given input and/or output. Consider the case of a given input. Then it is possible to restrict the computation of the token flow unfolding to such LPO-runs having the given input. Cycles not having empty input should not be cut (as it is the case for the computation of the finite prefix), but unfolded as many times as needed to get or to exceed the given input. When constructing such a restricted unfolding, special care need transitions having empty input. In particular, cycles with empty input must be cut after their first occurrence and stored for the weight computation. Analoguous considerations hold for the case of a given output and the case of a given input-output-pair.

Finally, if an input-output-pair belongs to the behavior of a PNT, its weight is computet as the supremum of the weights of all LPO-runs producing the input-output-pair. Note that the set of all such LPO-runs may be infinite if there are cycles with empty input and empty output - in this case the weight computation needs special care and depends on the used concurrent semiring.

6 Tool Support

For the modular construction of concrete PNTs in case studies and practical applications and as a basis for the implementation and evaluation of algorithms for analysis, simulation and optimisation of PNTs we are developing the tool $\text{PNT}_{\varepsilon}^{\text{OOL}}$. Its basic functionality was developed in the bachelor thesis [32] and presented in [18,24].

$\text{PNT}_{\varepsilon}^{\text{OOL}}$ is a python [36] library and implemented within the framework SNAKES supporting the rapid prototyping of new Petri net formalisms and providing many basic Petri net components and functionality [27,28]. $\text{PNT}_{\varepsilon}^{\text{OOL}}$ is mainly targeted at researchers in the area of PNTs. By the use of SNAKES it is relatively easy to implement and evaluate extensions, variations and new algorithms, as for example: Composition operations, algebraic weight structures, simulation algorithms and optimisation algorithms.

Constructed PNTs can be exported in an XML-format which is based on the standard PNML format developed for basic Petri net variants [33]. Moreover, PNTs can be visualised and pictures can be exported in all standard formats.

[4] Note that this implies that the sequential language of a PNT is regular. In order to deal with more general languages it would be necessary to use more general net classes combined with the concept of cleanness, as for example inhibitor nets.

The support of graphical output serves both as a possibility to check the implementation and as a handy utility in the process of writing scientific papers. PNT$_\varepsilon^{\circ\circ\mathrm{L}}$'s functionality supports fast construction of concrete example PNTs for case studies. PNML export can be used to analyse constructed example PNTs with other Petri net tools.

In the context of our research activities, PNT$_\varepsilon^{\circ\circ\mathrm{L}}$ serves as a scientific prototype for the development of an open library openPNT of efficient algorithms for the construction, composition, simulation and optimisation of PNTs which can be used in real world examples.

PNT$_\varepsilon^{\circ\circ\mathrm{L}}$ can be downloaded from our website [17] as a ZIP-archive. Assumed you have a working installation of Python, SNAKES, Graphviz, and dot2tex you only need to copy the py-files into the `plugins` sub-directory of your SNAKES installation.

6.1 Functionality

In this subsection we show how PNT$_\varepsilon^{\circ\circ\mathrm{L}}$ is used to construct and compose PNTs. We list the source code of the examples.

To use PNT$_\varepsilon^{\circ\circ\mathrm{L}}$ one has to create a text file and put the following code into it. These lines load the SNAKES library and the `transducer`-plugin.

```
1  import snakes.plugins
2  snakes.plugins.load(['transducer'], 'snakes.nets', 'pnts')
3  from pnts import *
```

Using the class method `N` from the class `PetriNet` a PNT consisting of a source place with one token, a sink place, and a single transition in between can be created (line 4). The parameters of this single transition are provided as named parameters to `N`.

In lines 5 and 6 graphical output of the PNTs is generated. The format of the output is controlled by the extension of the file-name given to the method `draw` as first argument. The orientation of the generated graphs is controlled by the parameter `leftright` of the method `draw`. This parameter is only effective if TikZ-output is to be created. For more available export formats one may consult the documentation of the Graphviz [12] package which is utilised by SNAKES for the export.

A PNT can be saved in PNML-format using the function `savePNML` – a wrapper of SNAKES methods – taking a file-name as second argument (line 7). For loading we provide `loadPNML`.

```
4  n = PetriNet.N('N1', weight = .5, input_symbol = eps,
       output_symbol = 'b')
5  n.draw("N.tikz", leftright = True)
6  n.draw("N.png")
7  savePNML(n, "N.pnml")
```

It is also possible to construct the same PNT (and also more complex PNTs) by adding all components (places with markings, transitions, edges with weights) separately:

```
8   n = PetriNet('N1')
9   n.add_place(Place('p_I', 1), is_source = True)
10  n.add_place(Place('p_F'), is_sink = True)
11  n.add_transition( Transition('t_1', input_symbol=eps,
        output_symbol='b'),weight=0.5)
12  n.add_input('p_I', 't_1')
13  n.add_output('p_F', 't_1')
```

Finally, PNTs can be composed by several compsition operations. An example for the use the operation of parallel composition is shown in line 14.

```
14  n1 = n | n
```

The other available operators are: `*` for concatenation, `+` for union, `~` for closure, and `>` for language composition.

There are many possibilities to influence the graphical output, like renaming of transitions, using subscripts in transition names, adjustment of label positions, and more. Moreover, it is possible to change the marking of a PNT be firing transitions. For details we refer to [18].

6.2 Architecture

$\text{PNT}^{\text{OOL}}_{\varepsilon}$ is implemented as a bunch of plugins on top of SNAKES and thus as a Python library. Actually SNAKES implements so-called coloured Petri nets [19] where Python objects and expressions are used for the annotations. However $\text{PNT}^{\text{OOL}}_{\varepsilon}$ does not use most of these features.

A plugin for SNAKES is a separate Python library which specialises already defined classes or adds new classes to SNAKES. Plugins can be loaded and are stacked onto each other. This way a class hierarchy is established. A function `extend` has to be implemented and some rules have to be followed for which the interested reader should refer to the SNAKES homepage. A plugin can depend on other plugins and can even mention conflicting plugins.

In the following we briefly describe each of the plugins that comprise $\text{PNT}^{\text{OOL}}_{\varepsilon}$. The first and fourth plugin can be used independently from the other plugins while the remaining three build upon each other.

The `d2t`-Plugin. This plugin extends the features of the `gv`-plugin which is delivered with SNAKES. By the use of that plugin a representation of Petri nets in the `dot`-language from Graphviz [12] can be produced which is then processed by Graphviz to compute a layout and eventually produce a graphical output. Our `d2t`-plugin adds several features to the graphical output routine, like the possibilties to use subscripts in object names, to rename objects for the graphical output, and to use the export format Ti*k*Z.

The `pt`-Plugin. As already said, SNAKES implements coloured Petri nets. Since the underlying net of a PNT is actually a place/transition Petri net we decided to write a plugin which restricts the nets of SNAKES by only allowing those constructs which are needed for them.

The `terminal`-Plugin. By using the `pt`-plugin and adding a few features to the class `PetriNet`, this plugin implements Petri nets that have a single source place and single sink place. Although SNAKES delivers the `label`-plugin to add properties to any node of a net we decided to implement our own mechanism because we only need a fraction of its functionality.

With these properties it is possible to define several composition operations. While SNAKES delivers the `ops`-plugin which implements composition operations according to [4] we implemented our own mechanism because the definitions of the operations defer.

Additionally, we provide a notation to create a *singleton* net consisting of a source and a sink place and a transition in between. This feature is implemented as the class method `N` of the class `PetriNet`.

The `bisemiring`-Plugin. We implemented the class `Bisemiring` and the association of weights to transitions in a separate plugin. `Bisemiring` objects hold definitions of the set of weights, neutral elements and functions for binary addition, sequential multiplication and parallel multiplication of a bisemiring.

A PNT contains an object of class `Bisemiring` which is the Viterbi-bisemiring as default. Every transition has a weight which can be checked against the bisemiring of the net.

The `transducer`-Plugin. The last plugin builds on top of the `terminal`- and `bisemiring`-plugin and equips transitions with input- and output-symbols which can be arbitrary Python objects. An additional class implements the ε-symbol. The `N`-method is extended to support weights and input- and output-symbols for the single transition. This plugin implements the additional composition operation of language composition. Also the graphical output of transitions is changed using the functionality of `d2t`-plugin. The generated Ti*k*Z-code uses definitions from a separate `sty`-file to provide adaptable graphics.

7 Related Work

There are several less general models using weigths with underlying algebraic structure.

Weighted finite automata are classical non-deterministic finite automata in which transitions carry weights [8]. These weights may represent cost, time consumption or probability of a transition execution. The behaviour of such automata is defined by a function associating with each word the weight of its execution. For a uniform definition of the behaviour, the set of weights is

equipped with the underlying algebraic structure of a semiring. The multiplication operation of the semiring is used for determining the weight of a path, and the weight of a word is obtained by the sum of the weights of its underlying paths. If each transition additionally is equipped with an output symbol, the resulting automaton is called a transducer. Such transducers are used for the translation between languages over different alphabets for example in natural language processing. For weighted finite automata and transducers (also called finite state transducers or FSTs) there are efficient implementations of composition and optimisation operations in standard libraries [26,41].

There are generalisations of weighted finite automata to weighted automata over discrete structures other than finite words, some of them introducing concurrency into the model through considering LPOs not consisting of a total order on their symbols but of a partial order.

In [11] an overview is given on weighted finite automata (and transducers) processing tree structures. They are used to recognise weighted context-free languages with weights coming from semirings and do not consider concurrency.

In [10] weighted asynchronous cellular automata accepting weighted traces, a special restricted kind of LPOs, are described. Here also only semirings are used to describe weights, i.e. no difference is made between the combination of weights of transitions occurring in sequential order and occurring in parallel.

In [20] weighted branching automata accepting weighted sp-LPOs are introduced. Here, weights come from bisemirings where the algebraic structure of semirings is extended by a third operation of parallel multiplication (which in this case needs no unit) used for the combination of weights of concurrent transition occurrences.

For all these automata models there are widely developed theories concerning equivalent representations as rational expressions or logic formulae, useful composition operations and closure properties [8].

Another extended automata model are Q-Automata [6] whose computations are step sequences. Q-Automata are coined for application in quality management with weights modelling costs and coming from a bisemiring, whose parallel multiplication may not be commutative.

PNTs, as introduced in this paper, are a natural generalisation of all these automata based weighted transducer models working on finite words, traces or sp-LPOs. If a semiring can be extended to a concurrent semiring, then each FST over this semiring is output equivalent to a PNT [25]. However, since not each semiring can be extended to a concurrent semiring, not each FST can be represented by an equivalent PNT. In [25] we examine several conditions of semirings, which allow an extension to concurrent semirings.

There are already several publications introducing PNTs and applying them in different application areas [34,37,38]. However, these are mainly case studies lacking a common basic formal definition and without any theoretical development. Moreover, these publications only make use of sequential semantics of PNTs and do not consider weights.

Another Petri net model with transitions having assigned weights are stochastic Petri nets (SPNs). SPNs introduce a temporal specification of probabilistic nature and are applied to the performance analysis of timed systems. The weights have no underlying algebraic structure and are used to compute firing probabilities of untimed transitions.

8 Further Work

Up to now, we have developed a basic theoretical framework of PNTs and basic tool support providing the computation of weights of PNT-runs, the computation of PNT-languages and several composition operations on PNTs. Moreover, we applied PNTs in some case studies in the field of semantic dialogue modelling.

In order to apply PNTs to practical relevant problems, there are important further research steps in several directions. First, the presented theoretical framework needs to be completed:

- There are several additional composition operations of FSTs (for example inversion or reversal) which need to be examined also w.r.t. PNTs.
- In order to get more efficient algorithms, optimisation techniques must be developed as in the case of FSTs (for example elimination of ε-transitions or pushing and merging of weights alongs paths).
- For analysis purpose, we need to examine which classical Petri net properties (as for example boundedness) are consistent with composition operations.

The presented simulation algorithms need to be improved as described in section 5. Moreover, for practical application in the field of semantic dialogue modelling and speech recognition the algorithms developed so far need to be improved and extended:

- We need on-the-fly simulation algorithms computing the N best runs of a PNT (similar to N-best-paths algorithms for FSTs in the field of natural language processing).
- It is necessary to develop semi-automatic procedures to construct PNTs for the translation between the syntactic and semantic level from experimental audio data (generated in Wizard-of-Oz experiments), for example using Petri net synthesis methods [21].
- Possibily, algorithms can be fine tuned concerning the concurrent Viterbi semiring (used in this application field).

As described, we use $\text{PNT}_{\varepsilon}^{\text{OOL}}$ for the quick construction of PNTs for the use in case studies and as a scientific prototype for the development of an open library openPNT of efficient algorithms for the construction, composition, simulation and optimisation of PNTs which can be used in real world examples (similar to the open library openFST [1] for FST-algorithms).

Finally, the framework can be extended w.r.t. several aspects:

- It is possible to consider other Petri net classes, either with the aim to increase the expressiveness of the model (for example using inhibitor nets), or with the aim to restrict expressiveness in order to get more effective algorithms and improved analysis and compositionality properties (for exapmle using free choice nets).
- On the other side, more usefull examples of concurrent semirings can be collected and described. Moreover, the algebraic structure of concurrent semirings used as the weight model can be generalized in order to broaden the field of possible applications.
- Concepts which are more general than cleanness can be considered for ensuring compositionality (for example adapting generlized versions of soundness).

References

1. Allauzen, C., Riley, M.D., Schalkwyk, J., Skut, W., Mohri, M.: OpenFst: a general and efficient weighted finite-state transducer library. In: Holub, J., Žd'árek, J. (eds.) CIAA 2007. LNCS, vol. 4783, pp. 11–23. Springer, Heidelberg (2007)
2. Aéma, P., Balbo, G. (eds.): Application and Theory of Petri Nets 1997. Lecture Notes in Computer Science, vol. 1248. Springer, Heidelberg (1997)
3. Bergenthum, R., Mauser, S., Lorenz, R., Juhás, G.: Unfolding Semantics of Petri Nets Based on Token Flows. Fundam. Inform. **94**, 331–360 (2009)
4. Best, E., Devillers, R.R., Hall, J.G.: The Box Calculus: a New Causal Algebra with Multi-label Communication. In: Rozenberg [30], pp. 21–69
5. Boudol, G., Castellani, I.: On the semantics of concurrency: partial orders and transition systems. In: Ehrig, H., Kowalski, R.A., Levi, G., Montanari, U. (eds.) TAPSOFT '87. LNCS, vol. 249. Springer, Heidelberg (1987)
6. Chothia, T., Klejin, J.: Q-Automata: Modelling the Resource Usage of Concurrent Components. Electronic Notes in Theoretical Computer Science **175**(175), 153–167 (2007)
7. Droste, M., Kuich, W.: Semirings and Formal Power Series. In: Droste et al. [8], ch. 1, pp. 3–28 (2009)
8. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. Monographs in Theoretical Computer Science. Springer (2009)
9. Esposito, A., Esposito, A.M., Vinciarelli, A., Hoffmann, R., Müller, V.C. (eds.): Cognitive Behavioural Systems. LNCS, vol. 7403. Springer, Heidelberg (2012)
10. Fichtner, I., Kuske, D., Meinecke, I.: Traces, series-parallel posets, and Ppictures: a weighted study. In: Droste et al. [8], ch. 10, pp. 405–452 (2009)
11. Fülöp, Z., Vogler, H.: Weighted tree automata and tree transducers. In: Droste et al. [8], ch. 9, pp. 313–404 (2009)
12. Gansner, E.R., North, S.C.: An open graph visualization system and its applications to software engineering. SOFTWARE - PRACTICE AND EXPERIENCE **30**(11), 1203–1233 (2000)
13. Gischer, J.L.: The Equational Theory of Pomsets. Theoretical Computer Science **61**, 199–224 (1988)
14. Grabowski, J.: On Partial Languages. Fundamenta Informaticae **4**(2), 428–498 (1981)
15. Hack, M.: Petri net languages. Technical Report Memo 124, computation structures group, massachusetts institute of technology (1975)

16. Hoare, T., Möller, B., Struth, G., Wehrman, I.: Concurrent Kleene Algebra and its Foundations. The Journal of Logic and Algebraic Programming **80**, 266–296 (2011)
17. Huber, M.: PNTool-Homepage (2014). www.informatik.uni-augsburg.de/lehrstuehle/inf/mitarbeiter/huber/software/
18. Huber, M., Lorenz, R.: Constructing Petri Net Transducers with PNTool. In: Moldt, D., Rölke, H. (eds) Proceedings of the International Workshop on Petri Nets and Software Engineering, Co-located with 35th International Conference on Application and Theory of Petri Nets and Concurrency (PetriNets 2014) and 14th International Conference on Application of Concurrency to System Design (ACSD 2014), Tunis, Tunisia, June 23–24, 2014, vol. 1160 of CEUR Workshop Proceedings, pp. 339–341. CEUR-WS.org (2014)
19. Jensen, K.: Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, vol. 1 of EATCS Monographs in Theoretical Computer Science. Springer (1992)
20. Kuske, D., Meinecke, I.: Branching Automata with Costs - A Way of Reflecting Parallelism in Costs. Theoretical Computer Science **328**, 53–75 (2004)
21. Lorenz, R., Desel, J., Juhás, G.: Models from Scenarios. T Petri Nets and Other Models of Concurrency **7**, 314–371 (2013)
22. Lorenz, R., Huber, M.: Petri Net Transducers in Semantic Dialogue Modelling. In: Proceedings of "Elektronische Sprachsignalverarbeitung (ESSV)", vol. 64 of Studientexte zur Sprachkommunikation, pp. 286–297 (2012)
23. Lorenz, R., Huber, M.: Realizing the Translation of utterances into meanings by petri net transducers. In: Proceedings of "Elektronische Sprachsignalverarbeitung (ESSV)", vol. 65 of Studientexte zur Sprachkommunikation (2013)
24. Lorenz, R., Huber, M., Straßner, D.: Constructing petri net transducers with PNTool. In: Proceedings of "Elektronische Sprachsignalverarbeitung (ESSV)", vol. 71 of Studientexte zur Sprachkommunikation (2014)
25. Lorenz, R., Huber, M., Wirsching, G.: On weighted petri net transducers. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 233–252. Springer, Heidelberg (2014)
26. Mohri, M.: Weighted Automata Algorithms. In: Droste et al. [8], ch. 6, pp. 213–254 (2009)
27. Pommereau, F.: Quickly Prototyping Petri Nets Tools with SNAKES. In: SimuTools, p. 17. ICST (2008)
28. Pommereau, F.: The SNAKES toolkit (2013). https://www.ibisc.univ-evry.fr/fpommereau/SNAKES
29. Pratt, V.: Modelling Concurrency with Partial Orders. Int. Journal of Parallel Programming **15**, 33–71 (1986)
30. Rozenberg, G. (ed.): APN 1992. LNCS, vol. 609. Springer, Heidelberg (1992)
31. Schlosser, C.: Entwurf und Implementierung von Algorithmen zur gewichteten Übersetzung partieller Wörter mit Petrinetz-Transduktoren. Master thesis, Augsburg University (2014)
32. Strassner, D.: Prototypische Implementierung von Petrinetz-Transduktoren mit SNAKES. Bachelor thesis, Augsburg University (2013)
33. P. team. PNML.org: The Petri Net Markup Language home, p. 8 (2011). http://www.pnml.org
34. van Biljon, W.R.: Extending Petri nets for specifying man-machine dialogues. Int. J. Man-Mach. Stud. **28**(4), 437–455 (1988)
35. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Aéma and Balbo [2], pp. 407–426

36. van Rossum, G., Drake, F.L.: The Python Language Reference Manual. Python-Labs, Virginia (2001)
37. Wang, F.-Y., Mittmann, M., Saridis, G.N.: Coordination specification for CIRSSE robotic platform system using Petri net transducers. Journal of Intelligent and Robotic Systems **9**, 209–233 (1994)
38. Wang, F.-Y., Saridis, G.N.: A model for coordination of intelligent machines using Petri nets. In: Proceedings of the IEEE International Symposium on Intelligent Control, pp. 28–33. IEEE Comput. Soc. Press (1989)
39. Wirsching, G., Huber, M., Kölbl, C.: Zur logik von bestenlisten in der dialogmodellierung. In: Proceedings of "Elektronische Sprachsignalverarbeitung (ESSV)", vol. 61 of Studientexte zur Sprachkommunikation, pp. 309–316 (2011)
40. Wirsching, G., Huber, M., Kölbl, C., Lorenz, R., Römer, R.: Semantic dialogue modeling. In: Esposito et al. [9], pp. 104–113
41. Wolff, M.: Akustische Mustererkennung. Habilitation (2009)