LITTLE DROP OF MULLIGATAWNY SOUP, MISS SOPHIE? AUTOMATIC SPEECH UNDERSTANDING PROVIDED BY PETRI NETS

Markus Huber[1,2], *Ronald Römer*[1], *Matthias Wolff*[1]
[1]*BTU Cottbus-Senftenberg,* [2]*InnoTec21 GmbH Leipzig*
markus.huber@b-tu.de

Abstract: This paper contributes to seamless automata-based integration of speech recognition and natural language understanding. We describe a new transducer between sequential utterances and partially-ordered semantic structures which can be composed with finite state machines. After a brief discussion of the limitations of previous approaches and some mathematical preliminaries, we will recapitulate the application of Petri net transducers (PNTs) to the bidirectional translation between sequences and partial orders. Then we will show how recursive hierarchical PNTs can be used to create partially-ordered semantic structures of arbitrary width and how to build a seamless speech signal-to-semantics recognition network.

1 Introduction

Speech based cognitive user interfaces require a bidirectional translation between speech signals and representations of speech meaning. While low-level speech representations (phone strings, words, texts) are sequential, speech semantics is, in general, non-sequential. Therefore we use labelled partial orders as semantic carriers. In [1] we proposed an algorithm which creates such structures from specially tailored speech recogniser output strings. The drawback of this method is that we must prematurely *decide* for an output string (or a set of output strings) and thus cannot benefit from semantic prior knowledge. We showed that Petri net transducers (PNTs) can overcome this problem as the they can (a) bidirectionally translate between sequential utterances and partially-ordered semantic structures [2], (b) be composed with finite state machines [3], and (c) also process weights [4, 5].

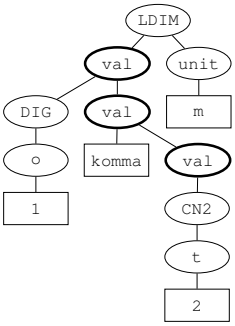


Figure 1 – Semantic structure "1.2 meters" (cf. figure 2 of [1]).

Translating totally-ordered sequences to partially-ordered semantic carriers involves a non-trivial reduction of structural constraints, namely removing the order between children of ordered parents. In figure 1 the nodes `1`, `komma` and `2` are not ordered while the nodes `val` preserve the order of the corresponding parts inside the utterance. In [3, 2] we proposed an approach based on the separation of input and output words. However, this only works with acyclic language models which are not even sufficient for simple applications in very limited domains (cf. [1]). If using a cyclic language model, the syntax-semantics transducer must be able to create partially-ordered semantic structures of potentially unlimited width. We will show that recursive hierarchical PNTs have this crucial capability.

2 Mathematical Preliminaries

The set of all *multisets* over a set X is the set $\mathbb{N}_0{}^X$ of all functions $f : X \to \mathbb{N}_0$. 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 satisfying $(m(x) = 1 \iff x \in A) \wedge (m(x) = 0 \iff x \notin A)$. A multiset m satisfying $m(a) > 0$ for exactly one element a we denote by $m(a)a$.

Given a binary relation $R \subseteq X \times Y$ and sets $A \subseteq X$ and $B \subseteq Y$, we denote the *image of A* by $R(A) = \{y \in Y \mid \exists x \in A : (x,y) \in R\}$ and the *preimage of B* by $R^{-1}(B) = \{x \in X \mid \exists b \in B : (x,b) \in R\}$. We denote by $\mathrm{Dom}(R) = R^{-1}(Y)$ the *domain of R* and call $R(X)$ the *image of R*. For $X' \subseteq X$ and $Y' \subseteq Y$ the restriction of R onto $X' \times Y'$ is denoted by $R|_{X' \times Y'}$.

Let $\mathscr{A}$ be a finite set of symbols called an *alphabet*. A *step over* $\mathscr{A}$ is a multiset over $\mathscr{A}$. A *step sequence* over $\mathscr{A}$ is an element of $(\mathbb{N}_0{}^{\mathscr{A}})^*$ where ε denotes the *empty step sequence*.

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 $m(p) \in \mathbb{N}_0$ of tokens, i.e. a marking is a multiset over P representing a distributed state. A *marked net* is a net $N = (P,T,F)$ together with an *initial marking* m_0.

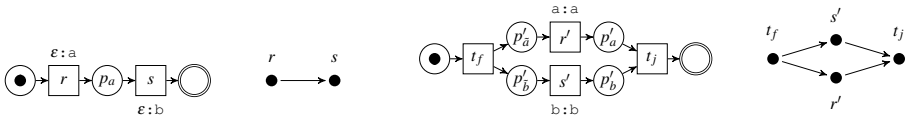
A *place/transition Petri net* (*PT-net*) is a 4-tuple $N = (P,T,F,W)$, where (P,T,F) is a net and $W : (P \times T) \cup (T \times P) \to \mathbb{N}_0$ is a *flow weight function* satisfying $W(x,y) > 0 \Leftrightarrow (x,y) \in F$. For (transition) steps τ over T we introduce the two multisets of places ${}^\bullet\tau(p) = \sum_{t \in T} \tau(t)W(p,t)$ 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 τ occurs in m, 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 its occurrence leads to m'. A *step execution in m* is a finite step sequence over T $\tau_1 \ldots \tau_n$ such that there are markings $m_1, \ldots, m_n$ with $m \xrightarrow{\tau_1} m_1 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_n} m_n$. The markings which can be reached from the initial marking m_0 via step executions are called *reachable*.

A *directed graph* is a pair $G = (V, \to)$, where V is a finite *set of nodes* and $\to \subseteq V \times V$ is a binary relation over V, called the *set of edges*. For a node $v \in V$ its *preset* is the set ${}^\bullet v = {\to}^{-1}(\{v\})$ and its *postset* the set $v^\bullet = {\to}(\{v\})$. A *path* is a sequence of (not necessarily distinct) nodes $v_1 \ldots v_n$ $(n > 1)$ such that $v_i \to v_{i+1}$ 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 $(V, \to')$ is an *extension* of an acyclic directed graph $(V, \to)$ if $\to \subseteq \to'$.

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 $(\forall u,v,w \in V : (u < v \wedge v < w) \Rightarrow u < w)$. We identify a finite partial order $<$ over V with the directed graph $(V,<)$. Given a partial order $po = (V,<)$ we call two nodes $v, v' \in V$ *independent* if $v \not< v'$ and $v' \not< v$. By $\mathrm{co}_< \subseteq V \times V$ we denote the set of all pairs of independent nodes of V. A set $A \subseteq V$ of pairwise independent nodes is called an *antichain*. The *width* of po is defined by $\mathrm{wd}(po) = \sup\{|A| \mid A \text{ is antichain in } po\}$.

A *labelled partial order* (*LPO*) *over a set X* is a 3-tuple $(V,<,l)$, where $(V,<)$ is a partial order and $l : V \to X$ is a labelling function on V. In most cases, we only consider LPOs up to isomorphism, i.e. only the labelling of nodes is of interest, but not the node names. 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 $v < w \Leftrightarrow I(v) <' I(w)$. In figures, we normally do not show the names of the nodes of an LPO, but only their labels and we often omit transitive arrows for a clearer presentation. A *step-wise linear LPO* is an LPO $(V,<,l)$ where the relation $\mathrm{co}_<$ is transitive. The maximal sets of independent nodes 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 *lpo*$'$ which is an extension of *lpo*.

We use LPOs over T to represent single non-sequential runs of PT-nets, i.e. the nodes of an LPO represent transition occurrences. For a marked PT-net $N = (P,T,F,W,m_0)$ an LPO $lpo = (V,<,l)$ over T is an *LPO-run* 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 $m' = m - \sum_{v \in V} {}^\bullet l(v) + \sum_{v \in V} l(v)^\bullet$. We denote the occurrence of an LPO-run *lpo* by $m \xrightarrow{lpo} m'$.



(a) – Generator N_{seq} with sequential run.

(b) – Transducer N_{par} with non-sequential run.

Figure 2 – Example Petri net transducers.

3 Petri Net Transducers

According to [4] Petri Net Transducers are PT-nets where each transition is augmented with an input and an output label. As with weighted finite-state transducers each transition additionally carries a weight drawn from a bisemiring. For this article the weights are due to limited space not of interest and we omit them from the definition.

A Petri Net Transducer (PNTs) is defined as a tuple $N = (P,T,F,W,p_I,p_F,\Sigma,\sigma,\Delta,\delta)$, where

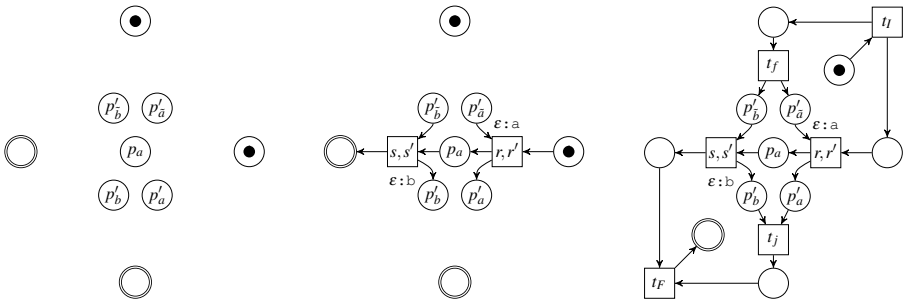
- (P,T,F,W,m_0) with $m_0 = p_I$ is a marked PT-net (called the *underlying PT-net*), $p_I \in P$ is the *source place* satisfying ${}^\bullet p_I = \emptyset$ and $p_F \in P$ is the *sink place* satisfying $p_F{}^\bullet = \emptyset$,
- Σ is a set of *input symbols* and $\sigma : T \to \Sigma \cup \{\varepsilon\}$ is the *input mapping*,
- Δ is a set of *output symbols* and $\delta : T \to \Delta \cup \{\varepsilon\}$ is the *output mapping*.

The marking p_I is called the *initial marking* and the marking p_F is called the *final marking*. A PNT is called *clean* if the final marking where only the place p_F is marked by one token is the only reachable marking m with $m(p_F) > 0$, i.e. the only reachable marking which marks p_F. Cleanness ensures that PNT semantics are closed under (sequential) product and closure.[1]

Figure 2 shows on the left the generator N_{seq} producing the sequence `ab` resulting from the projection of its run through δ. On the right is the transducer N_{par} with its so-called *concurrent* transitions r' and s'. Transitions with no symbols are so-called *ε-transitions* having ε as input and output. After projecting through σ resp. δ ε-labelled nodes get deleted together with their adjacent edges. Hence N_{par} reads and writes the partial order $(\{\mathtt{a},\mathtt{b}\},\emptyset)$.

Since N_{par} can read its input in any order N_{seq} – where the output is ordered – can be composed with it. In figure 3 the composition $N_{seq} \circ N_{par}$ is shown. The transducer N_{par} is

[1] That is not completely right since transitions without input places are still enabled when the final marking has been reached. However for the transducer to be clean such transitions cannot have output places as well. Therefore we forbid such transitions which is no real restriction for any practical example.

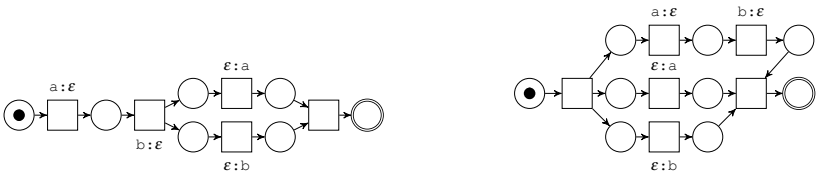


(a) – Union of places.

(b) – Merging of transitions.

(c) – Adding of ε-transitions.

Figure 3 – Composition of N_{seq} and N_{par}.



(a) – Sequential separation of input and output. **(b)** – Parallel separation of input and output.

Figure 4 – PNTs translating from the sequence ab to the partial order $(\{a,b\},\emptyset)$.

depicted top-down and the generator N_{seq} in the middle from right to left. As a first step the places of both PNTs are united in figure 3a. In the next step transitions with matching symbols get merged. As one can see in figure 3b the transitions r and r' as well as s and s' have been merged and the new transitions inherit its arcs from both PNTs, input symbols are taken from N_{seq} and output symbols from N_{par}. As a last step transitions with ε-output from N_{seq} and ε-input from N_{par} are added with their arcs and the two PNTs are glued together with new source and sink places as well as an initial transition t_I and a final transition t_F.

Note that the independence of r' and s' from N_{par} is not preserved during the composition. The ordering of r and s from N_{seq} – enforced by the place p_a and its arcs – wins and the resulting generator produces the sequence ab. In general composition can add order but cannot remove it. For a semantics-syntax transducer this is sufficient (cf. [6]).

4 Translation from Sequences to Partial Orders

Since the aim of this paper is to translate from sequences to partial orders the result from the last section is not satisfying. Clearly the problem arises from the fact that input and output in N_{par} are processed by the same transitions so dependency from the input by the means of N_{seq} is propagated to the output of $N_{seq} \circ N_{par}$.

The natural solution to this is separating the processing of the input from the processing of the output within N_{par} as shown in the following subsection. In subsection 4.2 we introduce an alternative solution where transitions can be isolated from one another.

4.1 Separation of Input and Output

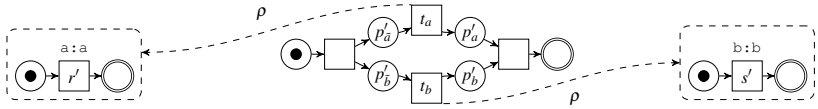
One possible way to achieve the separation is to have a PNT processing the input sequentially followed by a PNT processing the output like in figure 4a. This idea was shown in detail in [2]. When the run of the PNT is projected through σ then its second part is deleted and the sequence ab remains. Projecting the run through δ deletes the first part and the partial order $(\{a,b\},\emptyset)$ remains. Composition with N_{seq} does not alter the output transitions leaving them concurrent.

Another way is to separate input and output in parallel as shown in figure 4b. Here the input is processed in the upper part and the output in the lower part of the transducer. Projections of its run lead to the same results as in the case of sequential separation.

4.2 Hierarchical Petri Net Transdcuers

Instead of separating input and output one can also try to separate transitions from one another. This can be done with hierarchical PNTs which introduce a refinement operation where a transition is replaced by a PNT. Restriction of the runs then deletes edges which cross the borders between replacement PNTs resulting in the preservation of independence.

Based on the notations from [7] one can define for a given PNT $N_0 = (P_0, T_0, F_0, W_0, p_I, p_F, \Sigma_0, \sigma_0, \Delta_0, \delta_0)$ a *hierarchical PNT (hPNT)* as a 5-tuple $H = (N_0, \mathcal{N}, \rho, F_c, W_c)$ where



(a) – Refinement PNT N_1. **(b)** – Initial PNT N_0. **(c)** – Refinement PNT N_2.

Figure 5 – The hierarchical PNT H_{par} where the transitions t_a and t_b get refined by N_1 resp. N_2.

- N_0 is the *initial PNT*,
- $\mathcal{N} = \{N_1, \ldots, N_k\}$ is a family of *refinement PNTs* where $P_0, \ldots, P_k$ resp. $T_0, \ldots, T_k$ are pairwise disjoint,
- $\rho : T_0 \to \{1, \ldots, k\}$ is a partial *refinement function* which associates transitions from the initial PNT with PNTs from $\mathcal{N}$. A transition $t \in T_0$, if $t \in \mathrm{Dom}(\rho)$ holds, is refined by the PNT $N_{\rho(t)}$. Any transition $t \in T_0$ for which $t \notin \mathrm{Dom}(\rho)$ holds is called *simple*.
- $F_c \subseteq (P_0 \times \bigcup_{i=1}^{k} T_i) \cup (\bigcup_{i=1}^{k} T_i \times P_0)$ is the *crossing flow relation* which allows to connect transitions from refinement PNTs to places from the initial PNT and
- W_c is the corresponding *crossing flow weight function* as introduced for PT-nets in section 2.

In figure 5 the example hPNT H_{par} is shown with its initial PNT in figure 5b and its refinement PNTs for the concurrent transitions t_a resp. t_b in figures 5a and 5c. We demonstrate the refinement operation on transition t_a. We substitute t_a with two new ε-transitions i_{t_a} (put into the set ${}^\bullet T$) and o_{t_a} (put into the set $T^\bullet$) where i_{t_a} only inherits the incoming arcs of t_a and o_{t_a} only its outgoing arcs. Then we connect i_{t_a} with place p_{I_1} since $\rho(t_a) = 1$ and place p_{F_1} with transition o_{t_a}. After also refining t_b we have a PNT like N_{par} from figure 2b with four additional ε-transitions which do not alter input or output. The formal procedure of refining is given in the following box.

For a given hPNT $H = (N_0, \mathcal{N}, \rho, F_c, W_c)$ let K be the image of ρ and set $K_0 = K \cup \{0\}$. Let T_s be the set of simple transitions of N_0, F_s the flow relation of N_0 restricted to all its places and simple transitions and W_s the flow weight of N_0 restricted to all its places and simple transitions. H defines a PNT $N = (P, T, F, W, p_I, p_F, \Sigma, \sigma, \Delta, \delta)$ with

- $P = \bigcup_{i \in K_0} P_i$,
- $T = T_s \cup \bigcup_{i \in K} T_i \cup {}^\bullet T \cup T^\bullet$ where
  - ${}^\bullet T$ contains a transition i_t for every transition $t \in \mathrm{Dom}(\rho)$ and
  - $T^\bullet$ contains a transition o_t for every transition $t \in \mathrm{Dom}(\rho)$,
- $F = F_s \cup F_c \cup \bigcup_{i \in K} F_i \cup {}^\bullet F \cup F^\bullet$ with
  - ${}^\bullet F = \{(p, i_t) \in P_0 \times {}^\bullet T \mid (p,t) \in F_0\} \cup \{(i_t, p_{I_{\rho(t)}}) \mid i_t \in {}^\bullet T\}$ and
  - $F^\bullet = \{(o_t, p) \in T^\bullet \times P_0 \mid (t,p) \in F_0\} \cup \{(p_{F_{\rho(t)}}, o_t) \mid o_t \in T^\bullet\}$,
- $W = W_s + W_c + \sum_{i \in K} W_i + {}^\bullet W + W^\bullet$ with
  - ${}^\bullet W = \sum_{(p,i_t) \in P_0 \times {}^\bullet T} W_0(p,t)(p,i_t) + \sum_{i_t \in {}^\bullet T} (i_t, p_{I_{\rho(t)}})$ and
  - $W^\bullet = \sum_{(o_t,p) \in T^\bullet \times P_0} W_0(t,p)(o_t,p) + \sum_{o_t \in T^\bullet} (p_{F_{\rho(t)}}, o_t)$,
- $p_I = p_{I_0}$ and $p_F = p_{F_0}$,

- $\Sigma = \bigcup_{i \in K_0} \Sigma_i$,
- $\sigma|_{T_s} \equiv \sigma_0$, $\forall i \in K : \sigma|_{T_i} \equiv \sigma_i$, $\sigma|_{\bullet T} \equiv \varepsilon$, $\sigma|_{T^\bullet} \equiv \varepsilon$,
- $\Delta = \bigcup_{i \in K_0} \Delta_i$,
- $\delta|_{T_s} \equiv \delta_0$, $\forall i \in K : \delta|_{T_i} \equiv \delta_i$, $\delta|_{\bullet T} \equiv \varepsilon$, $\delta|_{T^\bullet} \equiv \varepsilon$.

The resulting PNT N is called the *interface of* H and the set of runs of N is per definition the set of runs of H. Since an hPNT defines a PNT any hPNT can be used as a refinement PNT justifying the notation of *hierarchical* PNTs. Also an hPNT is called *clean* if its interface is clean which is obviously the case when the initial PNT and all used refinement PNTs are clean.

Composition of a PNT with an hPNT is always done against its interface. There is only a subtle change in the procedure since the result is an hPNT again. We will show the example $N_{seq} \circ H_{par} = H'$. At first the places P_{seq} and P_0 are united resulting in P_0'. Then matching transitions get merged and put accordingly into the sets T_0', T_1' or T_2'. Since all places from N_{seq} are now in P_0' the inherited arcs from F_{seq} and W_{seq} go into F_c' and W_c'. Then transitions with ε-output from T_{seq} are added to T_0', transitions with ε-input from T_0, T_1 and T_2 are added to H' accordingly and the two PNTs are glued together as described in section 3. At last F_c and W_c are adjusted and taken over to H' as well as ρ. Composition of two hPNTs is subject to further research since it is not clear what the desired result could be.

Note that again the independence of r' and s' from H_{par} is not preserved within the result of the composition. Considering input and output of the interface nothing changed.

However, with hPNTs there are other possibilities for the definition of input and output. Given an hPNT H, its interface N and a run $lpo = (V, <, l)$ of N we can exclude dependencies by simply restricting $<$ which is defined on $V \times V$ to only respect dependencies of interest. So let K be the image of ρ and set $K_0 = K \cup \{0\}$. Then for any $i \in K_0$ the restriction of $<$ onto $l^{-1}(T_i) \times l^{-1}(T_i)$ eliminates all edges between nodes that are not both labelled by a transition from T_i. To preserve the edges added by the refinement we have to include the sets ${}^\bullet T$ and $T^\bullet$. So if we restrict $<$ onto $\bigcup_{i \in K_0} (T_i \cup {}^\bullet T \cup T^\bullet) \times (T_i \cup {}^\bullet T \cup T^\bullet)$ we isolate the refinement PNTs from each other and besides the connections resulting from the refinement also from N_0.

For our example of $N_{seq} \circ H_{par}$ the restriction of its run eliminates the edge between the nodes labelled by r, r' resp. s, s' which was introduced by the composition operation. The projection through δ then yields the partial order $(\{\mathtt{a}, \mathtt{b}\}, \emptyset)$.

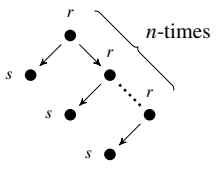


Figure 6 – LPO lpo_a with width n.

Now consider lpo_a from figure 6. The set of all nodes labelled by s is an antichain of cardinality n. Two nodes labelled by r are always ordered. Any set of more than n nodes must contain at least one node labelled by r but there are at most $n-1$ nodes which are independent of such a node. Clearly lpo_a has width n. Note that the set of all nodes labelled by r is a sequence of length n.

For PNTs there exists the well-known closure operation which creates some cycle inside the PNT. This way a single transition is sufficient for producing arbitrarily long sequences. On the other hand a PNT producing an antichain for every $n \in \mathbb{N}$ does not exist since it would need to have arbitrarily many concurrent transitions. So there can be no PNT producing LPOs like in figure 6 for every $n \in \mathbb{N}$. An hPNT would need to have arbitrarily many refinement PNTs to eliminate arbitrarily many unwanted edges. So there can be none either.

5 Recursive Hierarchical Petri Net Transducers

While a PNT as a finite object can produce arbitrarily long sequences the production of arbitrarily wide LPOs requires a potentially infinite object. Extending refinement to a recursive operation like for example the Petri box calculus [8] did combined with a termination condition allows us to add concurrent transitions on demand while the resulting transducer stays a finite object.

A *recursive hPNT (rhPNT)* is a 7-tuple $R = (N_0, \mathcal{N}, \rho, F_c, W_c, T_t, T_i)$ where

- $(N_0, \mathcal{N}, \rho|_{T'}, F_c, W_c)$ is an hPNT (called the *underlying hPNT*) where $T' = \{t \in T_0 \mid t \notin \mathrm{Dom}(\rho) \vee \rho(t) \neq 0\}$ is the set of those transitions which are simple or do not get refined by the initial transducer (called *recursive transitions*),
- ρ is therefore a partial function on T_0 where now also the value 0 is allowed,
- $T_t \subseteq T$ is a set of *trigger transitions* where $T = \bigcup_{i \in \{0\} \cup \rho(T_0)} T_i$ and
- $T_i \subseteq T$ is a set of *ignore transitions*.

In figure 7 the example rhPNT R_ω is shown where the recursive transition t_r gets refined by the initial PNT. All other transitions are simple and there are no refinement PNTs. At a first glance it seems obvious how R_ω can have LPO-runs where for arbitrary $n \in \mathbb{N}$ a sub-LPO isomorphic to lpo_a from figure 6 exists. However, the formal definition is far from obvious since the general interface would be an infinite object. Also composition with PNTs which are no state machines or even hPNTs or rhPNTs is subject to further research.

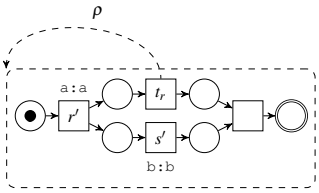


Figure 7 – Recursive hPNT R_ω.

We describe the composition ${N_{seq}}^2 \circ R_\omega$ with ${N_{seq}}^2 = N_{seq} \otimes N_{seq}$ where $\otimes$ is the concatenation of PNTs. ${N_{seq}}^2$ generates the sequence `abab`. The set of trigger transitions of R_ω is $T_t = \{r'\}$ its set of ignore transitions is $T_i = \{r', s'\}$. The initial interface of R_ω is the interface of its underlying hPNT. We start with the places as before and continue with merging of transitions. This time we must respect their order inside ${N_{seq}}^2$. We merge the first transition with output `a` with r' and the first transition with output `b` with s'. Since r' is a trigger transition we mark it. When we then try to merge the second transition with output `a` with r' the marked trigger transition tells us to do a recursive refinement first. During this refinement an isomorphic copy of R_ω where the places and transitions are renamed is put as a new refinement PNT into $\mathcal{N}$. The refinement function ρ is adjusted to associate t_r accordingly and the interface is regenerated. We resume the merging of the second transition with output `a`. Since we already did a recursive refinement and r' is an ignore transition the only matching transition is the corresponding one from the new refinement PNT which we mark since it is a trigger transition there. Since s' is also an ignore transition the second transition with output `b` can only be merged with the matching one of the new refinement PNT. After completing the composition as before we have an rhPNT where we can use the restriction technique from subsection 4.2 to get an output isomorphic to lpo_a from figure 6 with $n = 2$.

6 Outlook

We introduced recursive hierarchical Petri net transducers for the translation of totally-ordered utterances into partially-ordered semantic structures for the composition with a speech recogniser build of finite state machines. The semantic structures can be of arbitrary width when a cyclic language model is used. Due to results from [4, 5] this is also possible in a weighted setting.

We can now compute all weighted semantic structures corresponding to a speech signal within a hierarchical speech processing system without any premature decision (cf. [2]) even for semantic structures like in figure 1.

Moreover we are able to prime the recognition network by composition with a semantic structure representing an expectation on the next input. This expectation is a truly semantic one but adjusts the recogniser down to all low-level speech representations.

So the Bavarian question “A Supp’n?” (which happens to be the abbreviation of **A**utomatic **S**peech **U**nderstanding **p**rovided by **P**etri **N**ets) can yield the same recogniser adjustments as the title of this paper since their semantics are the same.

References

[1] WIRSCHING, G. and M. WOLFF: *Semantische dekodierung von sprachsignalen am beispiel einer mikrofonfeldsteuerung*. In R. HOFFMANN (ed.), *Elektronische Sprachsignalverarbeitung 2014, Tagungsband der 25. Konferenz Dresden, 26. - 28. März 2014*, pp. 104 – 109. 2014.

[2] LORENZ, R. and M. HUBER: *Realizing the Translation of Utterances into Meanings by Petri Net Transducers*. In P. WAGNER (ed.), *Proceedings of ”Elektronische Sprachsignalverarbeitung (ESSV)”*, vol. 65 of *Studientexte zur Sprachkommunikation*, pp. 103 – 110. 2013.

[3] LORENZ, R. and M. HUBER: *Petri net transducers in semantic dialogue modelling*. In M. WOLFF (ed.), *Proceedings of ”Elektronische Sprachsignalverarbeitung (ESSV)”*, vol. 64 of *Studientexte zur Sprachkommunikation*, pp. 286 – 297. 2012.

[4] LORENZ, R., M. HUBER, and G. WIRSCHING: *On weighted petri net transducers*. In G. CIARDO and E. KINDLER (eds.), *Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings*, vol. 8489 of *Lecture Notes in Computer Science*, pp. 233–252. Springer, 2014. doi:10.1007/978-3-319-07734-5_13. URL `http://dx.doi.org/10.1007/978-3-319-07734-5_13`.

[5] LORENZ, R.: *Modeling quantitative aspects of concurrent systems using weighted petri net transducers*. In R. R. DEVILLERS and A. VALMARI (eds.), *Application and Theory of Petri Nets and Concurrency - 36th International Conference, PETRI NETS 2015, Brussels, Belgium, June 21-26, 2015, Proceedings*, vol. 9115 of *Lecture Notes in Computer Science*, pp. 49–76. Springer, 2015. doi:10.1007/978-3-319-19488-2_3. URL `http://dx.doi.org/10.1007/978-3-319-19488-2_3`.

[6] HUBER, M. and R. RÖMER: *Modellierung des Semantik-Syntax grenzübergangs kognitiver Systeme am Beispiel des ”Mouse-Maze”-Problems*. In G. WIRSCHING (ed.), *Proceedings of ”Elektronische Sprachsignalverarbeitung (ESSV)”*, Studientexte zur Sprachkommunikation, pp. 232 – 239. 2015.

[7] ZUBEREK, W. M. and I. BLUEMKE: *Hierarchies of place/transition refinements in petri nets*. In *Proceedings of ”Conference on Emerging on Technologies and Factory Automation”*, pp. 355–360. B b, 1997.

[8] BEST, E., R. R. DEVILLERS, and J. G. HALL: *The box calculus: a new causal algebra with multi-label communication*. In G. ROZENBERG (ed.), *Advances in Petri Nets 1992, The DEMON Project*, pp. 21 – 69. Springer, 1992.