On the Semantic Readings of Proof-Nets

Philippe de Groote, Christian Retoré

INRIA-Lorraine & CRIN-CNRS
615, rue du Jardin Botanique - B.P. 101
54602 Villers lès Nancy Cedex – FRANCE

e-mail: Philippe.de.Groote@loria.fr, Christian.Retore@loria.fr

A la mémoire de
Xavier de Groote

Abstract. The goal of this paper is to demonstrate how the very rich notion of proof-net may be used, in the framework of categorial grammars, as a unique structure that allows the syntactic and semantic aspects of sentence analysis to be unified. We first explain how the intuitionistic multiplicative proof-nets correspond exactly to the so-called linear λ-terms. This allows us to interpret proof-nets not only as syntactic structures but also as semantic readings à la van Benthem. Then, we generalize the correspondence between proof-nets and λ-terms to the complete categorial hierarchy, and we show how Montague-like semantics may be handled in this framework.

1 Introduction

If one were to summarize in a few words the logical principles underlying categorial grammars [15, 17, 24], these could well be: Parsing as Deduction and Grammar Theory as Proof Theory. Indeed, during the last decade, proof-theoretical investigations of categorial grammars have been extremely fruitful, e.g., [21, 24].

On the syntactic side, Roorda advocates the notion of proof-net as an appropriate parsing structure [21]. Proof-nets are a new proof-theoretic tool introduced by Girard in the framework of linear logic [6]. They allow several proofs of the sequent calculus to be represented by the same structure when they do not differ in an essential way. In this sense, they correspond to unambiguous representations of proofs. Moreover, their nice mathematical theory gives rise to new parsing algorithms [13, 16, 21].

On the semantic side, van Benthem uses the Curry-Howard correspondence as an interface between syntax and semantics [23]. This correspondence, which dates back to the sixties [9] (see also [5, 8]), establishes an isomorphism between natural deduction, on the one hand, and typed λ-calculus, on the other hand:

<table>
<thead>
<tr>
<th>Natural Deduction</th>
<th>Typed λ-Calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formulas</td>
<td>Types</td>
</tr>
<tr>
<td>Proofs</td>
<td>λ-Terms</td>
</tr>
<tr>
<td>Proof Normalization (cut elimination)</td>
<td>β-Reduction</td>
</tr>
</tbody>
</table>

In the framework of categorial grammars, a third column may be added to this table:

<table>
<thead>
<tr>
<th>Natural Deduction</th>
<th>Typed λ-Calculus</th>
<th>Categorial Grammars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formulas</td>
<td>Types</td>
<td>Semantic Categories</td>
</tr>
<tr>
<td>Proofs</td>
<td>λ-Terms</td>
<td>Semantic Readings, Semantic Recipes</td>
</tr>
<tr>
<td>Proof Normalization (cut elimination)</td>
<td>β-Reduction</td>
<td>Composition of Semantic Recipes</td>
</tr>
</tbody>
</table>
Now, in the framework of linear logic, the essence of the Curry-Howard isomorphism is stated by Girard as follows:

<table>
<thead>
<tr>
<th>Formulas</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proofs</td>
<td>Proof-Nets</td>
</tr>
<tr>
<td>Proof Normalization (cut elimination)</td>
<td>Proof-Net Evaluation</td>
</tr>
</tbody>
</table>

Therefore, by a simple juxtaposition of the two tables above, one sees that proof-nets may play the semantic part that is usually played by \( \lambda \)-terms. In this paper we introduce and illustrate by several examples this new point of view.

The next section gives a brief introduction to the notion of proof-net, and explains how linear \( \lambda \)-terms may be represented as proof-nets by using a notion of polarity. Section 3 shows how to get semantic readings of categorial principles, such Montagovian type raising, directly from a proof-net. In Section 4, we explain how to adapt the notion of proof-net to the different logics of the categorial hierarchy, i.e., the Lambek calculus \( L \), the intuitionistic logic \( I \), and van Benthem’s intermediate logics \( LP \), \( LC \), and \( LPC \). Finally, Section 5 provides a short but complete example of a syntactic and semantic analysis based on proof-nets, in the spirit of Montague’s PTQ grammar [14].

While this paper tries to be as self contained as possible, we assume that the reader has some familiarity with categorial grammars [1, 17, 18], the Lambek calculus [12, 15, 24], and linear logic [6, 7, 22].

2 Correspondence between Linear \( \lambda \)-Terms and Intuitionistic Multiplicative Proof-Nets

2.1 A Brief Introduction to Multiplicative Proof-Nets

The notion of proof-net has been introduced by Girard [6] as the most suitable way of representing proofs in linear logic. With respect to sequential derivations, proof-nets have at least two advantages: firstly, they are more compact; secondly, they allow sequential proofs that differ in an inessential way to be identified.

Roughly speaking, proof-nets are obtained from sequential derivations by considering only the active formulas and by linking together the formulas that occur in the same axiom. Consider, for instance, the following derivation where the active formulas are framed:

\[
\vdash A \perp, A \quad \vdash B \perp, B \\
\vdash B, (A \otimes B \perp), A \perp \\
\vdash C \perp, C \\
\vdash B, ((A \otimes B \perp) \otimes A \perp) \\
\vdash C \perp, B, ((C \otimes ((A \otimes B \perp) \otimes A \perp)) \\
\vdash (C \perp \otimes B), (C \otimes ((A \otimes B \perp) \otimes A \perp))
\]

This derivation may be transformed into the following proof-net:

\[
\begin{align*}
A & \quad B \perp & \quad (A \otimes B \perp) & \quad A \perp \\
(C \perp \otimes B) & \quad C & \quad ((A \otimes B \perp) \otimes A \perp) & \quad (C \otimes ((A \otimes B \perp) \otimes A \perp))
\end{align*}
\]
More abstractly, the above proof-net may be identified with the following graph:

![Graph](image)

which represents the core of Derivation (1).

On the formal side, Girard defines first the notion of proof structure, which corresponds to a class of graphs akin to Graph 3. Then, a global geometrical criterion allows the proof-nets, which are the graphs that correspond actually to sequential proofs, to be discriminated from the other proof-structures (see [3, 6, 10, 20] for instances of such criteria and for more details).

### 2.2 Intuitionistic Multiplicative Proof-Nets

The examples in the previous section are taken from the so-called classical multiplicative linear logic. Now, logics such as the Lambek calculus are intuitionistic in the technical sense that they are defined by means of sequent calculi whose sequents are made of several antecedent formulas and only one succedent formula. In order to accommodate the notion of proof-net to such logics, one must use a notion of input (⊙) and output (○) polarities [2, 10]. The idea is that the input (or negative) polarities correspond to those occurrences of formulas that appear in the antecedents of the sequents while the output (or positive) polarities correspond to the occurrences of formulas that appear in the succedents.

More precisely, consider the following table that defines the notion of links for the proof-structures of the intuitionistic implicative linear logic (IILL, also known as van Bentham’s LP*), whose only connective is the linear implication \(\rightarrow\).

<table>
<thead>
<tr>
<th>Name</th>
<th>Axiom</th>
<th>Tensor</th>
<th>Par</th>
<th>Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK</td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>PREMISES</td>
<td>none</td>
<td>(p_1, p_2)</td>
<td>(p_1, p_2)</td>
<td>(p_1, p_2)</td>
</tr>
<tr>
<td>CONCLUSIONS</td>
<td>(c_1, c_2)</td>
<td>(c)</td>
<td>(c)</td>
<td>none</td>
</tr>
<tr>
<td>TYPES</td>
<td>(c_1 : \overrightarrow{A})</td>
<td>(p_1 : \overrightarrow{A})</td>
<td>(p_1 : \overrightarrow{A})</td>
<td>(p_1 : \overrightarrow{A})</td>
</tr>
<tr>
<td></td>
<td>(c_2 : \overrightarrow{A})</td>
<td>(p_2 : \overrightarrow{B})</td>
<td>(p_2 : \overrightarrow{B})</td>
<td>(p_2 : \overrightarrow{B})</td>
</tr>
<tr>
<td></td>
<td>(c : \overrightarrow{(A \rightarrow B)})</td>
<td>(c : \overrightarrow{(A \rightarrow B)})</td>
<td>(c : \overrightarrow{(A \rightarrow B)})</td>
<td>(c : \overrightarrow{(A \rightarrow B)})</td>
</tr>
<tr>
<td>POLARITIES</td>
<td>(c_1 : \text{input})</td>
<td>(p_2, c : \text{input})</td>
<td>(p_1 : \text{input})</td>
<td>(p_2, c : \text{input})</td>
</tr>
<tr>
<td></td>
<td>(c_2 : \text{output})</td>
<td>(p_1 : \text{output})</td>
<td>(p_2, c : \text{output})</td>
<td>(p_1 : \text{output})</td>
</tr>
</tbody>
</table>

Proof-structures are then defined to be graphs made of links such that:

1. any premise of any link is connected to exactly one conclusion of some other link;
2. any conclusion of any link is connected to at most one premise of some other link;
3. input (resp. output) premises are connected to input (resp. output) conclusions of the same type.
Then, as we have already pointed out, a correctness criterion allows one to distinguish the proof-nets among the proof-structures. In fact, this correctness criterion ensures that the proof-nets are those proof-structures that may be sequentialized into Gentzen-like derivations. In particular, it ensures that any proof-net has exactly one output conclusion.

The proof-net formalism also captures the dynamics of proofs: cut elimination may be performed directly on the proof-nets without any reference to the sequent calculus. Moreover, the cut elimination process is specified by simple graph rewriting rules. In the case of ILLL, these rewriting rules, which are purely local, are the following:

![Diagram](image)

2.3 Encoding of the Linear \( \lambda \)-Terms into the Intuitionistic Multiplicative Proof-Nets

The proof-nets introduced in the previous section give a way of representing the proofs of intuitionistic implicative linear logic. Another way of representing these proofs is given by the Curry-Howard isomorphism. It consists of using the so-called linear \( \lambda \)-terms. This raises immediately the following question: what is the relationship between these two different formalisms? The answer is quite simple: the linear \( \lambda \)-terms may be encoded as proof-nets and, in the case of linear \( \lambda \)-terms in normal form, the resulting correspondence is one-one.

As is well known, \( \lambda \)-terms in normal form may be defined by the following grammar:

\[
\begin{align*}
\mathcal{A} &::= x \mid (\mathcal{A} \mathcal{T}) \\
\mathcal{T} &::= \mathcal{A} \mid \lambda x. \mathcal{T}
\end{align*}
\]

By adding the constraint that each \( \lambda \) must bind exactly one variable occurrence, one defines the linear \( \lambda \)-terms that correspond, through the Curry-Howard isomorphism, to the proofs of ILL.

Now, the encoding of these terms into proof-nets obeys the following principles:

1. to any \( \lambda \)-term of type \( \mathcal{A} \) with free variables \( x_i \) of types \( \mathcal{A}_i \) corresponds some proof-net whose unique output conclusion is of type \( \mathcal{A} \) and whose input conclusions, which are of types \( \mathcal{A}_i \), may be labelled with the variables \( x_i \).
2. to any \( \lambda \)-term defined by the non-terminal \( \mathcal{A} \) corresponds some proof-net whose unique output conclusion is the output conclusion of an axion-link.

Keeping these two invariants in mind, one may define the encoding by induction on the above grammar.

**Case 1:** variable. The proof-net encoding a variable \( x \) is made of an axion-link:

![Diagram](image)
Case 2: application. Let $\Pi_1$ and $\Pi_2$ be the two proof-nets encoding respectively $\mathcal{A}$ and $\mathcal{T}$:

\[
\begin{array}{c}
\Pi_1 \\
(\mathcal{B}\rightarrow\mathcal{C})^+ \\
x_1:A_1^- x_2:A_2^- x_n:A_n^- \\
\Pi_2 \\
y_1:B_1^- y_2:B_2^- B^+ \\
\end{array}
\]

The proof-net encoding $(\mathcal{A}\mathcal{T})$ is obtained by gluing $\Pi_1$ and $\Pi_2$ with a tensor-link as follows:

\[
\begin{array}{c}
\Pi_2 \\
y_1:B_1^- y_2:B_2^- C^+ \\
\circ \\
\Pi_1 \\
x_1:A_1^- x_2:A_2^- x_n:A_n^- \\
\end{array}
\]

Case 3: abstraction. Let $\Pi$ be the proof-net encoding $\mathcal{T}$:

\[
\begin{array}{c}
\Pi \\
(\mathcal{B}\rightarrow\mathcal{C})^+ \\
x_1:A_1^- x_2:A_2^- x:B^- C^+ \\
\end{array}
\]

The proof-net encoding $\lambda x.\mathcal{T}$ is obtained by adding a $\text{par}$-link as follows:

\[
\begin{array}{c}
\Pi \\
(\mathcal{B}\rightarrow\mathcal{C})^+ \\
x_1:A_1^- x_2:A_2^- \\
\end{array}
\]

The above encoding is not merely syntactic: it relies on an actual correspondence that also takes into account the dynamics of the $\lambda$-calculus. Indeed, $\beta$-redexes may be represented by using cuts as follows.
Let $\Pi_1$ and $\Pi_2$ be the proof-nets encoding respectively $\lambda x. T_1$ and $T_2$:

$\Pi_1$

$\Pi_2$

$x_1:A_1^- x_2:A_2^- \xrightarrow{\lambda} y_1:B_1^- y_2:B_2^- B^+$

$(B \to C)^+$

The proof-net encoding $((\lambda x. T_1) T_2)$ is obtained as follows:

$\Pi_1$

$\Pi_2$

$x_1:A_1^- x_2:A_2^- \xrightarrow{\lambda} y_1:B_1^- y_2:B_2^- C^+$

Then, the process of cut elimination, as specified at the end of Section 2.2, amounts to the reduction of the $\beta$-redexes.

3 Semantic Readings as Proof-Nets

As noticed by van Benthem, $\lambda$-terms provide a semantic reading of categorial laws such as Montague type raising, Geach composition, argument lowering, etc.

Consider, for instance, Montague type raising, i.e.,

$e \vdash ((e, t), t)$

or, using Girard’s notation,

$e \vdash ((e \to t) \to t)$.

Its semantic reading is provided by the $\lambda$-term $\lambda x. xy$, where $y$ is a free variable of type $e$.

In the case of a sequential proof, it is necessary to decorate each sequent with a $\lambda$-term in order to get this semantic reading:

$\frac{x : (e \to t) \vdash x : (e \to t) \quad y : e \vdash y : e}{y : e, x : (e \to t) \vdash xy : t}$

When using proof-nets, however, one may get the semantic reading directly. Consider the proof-net that proves Montague type raising:

$e^- \quad ((e \to t) \to t)^+$
As we will see, a simple traversal of this proof-net will provide the semantic reading. This traversal, which follows Lamarche’s dependency paths [10], may be specified by a simple set of instructions:

1. enter the proof-net by its unique output conclusion;
2. follow the path specified by the output polarities until an axiom-link is eventually reached; this path, which is ascending, is made of par-links that correspond to successive λ-abstractions;
3. cross the axiom-link following the output-input direction;
4. follow the path specified by the input polarities; this path, which is descending, is made of tensor-links that correspond to successive applications; it ends either on some input conclusion of the proof-net, or on the input premise of some par-link; in both cases, the end of the path coincides with the head-variable of the corresponding λ-term; in the first case (input conclusion), this head-variable is free; in the second case (premise of a par-link) this head-variable is bound to the λ corresponding to the par-link;
5. in order to get all the arguments to which the head-variable is applied, start again the same sort of traversal from every output premise of the tensor-links that have been encountered during the descending phase described in 4;

It is worth noting that the above traversal algorithm does not make sense on every proof-structure. For instance, one may easily imagine proof-structures some links of which would never have been visited during the traversal. Another possible problem is when reading the head-variable: the descending path that follows the input polarities could end on the input premise of a par-link that would not have been visited before, i.e., a par-link that would not correspond to a λ-abstraction. But these “pathological” proof-structures, for which the reading algorithm does not work, are precisely the ones that are rejected by the correctness criterion. In other words, they are not proof-nets.

As a further illustration, consider the following consequence of Geach composition rule:

\[(a \rightarrow b) \leftarrow (c \rightarrow a) \rightarrow (c \rightarrow b),\]

to which is associated the following proof-net:

\[(a \rightarrow b)^- \; (c \rightarrow a) \rightarrow (c \rightarrow b)^+\]

Let us try to apply the reading algorithm on this example:

1. we enter the proof-net by its output conclusion (i.e. the conclusion of type \((c \rightarrow a) \rightarrow (c \rightarrow b)^+\)), and go up, following the output polarities; we cross two par-links that correspond to two successive λ-abstractions, say \(\lambda x\) and \(\lambda y\); hence, the λ-term that we are reading has the form \(\lambda x.\lambda y. T_1\)
2. we follow the axiom-link in the output-input direction, and go down, following the input polarities; we cross the leftmost tensor-link, and we end on the input conclusion of type \((a \rightarrow b)^-\); this input conclusion corresponds to a free head-variable, say \(z\); therefore, we are reading a λ-term of the form \(\lambda x.\lambda y. (z \; T_2)\).
3. in order to read the argument to which is applied the head-variable \( z \) (i.e. the \( \lambda \)-term \( T_2 \)), we again start the process from the output conclusion of the tensor-link that we just crossed; we follow the leftmost axiom-link, cross the second tensor-link and we end on the input premise of the par-link corresponding to \( \lambda x \); hence, we have read a \( \lambda \)-term of the form \( \lambda x. \lambda y. (z \, x \, T_3) \);

4. similarly, we read the \( \lambda \)-term corresponding to \( T_3 \) and we get the complete reading of the proof-net: \( \lambda x. \lambda y. (z \, x \, y) \).

Thus, we have shown how to obtain semantic readings from proof-nets by a simple traversal following Lamarche's dependency paths. In fact, this traversal of the proof-nets is so simple (linear time) that one can say that the semantic reading is no longer provided by a \( \lambda \)-term but by the proof-net itself. In other words, we argue that we no longer need the \( \lambda \)-terms anymore since we have the proof-nets at our disposal. This point of view will make more sense when working with logics more powerful than ILL in which the correspondence between \( \lambda \)-terms and proof-nets is no longer one-one. Indeed, for such logics, the notion of proof-net is much richer than that of \( \lambda \)-term.

4 Proof-Nets for the Categorial Hierarchy

4.1 The Categorial Hierarchy

In [24] van Benthem defines the following categorial hierarchy, starting on the left with the Lambek calculus and ending on the right with the intuitionistic implicational logic:

```
  LP     LPC       IL
 /\      \       \
L       ILC      
  \       /\      \        /\      \        
  \       /\      \        /\      \        
```

Each of these implicational calculi may be obtained from another by adding or removing one or more structural rules. For instance, starting from \( L \), one gets \( LC \) by admitting the contraction-rule. Then, LPC is obtained by adding the permutation-rule. Finally, one reaches IL by adding to LPC by the weakening-rule.

In Section 2, we have introduced the notion of proof-net in the framework of ILL, which is another name for \( LP \). Therefore, in order to adapt this notion to the Lambek calculus, we must explain how to reject the permutation-rule. This will be explained briefly in Section 4.2.

On the other hand, in order to adapt the notion of proof-net to \( LC \), LPC, and IL, we must allow for the structural rules of contraction and weakening. This is done, in linear logic, by using Girard's modal operator "". Hence, we will consider, in Section 4.3, a the fragment of intuitionistic linear logic that contains "" and "" as the only connectives. This fragment is called intuitionistic implicational exponential linear logic (IELL, for short).

4.2 Proof-Nets for the Lambek calculus

Proof-nets for the lambek calculus have been defined by Roorda in his thesis [21] and are presented in detail in [11, 20].

In order to deal with the non-commutativity of \( L \), one must distinguish between the left and right premises of the links. Consequently, one gets two different sorts of tensor-links, corresponding to the formulas \( (A \setminus B)^- \) and \( (A/B)^- \), and two different sorts of par-links, corresponding to the formulas \( (A\setminus B)^+ \) and \( (A/B)^+ \). One must also take into account the fact that the formulas in Lambek's sequents are ordered. This gives rise to an order on the

---

1 There is actually one difference: in LP, the empty antecedent is not admitted in the sequents.
conclusions of the proof-nets (a cyclical order, to be precise). Then, in order to adapt the correctness criterion, one has to add a planarity requirement: the axiom-links may not cross one another.

4.3 Proof-nets for Intuitionistic Implicative Exponential Linear Logic

Girard’s unary connective ‘!’ is a modal operator that allows for the structural rules of contraction and weakening. It obeys the following logical rules:

\[
\begin{align*}
A, \Gamma & \vdash C & \text{(dereliction)} & & !A_1, \ldots, !A_n & \vdash !C & \text{(promotion)} \\
& & & & !A, !A, \Gamma & \vdash C & \text{(contraction)} & & !A, \Gamma & \vdash !C & \text{(weakening)}
\end{align*}
\]

Consequently, intuitionistic implication ‘\(\rightarrow\)’ may be defined as follows:

\[(A \rightarrow B) = (!A \rightarrow B).\]

Now, in order to accommodate the proof-nets with this modal operator, new sorts of links must be introduced:

<table>
<thead>
<tr>
<th>NAME</th>
<th>Dereliction</th>
<th>Contraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK</td>
<td>p</td>
<td>P₁, P₂</td>
</tr>
<tr>
<td></td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td>PREMISES</td>
<td>p, c</td>
<td>P₁, P₂, c</td>
</tr>
<tr>
<td>CONCLUSIONS</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>TYPES</td>
<td>p : A⁻</td>
<td>p₁, p₂, c : !A⁻</td>
</tr>
<tr>
<td></td>
<td>c : !A⁻</td>
<td></td>
</tr>
<tr>
<td>POLARITIES</td>
<td>p, c : input</td>
<td>p₁, p₂, c : input</td>
</tr>
</tbody>
</table>

As for promotion and weakening, simple links are not sufficient. The problem with the promotion rule is that it is contextual:

In order to circumvent these difficulties, one uses boxes: A box is a part of the proof structure, the interior of which is itself a proof structure. The conclusions of a box are called its doors. There are two kinds of boxes: promotion-boxes and weakening-boxes.

\[\text{promotion-box : }\]

\[\text{weakening-box : }\]

Then, the notion of correctness is defined by induction on the nesting of the boxes:

1. the proof-net obtained by replacing each box by a n-ary axiom-link whose conclusions are the doors of the boxes satisfies the usual correctness criterion;
2. the interior of each box is correct.

On the dynamic side, the presence of the connective “!” gives rise to new cut-elimination cases that correspond to new graph rewriting rules:

where, strictly speaking, the $n$-ary weakening box in the lefthand side of the second rule correspond to $n$ nested ordinary weakening boxes.

### 4.4 Encoding simply-typed $\lambda$-terms

The intuitionistic implicative logic corresponds, through the Curry-Howard isomorphism, to the simply-typed $\lambda$-calculus. Hence, the simply-typed $\lambda$-terms may be encoded into the HELL proof-nets.

The translation is very similar to the one described in section 2. The main difference is that a $\lambda$-term whose free variables are of type $A_i$ is now translated into a proof-net whose input conclusions are of type $!A_i$. This allows different input conclusions of the same type to be contracted, which is a way of taking the non-linearity of the terms into account. Then, the translation has to be adapted by using contraction-links, dereliction-links, weakening-boxes, and promotion-boxes when needed.

Conversely, the traversal algorithm that provides the $\lambda$-term corresponding to a proof-net is almost the same as the one described in Section 3. The only difference is during the
descending phase: after having crossed a bunch of tensor-links, one may cross a dereliction-
link followed by several contraction-links before reaching the head-variable.

For more details on this topic, see [2, 4, 19].

5 Semantic Recipes as Proof-Nets

In this section we provide an example of the use of proof-nets as a uniform framework that
allows syntactic and semantic analysis to be unified. This example, which is more elaborate
that those of Section 3, is in the spirit of Montague PTQ Grammar[14].

On the syntactic side, we use the Lambek calculus together with three basic types: n, sn, and s.

On the semantic side, we use HELL with the two Montagovian basic types e and t. For
the purpose of our example, we also assume the existence of the following typed constants
(with their obvious intended meanings):

\[ \forall : ((e \rightarrow t) \rightarrow t), \quad \circ : (t \rightarrow (t \rightarrow t)), \quad \text{barber} : (e \rightarrow t), \quad \text{shave} : (e \rightarrow (e \rightarrow t)) \]

Technically, these constants will be handled as free variables: they will decorate input
conclusions of proof-nets.

We also assume that the following homomorphism \( \mathcal{H} \) between syntactic and semantic
types is given:

\[
\mathcal{H}(n) = (le \rightarrow t) \quad \mathcal{H}(sn) = !e \quad \mathcal{H}(s) = t
\]
\[
\mathcal{H}(A\backslash B) = \mathcal{H}(A) \rightarrow \mathcal{H}(B) \quad \mathcal{H}(A/B) = \mathcal{H}(B) \rightarrow \mathcal{H}(A)
\]

Then, we consider a lexicon made of words to which syntactic types and semantic
proof-nets are attached. Each of these semantic proof-nets is such that:

1. the type of its unique output conclusion is the semantic type of the corresponding word,
   i.e., the homomorphic image of the the associated syntactic type;
2. its input conclusions (if any) are decorated with constants.

<table>
<thead>
<tr>
<th>WORD</th>
<th>SYNTACTIC CATEGORY</th>
<th>SEMANTIC CATEGORY</th>
<th>SEMANTIC PROOF-NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>barber</td>
<td>n</td>
<td>((le \rightarrow t))</td>
<td>(\Pi_{\text{barber}})</td>
</tr>
<tr>
<td>every</td>
<td>((s/(np\backslash s))/n)</td>
<td>((le \rightarrow t) \rightarrow ((le \rightarrow t) \rightarrow t))</td>
<td>(\Pi_{\text{every}})</td>
</tr>
<tr>
<td>himself</td>
<td>(((np\backslash s)/np\backslash (np\backslash s)))</td>
<td>((le \rightarrow (le \rightarrow t)) \rightarrow (le \rightarrow t))</td>
<td>(\Pi_{\text{himself}})</td>
</tr>
<tr>
<td>shaves</td>
<td>(np\backslash (s/np))</td>
<td>((le \rightarrow (le \rightarrow t)))</td>
<td>(\Pi_{\text{shaves}})</td>
</tr>
</tbody>
</table>

where the proof-nets \(\Pi_{\text{barber}}, \Pi_{\text{every}}, \Pi_{\text{himself}}, \Pi_{\text{shaves}}\) are respectively the following:
These semantic proof-nets correspond to semantic recipes. For instance, by applying the reading algorithm of Sections 3 and 4 on $\Pi_{\text{every}}$, one obtains the following $\lambda$-term:

$$\lambda P.\lambda Q.\forall (x. \top(P x)(Q x))$$

or, using a more usual notation for the connectives $\forall$ and $\top$:

$$\lambda P.\lambda Q.\forall x.(P x) \top (Q x)$$

Now, consider the non-commutative proof-net resulting from the syntactic analysis of the sentence “every barber shaves himself”:

If one does not take the orientation of the links into account (identifying both \ and / with $\top$), we may think of the above proof-net as a proof-net of ILL or even IELL. Then, replacing each syntactic type by the corresponding semantic type does not affect the well-typedness of the proof-net because $\mathcal{H}$ is a homomorphism. This allows the semantic recipes to be plugged by means of cuts, the resulting proof-net being still well-typed:
Then, by cut elimination, one gets the semantic proof-net associated to the whole sentence:

whose semantic reading provides: $\forall x. (\text{barber } x) \supset (\text{shave } xx)$.

**Acknowledgment**

We would like to thank A. Cichon and F. Lamarche for helpful comments.

**References**
