Bidirectional Functional Semantics for Pregroup Grammars

Gabriel Gaudreault

Concordia University, Montreal, Quebec, Canada
gabriel.gaudreault@gmail.com

Abstract

Pregroup grammars are a recent descendant of the original categorial grammars of Bar-Hillel [1] and Lambek [10] in which types take the form of strings of basic types and left and right adjoints, as opposed to the non-commutative functional types of categorial grammars. Whereas semantic extraction is possible in other categorial grammars through the λ-calculus, this approach will not be feasible for pregroup grammars. In this paper, we show how to build a term calculus that could be used to fill this void. This system is inspired by the λ-calculus but differs in crucial aspects: it uses function composition as its main reduction strategy instead of function application and is bidirectional, i.e. the direction arguments are applied to terms matters. We show how this term calculus is one-to-one with a proper subset of pregroup types and give multiple examples to show how this system could be used to do semantic analysis in parallel to doing grammaticality checks with pregroup grammars.

1 Motivations

Semantic extraction from lexical expressions is commonly done in type-logical grammars [4] by using a version of the typed λ-calculus in which logical predicates are fed arguments in a well-defined way in coordination with the type reduction.Unfortunately, this option is not possible for pregroup grammars without too many additional constraints to the system. Semantic analysis in pregroups has been done, but using very different approaches. One of them is to place the semantics in free compact 2-categories [14], another one to use vector space models [8, 9, 16] and finally other more graphical approaches look directly at the reduction links and transfer the information accordingly [15].

Our approach could be seen somehow as a formalisation of the third approach, where the end of the contraction link connected to the adjoint type corresponds to an abstraction variable and the other end a value to be applied. What we are proposing is a system that takes into account the bidirectionally of argument passing and does so in a very natural way. To do so, we will redefine pregroups as logical structures with derivational rules and show how our semantic terms interact by using them as annotations for our types. We believe that the biggest advantage of a system like ours, which can also be adapted to work with other categorial grammars by modifying the reduction rules, is its way of handling arguments and co-arguments in independent ways, which better reflects the reduction process as opposed to using something like the λ-calclulus.

Pregroup grammars are also very flexible, e.g. there are no restrictions as to the order in which types such as $ab^l \cdot bc^l \cdot c$ should reduce – contraction of the b's first or of the c's first – and thus we will want our system to stay as simple complexity-wise as the original is, as it is one of its main selling points.

2 Pregroups & Pregroup Grammar

Pregroup grammars [11, 12] are elegant and computationally simple(r) categorial grammars in which types are made up of basic types and adjoints, as opposed to inverses such as is the case in the original syntactic calculus [10]. Part of what makes pregroup grammars attractive is the ease with which types can combine with eachother, giving us a greater degree of freedom and more flexibility than with other frameworks, such as for instance the syntactic calculus. At this point, pregroup grammars have already been used to analyse aspects of multiple different languages such as French, English, Italian, Latin, Persian, German and Turkish.

2.1 Definition of a Pregroup

A pregroup [12] $\mathbb{P} = (P, \rightarrow, ^r, ^l, \cdot, 1)$ is a partially ordered (by $\rightarrow$) monoid with a set of elements P, which is a set of basic types, in which every element $a \in P$ has a right and a left adjoint, $a^r \in P$ and $a^l \in P$ respectively, subject to $a \cdot a^r \rightarrow 1 \rightarrow a^r \cdot a$ and $a^l \cdot a \rightarrow 1 \rightarrow a \cdot a^l$. More precisely, it has the following properties:

Properties of a monoid

- Associativity:
$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$
- Identity:
$$a \cdot 1 = a = 1 \cdot a$$

Properties of partial orders

- Reflexitivity:
$$a \rightarrow a$$
- Antisymmetry:
$$\frac{a \rightarrow b \quad b \rightarrow a}{a = b}$$
- Transitivity:
$$\frac{a \rightarrow b \quad b \rightarrow c}{a \rightarrow c}$$

Property of compatibility on ordered monoids

$$\frac{a \rightarrow b}{c \cdot a \cdot d \rightarrow c \cdot b \cdot d}$$

We call $a \cdot a^r \rightarrow 1$, $a^l \cdot a \rightarrow 1$ contractions, and $1 \rightarrow a^r \cdot a$, $1 \rightarrow a \cdot a^l$ expansions. Frequently, we will omit $\cdot$ from our types for readability reasons, e.g. we will write ab instead of $a \cdot b$. The set of types closed under the ^{r} and ^{l} operations is called the set of simple types.

From this, we can deduce different interesting properties such as

$$a \rightarrow b \text{ iff } b^l \rightarrow a^l \text{ iff } b^r \rightarrow a^r$$

$$a^{rl} = a^{lr} = a$$

$$(a_1...a_n)^l = a_n^l...a_1^l$$

2.2 Pregroup Grammars

A pregroup grammar $\mathbb{G} = (\Sigma, P, \to, ^r, ^l, 1, \mathbb{T})$ consists of a lexicon Σ and a typing relation $\mathbb{T} \subseteq \Sigma \times \mathbb{F}$ between the alphabet and the pregroup freely generated by the simple types of P and the ordering relation $\to$. This means that each element of our lexicon will correspond to one or more words made up of simple types. For instance, we will have $(want, i\phi^l)$ – to be used in a sentence like *You want for Mark to lead a happy life* – and $(want, i\bar{j}^l)$ – to be used for *You want to eat ice cream.* The basic types we will be using are

s: declarative sentences	i: infinitives of intransitive verbs
j: infinitives of complete verb phrases	$\bar{j}$: complete infinitives with *to*
n: common nouns	$\bar{n}$: complete noun phrases
N: proper nouns	r: reflexives
π: subjects	$\bar{\pi}$: nominative pronouns
o: objects	$\bar{o}$: accusative pronouns
p: prepositional phrases	ϕ: quasi-sentence formed from infinitive

Some of the orders we will use are for instance $\bar{n} \to \pi_3$ and $\bar{n} \to o$, which would be interpreted as “Everything that is a complete noun phrase can also be a subject or as an object”; this is one very important property of pregroup grammars.

3 Proof Theory

3.1 Using Pregroup Types for Semantics

The proof theory we will define and use is a bit different than the one usually found in the literature for pregroups. For instance, as opposed to some other axiomatisations of pregroups, such as the one by Buszkowski [2, 3], we will take some distance from the algebraic definition of pregroups and aim for something more logical. Hence, instead of inference rules that mimic expansion and contraction rules such as

$$\frac{ab}{acc^l b}\,\eta$$

we will opt for introduction and elimination rules for our adjoints. Formalising the grammar as a deductive system will have the advantage that we can now make use of traditional logical methods to analyse its different properties.

We will also distance ourselves from the linear logical way of defining pregroup types, e.g. $A^r BC^l$ as $A^\perp \oplus B \oplus {}^\perp C$, or categorial way, $A^r BC^l$ as $A \setminus 0 \bullet B \bullet 0\ / C$ [3,5]. Instead of looking at pregroup types as concatenations of simple types, we will treat the adjoints as left and right implications, where the terms corresponding to the adjoint types have the power to bind the enclosed term.

The reasons here are twofold. First, we believe that to do semantics, the bonds between types should be stronger in some sense, and we cannot simply consider them as independent pieces. We have to make it clearer what parts of a type can contribute and how they can contribute to the meaning of of the whole. A term of type $A^r B$ will not be a pair of terms of type A^r and B, but a function from a term of type A to one of type B. In this sense, we could have used the notation $A \setminus B$ instead of $A^r B$, but we feel that the resulting system is still very much closer to pregroup types than categorial types, especially in the way types can combine.

The reason for those restrictions is also syntactic. Just like in categorial grammars, one reason one rarely sees types such as $B \bullet (A \setminus 1)$ is because once reducing it with an argument of type A, one gets a product of a term of type 1 and a term of type B, but it is hard to think of what something non-trivial of type 1 might be. It is also not clear how $A \setminus 1$ should contract in this case as it appears on the right edge of the type, and with what type it should contract. And so, we will not allow this kind of type to be formula in our system.

To see why we prefer to think of adjoints as implications, consider the typing relation (dances, $\pi_3^r s$). We would expect its semantic term to say something like: the meaning of the sentence we end up with should depend on whatever gets passed over when π_3 contraction happen. We feel that having a pair as the meaning for $\pi_3^r s$ would not really make sense as π_3^r is not a container of semantic data, its semantic role is functional: it tells us what kind of argument we should be expecting and where. We shouldn't think of a type such as $\pi_3^r s$ as a cartesian or tensor product or anything that could be split up in 2 parts. The full type for *dance* should be seen more as a linear functional type

$$\lambda x.dance(x) : \pi_3 \multimap s$$

And so in our system, what might seem like concatenation of pregroup types should actually be thought of as left and right implication, and our types will have the form $A_1^r ... A_n^r b C_1^l ... C_m^l$, where b is a basic type and its meaning dependent on its arguments of type A_i's and C_j's, which can be any type, not necessarily basic. We could also introduce some sort of cartesian product $A \otimes B$ for expressions with tuple meanings, although those do not often appear in pregroup analyses or even categorial grammars in general.

One important inconvenience when trying to turn pregroup types into logical systems is the fact that sometimes two lexical items will have their types reduce with each other but in two consecutive contractions. This is the case for instance with quantifiers like *somebody* which is looking for a sentence on its right but also has to act like a subject to contract with the verb's inverse subject type.

$$\begin{array}{cc} \text{somebody} & \text{dances} \\ s s^l \pi^{rl} & \pi^r s \\ \hline \multicolumn{2}{c}{s} \end{array}$$

The issue we're faced with here is that it is not clear in how many steps this should take place. In one big all-possible-contractions-should-happen-at-the-same-time? Or perhaps having one kind of "external" merging operation that would take care of reducing the subject first, merge *somebody* and *dances* together, then a second "internal" merging operation could reduce the internal s's in $ss^l s$. We will go in a different direction where we basically will not have to face this type of situation and do all reductions in one step.

The problematic is also interesting from a semantics point-of-view; how much discrete information should be shared by the two expressions? Should *somebody* be taking in only one or two arguments? The type π^{rl} it contains is a double adjoint, meaning that it is looking for an adjoint to contract with (or triple adjoint), so does it mean that the expression should expect two arguments, one corresponding to s^l and the other to π^{rl}? In this case, what should the argument corresponding to π^r look like? Notice though that double adjoint pregroup type π^{rl} is the same as π, so if we associate adjoints with "receivers" of information and basic types with the actual pieces of information passed around, then perhaps *somebody* should be first sending some information to *dances*, and then receive some from it. In this case, what seems to make the most sense would be that *somebody* send over a dummy variable to *dance* which would get abstracted over when passed back, something like:

$$\begin{pmatrix} ss^l\pi \\ (\lambda y.\forall x.y) \bullet x \end{pmatrix} \begin{pmatrix} \pi^r s \\ \lambda z.dance(z) \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} ss^l s \\ (\lambda y.\forall x.y)dance(x) \end{pmatrix} \rightarrow \begin{pmatrix} s \\ \forall x.dance(x) \end{pmatrix}$$

From a technical point of view though this seems somewhat complicated as normally this kind of binding would be illegal and some kind of α-renaming rewriting would go on first to change x's name in the quantifier to some other variable that's not free in y, then the term $dance(x)$ could be passed to it.

We believe that what happens is better reflected in the non-expanded form $s(\pi^r s)^l$, as expanding its type in a way breaks the relation between data – s – and its argument – π_3. Hence only one piece of information should be passed along during the contraction and not two.

Something else to note, is that in most cases, if a lexical item's type contains adjoints such as AB^lC^l it will either always contract in multiple steps, or it will always contract B and C is one swoop, in which case we believe that the non-expanded form $A(CB)^l$ is more appropriate. Hence why we feel that lacking distributivity rules, e.g. $(AB^l)^l \leftrightarrow B^{ll}A^l$ is not a problem, as it does not seem to represent anything of linguistic relevance.

This goes to other way too, it wouldn't make sense for a lexical item such as the ditransitive verb *give* to get the type $i(op)^l$ instead of the usual type ip^lo^l. It is hard to think of a situation where that non-distributed type would be appropriate, a situation where *give* contracts with another lexical item that is both a direct object and a prepositional phrase. It will basically always follow the pattern:

$$\begin{array}{ccccc} give & a & star & to & Bob \\ ip^lo^l & \bar{n}n^l & n & p\bar{n}^l & N \end{array}$$
$$\rightarrow \; ip^lo^l \quad \bar{n} \quad p\bar{n}^l \quad \bar{n}$$
$$\rightarrow \; ip^lo^l \quad o \quad p$$
$$\rightarrow \; ip^l \quad p \;\rightarrow\; i$$

(Note that we used the fact that $\bar{n} \rightarrow o$.) The same is also true with particles such as for that introduce quasi-sentences [12].

$$\begin{array}{cccccc} John & wants & for & Mary & to & live \\ N & \pi_3^r s\phi^l & \phi\bar{j}^lo^l & N & \bar{j}i^l & i \end{array}$$
$$\rightarrow \; \pi_3 \quad \pi_3^r s\phi^l \quad \phi\bar{j}^lo^l \quad o \quad \bar{j}$$
$$\rightarrow \; s\phi^l \quad \phi\bar{j}^l \quad \bar{j}$$
$$\rightarrow \; s\phi^l \quad \phi \;\rightarrow\; s$$

On the other hand, for the quantifier *somebody* to contract with two different lexical items would require it to contract first with a string of lexical items which have a combined type of π^r and then with one that has type s on its left edge. It is easy to find example for the second lexical item, but we have never encountered any examples that would fit the first type.

$$\begin{array}{ccc} somebody & ? & Edward\ likes\ pie \\ ss^l\pi & \pi^r & s \end{array}$$
$$\overline{\hspace{6cm}}$$
$$s$$

We can even go a bit further and predict that most of the time, a type with a double adjoint will also contain a simple adjoint next to it and that those two should actually show up as one adjoint type. For instance, never will we see

$$\begin{array}{cccc} \textit{John} & \textit{who} & \textit{Carl} & \textit{I like ice cream} \\ N & \bar{n}^r\bar{n}s^l\pi & \pi & s \\ \hline \multicolumn{4}{c}{\bar{n}} \end{array}$$

Similarly, it is hard to find a string of lexical items that would have a type of o^l in the following case:

$$\begin{array}{cccc} \textit{John} & \textit{whom} & \textit{I like ice cream} & ? \\ N & \bar{n}^r\bar{n}o^lls^l & s & o^l \\ \hline \multicolumn{4}{c}{\bar{n}} \end{array}$$

We can see that interrogative sentences also follow this pattern:

$$\begin{array}{ccc} \textit{was} & \textit{I} & \textit{seen?} \\ q_2(p_2o^l)^l\pi_1^l & \pi_1 & p_2o^l \end{array}$$

Assigning other types such as $q_2o^{ll}p_2^l\pi_1^l$ or $q_2(\pi_1p_2o^l)^l$ to *was* would not make as much sense. Hence we will stick to the contracted forms $A(BC^l)^l$ and $A(B^rC)^l$ instead of AC^{ll} or AC^lB so that we could have one only contraction of types per concatenation of lexical items.

3.2 Inference Rules

We first define the formulæ we will be using from now on.

Definition 1. *Formula:*
- If A is a basic type, then A is formula
- If A and B are formulæ then AB^l and A^rB are also formulæ
- Nothing else is a formula

Already, by defining our formulæ as such, we make it impossible to get a whole range of formulæ such as of those of the forms AB, where none are adjoints, or A^lB, or even AC^lB, which in the original pregroups is equal to the form $A(B^rC)^l$ after distribution of the ^{l}-operator and cancellation of the double left and right adjoints. That will not be a problem as most of those forms, e.g. $1/A$, $A \times (C \setminus B)$, also rarely show up in categorial grammars.

The system then makes it pointless to try to show relations such as $(AB)^l = B^lA^l$, which aren't really relevant linguistically to us, as those aren't even considered types (or formulæ) in our system. One thing that could be useful though is if we could show relations of the form $A(BC)^l = AC^lB^l$, which is not possible now, but even then, it is not clear whether we actually do need it or not.

Note that types such as A^rB^l could be recovered by adding the identity type 1 to the system, and types A^lB or BC^r by having an identity type and allowing product types.

The reader might have noticed that this way of defining pregroup types actually makes them very similar to types of the associative syntactic calculus [13] – $[AB^l] \Rightarrow [A]/[B]$ and so on – but we will see below that the system differs crucially in the way types combine. Whereas $\setminus/$-Elimination can be seen as function application, ^{rl}-Elimination acts more like function composition.

We give a natural deduction presentation of our type system. Here Γ stands for a possibly empty list of formulæ separated by commas. We also use Σ^r and Σ^l to signify a possibly empty sequence of right or left adjoints within a same formula, e.g. if $\Sigma^l = (EF)^l G^l$, then $A^r BC^l (EF)^l G^l = A^r BC^l \Sigma^l$.

$$\dfrac{\dfrac{\dfrac{\overline{\;B\;}\,u}{\vdots \qquad \vdots}}{A}}{AB^l}\,{}^lI^u \qquad\qquad \dfrac{AB^l \qquad B\Sigma^l}{A\Sigma^l}\,{}^lE$$

$$\dfrac{\dfrac{\dfrac{\overline{\;B\;}\,u}{\vdots \qquad\quad \vdots}}{A}}{B^r A}\,{}^rI^u \qquad\qquad \dfrac{\Sigma^r B \qquad B^r C}{\Sigma^r C}\,{}^rE$$

Note that A, B, and C do not have to be basic or even simple types, they can be just about any formula. We can rewrite the rules using the provability relation $\vdash$; read $\Gamma \vdash A$ as "A is provable in context Γ". Here Γ stands for a possibly empty list of formulæ separated by commas.

$$\dfrac{}{A \vdash A}\,init$$

$$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash BA^l}\,{}^lI \qquad\qquad \dfrac{\Gamma_1 \vdash AB^l \qquad \Gamma_2 \vdash B\Sigma^l}{\Gamma_1, \Gamma_2 \vdash A\Sigma^l}\,{}^lE$$

$$\dfrac{A, \Gamma \vdash B}{\Gamma \vdash A^r B}\,{}^rI \qquad\qquad \dfrac{\Gamma_1 \vdash \Sigma^r B \qquad \Gamma_2 \vdash B^r C}{\Gamma_1, \Gamma_2 \vdash \Sigma^r C}\,{}^rE$$

From those, it is easy to derive relations such as:

$$\dfrac{\dfrac{}{A \vdash A}}{\vdash AA^l} \qquad \Rightarrow \qquad \begin{array}{c} {}^lExpansion: \\ \vdash AA^l \end{array}$$

$$\dfrac{A \vdash B \qquad \dfrac{B \vdash C}{\vdash B^r C}}{A \vdash C} \qquad \Rightarrow \qquad \begin{array}{c} Transitivity \\ A \vdash B \qquad B \vdash C \\ \hline\hline A \vdash C \end{array}$$

$$\dfrac{\dfrac{}{AB^l \vdash AB^l} \qquad \dfrac{\dfrac{}{BC^l \vdash BC^l}}{\vdash BC^l (BC^l)^l}}{AB^l \vdash AC^l (BC^l)^l} \qquad \Rightarrow \qquad \begin{array}{c} Geach's Law \\ AB^l \vdash AC^l (BC^l)^l \end{array}$$

$$\dfrac{A \vdash B \qquad \dfrac{\dfrac{}{AD^l \vdash AD^l} \qquad C \vdash D}{AD^l \vdash AC^l}}{AD^l \vdash BC^l} \qquad \Rightarrow \qquad \begin{array}{c} Functoriality \\ A \vdash B \qquad C \vdash D \\ \hline\hline AD^l \vdash BC^l \end{array}$$

Note that for most of our linguistic purposes we will not need to use the introduction rules, just like we do not need to use expansions in the regular pregroup grammars to test grammaticality, only contractions.

An extra subtyping rule is also derivable in the system if we decide to encode ordering relations $A \to B$ as $A \vdash B$:

$$\dfrac{\Gamma \vdash \Sigma_1^r A \Sigma_2^l \qquad \dfrac{\dfrac{\dfrac{A \vdash B}{\vdash B A^l}\,{}^l I \qquad \dfrac{}{A\Sigma_2^l \vdash A\Sigma_2^l}\,\mathit{init}}{A\Sigma_2^l \vdash B\Sigma_2^l}\,{}^l E}{\vdash (A\Sigma_2^l)^r B \Sigma_2^l}\,{}^r I}{\Gamma \vdash \Sigma_1^r B \Sigma_2^l}\,{}^r E$$

$$\dfrac{\Gamma \vdash \Sigma_1^r a \Sigma_2^l \qquad a \vdash b}{\Gamma \vdash \Sigma_1^r b \Sigma_2^l}\,\mathit{Subtyping}$$

4 τ-Calculus

4.1 System Description

The term calculus we will be using in the semantic layer is very much inspired by the λ-calculus in both its structure and the way it will be used for semantics. At its core it is simply a way to represent functions that require both a variable (left) and co-variable (right), and thus can be a useful framework for semantics or syntax as it makes it more natural to deal with arguments coming from different sides.

Just as in the λ-calculus, the two fundamental operations on terms are application and abstraction, which will show up in this calculus as left and right application and left and right abstraction. These two sets of operations, namely left application and abstraction, and right application and abstraction, are completely disjoint from eachother. There is no way for instance to reduce a right abstraction using a left application. One of the main differences from the λ-calculus is the way abstraction inside of an incoming term is in some cases pushed to the exterior layer of the term to abstract over it. We call this term calculus τ-calculus.

One specificity of this calculus is how the terms are typed by pregroup types, as opposed to regular function types $A \to B$.

We define our types recursively starting with a set of basic types – a type for entities $\mathbf{e}$ and one for propositions $\mathbf{p}$ – from which we can build functional types. As said before, adjoints are not usually described as "functional", but we think this is appropriate considering how we use them.

Definition 2. *Semantic types*

$$\textbf{Type}\ \sigma, \phi \ := \ \mathbf{e} \mid \mathbf{p} \mid \sigma^r \phi \mid \sigma \phi^l$$

We now define the set of τ-terms of type σ, $\textbf{Term}_\sigma$. In the following, Ω^l, Ω^r stand for possibly empty sequences of left and right adjoints respectively, and $\textbf{Var} = \bigcup \textbf{Var}_\sigma$ and $\textbf{Con} = \bigcup \textbf{Con}_\sigma$ are set of variables and constants.

Definition 3. $\mathbf{Term}_\sigma$ Formation Rules

$$\frac{x \in \mathbf{Var}_\sigma}{x \in \mathbf{Term}_\sigma} \qquad \frac{c \in \mathbf{Con}_\sigma}{c \in \mathbf{Term}_\sigma}$$

$$\frac{t \in \mathbf{Term}_{\mathbf{\Omega^r}\phi} \qquad s \in \mathbf{Term}_{\phi^{\mathbf{r}}\pi}}{(t)s \in \mathbf{Term}_{\mathbf{\Omega^r}\pi}} \qquad \frac{t \in \mathbf{Term}_{\pi\phi^{\mathbf{l}}} \qquad s \in \mathbf{Term}_{\phi\mathbf{\Omega^l}}}{t(s) \in \mathbf{Term}_{\pi\mathbf{\Omega^l}}}$$

$$\frac{t \in \mathbf{Term}_\sigma \qquad x \in \mathbf{Var}_\phi}{t|x\rangle \in \mathbf{Term}_{\sigma\phi^{\mathbf{l}}}} \qquad \frac{x \in \mathbf{Var}_\phi \qquad t \in \mathbf{Term}_\sigma}{\langle x|t \in \mathbf{Term}_{\phi^{\mathbf{r}}\sigma}}$$

There is no need for parentheses around abstraction brackets as abstraction is associative in our calculus, just like pregroup types are.

$$\langle x|a|y\rangle = (\langle x|a)|y\rangle = \langle x|(a|y\rangle)$$

and

$$\mathbf{Term}_{\phi^r\sigma\pi^l} = \mathbf{Term}_{(\phi^r\sigma)\pi^l} = \mathbf{Term}_{\phi^r(\sigma\pi^l)}$$

Since our calculus is bidirectional, unlike the λ-calculus, we cannot really get rid of parentheses without allowing ambiguity.

The two kinds of abstraction here should simply be seen as two different λ-abstractions that differ in the incoming direction of their argument. For instance,

$$t[x]\ |x\rangle(s) =_\beta t[x := s]$$

$$(s)\langle x|\ t[x] =_\beta t[x := s]$$

but

$$\langle x|\ t[x]\ (s) \neq_\beta t[x := s]$$

By $t[x]$ we mean that x appears somewhere in t, and by $t[x := s]$ the replacement of all free occurrences of the variable x by the term s.

We define the set $\mathbf{Free}(t)$ of free variables of the τ-term t recursively.

Definition 4. $\mathbf{Free}(t)$

- $\mathbf{Free}(x) = \{x\}$, *if* $x \in \mathbf{Var}$
- $\mathbf{Free}(c) = \emptyset$, *if* $c \in \mathbf{Con}$
- $\mathbf{Free}(t(s)) = \mathbf{Free}((s)t) = \mathbf{Free}(s) \cup \mathbf{Free}(t)$, *if* $s, t \in \mathbf{Term}$
- $\mathbf{Free}(\langle x|t) = \mathbf{Free}(t|x\rangle) = \mathbf{Free}(t) \setminus \{x\}$, *if* $t \in \mathbf{Term}$, $x \in \mathbf{Var}$

We also define reduction rules, which are pretty standard except for β-reduction which reflects the way we can concatenate adjoints following type contraction.

Definition 5. *Reduction Strategies*

β-reduction

$$t|x\rangle\ (b|x_1\rangle ... |x_n\rangle)\ =_\beta\ (t)[x := b]\ |x_1\rangle ... |x_n\rangle,$$

$$\textit{if } t|x\rangle \in \mathbf{Term}_{\sigma\phi^{\mathbf{l}}} \textit{ and } b|x_1\rangle ... |x_n\rangle \in \mathbf{Term}_{\phi\pi_1^{\mathbf{l}}...\pi_n^{\mathbf{l}}}$$

$$(\langle x_1^r| ... \langle x_n^r|b)\ \langle x|t \ =_\beta\ \langle x_1^r| ... \langle x_n^r|\ (t)[x := b]$$

$$\textit{if } \langle x|t \in \mathbf{Term}_{\phi^{\mathbf{r}}\sigma} \textit{ and } \langle x_1^r| ... \langle x_n^r|b \in \mathbf{Term}_{\pi_1^{\mathbf{r}}...\pi_n^{\mathbf{r}}\phi}$$

α-renaming

$$s\ |x\rangle \ =_\alpha\ (s)[x := y]\ |y\rangle$$

$$\langle x|\ s \ =_\alpha\ \langle y|\ (s)[x := y]$$

η-reduction

if x doesn't appear free in s:

$$s(x)\ |x\rangle =_\eta\ s$$

$$\langle x|\ (x)s \ =_\eta\ s$$

Definition 6. *Term Substitution*

$$\begin{array}{lll}
(x)[x := u] & = u & x \in \mathbf{Var} \\
(y)[x := u] & = y & y \neq x, y \in \mathbf{Var} \\
(c)[x := u] & = c & c \in \mathbf{Con} \\
(s(t))[x := u] & = (s)[x := u]\ ((t)[x := u]) & s, t \in \mathbf{Term} \\
((s)t)[x := u] & = ((s)[x := u])\ (t)[x := u] & s, t \in \mathbf{Term} \\
(s|x\rangle)[x := u] & = s|x\rangle & s \in \mathbf{Term}, x \in \mathbf{Var} \\
((\langle x|s)\ [x := u] & = \langle x|s & s \in \mathbf{Term}, x \in \mathbf{Var} \\
(s|y\rangle)[x := u] & = (s)[x := u]|y\rangle & s \in \mathbf{Term}, x \in \mathbf{Var} \\
((\langle y|s)\ [x := u] & = \langle y|(s)[x := u] & s \in \mathbf{Term}, x \in \mathbf{Var}, y \neq x
\end{array}$$

Here's an example of β-reduction that shows how abstractions combine:

$$T(x)(y)\ |y\rangle\ |x\rangle\ (S(z)\ |z\rangle) \ =_\beta\ T(S(z))(y)\ |y\rangle\ |z\rangle$$

The corresponding type reduction would look something like:

$$\sigma\phi^l\psi^l\psi\pi^l \to \sigma\phi^l\pi^l$$

As a side note, notice that we could get rid of parentheses in our term calculus as they are in 1-1 relation with their semantic types. And so contractions $\sigma^l\sigma \to 1$ will happen in conjunction to reduction $|x\rangle a \to \epsilon$ (empty string), where any x on the left of the expression gets replaced by a. Although this requires us to be extra careful about variables to avoid ambiguity.

Let's look at an example.

$$\begin{array}{cccccccc}
\pi & \pi^r\sigma\phi^l & \phi & \phi^l & \psi & \psi^r\phi & \psi^l & \psi \\
a & b & x & |x\rangle & z & c & |z\rangle & d
\end{array}$$

$$\to \begin{array}{cccccc}
\pi & \pi^r\sigma\phi^l & \phi & \phi^l & \psi & \psi^r\phi \\
a & b & x & |x\rangle & d & c
\end{array}$$

$$\to \begin{array}{cccc}
\sigma\phi^l & \phi & \phi^l & \phi \\
ab & x & |x\rangle & dc
\end{array}$$

$$\to \begin{array}{cc}
\sigma\phi^l & \phi \\
ab & dc
\end{array} \to \begin{array}{c}
\sigma \\
abdc
\end{array}$$

We now know the type of the whole string, σ, and can deduce which terms are passed to what other terms by looking at their respective types: b has type $\pi^r\sigma\phi^l$, and so a must be its left argument; c has type $\psi^r\phi$, and so d must be its argument, which put together gives out a type ϕ, from which we deduce that the whole dc is the right argument of b.

Not having parentheses requires us though to figure out the reduction links in the types first. For instance, if we had the string of terms:

$$abc$$

and knew that $a : \sigma$ and $b : \sigma^r\phi$, we could be tempted to treat a as b's argument and make a term of type ϕ. Although if c's type turns out to be $(\sigma^r\phi)^r\sigma^r\pi$ then they are actually both arguments to c and that a should not be passed to b. And so we have to know the types of every subterm and figure out how they contract before doing term application.

4.2 Annotated Proof Theory

We can now put both pieces together and annotate the typing rules using our functional terms. We write $\Gamma \vdash a : A$ for "The term a has type A in context Γ".

$$\frac{}{x : A \vdash x : A}\ init$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash b|x\rangle : BA^l}\ {}^lI \qquad \frac{\Gamma_1 \vdash a : AB^l \qquad \Gamma_2 \vdash b : B\Sigma^l}{\Gamma_1, \Gamma_2 \vdash a(b) : A\Sigma^l}\ {}^lE$$

$$\frac{x : A, \Gamma \vdash b : B}{\Gamma \vdash \langle x|b : A^rB}\ {}^rI \qquad \frac{\Gamma_1 \vdash a : \Sigma^rA \qquad \Gamma_2 \vdash b : A^rB}{\Gamma_1, \Gamma_2 \vdash (a)b : \Sigma^rB}\ {}^rE$$

For more clarity, here's what ${}^lE$ rule looks like if we perform β-reduction directly:

$$\frac{\Gamma_1 \vdash a|x\rangle : AB^l \qquad \Gamma_2 \vdash b|x_n\rangle...|x_1\rangle : BC_n^l...C_1^l}{\Gamma_1, \Gamma_2 \vdash\ a[x := b]\ |x_n\rangle...|x_1\rangle : AC_n^l...C_1^l}\ {}^lE$$

4.3 Linguistic Justifications

Let us reitereate the two main differences between our τ-calculus and the λ-calculus:

1. Our system has term composition as main method of reducing terms as opposed to term application
2. Our system is bidirectional: The two sides of a term act independently of one another

The first difference was essential as we want to model pregroup contractions as closely as possible, which can be naturally thought of as function composition.

For instance, passing terms $g : BC^l$ to $f : AB^l$ gives us:

$$f : AB^l\ (\ g : BC^l) \to f \circ g : AC^l$$

which, if we think of $f : AB^l$ as a function of type $B \to A$, and g accordingly, gives us the composition $f \circ g$ of both functions of type $C \to A$.

This is as opposed to type-logical grammars [4], where composing functions is usually a multi-step process where extra hypotheses are introduced then discarded ($x : C$ in the following)

$$\dfrac{\dfrac{f : A/B \qquad \dfrac{g : B/C \qquad \dfrac{}{x : C}\,\mathrm{u}}{g(x) : B}\,/E}{f(g(x)) : A}\,/E}{\lambda x.f(g(x)) : A/C}\,/I^u$$

In pregroups, all of it takes place in only one step, hence we want our term-calculus to reflect this and made it so that some of the incoming abstracted variables end up abstracting the resulting term:

$$\dfrac{f(x)|x\rangle : AB^l \qquad g(y)|y\rangle : BC^l}{f(g(y))|y\rangle : AC^l}\,{}^lE$$

The consequence for the semantic layer is now that terms can combine as freely as their corresponding types do, as a functional term does not have to wait anymore for a non-functional term to be applied to it, but can combine with other functions to create new constituents.

$$\dfrac{\dfrac{\overset{the}{\imath(x)|x\rangle : \bar{n}n^l} \qquad \overset{green}{green(y)|y\rangle : nn^l}}{\imath(green(y))|y\rangle : \bar{n}n^l}\,{}^lE \qquad \overset{apple}{apple : n}}{\imath(green(apple)) : \bar{n}}\,{}^lE$$

$$\dfrac{\overset{the}{\imath(x)|x\rangle : \bar{n}n^l} \qquad \dfrac{\overset{green}{green(y)|y\rangle : nn^l} \qquad \overset{apple}{apple : n}}{green(apple) : n}\,{}^lE}{\imath(green(apple)) : \bar{n}}\,{}^lE$$

The nodes of the tree tell us something about the kind of constituent the concatenation of the lexical items at its leaves form and tell us also about its meaning, e.g.

the green:

- a noun phrase still looking for a noun to its right to be completed, just like *the* or *the very big beautiful*
- represents a single green entity predicate still looking for that entity which will come from the right.

Here we do not mean constituent as in the generative sense, but a more general notion of constituent: a string of lexical items is a constituent of syntactic category C with meaning s, when the lexical items can be combined together in order using inference rules to reach a pair of syntactic type and semantic term (s, C).

The same string of lexical items can form different constituents with different meanings:

$$\dfrac{\overset{flies}{plural(fly(x)|x\rangle) : n} \qquad \dfrac{\overset{in}{\langle y|(y)in(x)|x\rangle : n^r n\bar{n}^l} \qquad \dfrac{\overset{the}{\imath(x)|x\rangle : \bar{n}n^l} \qquad \overset{sky}{sky : n}}{\imath(sky) : \bar{n}}\,{}^lE}{\langle y|(y)in(\imath(sky)) : n^r n}\,{}^lE}{(plural(fly(x)|x\rangle)in(\imath(sky)) : n}\,{}^lE$$

$$\dfrac{\dfrac{\overset{flies}{\langle z|fly(z) : \pi^r s} \qquad \overset{in}{\langle y|(y)in(x)|x\rangle : s^r s\bar{n}^l}}{\langle z|(fly(z))in(x)|x\rangle : \pi^r s\bar{n}^l}\ {}^lE \qquad \dfrac{\overset{the}{\imath(x)|x\rangle : \bar{n}n^l} \qquad \overset{sky}{sky : n}}{\imath(sky) : \bar{n}}\ {}^lE}{\langle z|(fly(z))in(\imath(sky)) : \pi^r s}\ {}^lE$$

In the first case *flies in the sky* is a noun phrase with meaning that there are many flies in the sky, while in the second case, it is a sentence missing a subject, or verb phrase, with meaning that something unknown yet is flying in a sky.

The example above also exemplifies how our system can put constraints on the direction arguments are applied to eachother without help from the syntax. Being order sensitive also has other advantages that traditional categorial grammars do not have.

Take the transitive verb *like* for instance. Its semantic representation is now $\langle x|(x)like(y)|y\rangle$ – or *like* in η-reduced form. It is a truth predicate expecting two entities: one coming from the right and one coming from the left; as opposed to its λ counterpart $\lambda x.\lambda y.like(y,x)$ which takes both entities from the same side, its object coming first. This has the advantage that a subject could now be passed to the verb as soon as it is available, something that is not possible in categorial grammars without resorting to introducing and then eliminating hypotheses or adding some kind of permutation operator to the system.

$$\dfrac{\dfrac{\overset{John}{J : N}}{J : \pi} \qquad \overset{likes}{\langle x|(x)like(y)|y\rangle : \pi^r so^l}}{(J)like(y)|y\rangle : so^l}\ {}^rE$$

$$\dfrac{\dfrac{\overset{John}{J : N} \qquad \dfrac{\overset{likes}{\lambda x.\lambda y.like(y,x) : (N \setminus S)/N} \qquad \dfrac{}{u : N}\ \text{u}}{\lambda y.like(y,u) : N \setminus S}\ /E}{like(J,u) : S}\ \setminus E}{\lambda u.like(J,u) : S/N}\ /I^u$$

4.4 Untyped τ-Calculus

Even though the λ-calculus was originally an untyped system [6] and later came a version with types [7], it will not be possible in our case to have an untyped τ-calculus without some kind of modifications to the system. It is very easy to see why. For instance, consider the following terms:

$$a(x)(y)|y\rangle|x\rangle \qquad\qquad b(z)|z\rangle$$

And consider two of the possible terms resulting from applying the second to the first:

$$1.\ a(b(z))(y)\ |y\rangle\ |z\rangle \qquad\qquad 2.\ a(b(z)|z\rangle)(y)\ |y\rangle$$

Unless we know their types before the reduction, then we cannot decide which one is the right one. For suppose we have $a : \sigma\pi^l\phi^l$, $x : \phi$, $y : \pi$, $b : \phi\psi^l$, and $z : \psi$, then we will get:

$$\dfrac{a(x)(y)\ |y\rangle\ |x\rangle \in \mathbf{Term}_{\sigma\pi^l\phi^l} \qquad b(z)|z\rangle \in \mathbf{Term}_{\phi\psi^l}}{a(b(z))(y)\ |y\rangle\ |z\rangle \in \mathbf{Term}_{\sigma\pi^l\psi^l}}$$

which is a valid derivation, but

$$\frac{a(x)(y)\ |y\rangle\ |x\rangle \in \mathbf{Term}_{\sigma\pi^{\mathbf{l}}\phi^{\mathbf{l}}} \qquad b(z)|z\rangle \in \mathbf{Term}_{\phi\psi^{\mathbf{l}}}}{a(b(z)|z\rangle)(y)\ |y\rangle \in \mathbf{Term}_{\sigma\pi^{\mathbf{l}}\psi^{\mathbf{l}}}}$$

is not, as $a(b(z)|z\rangle)(y)\ |y\rangle$ is a term with type $\sigma\pi^l\phi^l\phi\psi^l\psi\psi^l\pi\pi^l \to \sigma\pi^l\psi^l\pi\pi^l$ which cannot be reduced to $\sigma\pi^l\psi^l$.

If instead we had the types $a : \sigma\pi^l(\phi\psi^l)^l$ and $x : \phi\psi^l$, then the term resulting from the application would be second term 2, and 1 would be of the wrong type.

5 Pregroup$_\tau$ Grammars

We give a brief overview of how to use this new semantic calculus in parallel with the syntax. For the most part, the way we do semantics will seem very similar to how it is usually done in type-logical grammars. We:

1. Introduce logical constants in our calculus, which will later on be interpreted in a "higher-order pregroup" model
2. Assign τ-terms to the lexical items in our dictionary as an addition to their syntactic types, i.e. pregroup types
3. Perform grammaticality check of a sentence by contracting the syntactic types according to the elimination rules we defined above
4. In parallel to the grammaticality checks, assemble the meaning of the string of words by putting together the different semantic terms in accordance with the application rules

Given a string of lexical items we aim to get a tuple $s : A$ corresponding to the grammatical category A of the expression and its meaning s.

5.1 Logical Constants & (Term, Type) Assignment

We will also use the usual logical connectives in our representations which we will treat as logical constants; we call them "logical" as their interpretation is the same in all models. Starting from the two basic types $\mathbf{e}$ for entities and $\mathbf{p}$ for propositions, we can define, for instance:

- $\neg$ is a constant of type $\mathbf{pp}^l$
- $\wedge$ is a constant of type $\sigma^r\sigma\sigma^l$, where σ is any type
- $\forall$ is a constant of type $\mathbf{p}(\mathbf{p}\sigma^l)^l$, where σ can be any type

From those we can define $\to$, $\vee$, $\exists$, $\leftrightarrow$ in the usual way, e.g. $\exists = \neg\forall(\neg P(x)|x\rangle)|P\rangle$. One can check that the resulting type is indeed right $(\mathbf{pp}^l)(\mathbf{p}(\mathbf{p}\sigma^l)^l)(\mathbf{pp}^l)(\mathbf{p}\sigma^l)\sigma\sigma^l(\mathbf{p}\sigma^l)^l = \mathbf{p}(\mathbf{p}\sigma^l)^l$.

We can also define other useful constants, such as $\imath$ the definite description operator (we follow [4]). First, we define $\exists^2 = \exists(P(x) \wedge Q(x)|x\rangle)|Q, P\rangle$, then $definite = \exists(\forall(P(x) \leftrightarrow x = y|x\rangle)|y\rangle)|P\rangle$, and then finally $\imath = \exists^2(P)(Q) \wedge definite(P)|Q, P\rangle$

Here are some example term and type assignments. As is tradition in pregroup grammars, we use lower case for syntactic types. We refer the reader to [12] for more explanation on the syntactic typing.

Here are some lexical items we will be using in our derivations and their term and type assignment. They should look familiar to most readers. They are written in non-η-reduced form for clarity.

Lexical item	Syntactic type	Semantic term
Antoine	n	A
loves	$\pi_3^r so^l$	$\langle x\|(x)love(y)\|y\rangle$
the	$\bar{n}n^l$	$\imath(x)\|x\rangle$
cat	n	$cat(x)\|x\rangle$
very	$nn^l(nn^l)^l$	$very(x)(y)\|y,x\rangle$
want (inf)	ij^l	$\langle x\|(x)want(y)\|y\rangle$
emphatic	$n(n)^l$	$empathic(x)\|x\rangle$
empathic	$(aa^l)^r nn^l$	$\langle y\|y(empathic(x))\|y\rangle$
to	$\bar{j}i^l$	$x\|x\rangle$
will	$\pi^r s_1 j^l$	$\langle x\|will((x)y)\|y\rangle$
will	$q_1 j^l \pi^l$	$will((x)y)\|y,x\rangle$
whom	$\bar{q}(q\hat{o}^l)^l$	$y(x)\|x,y\rangle$
every	$s(\pi_3^r s)^l n^l$	$\forall(y(z)\rightarrow x(z)\|z\rangle)\|x,y\rangle$
much	$\bar{n_0}n_0^l$	$much(x)\|x\rangle$
who	$n_2^r n_2(\pi_2^r s)^l$	$\langle y\|y\wedge(y)x\|x\rangle$

5.2 Semantic Derivations

We show some example sentence derivations in natural deduction forms.

5.2.1 Declarative sentence with transitive verb

$$\dfrac{\dfrac{\overset{\textit{Antoine}}{A:\bar{n}}}{A:\pi_3}\qquad\dfrac{\overset{\textit{likes}}{\langle x|(x)like(y)|y\rangle:\pi_3^r so^l}\qquad\dfrac{\overset{\textit{the}}{\imath(x)|x\rangle:\bar{n}n^l}}{\imath(x)|x\rangle:on^l}}{\langle x|(x)like(y)|y\rangle(\imath(x)|x\rangle):\pi_3^r sn^l}}{(A)\langle x|(x)like(y)|y\rangle(\imath(x)|x\rangle):sn^l}\quad\textcircled{1}$$

$$\dfrac{\dfrac{\overset{\textit{very}}{very(z)|z\rangle:nn^l(nn^l)^l}\qquad\overset{\textit{tall}}{tall(x)|x\rangle:nn^l}}{very(z)|z\rangle(tall(x)|x\rangle):nn^l}\qquad\overset{\textit{cat}}{cat:n}}{very(z)|z\rangle(tall(x)|x\rangle)(cat):n}\quad\textcircled{2}$$

$$\dfrac{\textcircled{1}\qquad\textcircled{2}}{(A)\langle x|(x)like(y)|y\rangle(\imath(x)|x\rangle)(very(z)|z\rangle(tall(x)|x\rangle)(cat)):s}$$

This last term reduces to

$$(A)\langle x|(x)like(y)|y\rangle(\imath(x)|x\rangle)(very(z)|z\rangle(tall(x)|x\rangle)(cat))$$

$$\rightarrow_\beta (A)like(\imath(x))|x\rangle(very(tall(x))|x\rangle(cat))$$

$$\rightarrow_\beta (A)like(\imath(x))|x\rangle(very(tall(cat)))$$

$$\rightarrow_\beta (A)like(\imath(very(tall(cat))))$$

The reader can check that there are as many orders of reducing this sentence and extracting its semantics as there are orders of doing the pregroup reductions alone; the semantic generation does not add more complexity to the parsing.

For the next examples, we will only show the reduced forms of the terms.

5.2.2 Interrogative sentence

$$\dfrac{\dfrac{\begin{array}{c}\mathit{who}\\ (y)x|y,x\rangle : \bar{q}(\pi_3^r s)^l\end{array} \qquad \dfrac{\begin{array}{c}\mathit{hid}\\ \langle x|\mathit{past}((x)\mathit{hide}(z))|z\rangle : \pi_3^r s_2 o^l\end{array}}{\langle x|\mathit{past}((x)\mathit{hide}(z))|z\rangle : \pi_3^r s o^l}}{\mathit{past}((y)\mathit{hide}(z))|z\rangle : \bar{q}o^l} \qquad \dfrac{\dfrac{\begin{array}{c}\mathit{the}\\ \imath(x)|x\rangle : \bar{n}n^l\end{array} \qquad \begin{array}{c}\mathit{chips}\\ \mathit{chips}(x)|x\rangle\end{array}}{\imath(\mathit{chips}(x)|x\rangle) : \bar{n}}}{\imath(\mathit{chips}(x)|x\rangle) : o}}{\mathit{past}((y)\mathit{hide}(\imath(\mathit{chips}(x)|x\rangle))) : \bar{q}}$$

5.2.3 Declarative sentence with quantifiers

Let's now look at a sentence involving quantifiers: *Somebody danced.*

$$\dfrac{\begin{array}{c}\mathit{somebody}\\ \exists(\mathit{person}(y) \to (y)x|y\rangle)|x\rangle : s(\pi_3^r s)^l\end{array} \qquad \dfrac{\begin{array}{c}\mathit{danced}\\ \langle x|\mathit{past}((x)\mathit{dance}) : \pi_3^r s_2\end{array}}{\langle x|\mathit{past}((x)\mathit{dance}) : \pi_3^r s}}{\exists(\mathit{person}(y) \to \mathit{past}((y)\mathit{dance})|y\rangle)) : s}$$

Our system also allows us to get multiple readings for quantified sentences such as *Everybody loves somebody*. Two different readings are possible for this one: 1) everybody has a special someone, and those special someones can all be different from one another or 2) there is one specific person who is loved by everybody.

$$\dfrac{\begin{array}{c}\mathit{everybody}\\ \forall((\mathit{person}(z) \to (z)x|z\rangle))|x\rangle : s(\pi_3^r s)^l\end{array} \qquad \dfrac{\begin{array}{c}\mathit{loves}\\ \langle x|(x)\mathit{love}(y)|y\rangle : \pi_3^r s o^l\end{array} \qquad \begin{array}{c}\mathit{somebody}\\ \langle z|\exists((\mathit{person}(y) \to z(y))|y\rangle) : (so^l)^r s\end{array}}{\langle x|\exists((\mathit{person}(y) \to (x)\mathit{love}(y))|y\rangle) : \pi_3^r s}}{\forall((\mathit{person}(z) \to \exists((\mathit{person}(y) \to (z)\mathit{love}(y))|y\rangle))|z\rangle) : s}$$

The term is simply a more detailed way of saying $\forall z.\exists y.(z)\mathit{love}(y)$. The second reading, $\exists x.\forall z(z)\mathit{love}(x)$, follows.

$$\dfrac{\dfrac{\begin{array}{c}\mathit{everybody}\\ \forall((\mathit{person}(z) \to (z)x|z\rangle))|x\rangle : s(\pi_3^r s)^l\end{array} \qquad \begin{array}{c}\mathit{loves}\\ \langle x|(x)\mathit{love}(y)|y\rangle : \pi_3^r s o^l\end{array}}{\forall((\mathit{person}(z) \to (z)\mathit{love}(y)|z\rangle))|y\rangle : so^l} \qquad \begin{array}{c}\mathit{somebody}\\ \langle z|\exists((\mathit{person}(x) \to z(x))|x\rangle) : (so^l)^r s\end{array}}{\exists((\mathit{person}(x) \to \forall((\mathit{person}(z) \to (z)\mathit{love}(x)|z\rangle)))|x\rangle) : s}$$

6 Conclusion

In this article, we proposed a novel way of doing semantics within pregroup grammars that is simple and shares many similarities with other semantic frameworks already in place. The calculus we created can be used at the same time as doing grammaticality checks through pregroup reductions and does not seem to be adding too much complexity to the overall system.

One of the down sides of our approach is that we had to restrict the format of pregroup types it could be used with, but as we explained, we feel that those types and relations left behind do not bear significant linguistic relevance.

This was a first step towards what we hope could become a standard way of doing pregroup grammars semantics and still have a lot of work ahead of us to investigate the system's properties in more depth. It is also not quite clear what those differences in directionality and reduction strategies will imply for the semantic analysis of sentences and how powerful the system really is.

References

[1] Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29(1):47–58, 1953.

[2] W. Buszkowski. Sequent systems for compact bilinear logic. *Mathematical Logic Quarterly*, 49(5):467–474, 2003.

[3] W. Buszkowski and A. Mickiewicz. Lambek calculus and substructural logics. *Linguistics Analysis*, 36(1):15–48, 2003.

[4] B. Carpenter. *Type-Logical Semantics*. MIT Press, 1998.

[5] C. Casadio and J. Lambek. A tale of four grammars. *Studia Logica*, 71, 2002.

[6] A. Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366.

[7] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940.

[8] S. Clark, B. Coecke, and M. Sadrzadeh. A compositional distributional model of meaning. In *Proceedings of the Second Quantum Interaction Synposium*, pages 133–140, 2008.

[9] B. Coecke, E. Grefenstette, and M. Sadrzadeh. Lambek vs. lambek: Functorial vector space semantics and string diagrams for lambek calculus. *Annals of Pure and Applied Logic*, 164(11):1079–1100, 2013.

[10] J. Lambek. The mathematics of sentence structure. *American Mathematics Monthly*, 65:154–169, 1958.

[11] J. Lambek. Type grammar revisited. In G. Perrier A. Lecomte, F. Lamarche, editor, *Logical Aspects of Computational Linguistics*, volume 1582 of *Lecture Notes in Computer Science*, pages 1–27. Springer Berlin Heidelberg, 1999.

[12] J. Lambek. *From Word to Sentence: A Computational Algebraic Approach to Grammar*. Polimetra, 2008.

[13] R. Moot and C. Rétoré. *The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics*, volume 6850 of *Theoretical Computer Science and General Issues*. Springer-Verlag Berlin Heidelberg, 2012.

[14] A. Preller. Category theoretic semantics for pregroup grammars. In *Logical Aspects of Computational Linguistics*, volume 3492 of *Lecture Notes in Computer Science*, pages 238–254. Springer Berlin Heidelberg, 2005.

[15] A. Preller. Toward discourse representation via pregroup grammars. *Journal og Logic, Language and Information*, 16(2):173–194, 2007.

[16] A. Preller and M. Sadrzadeh. Semantic vector models and functional models for pregroup grammars. *Journal of Logic, Language and Information*, 20(4):419–443, 2011.