Derivational Event Semantics for Pregroup Grammars

Gabriel Gaudreault

A Thesis

in

The Individualized Program

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Arts (Individualized Program - Classics, Languages, Linguistics)

Concordia University

Montreal, Quebec, Canada

October 10, 2016

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Gabriel Gaudreault

Entitled: Derivational Event Semantics in Pregroup Grammars

and submitted in partial fulfillment of the requirements for the degree of

Master of Arts (Individualized Program - Classics, Languages, Linguistics)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____ Chair

Deborah Dysart-Gale, Ph.D.

_____ Examiner

Charles Reiss, Ph.D.

_____ Examiner

Daniela Isac, Ph.D.

_____ Supervisor

Alan Bale, Ph.D.

Approved by _____ Graduate Program Director

Ketra Schmitt, Ph.D.

_____ 2016 _____ Dean of Graduate Studies

Paula Wood-Adams, Ph.D.

Abstract

Derivational Event Semantics for Pregroup Grammars

Gabriel Gaudreault

The focus of this research project is the development of a derivational system for event semantics over pregroup grammars. More concretely, it is shown how by extending the usual pregroup framework with a semantic layer and by assigning explicit event variables to the basic syntactic types of an expression, one can get semantic extraction from pregroup derivations without too many complications.

The resulting meaning is neo-davidsonian and conjunctivist in form, that is, the meaning is analysed in terms of events, and a single logical operator is used for meaning combination: the conjunction $\wedge$.

Using conjunctions as sole mean of meaning combination makes it harder at first to analyse certain constructions, but this is a small price to pay for the level of generality and overall derivational simplicity that is obtained in the end by equating syntactic combination — pregroup contractions — with meaning conjunction.

The issue of having non-functional types in this work is circumvented by using a unification operation over event and entity variables rather than abstraction/application operations à la λ-calculus, usually used in formal semantics.

Ackowledgements

Three years ago I left McGill University with a B.Sc. in mathematics and computer science to focus on the study of natural language structures. It's been an interesting ride, one in which I learned a lot about the field, got to present my work abroad, and met plenty of passionate and fascinating people. I can safely say that I got everything I wanted out of those years and have to thank all the people who have contributed to this to various degrees.

My program committee: Alan, Charles, and Dana. I introduced myself to you guys one month before the application deadline with a less than clear idea in mind of what linguistics was about. You still helped me to get my foot in and provided me with whatever resource I have needed since then. Although I may have overstayed my welcome by a year, you still allowed me to take my time and didn't pressure me needlessly. I wouldn't be as happy with the final product today if it had been different, I'm very grateful.

Ma famille. Bien que la question "Mais qu'est-ce que Gabriel fait?" semble avoir été une source continuelle d'interrogation ces 6 dernières années, vous avez toujours été là pour moi à m'offrir plus que votre support inconditionnel. Vous avez maintenant une partie de la réponse à la question entre vos mains, j'espère qu'elle saura vous satisfaire!

Joachim Lambek. Meeting you and having you supervise me in my second year of undergraduate studies definitely changed my academic path. You were a very generous man; your curiosity and your intellect was an inspiration to me. Without your coaching back then I would not be standing here today handing in a thesis on pregroup grammars! Thank you for everything. May you rest in peace.

Victoria. Your presence in the second half of those three years was so valuable to me. Your life experience — because you're old — and clear-headedness in more stressful moments was an inspiration and helped me so much in taking this work in the direction I intended.

Anyone I've met at conferences. I'm aware that probably none of them are going to read this, but they played such an important role. This group is divided in two. First, there were all those students whom I have met there, who showed me I wasn't alone in playing that game and that there were people interested in formal approaches to linguistics. Second, getting to meet all those researchers whose papers I'd previously read and having actual discussions with them, sometimes

even having them comment on my own posters/presentations, meant so much to me. I would end up coming back to Montréal with an enormous boost of motivation every time.

All those people who came over my apartment, saw the cryptic writings on the wall and asked apprehensively what those were about. Like the group I mentioned above, you're probably not going to read this, but your interrogations did help me in more than one ways. Trying to summarize in layman's terms what the hell the mysterious symbols were about was often harder than expected, and compelled me to ponder questions such as "Why should anyone care about those fancy diagrams and formulæ". Also, every time one of you would come up to me and say "Isn't this just the same stuff that was on your white board a month ago?" would be like a slap in the face and motivation to get up, erase the board, and do some work.

Coffee places. Once again, they are probably not going to read this, but I'm very grateful to all those coffee places I spent days at, for letting me sit on those chairs hours at a time staring at my laptop trying to get some work done. Your coffee fuelled me and your bright lighting at 11pm kept me awake.

Concordia University. None of this would have been possible without Concordia's immense support. They opened their doors to me and granted me total control over my field of study and research subjects. This also wouldn't have been feasible without the generous grants given to me by the School of Graduate Studies.

Contents

Chapter 1

Introduction

1.1 Introduction

The study of word and sentence meaning can be approached from multiple angles. One could be more interested in the specific meaning of natural language expressions and in questions such as

(1) What do people mean when they say the word *cat*?

or

(2) How is the meaning of the word *cat* related to that of the word *dog*?

Although it could also be of interest to investigate how to go from an abstract representation of an expressions' semantic values and get the more global meaning corresponding to the combination of those expressions, for instance when forming a sentence. This could raise questions of the sort:

(3) Given the meanings of expressions *the cat* and *does not like Paul*, what can we say about the meaning of the sentence *the cat does not like Paul* ?

In essence, this is what the focus of this present work will be: to show a way of deriving the semantic representation of an expression from its smaller parts. More precisely, this thesis will lay down a proposal for a new derivational semantics for Pregroup Grammars based on recent work in Event Semantics. It will be to show how the elements crucial for an event analysis can be encoded to work with the pregroup grammar framework.

The system that will be used to handle the meaning of the expressions and that will be built on top of the pregroup derivations, takes inspiration from the work of linguists such as Davidson (1967), Parsons (1990; 1988), Schein (1993; 2002), and Pietroski (2005; 2006). In particular, instead

of adopting a functional stance, such as is often the case in formal semantics, this work will follow the Conjunctivist approach to semantics developed, mainly, by Paul Pietroski (2003; 2005), where predicate conjunction is used as the sole means of semantic combination.

This section's purpose will be to describe briefly each of the different parts onto which will be built the final semantico-syntactic derivational system and highlight their key characteristics.

1.2 Overview of Event Semantics

Conjunctivism is the idea that as smaller expressions concatenate, their meanings simply conjoin. This approach breaks with the more dominant ways of doing semantics that originated with Frege (Frege, 1967). Traditionally, a greater interplay between the semantic values of the parts of an expression is assumed to take place: when two expressions combine, one is usually treated as the giver and the other one as the receiver. For instance, the verb phrase *likes Michael* could be thought of as an incomplete semantic relation between *Michael* and something to be filled. A value such as *the ghost* can then be passed to it as an input to fill that spot and stand for the thing that likes Michael.

Conjunctivism, on the other hand, relies on the assumption of hidden event and entity variable layers over which lexical expressions take scope. To understand why and how this idea came to be, it will first be important to have an understanding of two of the major developments of this branch of semantics: Davidson's characterization of action sentences in terms of events (1967) and Parsons' subatomic analysis of events (1990).

The traditional way of looking at a verb of action such as *kiss* is as a logical function that takes in two arguments – the subject and the object – and returns a truth value.

(4) $kiss(x, y) = \top \iff x \text{ kisses } y \iff (x, y) \in kiss_{ext}$

where $kiss_{ext}$ is the set of pairs of people kissing.

For instance, if there was a situation where John likes Mary but John does not like Paola, the extension of the verb — set of values — could be defined as

(5) $\text{kiss}_{ext} = \{(\text{J}, \text{M})\}$

The values of the following sentences could then be found by translating them into the appropriate logical form and checking if the conditions hold:

(6) $[\![John\ kisses\ Mary]\!] = kiss(J, M) = (J, M) \in kiss_{ext} = \top$

but

(7) $[\![John\ kisses\ Paola]\!] = kiss(J,P) = (J,P) \in kiss_{ext} = \bot$

Another possible analysis would be in terms of *events*:

(8) John kisses Mary $\iff$ there is an event at which John kisses Mary

i.e.

(9) $[\![John\ kisses\ Mary]\!] = \exists e.kiss(e,J,M)$

Doing things this way makes it much easier to deal with questions such as verb arity and sentential adjuncts, as constructions like temporal, locative and manner adjuncts can now be redefined as independent predicates over events. For instance,

(10) $[\![John\ danced\ yesterday\ at\ the\ ball]\!] = \exists e.danced(e,John) \wedge yesterday(e) \wedge Location(e, the\ ball)$

This turns out to be very useful when looking at analytical entailments, as the semantic representation of (10) now logically entails the one in (11), by simply removing one of (10)'s clause.

(11) $[\![John\ danced\ yesterday]\!] = \exists e.danced(e,John) \wedge yesterday(e)$

This kind of representation is not only limited to adjoints. Having the subject and object's semantic values be disconnected from the verb's is also possible through the introduction of predicates representing the thematic role they play in that event (Parsons, 1990).

(12) $[\![John\ kissed\ Julia]\!] = \exists e.Agent(e,John) \wedge kissed(e) \wedge Theme(e,Julia)$

1.3 Conjunctivist Approach to Event Analyses

It seems that a lot of a sentence's internal structure can be represented using the conjunction operator and hidden entity and event layers. One may then wonder if equating meaning combination with logical conjunction could cover enough ground to become an alternative theory to Frege's Functionalism. This is what Paul Pietroski set to investigate with Conjunctivism (Pietroski, 2005).

Pietroski's proposal is that the semantics of any expression in natural language consists of a finite conjunction of the meaning of its parts. Although this approach is a fairly recent one, it is based on work form logicians and semanticists such as Frege (1967) and Boolos (1984).

The crux of Pietroski's argument relies on plural quantification and on the use of plural variables to model plurality and quantification. Plural quantification is an interpretation of monadic second-order logic in which the monadic predicate variable is not interpreted as a set of things, but instead as taking multiple values. This is a subtle distinction, but with philosophical importance in this case (see section 5).

What this means concretely, is that it will be possible to translate a sentence such as (13) directly as (14) in a way that is coherent given a special interpretation provided for the plural logic.

(13) The boys dance

(14) $\exists E.\exists X.the_{agent}(E, X) \wedge boys(X) \wedge dance(E)$

The reason this is interesting is that this kind of sentence usually gets one of the following translations:

(15) a. $dance(Carl) \wedge dance(Paul) \wedge dance(Antoine) \wedge ...$

b. $\exists e_0 ... \exists e_n.dance(e_0) \wedge dance(e_1) \wedge ... \wedge agent(e_0, Carl) \wedge agent(e_1, Paul) \wedge ...$

c. $\forall x.boy(x) \rightarrow dance(x)$

Those are valid translations, but do not fit the conjunctivist framework.

1. (15a) does not show a clear division between the semantic information contributed by each word of the sentence: the semantic value of *dance* takes each of the boys' values as arguments, they are not disconnected and cannot be represented as independent conditions on the truth value of the sentence. The translation is also not as direct: it seems like the sentence that is being translated is *Carl dances and Paul dances and Antoine dances and ...* rather than *The boys dance*. Plus, *the*'s contribution to the representation is unclear.

2. In this case the semantic value of *dance* and of each of the boys is clearly delimited and can be represented as a conjunction of two complex clauses, but is not very efficient: how does one know how many events should appear in the sentence? This is totally independent of the number of words present in the sentence. Having direct access to the whole domain of boys is in this case also crucial, just like in the above case.

3. (15c) seems like a better translations than the other two, though it requires the use of two extra logical operations $\forall$ and $\rightarrow$. Derivationally speaking, their presence would require either introducing special combination rules to handle cases where expressions get combined

but where their meanings do not conjoin. This is usually handled by using functional semantic values, which is a more powerful and flexible approach, but requires more resource.

It will be shown later how a formula such as (14) can be interpreted as a series of conditions to be checked:

- Are all entites X's agents of some of the events E's?
- Are all entities X's boys?
- Are all events E's events of dancing?

This way one can get a nice correspondence between lexical items and their meanings: semantic values of expressions are independent of one another — they can be seen as blocks of information ranging over a common context — and combining expressions results in a conjunction of the semantic values of those expressions.

Pietroski's treatment of quantification will be somewhat different than the one described in the present work, as, for instance, the pregroup framework does not support movements and traces, and, has in general, very few lexical items with null phonology, if any. One of the main goals of this work is to show that an implementation is possible within the pregroup framework. Hence the focus will instead be on building one that fits the pregroup syntax.

1.4 Minimalist Interpretation

Conjunctivism is not only interesting from a technical point-of-view, but is also a biolinguistically motivated pursuit that has close ties with the Minimalist Program (Chomsky, 1995). In Pietroski 2008, Pietroski describes how it could be given this cognitive interpretation:

> *Open-class lexical items are instructions to fetch monadic concepts that may have been abstracted in the course of acquisition; and the meaning of a phrase is an instruction to build a conjunctive monadic concept from fetchable elements, given a few relational/thematic concepts and an operation of existential closure* (2008)

It is also interesting to look at parallels between the minimalist syntactic operations and the way information is shared across a conjunctivist analysis. For instance, *merge* (Chomsky, 1995) in its simplest form is usually described as an operation that forms a set out of two objects and assigns them a label dependent on the head, e.g.

$$\{\alpha, \beta\}_{l_1}$$

$$\alpha_{l_1} \quad \beta_{l_2}$$

A similar description could be given for the inner workings of conjunctive semantics, where the objects are then internal monadic concepts and the label stands for the variable they are predicating over. In this case though an extra condition is added during concatenation that forces labels to take the same value:

$$\{P, Q, x = y\}_x$$

$$P_x \quad Q_y$$

The interpretable features sent to LF during spell-out would then correspond to individual meaning predicates $P(x)$ and restrictions on the values of their arguments $x = y$. Movement could also be modeled under the economy of derivation principle, as being the process through which predicates can reach variables that emerge at a higher node in the tree and are not reachable from the predicate's original position.

1.5 Why Pregroup+Conjunctivism is a Happy Marriage

Pregroup grammars (Lambek, 2008) are descendents of Lambek's Syntactic Calculus (Lambek, 1958), where the grammatical categories assigned to expressions are formed recursively over a set of basic types or features. Expressions can then combine together when their corresponding types satisfy certain mathematical relations. In the case of pregroup grammars, a type (a) contracts on the right with its right-adjoint (a^r) and on the left with its left-adjoint (a^l). For instance, the determiner *the* combines with a noun phrase n on its right to form a determiner phrase, which is represented as (16). The final noun phrase type is the result of noun adjoint finding another noun to combine with.

(16)

the	cat	$\rightarrow$	the cat
$\bar{n}n^l$	n	$\rightarrow$	$\bar{n}$

One of the major inconveniences of using pregroup grammars to do semantics is that pregroup types are built *freely*, which is a fancier way of saying that they are strings of basic types. For instance, consider the possible types assigned to the subject position quantifier *every* in different grammatical formalisms:

1. Traditional Categorial Grammars (Carpenter, 1992): $(S/(N \setminus S))/N$
2. Minimalist Grammars (Stabler, 1997): `=N D -CASE`
3. Pregroup Grammars (Lambek, 2008): $s(\pi^r s)^l n^l$

In the first two cases, the order in which the types or features are used is well-defined and unique:

1. Type-elimination follows nestedness: The quantifier must be joined to a noun phrase on the right, then to a verb phrase on the right
2. Feature-checking is from left to right: The quantifier must be joined to a noun phrase, after which it could be used as a determiner and finally move by being selected by a higher node with a selectional case feature

On the other hand, pregroup types aren't ordered: any basic type present in a type could theoretically be contracted at any point if it appears on the edge of the type. For instance, from

(17)

John knows that	*every*	*boy*	*dances*
$s\bar{s}^l$	$s(\pi^r s)^l n^l$	n	$\pi^r s$

the next step of the derivation could either be 18 or 19

(18)

John knows that every	*boy*	*dances*
$s(\pi^r s)^l n^l$	n	$\pi^r s$

(19)

John knows that	*every boy*	*dances*
$s\bar{s}^l$	$s(\pi^r s)^l$	$\pi^r s$

where in the first case, *every* concatenates with the expression on its left, and in the second, with the one on its right.

The notion of constituents in pregroup grammar is very flexible: as long as lexical items can contract their types, the resulting expression is considered a constituent.

The very liberal type structure of pregroups is essentially the reason why traditional approaches to semantics do not work in that framework. For instance, consider the type of a finite transitive verb

(20)

kicked
$\pi^r s o^l$

where π corresponds to the subject and o to the object. In Montagovian semantics, such a verb would correspond to a relation between two entities, and would get assigned meaning:

(21)

$$\lambda x.\lambda y.kicked(y, x) : e \to e \to t$$

The order in which the subject and object get passed to the verb are very important, as a situation where I kick someone is very different from a situation where I get kicked by someone. But pregroup grammars cannot, in this sense, put constraints on which type gets contracted first.

The solution that will be investigated for this project will be to assign different variables to each basic type of an expression's type, so that when two expressions concatenate following a contraction of their types, their corresponding variables will unify, i.e. an equality constraint on those variables will be added to the semantics of the global expression.

1.6 Game Plan

To reiterate, the goal with this project is to show how an event analysis of natural language sentences could be approached from a derivational point-of-view using pregroup grammars as syntax. There will not be any fine analyses of natural language phenomena, but instead the aim will be to convince the reader that, structurally, the system is both powerful and simple enough to do interesting analyses if he/she were inclined to do so, and that it makes for a very natural semantic system for the pregroup framework.

An important principle that will be followed is that of simplicity, which is one of the strengths of Conjunctivism, along with

- Its intuitiveness: meanings conjoin as words combine
- Its descriptive power: capable enough to provide linguistic analyses for phenomena as complicated as quantification
- Its extreme compositionality: the meaning of the whole is the sum of the meanings of the parts, and the meaning of each part stays independent of whatever meaning gets conjoined to it

On the other hand, pregroup grammars are probably the ultimate categorial grammar in terms of simplicity and intuitiveness, and are also as powerful as any context-free formalism and thus powerful enough to do interesting linguistic analysis with. They are also a prime example of non-functional formal grammar which makes it even more interesting to try to define a semantics for them, as the main formal approach to semantics being functional fails in suitability.

The origins of categorial frameworks and their relations to areas of formal logic and mathematics will be looked at in the next section. It will be important to understand what makes pregroup grammars interesting and different from the majority of other formal grammar frameworks out there. Then the development of formal semantics, from Montague to Parsons, and to Pietroski, and what each brought to the table will be discussed. It will be shown that because of their non-functional types, pregroup grammars do not tend to provide for a good syntactic structure onto which one could do semantics, thus a different approach is in order.

The next chapter will then provide an in-depth description of a derivational system, based on the pregroup grammar types and derivation style, that allows one to start from event semantic representations of the meanings of individual expressions and build a full representation of the expression resulting from their concatenation. The final chapter will be a brief description of how to extend the underlying logic used in the meaning representation to account for plurality and quantification phenomena.

Chapter 2

Categorial Grammars

2.1 Categorial Grammars

A Categorial Grammar is a syntactic formalism where grammaticality judgements of expressions are obtained through formal derivations on the mathematical or logical types corresponding to each of its subparts. One of the attractive properties of those systems is that a great deal of linguistic coverage can be done using only a few derivation rules.

The original concept can be traced back to Edmond Husserl's Logical Investigations (Husserl, 1900), in which he described how words could be distributed into different classes, where words of the same class could be interchanged within context without affecting the coherency of the whole. This notion of coherency being a syntactic notion rather than semantic, words such as *cats* and *babies* could be said to be part of the same class, disjoint from the class containing *grey*.

(22) a. Those *cats* are great.

b. Those *babies* are great.

c. *Those *grey* are great.

(23) a. That shark is *grey*.

b. *That shark is *cats*.

c. *That shark is *babies*.

Husserl made the important distinction between what he called *syncategorematic* and *nominal* expressions, i.e. expressions that become meaningful only after completion by other expressions and those already meaningful on their own. A proposition like *at*, for instance, could be said to

be incomplete syntactically, as it cannot really stand on its own without requiring some sort of complement, such as a noun phrase. This distinction lies at the core of the categorial approach to syntax.

There is no set number of nominal expressions (or *basic categories*) and different branches of the categorial grammar family will often differ in what categories they use. For instance, Lambek's Syntactic Calculus (Lambek, 1958) often uses only two or three categories, that of nouns n, sentences s, and noun phrases np, whereas Pregroup Grammars (Lambek, 2008) usually have dozens of those types, along with various interrelations.

The first attempt to formalise Husserl's ideas was done by Ajdukiewicz (Ajdukiewicz, 1967), who translated Husserl's nominal expressions into basic categories and the syncategorematic ones into something close to a fraction of basic categories. A finite verb such as *dances* would be assigned a syntactic category

(24)
$$\frac{\text{sentence}}{\text{noun phrase}}$$

which correspond to the intuition that completing it with a noun phrase results in a sentence expression, e.g. *The cat dances*. Completion here is accomplished by aligning words next to each other in a sentence and, in a sense, taking the *product* of their types, just like in arithmetic.

(25)
$$\frac{A}{B} \times B = A = B \times \frac{A}{B}$$

(26)
$$\begin{array}{ccccccc} \text{The} & \text{cat} & \text{dances} & & & & \\ \frac{n}{n} & n & \frac{s}{n} & \rightarrow & n\ \frac{s}{n} & \rightarrow & s \end{array}$$

That gave him a nice computational system that could be used to test the grammaticality of simple sentences.

Fraction	Category
Product	String Concatenation
Numerator	Basic Category (Category after Completion)
Denominator	Basic Category (Complement)

Modifications (Bar-Hillel, 1953; Lambek, 1958) were later made to the system to account for the non-commutativity of sentencial constructions.

In 1958, Lambek (1958) published his first article on what he named the Syntactic Calculus, his version of Ajdukiewicz's calculus. His system was grounded in logic and greatly inspired by Gentzen's (1934) work on deductive systems. For instance, Gentzen's rules for logical implication were now reformulated in this non-commutative setting as introduction and elimination rules for left and right arrows

Gentzen	Lambek	
$\rightarrow$	$\backslash$	$/$
$\frac{\Delta, A, \Gamma \vdash B}{\Delta, \Gamma \vdash A \rightarrow B}$	$\frac{\Delta, A \vdash B}{\Delta \vdash A \backslash B}$	$\frac{A, \Delta \vdash B}{\Delta \vdash B / A}$
$\frac{\Delta \vdash A \quad \Gamma \vdash A \rightarrow B}{\Delta, \Gamma \vdash B}$	$\frac{\Delta \vdash A \quad \Gamma \vdash A \backslash B}{\Delta, \Gamma \vdash B}$	$\frac{\Gamma \vdash B / A \quad \Delta \vdash A}{\Gamma, \Delta \vdash B}$

The types were now not only non-commutative but were also recursively defined, which allowed for finer grammatical analyses.

For instance, the relative pronoun *who* would now get the type $(n \setminus n)/(n \setminus s)$ that better reflects its syntactic distribution and how it combines with a syncategorematic expression – a sentence lacking a subject – which was not possible in Ajdukiewicz's system.

(27)

The	*cat*	*who*	*likes*	*Edward*
n/n	n	$(n \setminus n)/(n \setminus s)$	$(n \setminus s)/n$	n

The logical derivation rules also allow us to deduce theorems that can then be interpreted as relations between grammatical types, such as type raising (28) and Geach's Law (29).

(28) $$B \rightarrow (A/B) \setminus A$$

(29) $$A/B \rightarrow (A/C)/(B/C)$$

Many variants of the original calculus have since then been developed, each answering different needs (Pregroup Grammars, Lambek 2008, Combinatory Categorial Grammars, Steedman 2000, Abstract Categorial Grammars, de Groote 2001). The categorial system that will be used in this project is a recent descendant of the original calculus called Pregroup Grammars, which was also developed by Lambek.

2.2 Pregroup Grammars

The categorial framework that will be used for this project is called Pregroup Grammars and is a recent descendant of the original syntactic calculus which arose from the study of resource

sensitive logics (Lambek, 1999, 2008; Buszkowski, 2003a). The syntactic types used in a pregroup grammar form what is called a *pregroup*, which is a special kind of algebra that is ordered and has non-commutative inverses.

A *pregroup* (Lambek, 2008) $\mathbb{P} = (P, \rightarrow, ^r, ^l, \cdot, 1)$ is a partially ordered monoid on a set of elements P, called the set of basic types. What this means is that every element $a \in P$ has a corresponding right and left adjoint — $a^r \in P$ and $a^l \in P$ respectively — subject to left and right contraction rules

$$a \cdot a^r \rightarrow 1 \qquad a^l \cdot a \rightarrow 1$$

and left and right expansion rules

$$1 \rightarrow a^r \cdot a \qquad 1 \rightarrow a \cdot a^l$$

These types also satisfy the following monoidal and partial ordering properties:

- Associativity:

$$(a \cdot b) \cdot c = a \cdot (b \cdot c), \text{ for any } a, b, c \in \mathbb{P}$$

- Identity:

$$a \cdot 1 = a = 1 \cdot a, \text{ for } a \in \mathbb{P}$$

- Reflexitivity:

$$a \rightarrow a, \text{ for } a \in \mathbb{P}$$

- Antisymmetry:

$$\frac{a \rightarrow b \quad b \rightarrow a}{a = b}, \text{ for any } a, b \in \mathbb{P}$$

- Transitivity:

$$\frac{a \rightarrow b \quad b \rightarrow c}{a \rightarrow c}, \text{ for any } a, b, c \in \mathbb{P}$$

The set of types closed under the ^{r} and ^{l} operations is called the set of simple types.

A pregroup grammar $\mathbb{G} = (\Sigma, P, \rightarrow, ^r, ^l, 1, \mathbb{T})$ 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 $\rightarrow$. This simply means that each element of our lexicon corresponds 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*. Here are some common basic types:

s: declarative sentences	s_2: declarative sentence in the past tense
g: gerund	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/nominative noun phrases	$\bar{\pi}$: nominative pronouns
o: objects/accusative noun phrases	$\bar{o}$: accusative pronouns
p: prepositional phrases	ϕ: quasi-sentence formed from infinitive

and an example derivation (30).

(30)
$$\begin{gathered} \textit{He likes her} \\ \pi_3 \cdot (\pi_3^r \cdot s \cdot o^l) \cdot o \\ \to \pi_3 \cdot \pi_3^r \cdot s \cdot o^l \cdot o \\ \to 1 \cdot s \cdot 1 \to s \end{gathered}$$

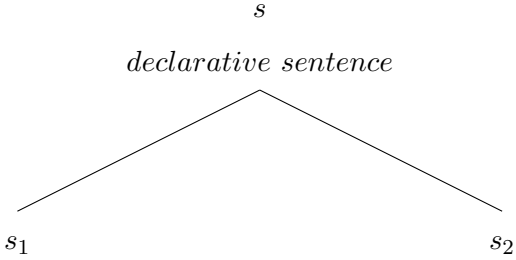
The fact that the structure is partially ordered also allows us to set a specific ordering of grammatical types such as the following.

$$\bar{n} \to \pi_3 \to \pi$$

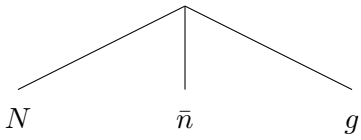
$$s_2 \to s \qquad s_1 \to s$$

where $a \to b$ means that a could also be used as b, e.g. a plural noun n_2 such as *cats* could be used as an object o or plural subject π_2, but not as a third person singular subject π_3.

It can help to think of some of those types in terms of conjunctive/disjunctive types or features



$$\pi_3$$
$$Name \vee Gerund \vee \ Noun\ Phrase \vee \ Small\ Clause \approx 3^{rd} PersonSubject$$



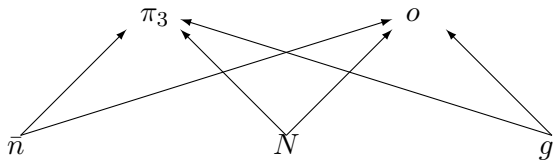
$$Name \quad NounPhrase \quad Gerund$$

and of ordering relations as conjunction of extra features

$$\begin{array}{ccc} \bar{n} & & o \\ & \rightarrow & \\ noun\ phrase & & noun\ phrase \wedge accusative\ case \end{array}$$

The types used here aren't definitive either, one could still get interesting results using, say, more basic types like $\{n, np, s\}$, or go full generative with $\{n, d, v, nom, wh, c, i, ...\}$.

Note that in general pregroups do not form a lattice. One of the reasons for that, is that meets and joins are not always uniquely defined. For instance, there is generally no real equivalent to a generative DP: there is a type for noun phrase $\bar{n}$, for proper nouns N, and for gerunds g, and also relations to the types for subjects π and objects o, but no type that would be the intermediate between the top and bottom layer.



Using the types we can now analyse various types of sentences, from simple ones like (31) to more complex ones like (32).

(31)
$$\begin{array}{ccccc} John & & likes & & Maria \\ N & & \pi^r so^l & & N \\ \rightarrow & \pi & & \pi^r so^l & o \\ & \rightarrow & so^l & & o \\ & & \rightarrow & s & \end{array}$$

(32)
$$\begin{array}{cccccccccc} & John & wants & for & the & cat & that & dogs & fear & to & live \\ & N & \pi_3^r s\phi^l & \phi\bar{j}^l o^l & \bar{n}n^l & n & n^r no^{ll} s^l & n_2 & \pi_3^r so^l & \bar{j}i^l & i \\ \rightarrow & \pi_3 & \pi_3^r s\phi^l & \phi\bar{j}^l o^l & on^l & n & n^r no^{ll} s^l & \pi_3 & \pi_3^r so^l & \bar{j}i^l & i \end{array}$$
$$\rightarrow \quad s\phi^l \quad \phi\bar{j}^l o^l \quad on^l \quad no^{ll} s^l \quad so^l \quad \bar{j}$$
$$\rightarrow \quad s\phi^l \quad \phi\bar{j}^l o^l \quad on^l \quad no^{ll} \quad o^l \quad \bar{j}$$
$$\rightarrow \quad s\phi^l \quad \phi\bar{j}^l o^l \quad on^l \quad n \quad \bar{j}$$
$$\rightarrow \quad s\phi^l \quad \phi\bar{j}^l o^l \quad o \quad \bar{j}$$
$$\rightarrow \quad s\phi^l \quad \phi\bar{j}^l o \quad \bar{j}$$
$$\rightarrow \quad s\phi^l \quad \phi$$
$$\rightarrow \quad s$$

Note the use of various syntactic ordering rules through the derivations, viz. $N \rightarrow \pi$, $N \rightarrow o$, and $\bar{n} \rightarrow o$. As pregroup types are merely concatenation of types, the order of contractions does not really matter and the latter derivation could have been derived more succinctly as in (33).

(33)
$$\begin{array}{cccccccccc} John & wants & for & the & cat & that & dogs & fear & to & live \\ N & \pi_3^r s\phi^l & \phi\bar{j}^l o^l & \bar{n}n^l & n & n^r no^{ll} s^l & n_2 & \pi_3^r so^l & \bar{j}i^l & i \end{array}$$
$$\rightarrow \pi_3\pi_3^r s\phi^l\phi\bar{j}^l o^l on^l nn^r no^{ll} s^l \pi_3\pi_3^r so^l \bar{j}i^l i$$
$$\rightarrow s\bar{j}^l n^l no^{ll} s^l so^l \bar{j} \rightarrow s\bar{j}^l o^{ll} o^l \bar{j} \rightarrow s\bar{j}^l \bar{j} \rightarrow s$$

What really matters in this kind of grammar are the derivation links that show us how the different lexical items combine with eachother in a given sentence.

(34)
$$\begin{array}{cccccccccccccc} A & & man & & will & & dance & & to & & save & & humanity \\ \bar{n} & n^l & n & \pi^r & s & i^l & i & i^r & i & i^l & i & o^l & n \end{array}$$
$$s$$

The arrows here represent contractions: the tip of the arrow points to the adjoint of the type appearing at the base of the arrow, with which it contracts. For instance, the first two words put together — *A man* — create an expression of syntactic category $\bar{n}$, as the adjoint n type of the determiner contracts with the n type of the noun.

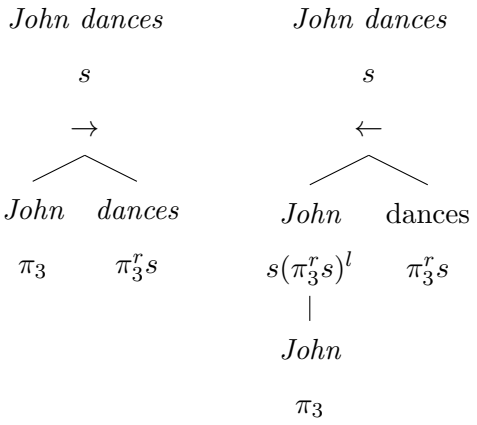
Those arrows can also be interpreted from a semantic point of view as information flow: the semantic values of *dance* and *save humanity* get related to each other by being passed along to the preposition *to*; one plays the role of an action or trigger and the other, the expected result of that action.

We can also derive many interesting relations using the pregroup properties, such as subject raising in (35), which was mentioned earlier.

(35) $\pi_3 \to \pi_3 \cdot 1 \to \pi_3 \cdot (\pi_3^r \cdot s) \cdot (\pi_3^r \cdot s)^l \to (\pi_3 \cdot \pi_3^r) \cdot s \cdot (\pi_3^r \cdot s)^l \to 1 \cdot s \cdot (\pi_3^r \cdot s)^l \to s \cdot (\pi_3^r \cdot s)^l$

This can interpreted as saying that an expression acting as a subject could be used as a sentence that is still left to be completed by a sentence lacking a subject, or VP. NP raising (Carpenter, 1998) is useful for trying to get a more generalized theory for NP typing, one that can help homogenize the types for noun phrases, generalized quantifiers, proper nouns, etc.

(36)



In the Montague framework, the terms would look something like (37) and could get derived directly from the typing rules of the simply typed λ-calculus.

(37) $A : e \vdash \lambda P.P(A) : (e \to t) \to t$

In general, most things that can be done in the syntactic calculus can be done in pregroup grammars, usually quicker; they also have a low computational complexity, lower than categorial grammars (Pentus, 2003). Although, in the end, both Lambek's original system and pregroup grammars are context-sensitive grammars (Lambek, 2008). One of the consequences of having a lower complexity is that it is sometimes harder to restrain derivations in pregroup grammars than in more traditional categorial grammars (Lambek, 2008); having less constrained derivations also makes pregroup parsing easier. Without going too much into details, let us have a look at the following derivations of the same sentence in each framework.

Faced with a sentence like (38), a syntactic calculus parser would normally have to wait until the very last word is known before starting the derivation, as *likes* requires its first argument to be an NP argument on its right, but then *the* needs to be completed by an N expression, and so on.

(38) John likes the big red dog

This is not only inconvenient, but adds a lot of complexity to the system both time-wise and memory-wise.

(39)

					red	*cat*	
				big	N/N	N	
			the	N/N	N		
		likes	NP/N	N			
He	$(NP \setminus S)/NP$	NP					
NP	$NP \setminus S$						
S							

On the other hand, as we have already seen above, pregroup contractions have a lot of flexibility on the order of contractions. This means that *he*'s π_3 type can contract with the verb as soon as the derivation starts. This then allows the verb phrase to contract its object adjoint type o^l with the noun phrase type $\bar{n}$ of *the*, right after this one has reduced to an object type o. The information carried from one contraction to another along the derivation is (in this case) a stack of no more than two types, viz. π_3, so^l, sn^l, s.

(40)

						red	*cat*
					big	nn^l	n
				the	nn^l	sn^l	
			likes	$\bar{n}n^l$	sn^l		
		He	$\pi_3^r so^l$	on^l			
		π_3	so^l	sn^l			
				s			

This makes constituent analysis more straightforward: any lexical items can simply be aligned next to each other and the contraction of their types tell you what constituent they form. For instance,

(41) John likes the big red

corresponds to a sentence looking for a noun on its right, sn^l. Finding that type in the syntactic calculus would require hypothetical reasoning: variables have to be introduced and then later discarded. A simple example is

(42) likes the

which has type $\pi_3^r sn^l$ corresponding to the concatenation of the two types. The corresponding derivation in the syntactic calculus would look something like this:

(43)
$$\dfrac{\dfrac{\overset{likes}{(NP \setminus S)/NP} \quad \dfrac{\overset{the}{NP/N} \quad \overline{N}}{NP}}{NP \setminus S}}{(NP \setminus S)/N}\, u-elim \qquad u-intro$$

which not only takes longer, but also places syntactic calculus parsing in a totally different complexity class than pregroup grammar parsing, which is more straightforward.

(44)
$$\dfrac{\overset{likes}{\pi_3^r so^l} \quad \overset{the}{on^l}}{\pi_3^r s}$$

For this reason, other categorial frameworks have included a composition operation

$$A/B * B/C \vdash A/C$$

as one of their main operation (Steedman, 2000).

2.3 Summary

The essential points to remember from this section are:

- Pregroup grammars — and categorial grammars in general — are grammatical formalisms that try to mathematically model the intuitive notion of syntactic incompleteness in natural language sentence construction.
- Traditional categorial grammars do this by introducing functional operators / and \, that behave in a way similar to fractions. Those operators are not interchangeable

  $$A \,/\, B \;\neq\; B \setminus A$$

  and the ways they can combine is very strict

  $$(A \setminus B) \,/\, C \;\neq\; A \setminus (B \,/\, C)$$

- The types in a pregroup grammar are also not interchangeable, nor commutative

  $$a \cdot b \;\neq\; b \cdot a$$

Rather than having fractional operators / and \ pregroup grammars instead make use of a left and a right type operator ^{r} and ^{l}, reminiscent of negative exponenents in arithmetic. This makes pregroup grammar types behave more like a list of types that can be attacked from any of its ends.

$$(a \cdot b) \cdot c \;=\; a \cdot (b \cdot c) \;=\; a \cdot b \cdot c$$

This could allow for the contraction of a transitive verb with a subject to happen before the verb contracts with its object, for instance. This could also be possible in traditional categorial grammars, but is a more complex operation that requires more resources.

Chapter 3

Formal Semantics

The purpose of this section is two-fold. First, the development of formal semantics will be discussed, starting from Montague's work in the 1970's (Montague, 1974; Partee, 1976). The key characteristics and advantages of using event layers to model action sentences will then be laid out and compared to the traditional approach. Conjunctivism, which is in essence a different flavour of event semantics, will also be introduced.

It will then be shown how event semantics could be modelled without too much work using a traditional categorial framework to handle the syntax. Finally, the question of doing formal semantics using pregroup grammars will be introduced.

3.1 Montague Semantics

Less than twenty years after Lambek's original publication, Richard Montague (Montague, 1974; Partee, 1976) started his own revolution in the field of semantics. He based his theory of formal semantics on notions from computability theory. More precisely, he made use of the λ-calculus, which was created originally as a general model of computable functions, to model semantic values of natural language expressions. As it will be shown, this area of formal logic is not completely disjoint from the one Lambek got inspiration from to create his own syntactic calculus.

Montague's system is based on the core notion of a function: everything can be considered a function and meanings are combined by passing one expression's semantic value as argument to another expression. For instance, the meaning of the verb *likes* now takes the form of a function that requires two arguments — two individuals — which are to be evaluated by the predicate $like(x, y)$ to test whether the first argument likes the second.

The semantic value of a lexical item in Montague's system is a pair $a : \alpha$ composed of a *term* a and a *type* α. On the term side, we find entities, logical predicates representing the concrete meanings of lexical items, and functions between those. Functions are represented as λ-abstractions:

$$\lambda x.f(x) : \text{this term takes an } x \text{ and passes it as an argument to } f$$

For instance, the truth value of (45) is found by passing *patrick* as an argument to $\lambda x.dance(x)$:

(45) Patrick dances

(46) $dance(Patrick)$

On the other hand, the types correspond to the domain from which the terms come, such as the set of entities $\mathbf{e}$, the set of truth values $\mathbf{t}$, or any set of functional types defined recursively on those types.

$$types\ \alpha,\ \beta\ :=\ \mathbf{e} \mid \mathbf{t} \mid \alpha \to \beta$$

Terms and types are related in the following way:

$$\lambda x.g : \alpha \to \beta \text{ , if } g \text{ is a term of type } \beta \text{ and } x \text{ a variable of type } \alpha$$
$$f(a) : \beta, \text{ if } f \text{ is a term of type } \alpha \to \beta \text{ and } a \text{ a term of type } \alpha$$

For instance, the function $\lambda x.mother\_of(x) : \mathbf{e} \to \mathbf{e}$ takes an individual to its mother, another individual.

(47) $$mother\_of(Georges) = Barbara \iff \text{Barbara is Georges' mother}$$

and the function $\lambda x.dance(x) : \mathbf{e} \to \mathbf{t}$ takes an individual and gives back a truth-value in return corresponding to whether or not the individual dances.

(48) $$dance(John) = \top \iff \text{John dances}$$

The meaning of a sentence is now represented as multiple higher-order logical predicates nested within one another, e.g. the logical representation of the sentence *The big black dog met John in Seattle* would be something like

(49) $$in(Seattle, \lambda x.meet(x, John), the(big(black(dog))))$$

Later, work by van Benthem (van Benthem, 1991) showed that this system could also be put in close correspondence with Lambek's syntactic system for a dual semantic and syntactic analysis of expressions.

Lambek's Syntactic Calculus	Montague Semantics
Grammaticality Testing	Semantic Extraction
Syntactic Types	Typed λ-Terms
Basic Syntactic Categories: n, s, ...	Basic Semantic Types: **e**, **t**,...
$\backslash$/-Elimination	Function Application + $\rightarrow$-Elimination
$\backslash$/-Introduction	Function Abstraction + $\rightarrow$-Introduction

To the /-Elimination rule for instance

$$\frac{\Gamma \vdash A/B \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A}$$

corresponds the functional application rule, hence we can rewrite them together as:

$$\frac{\Gamma \vdash \lambda x.f(x) : A/B \qquad \Delta \vdash a : B}{\Gamma, \Delta \vdash f(a) : A}$$

Montague's system and its use in categorial grammars would become very powerful and important tools for future semanticists.

3.2 Event Semantics

Around the same years as Montague was laying down his functional semantics, important work was done by Donald Davidson and other linguists (Davidson, 1967; Vendler, 1957; Castañeda, 1967) on the semantics of action sentences and the study of the classification of verbs. These notions can also be put together with Montague's framework to provide for finer formal semantic analyses, which is the way event semantics will be approached in this section.

One of the most important distinctions when classifying verb classes is that of eventive versus stative verbs.

A sentence such as

(50) *John danced at the ball*

could be said to be making a claim about the existence of an event which John danced at and which took place at the ball. Events are usually thought of as *activities* and verbs as descriptions of those events.

On the other hand, it does not make as much sense to analyse the sentence

(51) *John knows who killed JFK*

as an event or activity in which John knew something. Saying that John is in a state of knowing something seems more natural. States are something people or things are *in*, whereas events something that is done. Most verbs fit more naturally in one category than in the other, and many more divisions are possible than just these two classes.

Davidson's insight was to posit a new semantic type, that of *events*, which is something that verbs take as argument in addition to whatever internal and external arguments they take.

(52) $[\![John\ kicked\ Henry]\!] = \exists e.kick(e, John, Henry)$

Using this event, modifiers can then, be represented as predicates conjoined to the verb and taking scope over the event. This is similar to the way adjectives can be joined to nouns and take scope over an entity. For instance, (53) describes an entity that is a dog and is passionate, whereas the adverb *passionately* in the sentence (54) describes an event of dancing that is passionate.

(53) $[\![\ a\ passionate\ dog\ ]\!] = \exists x.Indefinite(x) \wedge passion(x) \wedge dog(x)$

(54) $[\![\ John\ dances\ passionately\ ]\!] = \exists e.dance(e, John) \wedge passion(e)$

This can be extended to an analysis of almost any kind of modifiers such as locative, modal or temporal ones.

(55) $[\![John\ kissed\ Maria\ in\ Chicago]\!] = \exists e.kiss(e, John, Maria) \wedge Loc(e, Chicago)$

This also has an enormous advantage over other analyses that would interpret, for instance, the adjunct as a modifier of the subject (Carpenter, 1998).

(56) $kiss(John, Maria) \wedge in(Chicago, John)$

The reason is that not having a specific variable to refer to the event gives us too much room to play around with the predicates and get false entailments. We would not want to allow the following entailment:

$$\frac{\textit{John kisses Maria in Chicago} \qquad \textit{John punches Barry on the nose}}{\textit{John punches Barry in Chicago}}$$

which could be derived, for instance, from a naive translation of the sentences as in (57).

(57) a. $[\![\ John\ kisses\ Maria\ in\ Chicago\ ]\!] = kiss(John, Maria) \wedge Loc(Chicago)$

b. $[\![\ John\ punches\ Barry\ on\ the\ nose\ ]\!] = punch(John, Barry) \wedge Loc(Definite(nose))$

$$\frac{kiss(John, Maria) \wedge Loc(Chicago) \wedge punch(John, Barry) \wedge Loc(Definite(nose))}{Loc(Chicago) \wedge punch(John, Barry)}$$

Knowing that John kissed Maria in Chicago and that he punched Barry on the nose shouldn't allow us to infer that he punched Barry in Chicago. Maybe he only punched him in Detroit. That same entailment would not be possible in event semantics, as now the two verbs introduce two different events that cannot be fused: positing the existence of an event, and then of another one, does not imply that those two events are the same.

$$\frac{\exists e.kiss(e, John, Maria) \wedge Loc(e, Chicago) \qquad \exists e'.kiss(e', John, Barry) \wedge Loc(e', nose)}{\dfrac{\textbf{Does not entail}}{\exists e.punch(John, Barry) \wedge Loc(e, Chicago)}}$$

Modelling possible entailment relations is also more easily done once one starts treating adjuncts as distinct predicates bound by the same variable.

(58) a. John pinched Sarah

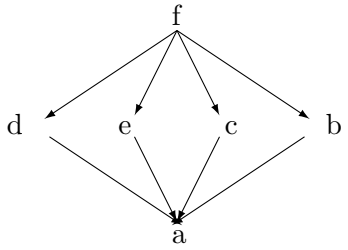
b. John pinched Sarah intensely

c. John pinched Sarah in the afternoon

d. John pinched Sarah when she wore that dress

e. John pinched Sarah at school

f. John pinched Sarah intensely at school in the afternoon when she wore that dress



Sentences (b) to (f) all entail sentence (a), and sentence (f) entails every other sentences. The fact that an adjunct like *at school* is now treated simply as a truth predicate $at(e, school)$ makes it easy to discard it and get a more general reading, e.g.

(59) $pinched(e, John, Sarah) \wedge at(e, school) \vdash pinched(e, John, Sarah)$

Notice also that the truth of sentences (b) and (c) would not, for instance, entail the truth of

(60) John pinched Sarah intensely in the afternoon

as the events specified in (b) and (c) could be distinct; first-order logic requires fresh variables to be used when doing ∃-elimination.

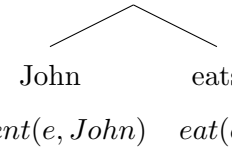
A major modification that was made to davidsonian semantics was Parsons' subatomic semantic approach (Parsons, 1988, 1989, 1990) to sentential structure, which decomposed even further the predicates' structure by making the way arguments combine in meaning more similar to the way adjuncts do. In his framework, a verb is now associated with a monadic predicate ranging over events only, and its external and possible internal arguments are related to it via the thematic role they play in the event.

(61) $[\![John\ kissed\ Cindy\ passionately]\!]$
$= \exists e.Agent(e, John) \wedge kiss(e) \wedge Past(e) \wedge Theme(e, Cindy) \wedge Passion(e)$

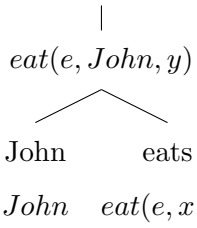
Here $Agent(e, John)$ can be read as *John is one of the possibly multiple agents of the event e.*

Reducing predicates' arities by eliminating the slot usually assigned to complements can help to account for polyvalence of some verbs and to simplify some entailments by not requiring a hidden existential closure operation on the missing argument.

(62) a. $Agent(e, John) \wedge eat(e)$



b. $\exists y.eat(e, John, y)$



One gets directly a complete semantic value without having to say anything about the missing object.

Not only can thematic relations be encoded as their own predicates, but things like tense and aspect can also get finer analyses by being cut up in smaller parts. The example below shows that by letting the auxiliary introduce an extra event variable, one can model the meaning of the past perfect tense: there was an event of kissing, John was the agent, Maria the patient, and there is a time at which the event was completed, and that event was in the past — it took place before now.

(63) $[\![\ John\ had\ kissed\ Maria\ ]\!]$
$= \exists e.\exists e'.Agent(e, John) \wedge kiss(e) \wedge Patient(e, Maria) \wedge e' < now \wedge Cul(e, e')$

3.3 Conjunctivism

Another development in event semantics that will be important for this project comes from Paul Pietroski (Pietroski, 2003, 2005). His approach is called *Conjunctivism* as it stands in opposition to semantic functionalism and the idea that *everything is a function* where lexical items are passed to one another as function arguments. Instead, meanings are combined together by using conjunctions and taking scope over the same variables.

There will not be much difference in form between the two approaches in the case of simple sentences such as

(64) $[\![John\ dances]\!] = \exists e.Agent(e, John) \wedge dance(e) \wedge Present(e)$

Although there will be for more complex sentence such as

(65) John kissed every girl

which is usually interpreted in event semantics as

(66) $\forall x.girl(x) \rightarrow \exists e.Agent(e, John) \wedge kissed(e) \wedge Patient(e, x)$

i.e. to every girl corresponds a liking event in which she is the theme and John the agent, and not as a conjunction of all predicates:

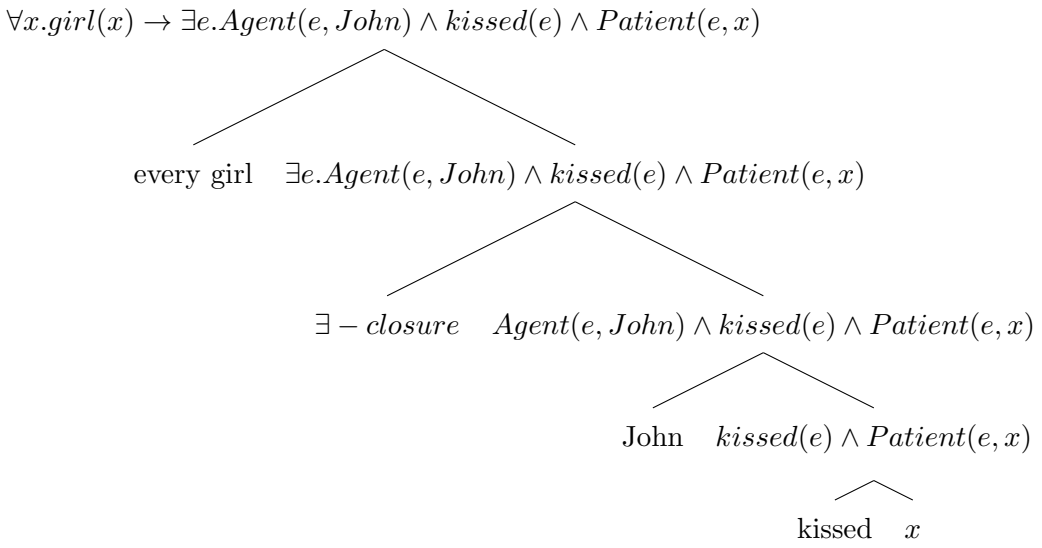
(67) $\exists e.\exists x.Agent(e, John) \wedge kissed(e) \wedge every(x) \wedge girl(x) \wedge Patient(e, x)$

which seems to be saying that there was a kissing event, where John was the agent, and there was a girl who was *every* and was also a patient.

Leaving aside the issue of modelling such a sentence using only conjunctions, let us have a look at the way form (66) could have been derived compositionally from its constituents.

The way this is usually approached is through an assignment of functional meaning predicates to every lexical item — similar to the way it is done in Montagovian semantics, but with different values — and then combining them in a certain way that is dictated by the syntax. For a sentence with quantifier, such as the one above, the process usually involves the use of quantifier raising

(Champollion, 2010, 2015; de Groote and Winter, 2015a), by which the DP with quantifier leaves a trace in the tree, then moves upward and takes scope over the whole expression, trace included.



(This derivation was simplified to get the general idea.)

Without movement, the derivation would output a form like

(68) $\exists e.Agent(e, John) \wedge kissed(e) \wedge (\forall x.girl(x) \rightarrow Patient(e, x))$

which describes a kissing event where John kissed every girl. But what if no girl got kissed? Then (65) is vacuously true, but (68) still requires the existence of a kissing event where John is the kisser to be true (as all parts of a conjunctive formula must be satisfied).

In the tree above, possibly half the combinations of two branches could be represented as some kind of conjunction of their values, e.g. *John*'s semantic value — $Agent(e, John)$ — can be conjoined with *kissed x*'s value — $kissed(e) \wedge Patient(e, x)$ — to form $Agent(e, John) \wedge kissed(e) \wedge Patient(e, x)$. Although this is usually modelled as a function taking in another function as argument

$$\begin{array}{cc} John & kissed\ x \\ \lambda e.Agent(e, John) & \lambda Q.\lambda e.kissed(e) \wedge Patient(e, x) \wedge Q(e) \end{array}$$

$$\rightarrow \quad \begin{array}{c} John\ kissed\ x \\ \lambda e.kissed(e) \wedge Patient(e, x) \wedge Agent(e, John) \end{array}$$

The other two term combination operations cannot be as simply represented as conjunctions of terms. The existential-closure operation takes the unbound event from the open sentence and

binds it with an existential event, whereas the quantified noun phrase *every girl* requires a more complex value, such as (69), which binds the entity value x from the trace in an appropriate way.

(69) $\lambda P.\lambda Q.\forall x.P(x) \rightarrow Q(x)$

The important point here is that modelling semantic values as functions is a very powerful tool as it allows a functional term to handle the semantic of its arguments in almost any way it decides to: it can conjoin it, bind it, and so on.

Another important point is that in this kind of framework, the application of existential-closure is a very sensitive task and is required to happen at a very specific point of the derivation.

For instance, it would be wrong to bind the event after quantification has happened, and interpret the sentence such as (70) as (71) as it would imply the existence of one event at which John kissed every girl. It would also evaluate a situation where John kissed half the girls on Monday and the other half on tuesday as false — although there is an interpretation of that variable under which the formula could make sense, as it will be shown later.

(70) John kissed every girl

(71) $\exists e.\forall x.girl(x) \rightarrow Agent(e, John) \wedge kissed(e) \wedge Patient(e, x)$

Instead, the right form should probably have the universal quantifier take a wider scope than the existential quantifier.

Similarly, negative sentences require that the negation operator have a wider scope than existential-closure. For instance,

(72) ⟦ No cat danced ⟧

a. $\exists e.\neg\exists x.Agent(e, x) \wedge cat(x) \wedge dance(e)$

b. $\neg\exists x.\exists e.Agent(e, x) \wedge cat(x) \wedge dance(e)$

Form (72a), that one would get by taking scope over the whole expression, seems wrong, as it expresses the existence of an event, one at which no cat danced, which is pretty easy to find if there ever was a time at which no cat danced. Although, by uttering *no cat danced* what one is really doing is denying the existence of an event, one at which some cat would have danced. On the other hand, form (72b) sounds just about right: there is no entity and no event such that this entity was a cat and was dancing.

The system that will be used to do semantics in this project will follow a different approach and assume that meanings of the parts all contribute essentially in the same way to the semantics of

an expression by being conjoined to each other. This approach is much more restricted as it does not support function abstraction of the sort we saw above.

The key component of this approach is the use of second-order variables and plural predicates, that will not only be useful to work with quantifiers, but for any expression involving plurality. The idea was originally developed by Frege (1967) and Boolos (1984), and then pushed forward by other semanticists such as Schein (1993) and, more recently, Pietroski (2005).

One of the main arguments for this approach is that natural language possesses two distinct kinds of logical quantifiers, one that is singular, e.g. (73), that corresponds to the usual first-order universal quantifier $\forall x$, but also one that is plural, that can be found in sentences such as (74), and for which it is hard to find a nice natural first-order interpretation.

(73) There is a cake on the table

(74) There are cakes on the table

These cases, it is argued, can be handled using plural variables, by assigning the logical form (75) to the sentence above.

(75) $\exists X.cake(X) \wedge Plur(X) \wedge \exists y.table(y) \wedge on(X, y)$

where X is a *plural*, or *second-order*, variable, i.e. it stands for more than one value.

Plural quantification is a valid interpretation of second-order logic (Pietroski, 2003) and allows us to naturally translate a sentence like (76) as (77).

(76) Two firemen kicked John

(77) $\exists E.\exists X.2(X) \wedge Agent(E, X) \wedge fireman(X) \wedge kick(E) \wedge Patient(E, John)$

which is read as:

- There is a plural event and a plural entity
- There are at least two values for the entity
- The values of the entity are agents of the values of the event
- The values of the entity are the firemen
- The values of the event are the kicking events
- John is a patient for the values of the event

We will refer to the *values of an entity* as *entities* from now on for conciseness, but it is important to understand that even though we are referring to them as a collection, we do not usually evaluate them as such. It does not make much sense to ask if a collection of three people is a $fireman$; each person in this collection has to be evaluated individually.

One of the most important and interesting aspect of this formalisation will be that it can model quantification as a relation between the two plural variables (event and entity).

(78) $[\![\text{ Every child cried }]\!] = \exists E.\exists X.every(E,X) \wedge Agent(E,X) \wedge child(X) \wedge cried(E)$

At first this might seem wrong: how could *every* test whether the relation between *child* and *cried* is true if it can only access the event E? This is where the magic of plural variables comes into action. Under a Conjunctivist translation, every lexical item corresponds to a truth predicate that restrains the set of possible values the entities and events could take. In this case, $Agent(E,X)$ makes sure that the agents of the events are children, whereas $every(E,X)$'s job is to check that all those children are actually acting as agents in some event. Note that this is slightly different than Pietroski's interpretation of values of verb phrases as $Frege\text{-}pairs$.

Now, letting E be a crying event where $jamie$ and $dorothy$ are agents, and X have values $jamie$ and $dorothy$, in a world where the things that cry consist of $dorothy$, $sandy$, $jamie$, and $jordan$ and the children $jamie$ and $dorothy$, makes the sentence true, as every predicate can be evaluated to hold under that assignment of values:

- $every(E,X)$: Is every value in X an agent of E? Yes
- $Agent(E,X)$: Is every agent of E an element of X ? Yes
- $child(X)$: Is X comprised of all and only the children? Yes
- $cried(E)$: Are the events crying events? Yes

3.4 Categorial Event Semantics

There has been work done on bridging the gap between the event and montagovian approaches to semantics (Champollion, 2010, 2015), even sometimes using some kind of categorial grammars as syntax (de Groote and Winter, 2015a; Winter and Zwarts, 2011a). The way things usually work is that the set of basic types gets extended, from entities and booleans $\{e,t\}$ to entities, booleans and events $\{e,t,v\}$, and new covert lexical items are added to the lexicon, e.g. $\exists$-closure, *agent*

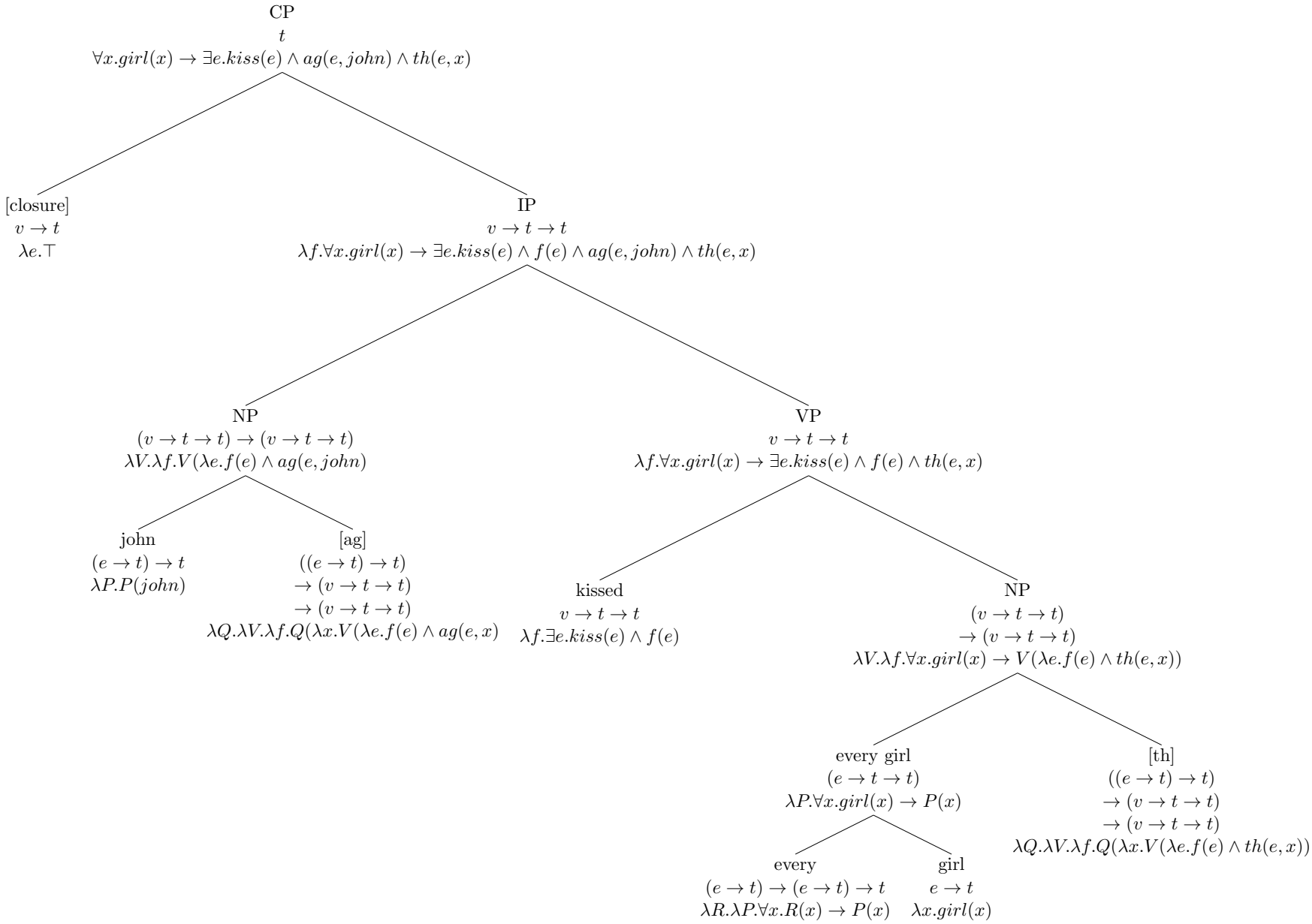


Figure 3.1: Quantification using event variables

and *theme*. Figure 3.1 is an example derivation using generative syntax of the quantified sentence *John kissed every girl*, taken from Champollion 2015.

The role of the existential-closure in this case is similar to the previous examples, though it is implemented somewhat differently. Throughout the derivation, extra conditions can be passed to the consequent of the semantic term of nodes like IP, VP and NP (second) — by substituting for the f variable — even after the event gets closed (VP node), which is very useful. The role of the existential-closure is then to stop the addition of new conditions to the semantic value of the IP, rather than actually closing the event variable, which is already closed.

3.5 Pregroup Semantics

Different methods have been developed to do meaning extraction of a sentence using the pregroup framework, but those methods are quite different from the traditional approach of labelling syntactic types with λ-terms that was described above, in that, they are not functional, i.e. they don't treat the types as corresponding to functions of some kind.

Pregroup grammars being defined *freely* on a set of elements forming a pregroup means that the types given to a specific lexical item form, in a way, just a string, or word, of simple types and no a priori relation is assumed to hold between them. That is a problem that Lambek-Van-Benthem grammars did not have to face.

Indeed, the translation from

$$(A \setminus B) \,/\, C$$

to

$$C \to A \to B$$

is pretty intuitive, as looking at the first type tells you that the first and only thing it can be cancelled with would be a term of type C, which would give the type $A \setminus B$, which can only be cancelled with a term of type A, to give a term of type B. In some ways, the only difference between the two types is the fact that the first ones care about the direction of its incoming arguments; it is non-commutative.

This very important characteristics allows one to assign λ-terms — which were developed originally to formalise the notion of a $function$ — to lexical items in order to handle the semantics of a sentence.

(79)
$$\dfrac{\begin{array}{cc} Mark & \dfrac{\begin{array}{cc} likes & Paola \\ \lambda x.y.like(y,x) : (N \setminus S)/N & P : N \end{array}}{\lambda y.like(y,P) : N \setminus S} \\ M : N & \end{array}}{like(M,P) : S}$$

In this case, the semantic type assigned to $like(x, y)$ is $e \to e \to t$, which is a function from individuals to a function from individuals to truth values.

Pregroup types are not as nice to work with in that sense as adjoint types can appear in a complex type in the same positions a non-adjoint type would, and contraction of types can be done from right or left in no particular order. Types such as $A^l B^r$ and $AB^{rr}CD^l E$ are as legal as more straightforward ones like $A^r B$ or BC^l.

Even trying to think of adjoints as some sorts of functions does not clarify the question: How should information travels within a type such as $AB^r C$? Which type corresponds to the argument? Which type is the $final$ one, the one left after completion by other types?

For instance the example above can be derived in two ways using pregroup types:

(80)
$$\dfrac{\begin{array}{ccc} Mark & likes & Paola \\ \dfrac{N}{\pi_3} & \dfrac{\pi_3^r s o^l}{\pi_3^r s} & \dfrac{N}{o} \end{array}}{s}$$

(81)
$$\dfrac{\begin{array}{ccc} Mark & likes & \\ \dfrac{N}{\pi_3} & \dfrac{\pi_3^r s o^l}{s o^l} & \dfrac{Paola}{\dfrac{N}{o}} \end{array}}{s}$$

One way of using functional types would be to have two different ones and to drop associativity:

$$\lambda x.y.like(y,x) : (\pi_3^r s) o^l \qquad \lambda x.y.like(x,y) : \pi_3^r (s o^l)$$

but this just goes against everything that makes pregroups special and interesting, and turns them into a notational variant of the syntactic calculus. Another possible way would be to change the mode of combination between terms and types to function composition and to introduce two classes of abstractions, those who take arguments from the left, and those who do from the right (Gaudreault, 2015).

Other reasons that makes a functional semantic for pregroup types not very attractive are these properties:

$$(a_n \cdot ... \cdot a_1)^l = a_1^l \cdot ... \cdot a_n^l \qquad (a_n \cdot ... \cdot a_1)^r = a_1^r \cdot ... \cdot a_n^r$$

$$(a^l)^r = a = (a^r)^l$$

which can be used in some cases to alter the type of quantifiers : $s(\pi_3^r s)^l = s s^l \pi_3^{rl} = s s^l \pi_3$. The

problem here is that adjoints — inverses in traditional categorial grammars — usually correspond to variable abstraction, and non-adjoint types to values, whereas we are now saying that a double-adjoint is also a non-adjoint type. it is not clear what values these should take. Also, how would one know looking at a type ab^lc that it is meant to be read as $(ab^l)c$ and not $a(b^lc)$, i.e. that the final completed type should be a and not c?

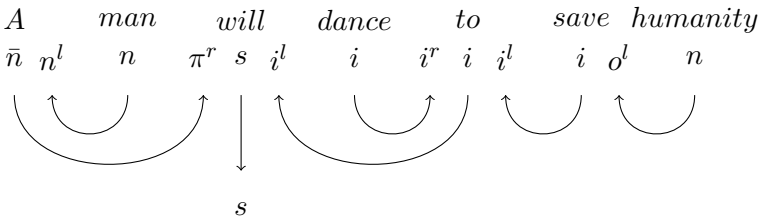
The standard semantics for pregroup grammars comes from the work of Anne Preller (2005; 2007). It takes advantage of the geometrical nature of pregroup derivations to extract the argument structure of a sentence by looking at its contraction links. Other ways are also possible (Clark et al., 2008; Coecke et al., 2013), though they will not be of relevance to our work.

The general idea is that one can assign logical predicates as semantic values to expressions, and associate the variables present in these predicates to some of the adjoint types contained in the expression's type. Then, after a contraction, the semantic value of of the expression with the non-adjoint type gets passed to the expression with the adjoint type, and replaces the value of the variable the adjoint type corresponds to.

For instance, *John* could get assigned the syntactic type π_3 with semantic value *John*. Then, when placed next to *cries*, which has semantic value $cry(x)$ and type $\pi_3^r s$, with x is linked to π_3, *John*'s type π_3 would contract with *cries*'s adjoint type π_3^r , and so its semantic value, *John*, would get passed to *cries*'s semantic value to replace the occurrence of x. Hence the final semantic value would be $cry(John)$, with type s.

Here is a lengthier example that was discussed previously

(82)



whose logical translation would be:

(83) will(a(man), to(dance,save(humanity)))

This translation is then interpreted either in a compact closed category with product or in a fashion similar to the Discourse Representation of Kamp-Reyle (1993), where meaning postulates are used to interpret the predicates. The basic semantic types she uses, or sorts, are similar to the ones that will be used in this project, i.e. boolean, individual, and set of individual types.

Chapter 4

Non-Functional Event Semantic Analysis

In this section, it will be shown how one can use the implicit event variables instantiated by lexical item's corresponding meaning predicates to derive the right neo-davidsonian representation of an expression. The idea is that those variables can be turned into explicit objects that can be unified over as the subexpressions are combined. To constrain the combination of expressions and ensure they act on the right event layer, pregroup grammars are used as the syntactic framework, on top of which is added the truth-conditional semantic layer.

The general process of unifying event variables is not required to take place in the pregroup framework, or even categorial framework, though pregroup grammars do offer some advantages over other syntactic frameworks for this type of analysis, especially since their types are non-functional and some syntactic relations are already defined in the system through type ordering, e.g. $N \rightarrow \pi_3$.

The focal point of the analysis will also be different in a sense from the more traditional approaches to semantics, as instead of focusing on truth condition combination and having meaning predicates used as arguments to other predicates, the variables themselves will be the ones moving around the syntactic trees, as they are in a way the only piece of information accessible from a predicate by another one. On the other hand, the semantic predicates' main role will be to constrain the possibilities of events to occur by restraining the possible values the event and entity variables can take.

It will sometimes be shown that more than one possible way of implementing a certain aspect of event analysis with the pregroup framework could be possible. Their respective technical advantages

will also be discussed.

4.1 Motivation

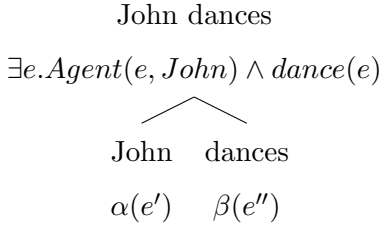
Let us start by looking at the event analysis of a simple sentence.

(84)

$$[\![John\ dances]\!] = \exists e.Agent(e, John) \wedge dance(e)$$

In this case, a single event e is shared by the two lexical items. The goal is to compositionally explain how to get to that denotation, so that one could define something like a derivational system into which words are fed and complex meanings are formed as outputs by that system.

(85)

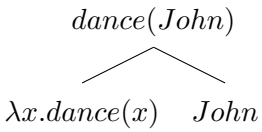


In the above case, α and β stand for the semantic representation of their respective expressions and take scope over an event argument. Leaving aside for a moment the question of what the exact values α and β stand for, an important question to answer is: where does the event e come from?

The main property of a variable is its mutability, i.e. it can take any value assigned to it. One does not have to know from the start what value the variable will be taking at the end.

In the following functional example, x does not have any intrinsic value at the beginning, it will take the value of whatever term gets passed to the expression.

(86)



After function application, the value of x is replaced by that of $John$.

Now, looking at the event translation, neither the subject $John$ nor verb phrase *dances* know what they will take scope over once the derivation ends. For instance, in a sentence such as (87)

they would not have the same event as argument: the subject *John* is related to a first event of *knowing*, while *dances* predicates over a totally different event where Sara is the agent instead of John. The goal is to figure out what they could have started with so that they end up with the right argument assignment.

(87) John knows that Sara dances

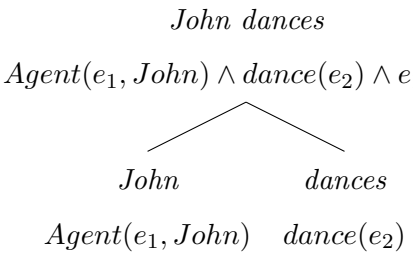
Assuming that they start for instance with values $Agent(e, John)$ and $dance(e)$, over the same event e, does not solve the problem: how did they know that they both take the exact same event as argument? Remember the last example: they won't always take the same event as argument.

The semantic system developed in this project does not have support for function application and uses conjunctions instead as meaning combination operation, whenever possible. One of the reasons for that, is that the syntactic framework that is used to structure the derivations does not support λ-expressions. Instead, getting the right variables in the right place will be achieved by making use of the operation of *unification* on variables.

Here is how the derivation of the logical form $\exists e.Agent(e, John) \wedge dance(e)$ will be assumed to take place:

- Distinct variables are instantiated by *john* and *dances*'s semantic predicates, to be taken as arguments: $Agent(e_1, John)$ and $dance(e_2)$
- Lexical items are concatenated, from which it follows that the variables associated with the syntactic categories that allowed the concatenation to take place are unified. In this case, e_1 and e_2 are unified, i.e. $e_1 = e_2$.

(88)



- A final process then takes place that binds the variables that were instantiated

The form (89) will often be automatically replaced by (90) for readability.

(89)

$$A(e_1) \wedge B(e_2) \wedge e_1 = e_2$$

(90)

$$A(e_1) \wedge B(e_1)$$

Note that actually keeping the two variables as distinct and binding each one — hence binding twice instead of once — does not actually make any difference in the meaning:

(91)

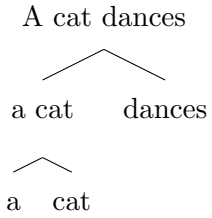
$$\exists x.\exists y.A(x) \wedge B(y) \wedge x = y \iff \exists x.A(x) \wedge B(x)$$

Now let's change things a bit and look at the sentence (92) which has logical form (93) with formation tree skeleton (94).

(92) A cat dances

(93) $\exists e.\exists x.Indefinite(x) \wedge cat(x) \wedge Agent(e, x) \wedge dances(e)$

(94)



It would be nice to have the derivation process be similar to the one described above, but that poses a problem, as doing so exactly the same way would give us the kind of logical form in (95).

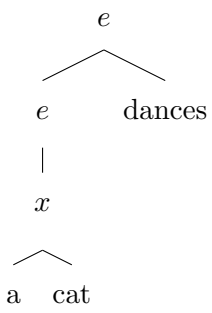
(95) $\exists e.Indefinite(e) \wedge cat(e) \wedge Agent(e, e) \wedge dances(e)$

The variable that the determiner phrase takes as argument should be a completely different one from the one taken by the verb. The relation between those two variables seems to be exactly what $Agent(e, x)$ is defining: that the variable from the noun phrase is an element of the variable taken in by the verb phrase; it corresponds to its agent.

This problem will be approached from two different angles. The first way, following Pietroski (2005) will be to assume that the type of the determiner+noun compound contains one accessible variable x, which, through a transformation from determiner phrase to subject – or by being assigned case – gets a new semantic constraint $Agent(e, x)$ and has its accessible event variable switched from x to e. In other words, a new grammatical role is now synonymous with a change of variable and a closure of the old variable. This solution has the advantage of being more

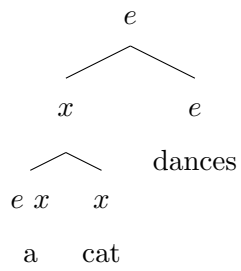
theoretically motivated, but also has its downsides when used in a syntactic framework like that of pregroup grammars.

(96)



The second way this could be dealt with this is by assigning different variables to the syntactic categories that form the type of the determiner a, so that when it first concatenates with *cat*, the syntactic category that allows for the operation to happen will be linked to x, and the one corresponding to the second concatenation will contain a different variable, e.

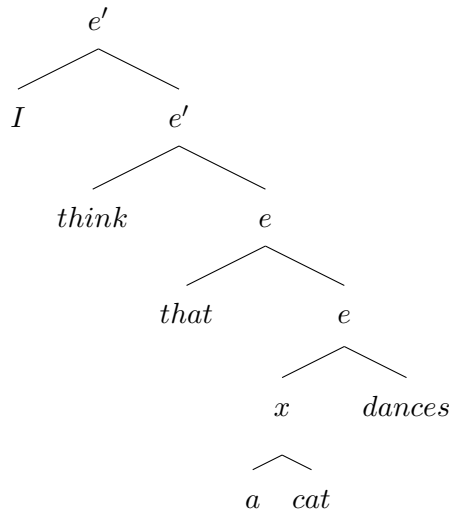
(97)



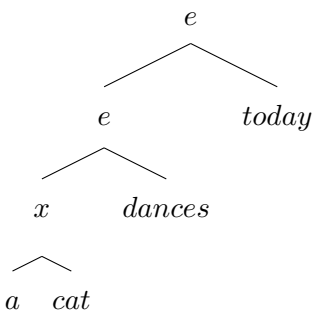
The variable over which two branches unify over is represented in the node.

Looking back at the previous tree, one see that e dominates it and might wonder what this implies. For simplicity, it can be assumed that this node, which is the final one in the tree, is of a basic category, e.g. s or C, as opposed to a concatenation of categories, and so that this category corresponds to a single variable, or has access to a single variable, e in this case. The reason is that if one were to use the expression within another expression or if one wanted to concatenate extra lexical items to it, what would be shared between the two would be the event variable e, and in no case the entity variable x. That's assuming the internal structure of that subtree is not modified.

(98)



(99)



(100) $[\![\ I\ think\ that\ a\ cat\ dances\ ]\!]$
$= \exists e'.Agent(e', I) \wedge think(e') \wedge \exists e.\exists x.Agent(e, x) \wedge cat(x) \wedge dance(e)$

(101) $[\![\ A\ cat\ dances\ today\ ]\!] = \exists e.\exists x.Agent(e, x) \wedge cat(x) \wedge dance(e) \wedge today(e)$

On the left, the sentence is included in the tree as an embedded clause, which semantically would be represented as the variable e now being attached to e', the event over which I and *think* take place, by being its theme, i.e. $Theme(e', e)$. On the right, the expression is concatenated with an adjunct, which shows that the event e is still possibly accessible from *a cat dances*.

In a way, no matter how that constituent — *a cat dances* — is used, the main information that will be shared and that could be quantified over, seems to be the event. This is similar to the way syntactic categories behave, in the sense that no matter how long an expression gets, if its syntactic category is A, it will always be possible to use it anywhere where an expression of category A could be used, no matter what other constituents it might contain.

4.2 Pregroup Grammars and Event Unification

4.2.1 Preamble

It will be now shown how to transpose that approach into the pregroup framework.

There seems to be more than one way of accomplishing the task at hand. For instance, existential-closure could be implemented using pregroup grammars by relating the operation to grammatical function assignment and have variables get bound when the expression gets assigned a thematic role, or it could also work by simply binding a variable once it is used for the last time

in a type. It could also be possible that all closure operations happen at the very end. As the system will not include the negation operator or any other logical operators than conjunction $\wedge$ and existentialization $\exists$, this will not be problematic from a logical standpoint. Similarly, it will be shown how implicit event variables could be introduced in two different ways within the syntactic types.

To stay in the categorial state-of-mind, the system will be as general and require as few rules as possible when it comes to generating the logical form: systematic combination rules will be defined, but constraints as to when and to what lexical items these rules can be applied is mostly left to the lexical items themselves, by carefully specifying their syntactic types in the right way.

This also means, for instance, that just like in the pregroup grammar framework, where lexical items can only interact with each other through their grammatical categories, no computation or process, be it on the syntactic types or semantic form, will be assumed to have happened prior to the words being put together.

Therefore, the starting point for a derivation of the sentence (102) should not have *John* already assigned the thematic role *goal* and *a cat*, *theme*. Doing so would be either assuming that some first look-over at the sentence was done by the verb *was given* prior to the start of the derivation, which then set the semantic value of the pronoun to *recipient*, or that *John* possesses a collection of different semantic values corresponding to each possible thematic role it could take, viz. *agent*, *goal*, *theme*, *source*, and so on, and magically chose the right role given no exterior information as to the context it appeared in.

(102) John was given a cat

Note that the problem would not be the assignment of multiple possible semantic values to *John*, which is supported in the system in the same way that a lexical item could be assigned different syntactic types depending on the context, but the fact that nothing in the syntax restricts it from taking, say, the role of a *source*, eventhough it is clearly wrong.

Another, perhaps more ad hoc, option that could possibly work is to have the expression's role specified in its grammatical type.

(103) $\begin{matrix} like \\ a^r s t^l \end{matrix}$

In this case a would be the type of an agent and t of a theme, which would then force auxiliaries to have a type of the form below.

(104) $X^r s(X^r i)^l$

which a type template, where X stands for any thematic type. The reason is that in a pregroup derivation, the auxiliary being the head of the sentence, take as argument the subject and the verb phrase. Therefore some information about the role of the subject, which is determined by the verb, has to be passed along to the auxiliary.

(105)

$John$	$will$	$like$	$Suzy$
a^r	$a^r s(a^r i)^l$	$a^r it^l$	t
$Agent(e, John)$	$will(e)$	$like(e)$	$Theme(e, Mary)$

(106)

$John$	was	$kissed$
t^r	$t^r s(p_2 t^l)^l$	$p_2 t^l$
$Theme(e, John)$	$Past(e)$	$kissed(e)$

In this case, the syntactic category of an expression determines its meaning, e.g. $t \Rightarrow Theme(e, Mary)$ and $a \Rightarrow Agent(e, John)$. This approach is perhaps more unorthodox, but could be interesting if one wanted to bypass the use of meaning postulates and derive the right thematic roles as the sentence is parsed, or simply find out more about the distribution of thematic assignments and how they relate to the syntactic structure itself, see (Baker, 1988; Hale and Keyser, 1993, 2001).

Instead, it will be assumed that a derivation starts when lexical items are put side-by-side with their syntactic and semantic information, that the nodes' specification is completely independent from each other, and that no thematic information appears in the syntactic layer. The syntactic types then act as constraints on the way the different semantic values can combine, and the end result is a raw representation of the semantics of the sentence, to which some work still has to be done to get the full representation.

This final representation can be qualified as *raw*, as some aspects of the meaning of a sentence cannot be reached simply by predicate combination, especially since no pre-derivational thematic assignments are assumed to have taken place. Concretely, this means that one might end up with the representation (107) for the sentence (108).

(107) $\exists e.Subject(e, John) \wedge Passive(e) \wedge Past(e) \wedge Kick(e) \wedge Time(e, Monday)$

(108) John was kicked on Monday

Then that extra meaning could be reached through the subsequent use of meaning postulates such as (109).

(109) $Subject(e, A) \wedge Passive(e) \wedge Kick(e) \vdash Patient(e, A)$

as thematic roles depend on multiple factors such as voice, grammatical functions and the nature of the verb itself.

That being said, *Agent*, *Theme* and other thematic role predicates will be used in this work to follow what is done the literature, but it would perhaps be more right to think of them as *Subject* and *Object* predicates, or even better as *Internal* and *External* argument predicates, which after all information is extracted from the sentence, are used to entail the right thematic role.

It is also common to analyse verbs such as *to break* as complex verbs containing subevents (Schein, 2002; Pietroski, 2005) such as (110), meaning that having an event of breaking, means to have a *State_of_being_broken* event which is what the event of breaking eventually becomes.

(110) $broke(e) := \exists e'.State\_of\_being\_broken(e') \wedge Become(e, e')$

There will be no deep analysis of neither complex events nor meaning postulates in this work, the focus will be mainly on getting the raw representation, which will continue being referring to as *logical form* unless some confusion arises.

This also means that the explicit meaning of a predicate will often be abbreviated for conciseness and because it is assumed that the semantic value of each expression represents a truth condition independent of the others, e.g. *boys* will be represented as $boys(x)$ instead of $boy(x) \wedge Plur(x)$.

4.2.2 Semantic Pregroup Types

A first way of assigning syntactic and semantic information to lexical items is shown, which is somewhat more general than the *Pietroski* way, which is covered in section 3.3.

Lexical items are paired with a syntactic type and truth-conditional meaning predicates, which take as arguments different values and variables. Those variables are instantiated when the lexical item is first used, and their value changes depending on the way types contract. Every new variable introduced is fresh, i.e. does not appear in any other expression.

The full value of a lexical item is then a tuple of the form:

(111)

$$((a_1, x_1) \cdot (a_2, x_2) \cdot ... \cdot (a_n, x_n), A)$$

where a_i is a pregroup type, x_i an event variable, and A a logical formula that stands for the expression's meaning.

For instance, the relative pronoun *whom*, will have the form (112a), which will be rewritten as (112b) for clarity. This type could be read as: the variable associated with the sentential and subject type is possibly different from the one associated to the noun types. This comes from the fact that *whom* is usually used as *theme* predicate over 2 distinct variables, one corresponding to an event and another corresponding to an entity.

(112) a. $(n^r, x) \cdot (n, x) \cdot (s^l, e) \cdot (\pi_3, e)$

b. $n^r_x n_x s^l_e \pi_{3,e}$

This is a good example to show that having only one available variable per lexical item does not work well with pregroup types: both the event and entity variable have to appear within its meaning predicate at some point of the derivation, but starting with either and trying to introduce the other at a later point brings many complications. Not to say that it is impossible, just much more painful.

In the case of (113a), *whom* will have semantic value $Theme(e, x)$, where one of the variables, e, is also shared on the right with *Caesar stabbed* and the other one, x, on the left with *cat*.

(113) a. the cat whom Caesar stabbed

b. $\exists x.\ the(e, x) \wedge cat(x) \wedge \exists e. Theme(e, x) \wedge Agent(e, Caesar) \wedge stabbed(e)$

Note that the *kind* or *type* of the variables will be irrelevant in this system. There will be no real difference between x, e, or any other variable, and only the constraints put on a variable can say something about it. Using specific characters to represent variables such as x and e only makes reading descriptions easier.

It is tempting to use, for instance, entity types and event types and try to copy what is done in Montagovian semantics, but in the end the types that would end up being required would be very different from the Montagovian ones. For instance, there is not going to be any boolean type passed around: in event semantics, one does not usually use existential quantification over truth values. Although some types that would be useful are *states*, *activities*, *accomplishments* and *achievements*, which are the usual ways of classifying eventualities (Vendler, 1957). The problem with using those, is that the boundaries are extremely blurry and the class depend on multiple factors.

(114) a. I built the house in an hour ⇒ accomplishment

b. I built houses for 10 years ⇒ activity

It also seems non-essential to try to pinpoint exactly what is an entity, what is an event and so on. Not only is it not 100% clear

(115) a. The cat is beige ⇒ cat : entity?

b. The destruction of the building took three days ⇒ destruction : entity?

c. The thunderstorm was intense ⇒ thunderstorm : entity?

d. John's destruction of the wall ⇒ destruction : entity?

e. John destroyed the wall ⇒ destroyed : event?

but it does not seem like types are even needed in this case, the contraction of types, in a way, takes care of making sure predication is over the right variables. There have been theories of typeless semantics or with a single type proposed (Carstairs-McCarthy, 1999; Partee, 2006), but it doesn't seem like it would add anything interesting to our analysis here or solve any problem.

It can also be argued that the syntactic structure's task is primarily to pass those bare variables around, and that being an event, a task, or an accomplishment are simply either subproperties of the predicates or the result of the interaction between predicates (Zwarts, 2005).

(116)

$$cat(x) \Rightarrow x = living$$

(117)

$$build + for \Rightarrow activity$$

Semantic incongruities could also be approached this way, for instance, the concept $rock(x)$ might contain a subconcept of the kind $non\text{-}animate(x)$, which then could interfere with a subconcept $animate(x)$ of a predicate $crying(x)$, and give the natural intuition that rocks cannot cry, by entailing the contradictory meaning.

(118)

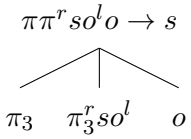
$$non\text{-}animate(x) \wedge animate(x)$$

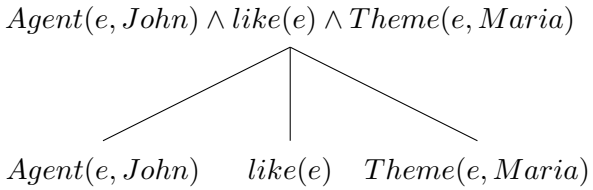
4.2.3 Semantic Combinations

This section will outline a method of combining lexical items' meanings given their new tuple types defined above.

As previously mentioned, a functional semantics for pregroup grammars is not the best option, unless major modifications are made to either the pregroup layer, to make it more "functional", or to the semantic layer, to include something like bidirectionality in the argument passing and some way of accounting for compositionality of types (Gaudreault, 2015). So instead, meaning will be treated here in a conjunctivist fashion, and the meaning of an expression will simply be defined as a conjunction of the meaning of the parts, with some interplay of the explicit variables.

This already seems to be more fitting to the pregroup framework, as here the syntactic types are simple concatenations of basic types, which gives us a nice kind of symmetry:

(119)


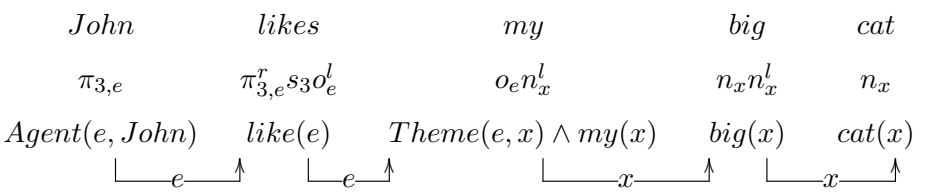
(120)


To get the right variables at the right places the variables will have to get unified over the contraction links. What this means is that whenever two syntactic types get contracted, their contained event variables will be forced to take the same value by conjoining an extra equality condition to the semantics. Here is a simple example to show how it works:

(121)

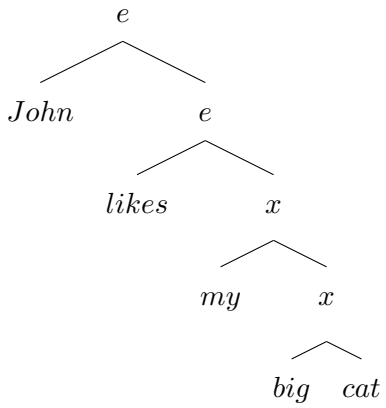
$$\begin{array}{ccccc} big & cat & \to & big\ cat \\ n_x n_x^l & n_y & \to & n_x \\ big(x) & cat(y) & \to & big(x) \wedge cat(y) \wedge x = y \end{array}$$

While contracting the types, the constraint that $x = y$ is added to the global meaning. As it was previously mentioned, the variables can also be replaced directly and the semantic part rewritten simply as $big(x) \wedge cat(x)$. What is nice about this is that a derivation can now be represented as semantic predicates linked to each other by the variables they share, which also corresponds to the contraction links. Put another way, the contraction links can be labelled using event variables.

(122)


Although binary tree representations are not very appropriate for pregroup derivations, because of the non-uniqueness of contraction orders, the above derivation could be drawn as the tree below.

(123)



The table below summarizes some of the correspondences between the two systems at this point:

Syntax	**Semantics**
Concatenation of basic types	Conjunction of logical predicates
Contraction of types	Unification on event variables

4.2.4 Syntactic/Semantic Hierarchy

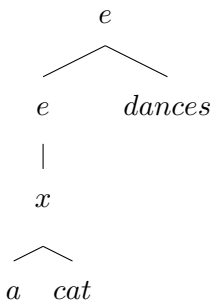
Let's have a look back at example (92) that was discussed at the very beginning of this chapter .

(124)

$$[\![A\ cat\ dances]\!] = \exists e.\exists x.Indefinite(e,x) \wedge cat(x) \wedge Agent(e,x) \wedge dances(e)$$

Underlyingly, the variables in this derivation must be shared in the way of (125) given that the types are usually distributed as in (126)

(125)



(126)

a	cat	dances
$\bar{n}n^l$	n	$\pi^r s$

The task at hand now is to find a way of going from the variable that is shared between *the* and *cat* to a fresh one that would then get unified with the one coming from *dances*. There is actually a very simple way of relating this to another pregroup operation and that is by extending the pregroup orderings, or syntactic hierarchy, to take into account semantic constructions.

To remind the reader, since pregroups are ordered structures, some grammatical relations can be explicitly defined as orders. For instance,

$\bar{n} \to o$ a determiner phrase can be used as object

$\bar{j} \to \pi_3$ an infinitive of complete verb phrase can be used as subject

A relation such $\bar{n} \to \pi_3$ could then be rewritten to include information about the variables present in the types and about the extra semantic relation they now play under this new syntactic type.

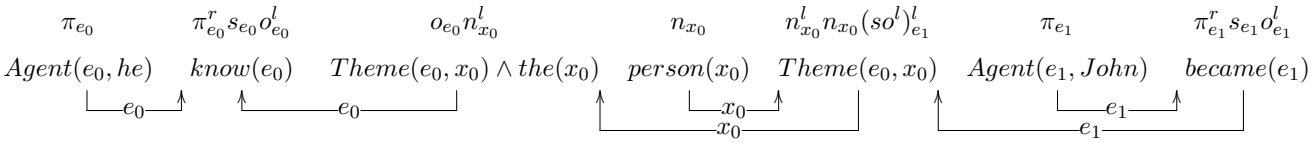
(127)

$\bar{n}_x$		$\pi_{3,e}$
$A[x]$	$\Rightarrow$	$Agent(e,x) \wedge A[x]$

which is to be interpreted as: using a determiner phrase as a third person subject means having its corresponding event variable used as the agent (or some other thematic role determined by the verb and tense used) of the event specified by the verb it will combine with.

(128)

He	*knows*	*the*	*person*	*whom*	*John*	*became*
π_{e_0}	$\pi^r_{e_1} s_{e_1} o^l_{e_1}$	$o_{e_4} n^l_{x_0}$	n_{x_1}	$n^l_{x_2} n_{x_2} (so^l)^l_{e_2}$	N_{john}	$\pi^r_{e_3} s_{e_3} o^l_{e_3}$
$Agent(e_0, he)$	$know(e_1)$	$the(e_4, x_0)$	$person(x_1)$	$Theme(e_2, x_2)$	$\top$	$became(e_3)$

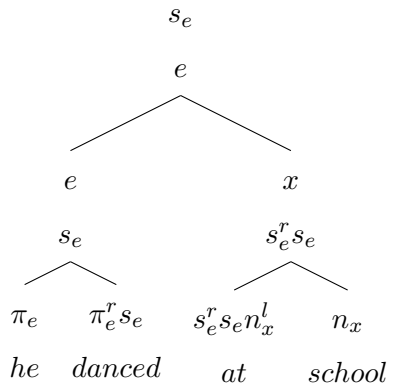


Note that it is not always necessary to go through a transformation of variable, as potentially distinct variables could be attached to the basic types of an expression, just like for the above case of the relative pronoun *whom*, or in a case like (129).

(129) $[\![He\ danced\ at\ school]\!] = Agent(e, he) \wedge danced(e) \wedge Loc(e, x) \wedge school(x)$

where all pieces naturally combine and the variables over which expressions are unified varies as the concatenation takes place.

(130)



In this case the variable at the top of a branching represents the variable that was unified.

The table of correspondences can now be updated:

Syntax	**Semantics**
Concatenation of basic types	Conjunction of logical predicates
Contraction of types	Unification on event variables
Type ordering	Conjunction of new semantic predicate

4.2.5 Existential-Closure

Existential-closure takes a bit more work to be implemented in this system, mainly because of the multiple directions a pregroup derivation can take depending on the order of the contractions.

Keep in mind that the only logical symbols used until this point are the conjunction $\wedge$ and existentialization $\exists$, hence existential-closure could always just be defined as a meta-operation that binds every variable used through a derivation at the very end of it, without affecting the truth conditions.

(131)

$$\exists e.dance(e) \wedge \exists x.cat(x) \wedge Agent(e,x) \iff \exists e.\exists x.dance(e) \wedge cat(x) \wedge Ag(e,x)$$

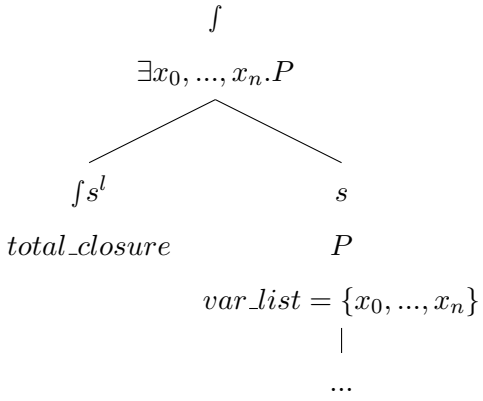
as long as only fresh variables are used, as otherwise there could be binding issues. For this reason, every raw translation will be equivalent to one of the form (132)

(132) $\exists x_0 ... \exists x_n . P_0 \wedge ... \wedge P_m$

Consequently, this means that a list of the instantiated variables used in our derivation could always be kept around and those variables bound them all at the very end. This could be justified

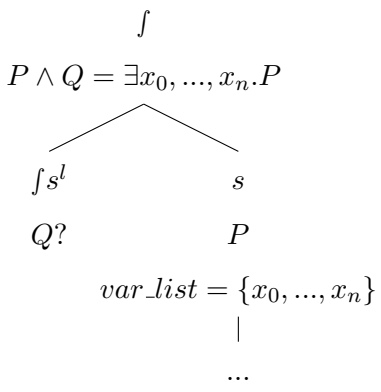
for instance by introducing a covert interpretation object *total_closure* that makes the semantics logically interpretable.

(133)



This would have to be an extra operation defined over the system, as it does not seem like there is any way to define *total_closure* so that the meaning above could be reached by simply conjoining *total_closure* with P. What it does is take the predicate, take the list, and then close each of the variable of the list one after the other, and that is, in a sense, more complicated than conjoining two predicates.

(134)



Before going any further, let's take a moment to look at the way types are constructed in pregroup grammars.

First, pregroup grammars do not provide official definitions for things like *complement* or *adjunct*, but they can still be extracted from looking at the type of an expression. An expression could be said to take a complement if it contains an adjoint type that has a different value than the value of its main type.

For instance,

Expression		Complement		
the		cat	$\Longrightarrow$	The expression takes in the complement
$\bar{n}n^l$	$\longleftarrow$	n		

and some examples of categories taking a complement:

Determiners:	$\bar{n}n^l$	n is the complement of the determiner
Finite intransitive verb:	$\pi^r s$	π is the complement of the verb
Relative clause:	$n^r n(\pi^r s)^l$	$(\pi^r s)^l$ is the complement of the clause

On the other hand, adjuncts do not affect the overall syntactic type of an expression, they will take in an expression of a certain type and output the same type it received. As opposed to complements, adjuncts are the ones with the adjoint type. For instance,

Expression		Adjunct		
cat		who danced	$\Longrightarrow$	the adjunct takes in the expression
n	$\longrightarrow$	$n^r n$		

and examples of adjuncts:

Adjectives:	nn^l	the expression is an n adjunct
Sentential adverbs:	$s^r s$	the expression is an s adjunct
Relative clause:	$n^r n(\pi^r s)^l$	the expression also acts as a right n adjunct

Existential-Closure $\Longleftrightarrow$ Grammatical Role

Following Pietroski (2005), one of the more theoretically motivated way of introducing closure on the variable would be to relate it to the introduction of grammatical role or to argument structure.

Trying to equate existential closure with argument taking and binding it right away does not seem to be possible in this framework as once the variable is bound, it cannot be accessed. Hence something like (135) would not work, as y would never get bound, and explicitly adding the equation $x = y$ creates a new unbounded x variable.

(135)

one	cat		one cat
$\bar{n}_x n^l_x$	n_y	$\rightarrow$	$\bar{n}_x$
$\exists x.one(x)$	$cat(y)$		$cat(y) \wedge x = y \wedge \exists x.one(x)$

One possible way to implement binding would be through the syntactic-semantic hierarchy.

(136)

$\bar{n}_x$		$\pi_{3,e}$
$A[x]$	$\Rightarrow$	$\exists x.Agent(e, x) \wedge A[x]$

Using this rule, the sentence (137) could then be analysed as (138).

(137) John danced

(138)

John	danced		John	danced
N_x	$\pi^r_{e_0} s_{e_0}$	$\longrightarrow$	π_{e_1}	$\pi^r_{e_0} s_{e_0}$
$John(x)$	$danced(e_0)$		$\exists x.Agent(e_1, x) \wedge John(x)$	$danced(e_0)$

	John danced
$\longrightarrow$	s_{e_0}
	$\exists x.Agent(e_0, x) \wedge danced(e_0)$

In general though, trying to define this kind of rules for pregroup grammars will end up creating more problems than it would solve. This mostly stems from the fact that grammatical roles are not clearly defined in the pregroup framework and word ordering blurs the way complements and adjuncts interact.

Let us look at some of those problems:

- Some grammatical relations are already encoded in the semantics of certain lexical items

(139)

building	of	a wall
n_x	$n^r_x n_x \bar{n}^l_y$	$\bar{n}_y$
$building(x)$	$Patient(x, y)$	$wall(y)$

	building	of wall
$\longrightarrow$	n_x	$n^r_x n_x$
	$building(x)$	$Patient(x, y) \wedge wall(y)$

In those cases, it is not clear when and how existential binding should take place.

This is also problematic for sentential adjuncts, such as *when* or *where*, which do take complements — the two sentences — but only bind one of the events, the inner one.

(140)

I cried	when	John danced
s_e	$s^r_e s_e s^r_{e'}$	$s_{e'}$
$cried(e) \wedge Agent(e, me)$	$overlap(e, e')$	$danced(e) \wedge Agent(e, John)$

$\Longrightarrow$

I cried	when John danced
s_e	$s^r_e s_e$
$cried(e) \wedge Agent(e, me)$	$overlap(e, e') \wedge danced(e) \wedge Agent(e, John)$

- Although the most important problem for this approach is that, since basic types on each boundary of a type are independent from each other contraction-wise, to equate binding of a variable with thematic transformation — which has to be done through type ordering — would force us to create major restrictions in the way types combine.

  For instance, using rule (136) defined above — which states that when a determiner phrase takes the role of an agent it binds the entity variable it contains and instantiate a new event variable — in the context where a determiner phrase is formed by a determiner and a noun, things can quickly go awry if the order of the contraction is not restricted.

(141)

some	cat		some	cat
$\bar{n}_x n^l_x$	n_y	$\rightarrow$	$\pi_e n^l_x$	n_y
$some(x)$	$cat(y)$		$\exists x.some(x) \wedge Agent(e, x)$	$cat(y)$

  The derivation is already doomed to fail: the variable that y is supposed to be unified with when contracting with n^l_x is already bound! The way to remedy to this situation would be to force the ordering relations to be used only once the type of an expression is a basic type — so to wait for all contractions to have taken place — but this makes the framework much more complex and might also be impossible to even put in place without a complete restructuring of the types.

  One of the main advantages of pregroup derivations is that it is possible to simply align all types corresponding to lexical items side-to-side in one string of types at the beginning of a derivation, forget about which type comes from where, and let the magic happen.

(142)

some cat danced when I cried
$\bar{n}_x n^l_x n_y \pi^r_{e_0} s_{e_0} s^r_{e_1} s_{e_1} s^l_{e_2} \pi_{e_3} \pi^r_{e_4} s_{e_4}$
$some(x) \wedge cat(y) \wedge danced(e_0) \wedge overlap(e_1, e_2) \wedge Agent(e_3, me) \wedge cried(e_4)$

$$\longrightarrow$$

$$\bar{n}_x \pi^r_{e_0} s_{e_1} s^l_{e_2} s_{e_4}$$
$$some(x) \wedge cat(x) \wedge danced(e_0) \wedge overlap(e_0, e_2) \wedge Agent(e_4, me) \wedge cried(e_4)$$

$$\longrightarrow$$

$$\pi_{e_5} \pi^r_{e_0} s_{e_1}$$
$$\exists x.some(x) \wedge cat(x) \wedge danced(e_0) \wedge overlap(e_0, e_4) \wedge Agent(e_4, me) \wedge cried(e_4) \wedge Agent(e_5, x)$$

$$\longrightarrow$$

$$s_{e_1}$$
$$\exists x.some(x) \wedge cat(x) \wedge danced(e_0) \wedge overlap(e_0, e_4) \wedge Agent(e_4, me) \wedge cried(e_4) \wedge Agent(e_0, x)$$

Using the ordering too soon would have the consequence that the variable contained in the type of cat would never get bound as the binding for the variable in *some* is already bound at the time they unify. This is similar to the problem that was faced above with the potential binding of the variable at the very beginning of the derivation, before any contraction could take place. Hence those types need to already be contracted and variables unified before using the ordering and closing the expression.

In this context too, a rule that says "use the syntactic ordering only when the type of an expression is basic" does not make any sense, as what is in the syntactic layer is a big bundle of basic types.

- Finally, one could also try to correlate binding of a variable with argument passing, by defining a rule that says something like:

  Once an argument gets passed to an expression,
  the variable carried by the argument is closed

  This rule might seem sound at first: complementizer phrases bind the event variable coming from the sentence they take as argument on the right, just like relative pronouns bind the event variable coming from the sentences on the right. But it fails on multiple other cases: variables in a noun phrase do not get bound when passed to a determiner phrase, they get

bound when taking the role of an agent. Similarly, verbs do not bind the (event) variable coming from the subject, it is still accessible for sentential adjuncts to use, for instance.

In the end, a rule that simply says:

Bind a variable after any operation on the types if the
variable does not appear in the resulting type

will work just perfectly.

To formalize the rule, notice that in pregroup grammars, modification on the types could be the result of two things: either contractions happened or the syntactic hierarchy was used. Therefore the contraction rule can be rewritten as (143).

(143)
$$\begin{array}{cccc} \alpha a_x & b_y\beta & \rightarrow & \alpha\beta \\ A[x] & B[y] & & \exists x.A[x] \wedge B[x] \end{array}$$

if x and y do not appear in any basic type of $\alpha\beta$, where a and b are adjoints of one another, otherwise there is simply no binding.

On the other hand, the order rule can be written as (144) given the ordering (145)

(144)
$$\begin{array}{ccc} \alpha a_x \beta & \rightarrow & \alpha b_y \beta \\ A & & \exists x.B \end{array}$$

(145)
$$\begin{array}{ccc} a_x & & b_y \\ A & \rightarrow & B \end{array}$$

if x does not occur in $\alpha b_y \beta$. If it does not occur in the type, no existential-closure shall take place.

This allows one to get the right forms when, for instance, a determiner phrase is used as subject and when an adjunct clause is created.

(146)
$$\begin{array}{cc} \bar{n}_x & \pi_e \\ the(x) \wedge cat(x) & \rightarrow \exists x.the(x) \wedge cat(x) \wedge Agent(e,x) \end{array}$$

(147)
$$\begin{array}{cccc} \text{when} & \text{I cried} & & \text{when I cried} \\ s^r_{e_0} s_{e_0} s^l_{e_1} & s_{e_2} & \rightarrow & s^r_{e_0} s_{e_0} \\ overlap(e_0,e_1) & Agent(e_2,me) \wedge cried(e_2) & & \exists e_1.overlap(e_0,e_1) \wedge Agent(e_1,me) \wedge cried(e_1) \end{array}$$

The only potential drawback of this approach is that the way to close the final event variable of a sentence would probably have to include introducing a new syntactic type for sentence-ready-to-be-interpreted and include instantiating a new variable that will not be used.

(148)
$$\begin{array}{ccc} \text{John danced} & & \text{John danced} \\ s_{e_0} & \rightarrow & \bar{s}_{e_1} \\ Agent(e_0, John) \wedge danced(e) & & \exists e_0.Agent(e, John) \wedge danced(e) \end{array}$$

Here is the final update to the table of correspondences between the syntactic and semantic systems:

Syntax	**Semantics**
Concatenation of basic types	Conjunction of logical predicates
Contraction of types	Unification on event variables
Type ordering	Conjunction of new semantic predicate
Variable is taken out of the types	$\exists$-closure

4.3 Semantic Pregroup Types: Alternative Way

Instead of assigning possibly distinct variables to every basic type in the type of an expression, what could be attempted is to assign a single variable to the expression and reach new variables using the syntactic-semantic hierarchy. This way of handling semantics is a direct translation from Pietroski (2005).

Alternative tuple-form:

(149) $((a_1, \cdot a_2, \cdot \ldots \cdot a_n), x, A)$

where a_i is a pregroup type, x an event variable, and A a logical formula that stands for the expression's meaning.

For instance,

(150)
$$\begin{array}{llll} one: & \bar{n}n^l & x & one(x) \\ should: & \pi^r s i^l & e & should(e) \\ big: & nn^l & x & big(x) \end{array}$$

The main issue with having a single variable is that restricting the use of that variable, so that it is only used in the right cases, is very hard to achieve since the order of contractions in a pregroup type is not unique.

For instance, consider the relative pronoun *who* with type $n^r n(\pi_3 s)^l$. Right away, this seems like having one variable will simply not be enough. Why?

- The lexical item's role is to relate two variables as $Agent(e, x)$
- After both contractions have taken place, x has to be the main event variable, i.e. the one coming from the noun, as it will be used later to construct a determiner phrase or some other construction, therefore *who* would have to start with e as corresponding event variable and find a way to transform it into x
- Although when starting with (151) nothing would stop a noun phrase on its left from contracting with it and unifying its entity variable with *who*'s event variable, which is clearly wrong

(151)
$$\begin{array}{c} e \\ n^r n(\pi^r s)^l \\ Agent(e, x) \end{array}$$

One might think that a possible remedy to that situation would be to start with a different type, such as $q(\pi^r s)^l$, which is the type used in wh-question constructions, and then introduce a rule $q \to n^r n$, that states that those constituents could be used as right noun adjuncts.

(152)
$$\begin{array}{ccc} \text{cat} & \text{who} & \text{danced} \\ x & e_0 & e_1 \\ n & q(\pi^r s)^l & \pi^r s \\ cat(x) & \top & danced(e_1) \end{array} \longrightarrow \begin{array}{cc} \text{cat} & \text{who danced} \\ x & e_0 \\ n & q \\ cat(x) & danced(e_0) \end{array}$$

$$\longrightarrow \begin{array}{cc} \text{cat} & \text{who danced} \\ x & y \\ n & n^r n \\ cat(x) & danced(e_0) \wedge Agent(e_0, y) \end{array} \longrightarrow \begin{array}{c} \text{cat who danced} \\ x \\ n \\ cat(x) \wedge danced(e_0) \wedge Agent(e_0, x) \end{array}$$

Right away, it can be seen that q would have to be divided into different classes, depending on its case, as that type will then be used to get the right thematic relation on the event.

A bigger problem is the fact that, using the ordering relation before the first contraction makes everything crumble once again.

$$\begin{array}{ccc} \text{cat} & \text{who} & \text{danced} \\ x & e_0 & e_1 \\ n & q(\pi^r s)^l & \pi^r s \\ cat(x) & \top & danced(e_1) \end{array} \longrightarrow \begin{array}{ccc} \text{cat} & \text{who} & \text{danced} \\ x & y & e_1 \\ n & n^r n(\pi^r s)^l & \pi^r s \\ cat(x) & Agent(e_0, y) & danced(e_1) \end{array}$$

(153)

$$\longrightarrow \begin{array}{c} \text{cat who danced} \\ x \\ n \\ cat(x) \wedge Agent(e_0, x) \wedge danced(x) \end{array}$$

The problem is that using the ordering prior to the contraction forces a change in the accessible event variable which makes the event variable of the verb phrase unify with the entity variable coming from the cat.

The use of the ordering and change of free event variable could be forced to only occur after all contractions have happened, but then all the same problems encountered in the last section were resurfaced. Trying to fit in a way of dealing with existential-closure is also not at all straight forward and most of the problems faced when trying to make it work with multiple variables would have to be faced again.

4.4 Summary of the Problems Encountered

The overall problem with trying to adapt the kind of semantic system that people like Pietroski uses is that syntactic types in pregroup grammars can combine in any order they want as long as they are on the boundary of the type. This is problematic, as syntax usually does not work this way, and a lot of ordering is encoded in the types, for instance by using internal/external argument relations. Hence, transformations on the variable that can be used in the meaning of an expression, and assignments of thematic relations do not work as well in the pregroup framework.

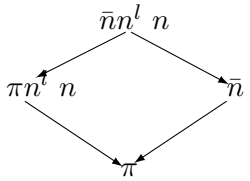
A simple example is how thematic relation assignment is usually dealt with. A generative way of representing how the phrase *my cat* can play the role of an agent could be as positing a covert agent-node that takes in the clause and changes its domain of predication from an entity to an event

(154) $[\![\ Agent\ [my\ cat]\ ]\!](e) = \exists x.Agent(e,x) \wedge [\![my\ cat]\!](x)$

This way, if something is under the scope of *agent*, it will only have access to x and not e, and on the other hand, x is bound inside the agent-node, hence if one is working outside of it, they will not have access to x, but only to e. By letting the determiner phrase be taken as an argument, the syntactic parsing is also restricted, as the concatenation of *my* and *cat* is something that can only happen lower in the tree, than the assignment of the agent role.

In pregroup grammars, the derivation could take multiple forms.

(155)



On the left, the ordering is used first on $\bar{n} \to \pi$, which corresponds, in a sense, to combining a covert agent node with the determiner, before the latter combines with the noun. On the right, the combination of the determiner and noun takes place before the expression takes on the role of an agent, which is equivalent to the unique generative representation discussed above.

This much more flexible way of combining expressions is the reason it is harder to structure existential closure and is the reason multiple variables per type are needed, as some of those operations take place in parallel, and sometimes multiple variables have to be accessible at the same time. More precisely, in this case, closing the variable as $\bar{n}$ goes to π before contracting the types (left path) blocks the entity variable coming from the noun to be unified with the determiner, which is dramatic.

Chapter 5

Plural Analysis

This section aims at providing a brief description of the way plurality can be handled using a conjunctivist semantic representation. It will be shown that using a 2-level logic allows one to capture some essential aspects of plurality and quantification.

More precisely, quantifiers will be shown to have a natural interpretation as conditions set on the events they relate to and the entities present in that event.

5.1 Plurality

At this point, enough machinery has been built up to do basic semantic analysis, but that is probably not enough to deal with simple cases involving plurality, which is what will be done here.

Consider the sentence

(156) My cats danced

How should the meaning of this sentence be represented? A basic analysis following last chapter's could help us produce this kind of reading:

(157) $\exists e.\exists x.my(x) \wedge cats(x) \wedge Agent(e,x) \wedge danced(e)$

But what does this translation really say? That there is both an event and an entity, and that this entity satisfies certain conditions, one of which being *cats*. For this to be sound, x would need to stand for multiple values at once, either in the form of sets or plurality. For instance, given support for sets, the following translations could be provided for the previous sentence:

(158) $\exists e.\exists x.c_0 \in x \wedge ... \wedge c_n \in x \wedge Agent(e,x) \wedge dance(e)$

(159) $\exists e.\exists x.\forall c.(c \in my\_cats \leftrightarrow c \in x) \wedge Agent(e, x) \wedge danced(e)$

where c_i's are my cats.

On the other hand, if one could somehow get access directly to *my cats*, they could always represent the sentence as:

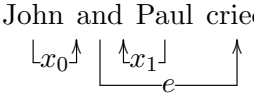
(160) $\exists e.Agent(e, c_1) \wedge ... \wedge Agent(e, c_n) \wedge danced(e)$

Although depending on the internal structure of the events, this last one might be problematic, as supposing that event variables can only stand for a single event, this form would directly imply the existence of a single event where all my cats danced. In a situation where all cats danced at different times, the sentence would instead have to be modelled as something like:

(161) $\exists e_0...\exists e_n.Agent(e_0, c_0) \wedge danced(e_0) \wedge ... \wedge Agent(e_n, c_n) \wedge danced(e_n)$

Which seems a bit too explicit and to be lacking generality. It also does not really fit the general scheme that has been followed until now, as the number of variables might clearly be greater the number of basic syntactic types present in a lexical item — the containers of variables — hence it is not clear where those variables and predicates would be extracted from, unless some kind of process that takes a sentence to its extensional form prior to its logical translation is assumed to take place.

Similarly, consider the sentence below



One way of doing the analysis would be to carry both subject entities through the whole derivation and duplicate the verb phrase predicate at the very end.

(162) $\exists e_0.\exists e_1.Agent(e_0, John) \wedge Agent(e_1, Paul) \wedge cried(e_0) \wedge cried(e_1)$

But again, it seems like this construction would benefit from more generality and would look better given some sort of support for sets or combination of values, both for entities and events, which would allow us to interpret the $Agent(e, x)$ predicate below in the right way.

(163) $\exists e.\exists x.John \in x \wedge Paul \in x \wedge Agent(e, x) \wedge cried(e)$

One useful modification would be to redefine events as a recursive structure, or even to define a special kind of algebra over events that supports a way of combining events (Bach, 1986; Schein,

1993; Pietroski, 2005; Lasersohn, 2006; Zwarts, 2005). For instance, events could be divided into basic events and complex events, which would be composed of multiple smaller events.

- Basic events: e_0, ..., e_n
- Complex events: $\{e, e'\}$, if e and e' are events

The above translation would then be sound under an interpretation where e is a complex event containing one subevent where John cried and another one where Paul cried, and where the interpretation of the verb and agency relation are distributive over complex events and entities, e.g. $cried(\{e_0, e_1\})$ to be evaluated as $cried(e_0) \wedge cried(e_1)$.

(164) $John \in x \wedge Paul \in x \wedge Agent(\{e_0, e_1\}, x) \wedge cried(\{e_0, e_1\})$
$\Longrightarrow \top$ when $Agent(e_0, John)$, $Agent(e_1, Paul)$, $cried(e_0)$, and $cried(e_1)$

Although defining exactly how many operations on sets and what their exact interpretation should consist of is not an easy question. Consider

(165) John and Paul danced and cried

How many events should be present in the translation? Should each agent be required to satisfy both cried and danced for the whole to be true or could John have been the one to do the dancing and Paul the crying?

Although these questions will for the most part stay unanswered, the following section will briefly show how plural variables could be used to provide for interesting analyses of some of the cases above, and of even more complex ones such as quantification. The logic used in the next sections will allow for analyses similar to those above, although instead of using sets, a logic with two levels will be used, which contrasts with the potentially infinite depth of sets. Events and entities will also be of two different kinds: singular and plural.

Not a lot in this section will be novel, it is rather a very condensed version of some of what one could find in the literature (Pietroski, 2005; Schein, 1993) in regards to modelling plurality in a Conjunctivist framework. The use of pregroup grammars does not affect the structure of the modelling in significant ways, it just shows how those representations could be reached in a simple compositional fashion.

5.1.1 2-Sorted Logic

To model plurality, a two-sorted logic, with sorts modelling singularity and plurality, is used. This logic is very close to a second-order logic, which is also used in the literature (Boolos, 1984; Pietroski, 2003, 2005; Schein, 1993) and which could have also been used in this section, although two-sorted logic seems more convenient and also more accessible to readers less familiar with formal logic.

One of the reasons for using this logic is that it ends up being more powerful than first-order logic(Boolos, 1984), as it can model very naturally sentences such as (166), as (167).

(166) Some critics admire each other

(167) $\exists X.\forall u.(u \prec X \rightarrow Critic(u)) \wedge \forall u.\forall v.(u \prec X \wedge Admire(u,v) \rightarrow v \prec X \wedge u \neq v)$

From a more foundational point of view, this logic is also interesting as it avoids a lot of the problems using set theory can bring, especially in regards to the potential depth of sets. Its variables and constants can only be of two kinds: singular or plural; and its corresponding membership relation — "$\prec$" — is restricted to singular elements on its left and plurals on its right. So such problematic sets as *the set of all sets who do not contain themselves* simply cannot be implemented in this kind of framework. If something is *one of* something else, it has to be singular, otherwise it is plural.

Alphabet

- Logical relations $=_{sg}$, $=_{pl}$, and $\prec$
- Sorts $\sigma = \{sg, pl\}$
- Singular constants a_i's
- Plural constants A_i's
- Singular variables x_i's
- Plural variables X_i's
- Sort function ν taking terms to sorts
- Functions f_σ defined on a specific domain of values $(s_0, ..., s_{n-1})$ that outputs s_n's of sort σ
- Predicates P defined on a specific domain of values $(s_0, ..., s_{n-1})$ that output a boolean s_n

In the case where P takes in a unique argument of the sort sg (pl), it will be written as P_{sg} (P_{pl} respectively). Lower case letters will also be used to represent singular values and upper case for plural values, when it is clear from the context.

Terms

- Singular constants and variables are singular terms
- Plural constants and variables are plural terms
- $f_\sigma(s_0, ..., s_{n-1})$ is a term of sort σ, given s_i's of the right sort

Expressions

- $t_\sigma =_\sigma s_\sigma$ is an expression, given terms t and s of the sort σ
- $t \prec T$ si an expression, given a singular term t and plural term T
- $P(s_0, ..., s_1)$ is an expression, given s_i's of the right sort
- $E_0 \circ E_1$ is an expression, given expressions E_i and $\circ$ a binary logical operation $\{\wedge, \vee, \rightarrow\}$
- $\neg E_0$ is an expression, given an expression E_0
- $\forall_\sigma x_\sigma.P$ and $\exists_\sigma x_\sigma.P$ are expressions, given an expression P and sort σ

Put simply, the difference between terms and expressions is that terms stand for a value of one of the sorts, whereas expressions are truth values.

As an aside, note that both singular and plural constants are, in a sense, unnecessary, as they could have also been defined as singular and plural nullary functions: given any constant c, let f_c be a function that takes no argument and outputs the value c, then f_c is for all intents and purposes the same as c.

Note the difference between the set-theoretic relation of membership as opposed to that of being *one of*.

$$x \in X := x \text{ is } \textit{an element of} \text{ the set } X$$

vs.

$$x \prec X := x \text{ is } \textit{one of} \text{ the } X\text{'s}$$

It's a subtle distinction, but consider the relation of *singer* to *singers* and that of *singer* to *choir*. On the one hand, it seems more natural to use *singers* when referring to all singers at once; you can say (168) if all singers in your group have blue hair, but (169) seems off.

(168) The singers have blue hair

(169) The choir has blue hair

Choir seems to be adding an extra layer on top of the multitude of singers; *choir* is a singular entity to which you can refer to and that can contain multiple elements. Hence you could say that a singer is *an element of* the choir, but is *one of* the singers.

Here are some examples of predicates that could be used to model natural language:

(170) a. $run_{sg}(x) := x$ is a singular entity that runs

b. $giraffe_{sg}(x) := x$ is a giraffe

c. $related_{pl}(X) := X$ is a plural variable with values that are related to each other

d. $three_{pl}(X) := X$ is a plural variable with three distinct values

e. $like(x_{sg}, y_{sg}) := x$ is a singular value who likes the singular value y

As it will be shown, there is probably no need to introduce more variations on the *like* predicate, e.g. $like(x_{sg}, X_{pl})$ which relates a singular value to a plural one, as those can usually be derived from the basic one defined above:

(171) $like(x, Y) \iff \forall y \prec Y.like(x, y)$

5.2 Plurality and Distributivity

To get the right interpretations, it is crucial to introduce an important rule that helps with distributive readings of predicates.

5.2.1 Distributivity of Singular Predicates

(172) $P_{sg}(T_{pl}) := \forall x.x \prec T_{pl} \leftrightarrow P_{sg}(x)$

This rule says that a plural value satisfies a singular predicate if and only if every one of its singular values satisfies the predicate.

This has the consequence that any singular monadic predicate can be seen more simply as a plural value: given the singular predicate $cat(x)$ that evaluates whether one or more entities are cats, let T_{cat} be such that it satisfies $c \prec T_{cat}$ if and only if c is a cat, then $cat(x)$ and $x \prec T_{cat}$ are interchangeable following the distributive axiom.

Going back to the situation above, the sentence

(173) My cats danced

can be translated as

(174) $\exists E.\exists x.my(X) \wedge cat(X) \wedge Plural(X) \wedge AGENT(E,X) \wedge danced(E)$

which is then interpreted as

(175) $\exists E.\exists x.\forall x.(x \prec X \leftrightarrow my(x) \wedge cat(x)) \wedge Plural(X) \wedge AGENT(E,X) \wedge \forall e.(e \prec E \leftrightarrow danced(e))$

Note that $Plural(X)$ is not affected by the variable distribution as it is a true plural truth predicate, i.e. its evaluation is not reducible to an evaluation on the singular values of an entity.

The predicate $AGENT(E,X)$ in this case is distributive and could have this kind of interpretation:

(176) $(\forall e \prec E.\exists x \prec X.agent(e,x)) \wedge (\forall x \prec X.\exists e \prec E.agent(e,x))$

which verifies that each event e has an agent in X, and that each x is the agent of some event in E. That second clause is required in this case to contrast it from the meaning

(177) Some of my cats danced

which only requires some of the cats to be agents of the dancing event. All values in X must be agents of E when someone utters *My cats danced*. This condition could also be modelled as a sub-condition of the determiner itself which would act as a quantifier restricting the way the event and entity variables relate to each other. That is the way it will be modelled later in this section.

Similarly,

(178) John and Paul cried

could now take the form

(179) $\exists E.\exists X.John \prec X \wedge Paul \prec X \wedge Agent(E,X) \wedge cried(E)$

interpreted as

(180) $\exists E.\exists X.John \prec X \wedge Paul \prec X \wedge Agent(E,X) \wedge \forall e.(e \prec E \leftrightarrow cried(e))$

or, treating proper names as predicates, which is more convenient in some cases,

(181) $\exists E.\exists X.\exists x.\exists y.John(x) \wedge Paul(y) \wedge x \prec X \wedge y \prec X \wedge Agent(E,X) \wedge \forall e.(e \prec E \leftrightarrow cried(e))$

5.2.2 Motivating distributivity

The motivation for this rule comes from the tendency for natural language to interpret predicates over plural values in a distributive way, that is, the predicate takes a true value on a plural value if

and only if the predicate holds for each element of that plural domain. See *cat* above: a collection of animals are *cats* only when each animal is a cat; there is no notion of being a *plural cat*.

Similar axioms are defined in the literature. For instance, Pietrokski (2005) would have the expression *my cats* in subject position be defined as a determiner phrase that is passed as argument to a covert EXT node that assigns to it the role of an external argument to the verb phrase. Distributivity then shows up through the interpretation of the argument nodes, which is defined as

(182) $[\![\ EXT\ [\ my\ cats\ ]\ ]\!](E) \iff \exists E.External(E, X) \wedge \forall x.(x \prec X \leftrightarrow [\![\ my\ cats\ ]\!](x))$

In his case, distributivity comes hand in hand with argument assignment, and case-by-case specifications of the template have to be made to handle the nested expression in the right way. The notion of semantic arguments in pregroup grammars, not being as clearly defined, and the fact that derivations cannot be uniquely represented as binary tress, i.e. the order of contractions is not uniquely determined (see previous section), forces distributivity to be encoded differently in this system.

The distributive interpretation, being only defined on singular predicates, allows plural predicates to define their own truth conditions. Plural predicates could then be used to model those more complex conditions such as collectivity, which shows up in situations where it could be argued that even though an ensemble satisfies a certain predicate, its individual values do not necessarily do so:

(183) The singers form a choir, BUT a single singer is not a choir

(184) The pens form a square, BUT a single pen does not form a square by itself

(185) The students gathered, BUT a single student cannot gather on its own

This situation could be analysed, for instance, by creating two different denotations available to the agency relation: one representing the distributive agency reading, seen earlier, and one representing collective agency readings, which takes on singular event variables and plural entity variables. It would also be possible to model the situation by using *collective* and *distributive* semantic features as predicates on event values, and posit the former as a subcondition of the verb *gather*

(186) $[\![\ gather\ ]\!](e) = gather(e) \wedge Collective(e)$

Then linking the $Distributive$ predicate to the distributive reading of $Agent$, and $Collective$ to the collective reading would make one of the readings of the sentence

(187) The students gathered

be contradictory

(188) $\exists E.\exists X.the\_students(E,X) \wedge (Agent_{dist}(E,X) \wedge Distributive(E)) \wedge (gathered(E) \wedge Collective(E))$
$\vdash Distributive(E) \wedge Collective(E) \vdash \bot$

but not the other one

(189) $\exists E.\exists X.the(E,X) \wedge students(X) \wedge (Agent_{coll}(E,X) \wedge Collective(E)) \wedge (gathered(E) \wedge Collective(E))$

Other predicates, such as $two(X)$ — which verifies that X has indeed two values — and $Plural(X)$ — which checks that a plural variable takes on more than one value — are also plural predicates. For instance, the sentence

(190) Two cats jumped

could not be interpreted as

(191) $\exists E.\exists X.\forall x.(x \prec X \leftrightarrow two(x) \wedge cat(x) \wedge Plural(x)) \wedge jumped(E)$

as a single cat cannot satisfy $two(x)$, nor can it satisfy being plural, as it is single by definition. Instead, those predicates are said to be collective, and make the expression take meaning

(192) $\exists E.\exists X.two(X) \wedge \forall x.(x \prec X \leftrightarrow cat(x)) \wedge Plural(X) \wedge jumped(E)$

where only the *cat* predicate has distributed meaning.

5.2.3 Use of the biconditional

A final point that has to be addressed is the biconditionality of the distributive rule. Consider the predicate

(193) $cat(X)$

It would seem natural to try to interpret this condition on the collection of values X as

(194) $*cat(X) :=$ all X's are cats

. For instance, given a domain of cats $\{c_0, c_1, c_2\}$, all of

(195) a. $cat(\{c_0\})$

b. $cat(\{c_0, c_1\})$

c. $cat(\{c_0, c_1, c_2\})$

could be argued to be true, as in each case, the values of the argument are cats.

What the rule implicitly says is that this one-way implication is not restrictive enough when it comes to handling a clash in sorts of the predicate and argument: all values of the plural argument have to be cats, but also any value that satisfies the singular predicate has to be a value of the argument.

This might seem counterintuitive at first and suspicious to want

(196) $cat(CATS)$

to be true, but not

(197) $cat(SMALL\_CATS)$

where $CATS$ and $SMALL\_CATS$ are the plural constants representing all cats and all small cats, and assuming there is a cat that is not a small cat.

The reason, put simply, is that the existential requirements of the event forms are not sufficient to accurately handle plurality.

To see why, look at the example below:

(198) The blue cat sleeps

The denotation of the determiner phrase is of the form

(199) $[\![\textit{ the blue cat }]\!]_e = \exists x.the(e,x) \wedge blue(x) \wedge cat(x)$

The role of the existential quantifier is to ask the question: does there exist a value in the model that makes the enclosed formula true once the quantified variable is replaced by that value. In this case, can one find a singular value that is also blue and a cat and is the unique relevant such value given the event e?

Now look at

(200) The blue cats sleep

The denotation of the determiner phrase is of the form

(201) $[\![\textit{ the blue cats }]\!]_E = \exists X.the(E,X) \wedge blue(X) \wedge cat(X) \wedge Plural(X)$

Within this constituent certain restrictions are placed on the values X can take, viz. being blue, being cat, and being plural. Those are all intersective properties, i.e. if certain values satisfy the conditions $C_0, ..., C_n$, then conjoining a new condition to the lot means trimming that collection of values that satisfied those previous conditions in accordance with the new condition.

(202) X_{n+1} := values of X_n that satisfy C_{n+1}

The final values of X are then compared to the values of the event E. The comparison method varies from one determiner to another: $some_{ag}$ asks that at least one element be an agent, $every_{ag}$ that all elements be agents, $most_{ag}$ that more than 50% of them be agents, and so on.

As this is a conjunctivist system, none of the predicates have access to what the other predicates stand for; they each apply their own restrictions to their variable arguments. This means that, interpreting a singular predicate with a plural value $P_{sg}(X)$ instead as a one-way conditional

(203) $\forall x.x \prec X \rightarrow P_{sg}(x)$

in a world where 2 out of 3 blue cats are sleeping, would make the evaluation of the sentence *the blue cats sleep* true. Why? Let X in (201) have values those two cats that are sleeping, then every clause in the denotation of (200) is satisfied by picking the event where these only these cats are sleeping

(204) $\exists E.\exists X.the_{agent}(E, X) \wedge [\![\ blue\ cats\ ]\!](X) \wedge sleep(E)$

a. Are the values of X blue cats? Yes

b. Are the values of X agents of the event? Yes

c. Are all values of E sleeping events? Yes

The problem is that the meaning of *the* is like that of *all of the*, but it has no way of actually verifying that the values X it receives cover all blue cats, because it cannot refer to blue cats. Hence if the role of *blue cats* is verifying that the values of the X are blue cats, and not that they are all the possible blue cats, then $the(E, X)$ would output the wrong truth value as it can only test that the values of X, which are only a fraction of the full collection of values that are blue cats in this case, are the agents of the values of the event.

The only thing *the* can do is check if its entity values are agents of the event values, it does not have direct semantic connection with what is inside of the noun phrase, ie. *blue* and *cats*. The expression *blue cats* is then the only part of the sentence that could possibly check any condition about being a blue cat, hence its role should be to find all possible values that are blue cats and not restrict it in any other way. The quantifier will then be the one making sure that the values of the noun phrase relate in a specific way with the values of the event, e.g. whether they are all agents of the event, or that most of them are agents, or that only some of them are agents.

Hence, as opposed to a first-order framework, finding a plural variable that satisfies the condition

is not enough, that variable must be, in a sense, the maximal variable in size that satisfies that condition. Using a biconditional relation allows us to get that requirement.

5.3 Quantification

Quantification has traditionally been analysed using higher-order truth functions verifying certain conditions between sets (Montague, 1974; Partee, 1976; Barwise and Cooper, 1981). For instance

(205) a. $[\![\ every_{functional} \ ]\!] := \lambda P.\lambda Q.P \subseteq Q$

b. $[\![\ some_{functional} \ ]\!] := \lambda P.\lambda Q.P \cap Q \neq \emptyset$

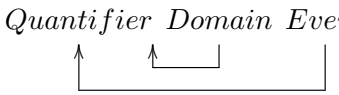
c. $[\![\ most_{functional} \ ]\!] := \lambda P.\lambda Q.|P \cap Q| \geq |\bar{P} \cap Q|$

where $\bar{P}$ is the complement of P, i.e. elements of the domain that are not in P.

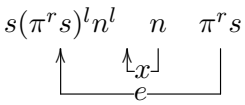
For instance, the denotation of *every* in the sentence *every boy likes baseball* would take in the set of boys and the set of people who like baseball, and verifies that the former is a subset of the latter.

A similar treatment of quantification could be provided within a conjunctivist system using plural variables by simply letting the quantifiers stand for restrictions on the way a certain entity domain is represented in the event, although taking full advantage of the event structure can provide for more interesting analyses.

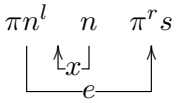
The general representation of a sentence with a (binary) quantifier in subject position then takes the form



which translated in the pregroup framework with annotated contractions looks like



Note that, as information is not passed explicitly from one node to another during pregroup contractions, but is instead shared by both parties, quantifier raising is not even necessary in this framework and the quantifier phrase could instead take the simpler type of a subject π_3, as in either case, event values are shared between the determiner and the verb phrase, which is really what's important in this semantic framework.



Here is a simple example derivation with the accusative *every*:

(206)

every	dog	cried		every dog cried
$\pi_E n^l_X$	n_X	$\pi^r_E s^l_E$	$\rightarrow$	s_E
$every_{ag}(X,E)$	$dog(X)$	$cried(E)$		$every_{ag}(X,E) \wedge dog(X) \wedge cried(E)$

The crux of the problem becomes finding how to interpret the quantifier itself. A translation of the functionist approach could be achieved by defining the *agent* and *every* predicates as:

(207) $Agent_{pl}(E,X) := \forall e.e \prec E \rightarrow \exists x.x \prec X \wedge Agent_{sg}(e,x)$

(208) $every(E,X) := \forall x.x \prec X \rightarrow \exists e.e \prec E \wedge Agent_{sg}(e,x)$

and

(209) $every_{ag}(E,X) := Agent(E,X) \wedge every(E,X)$

On the one hand, the $Agent_{pl}$ predicate restricts the value of the plural event to only those which have agents, and on the other, the $every(E,X)$ restricts the possible entities to those being the agent of some event.

This can be extended to a case with two quantifiers:

every cat kicked some dog

$$\pi_E \; n^l_X n_Y \; \pi^r_{E'} s_{E'} o^l_{E'} \; o_{E''} n^l_Z \; n_W$$
$$every_{ag}(X,E) \wedge cat(Y) \wedge kicked(E') \wedge some_{th}(E'',Z) \wedge dog(W)$$
$$\Rightarrow$$
$$\pi_E \; \pi^r_{E'} s_{E'} o^l_{E'} \; o_{E''}$$
$$every_{ag}(X,E) \wedge cat(X) \wedge kicked(E') \wedge some_{th}(E'',Z) \wedge cat(Z)$$
$$\Rightarrow$$
$$s_E$$
$$every_{ag}(X,E) \wedge cat(X) \wedge kicked(E) \wedge some_{th}(E,Z) \wedge dog(Z)$$

which roughly translates to saying that a possibly plural event of kicking had all and only the cats as agents and some dogs were the theme of those events. The reading where one specific dog was the theme of all those kickings could be reached by modifying the truth conditions of $some_{th}$ from:

(210) $[\![\ some_{th}\ ]\!] = Theme(E, X)\ \wedge\ \exists x.x \prec X \rightarrow \exists e.e \prec E \wedge Theme_{sg}(e, x)$

to

(211) $[\![\ some'_{th}\ ]\!] = Theme(E, X)\ \wedge\ \exists!\ x.x \prec X \rightarrow \exists e.e \prec E \wedge Theme_{sg}(e, x)$

where $\exists!\ x.A[x]$ stands for

(212) $\exists x.A[x]\ \wedge\ \forall y.A[y] \rightarrow y = x$

It is convenient to introduce a new variable in the representation of quantifiers which represents the subcollection of elements of the noun phrase's entity variables that satisfy the condition set on the event. This new variable is completely internal to the quantifier's meaning representation and doesn't affect in any way the structure of the derivations, i.e. it is never free and does not need to get bound.

The following templates can be used to find the truth conditions relative to a quantifier relative to the role it plays in a sentence:

$$\begin{aligned} Q_\theta(E, X) :=&\quad \exists Y.Q(Y, X)\ \wedge\ \forall y.(y \prec Y \rightarrow \exists e.e \prec E \wedge \theta_{sg}(e, y))\ \wedge\ \theta_{pl}(E, Y) \\ \theta_{pl}(E, Y) :=&\quad \forall e.(e \prec E \rightarrow \exists x.(x \prec X \wedge \theta_{sg}(e, x))) \\ \theta_{sg} \in&\quad \{Agent_{sg}, Theme_{sg}, Patient_{sg}, ...\} \\ Q \in&\quad \{every, some, most, two, ...\} \end{aligned}$$

The quantifiers Q can be defined as follows:

(213) a. $[\![\ every\ ]\!](Y, X) :=\ Y = X$

b. $[\![\ some\ ]\!](Y, X) := \emptyset \neq Y \subseteq X$

c. $[\![\ most\ ]\!](Y, X) := |\bar{Y} \cap X| \leq |Y \cap X|$

d. $[\![\ exactly\ two\ ]\!](Y, X)$
$:= \exists y.(y \prec Y \wedge \exists x.(x \prec Y \wedge x \neq y \wedge \nexists z.(z \prec Y \wedge z \neq y \wedge z \neq x))) \wedge Y \subseteq X$

e. $[\![\ at\ least\ two\ ]\!](Y, X) := \exists y.(y \prec Y \wedge \exists x.(x \prec Y \wedge x \neq y)) \wedge Y \subseteq X$

f. $[\![\ my_i\ ]\!](Y, X) := my(X, i) \wedge [\![\ every\ ]\!](Y, X)$

The meaning of a quantifier like $some_{ag}$ now reads:

(214) $[\![\ some_{ag}\ ]\!](E, X) = \exists Y.\emptyset \neq Y \wedge Y \subseteq X \wedge\ \forall y.(y \prec Y \rightarrow \exists e.e \prec E \wedge \theta_{sg}(e, y))\ \wedge\ \theta_{pl}(E, Y)$

i.e. there is a subcollection of the entities X that is not empty and that contains exactly the elements satisfying the event E. Having this extra variable is useful for more complicated quantifiers such as *most* that do not correspond directly to a first-order quantifier.

More complex readings such as the one for

(215) Two cats bit three dogs (each)

where two cats each bit three dogs (as opposed to three dogs in total that were bitten) are not as easily accounted for in this kind of framework, though it still seems to be possible to model them by using a different denotation for the quantifier, e.g.

(216) $\forall x.\exists e.(e \prec E \wedge Agent(e,x))$
$\rightarrow \exists y_1 \exists e_1.Theme(e_1,y_1) \wedge \exists y_2 \exists e_2.Theme(e_2,y_2) \wedge \exists y_3 \exists e_3.Theme(e_3,y_3)$

5.4 Summary

This section was simply meant to give a brief overview of the way plurality and quantification could be handled under Conjunctivism. As shown, the framework developed in the previous section works as well as I did without the plural logic interpretation of variables.

Some questions proper to Conjunctivism still have to be investigated, especially those dealing with quantification, though it is nice to see that some quantifiers can be naturally interpreted as standing for conditions on events and entities related to this event.

Chapter 6

Conclusion

The compositional semantic system defined in this work is an elegant and natural way of defining a semantics for pregroup grammars which relies on light machinery and intuitive operations. The resulting semantic representation follows the event semantics template, where implicit entity and event variables are introduced and used as argument by the different predicates forming the semantic value of the expression. There already existed semantics for pregroup grammars, based on work by Clark, Coecke, and Sadrzadeh (Clark et al., 2008), and work by Preller (Preller, 2005), but the approach here differs in important ways.

In a nutshell, the goal was to provide a system that would allow one to get the semantic representation of an expression in a compositional way, i.e. starting from the semantic values of each of its constituents, have a system that can derive automatically the semantic value of the whole expression. Following the conjunctivist approach was also a requirement, that is, the syntactic concatenation had to correspond to a a conjunction of semantic values. The advantage of this approach is that the role each subexpression plays in the meaning of a sentence is that of an independent condition that can be checked. Semantic values do not get passed around to other semantic values, as is usually the case in formal semantics, where one ends up with a semantic representation that is composed of logical predicates nested into one another.

The main difference between doing semantics in the pregroup framework as opposed to within a more traditional categorial grammar is that pregroup types cannot be uniquely translated into functions. This makes traditional formal semantic techniques useless, as those rely on following a strict order for argument passing. Pregroup types are in a sense simply a stack made of atomic syntactic categories, which can be used and combined from either side, in any order. A good

semantics for a pregroup grammars has to take this into account and have ways of getting access to this information without restricting the syntactic types.

For instance, the way Preller (Preller, 2005) solved this problem was by having the derivational system follow the contraction links and send the information from the base of the contraction arrow — the non-adjoint type — as an input to the tip of the arrow — the adjoint type. Hence, even though pregroup types cannot be represented as functions, one can still manage to recover the argument passing operation of functional semantics.

The system that was developed in this project went in a different direction. Here are its key elements.

- Entity and event variables are directly encoded in the simple types forming the complex syntactic type of a lexical item or expression.

- Those implicit variables are in a way instantiated by the semantic value of an expression: the only possible unbounded variables in an expression's semantic value have to be present in its syntactic type. For instance, $john$ in subject position would have semantic value $Agent(e, john)$ and e in the variable contained in its syntactic type π_e.

- When two expressions concatenate, their syntactic types contract, and their semantic values get conjoined. At the same time, the variables contained in those types that just contracted get unified. What this means is that whatever value these variables might end up taking in the overall representation will have to be the same. For instance, concatenating fat and cat has the consequence that the argument entities in the corresponding $fat(x)$ and $cat(y)$ predicates will have to be the same, i.e. the semantic value of the expression will be of the form $fat(y) \wedge cat(x) \wedge x = y$.

- The syntactic ordering already present in the pregroup framework can be extended to encode transformations in semantic values. This can be useful when introducing grammatical roles. For instance, using a noun phrase as subject, i.e. applying the pregroup ordering rule $\bar{n} \rightarrow \pi$, could be equated with the addition of an $Agent(e, x)$ clause to the semantic value of the noun phrase and a change of implicit variable contained in the syntactic type.

$$\begin{array}{ccc} \bar{n}_x & & \pi_e \\ A[x] & \Rightarrow & Agent(e, x) \wedge A[x] \end{array}$$

Bibliography

Ajdukiewicz, Kazimierz. 1967. Die syntaktische konnexität. In *Polish logic 1920-1939*, 207–231. Oxford: Oxford University Press. Translated from *Studia Philosophica*, 1, 1-27.

Bach, Emmon. 1986. The algebra of events. *Linguistics and Philosophy* 9:5–16.

Baker, Mark. 1988. *Incorporation: A theory of grammatical function changing*. Chicago University Press.

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

Barwise, John, and Robin Cooper. 1981. Generalized quantifiers and natural language. *Linguistics and Philosophy* 4:159–219.

van Benthem, Jan. 1991. *Language in action*. North Holland. 348 p.

van Benthem, Johan. 1989. Polyadic quantifiers. *Linguistics and Philosophy* 12:437–464.

van Benthem, Johan, and Alice ter Meulen, ed. 2011. *Handbook of logic and language*. Elsevier.

Boolos, George. 1984. To be is to be the value of a variable (or to be some values of some variables). *Journal of Philosophy* 81:430–449.

Buszkowski, W. 2003a. Lambek calculus and substructural logics. *Linguistics Analysis* 36:15–48.

Buszkowski, W. 2003b. Sequent systems for compact bilinear logic. *Mathematical Logic Quarterly* 49:467–474.

Carpenter, B. 1998. *Type-logical semantics*. MIT Press.

Carpenter, Bob. 1992. *The logic of typed features structures*. Cambridge Tracts in Computer 32. Cambridge University Press. 270 p.

Carstairs-McCarthy, Andrew. 1999. *The origins of complex language: An inquiry into the evolutionary beginnings of sentences, syllables, and truth*. Oxford University Press.

Casadio, C., and A. Kiślak-Malinowska. 2014a. Computing italian clitic and relative clauses with tupled pregroups. *Proceedings NLCS 2014* .

Casadio, C., and A. Kiślak-Malinowska. 2014b. Computing italian clitic and relative clauses with tupled pregroups. In *Proceedings NLCS 2014*.

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

Casadio, C., and J. Lambek, ed. 2008. *Computational algebraic approaches to natural language*. Polimetra.

Casati, R., and A. Varzi, ed. 2006. *Events*. Aldershot: Dartmouth.

Castañeda, Hector-Neri. 1967. Comments. In *The logic of decision and action*, ed. N. Rescher. University of Pittsburgh Press.

Champollion, Lucas. 2010. Quantification and negation in event semantics. In *Baltic International Yearbook of Cognition, Logic and Communication*, ed. MIchael Glanzberg Barbara Partee and Jurgis Skilters, volume 6, 1–23. Manhattan, KS: New Prairie Press.

Champollion, Lucas. 2015. The interaction of compositional semantics and event semantics. *Linguistics and Philosophy* 38:31–66.

Chomsky, Noam. 1995. *The minimalist program*. MA: MIT Press.

Church, A. 1932. A set of postulates for the foundation of logic. *Annals of Mathematics* 33:346–366.

Church, A. 1940. A formulation of the simple theory of types. *Journal of Symbolic Logic* 5:56–68.

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

Clark, S., B. Coecke, and M. Sadrzadeh. 2010. Mathematical foundations for a compositional distributional model of meaning. *Linguistic Analysis* 36.

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

Davidson, Donald. 1967. The logical form of action sentences. *Synthese* .

Dekker, Paul. 2003. Meanwhile within the frege boundary. *Linguistics and Philosophy* 26:547–556.

Dowty, David. 1991. Thematic proto-roles and argument selection. *Language* 67:547–619.

Dowty, David. 2000. The dual analysis of adjuncts/complements in categorial grammar. *ZAS Papers in Linguistics* 17.

Efrat Jaeger, Shuly Wintner, Nissim Francez. March 2005. Unification grammars and off-line parsability. *Journal of Logic, Language and Information* 14:199–234.

van Eijck, Jan. 2005. Normal forms for characteristic functions on n-ary relations. *Journal of Logic and Computation* 15:85–98.

Fodor, Jerry. 1970. Three reasons for not deriving "kill" from "cause to die" 4:429–438.

Frege, Gottlob. 1967. Begriffsschrit. In *From frege to Gödel*, ed. Jean van Heijenoort. Cambridge, MA: Harvard University Press.

Gaudreault, Gabriel. 2015. Bidirectional functional semantics for pregroup grammars. In *NLCS'15. Third Workshop on Natural Language and Computer Science*, ed. Makoto Kanazawa, Lawrence S. Moss, and Valeria de Paiva, volume 32 of *EPiC Series in Computer Science*, 12–28.

Gentzen, Gerhard Karl Erich. 1934. Untersuchungen über das logische schließen i. *Mathematische Zeitschrift* 39:176–210.

Gierasimczuk, N., and J. Szymanik. 2009. Branching quantification v. two-way quantification. *Journal of Semantics* 26:367–392.

de Groote, Philippe. 2001. Towards abstract categorial grammars. In *Association for Computational Linguistics 2001*.

de Groote, Philippe, and Yoad Winter. 2015a. A type-logical account of quantification in event semantics. In *New Frontiers in Artificial Intelligence*, ed. Tsuyoshi Murata, Koji Mineshima, and Daisuke Bekki, 53–65.

de Groote, Philippe, and Yoad Winter. 2015b. A type-logical account of quantification in event semantics. *New Frontiers in Artificial Intelligence* 53–65.

Hale, Kenneth, and Samuel Keyser. 1993. On argument structure and the lexical expression of syntactic relations. In *The view from building 20: Essays in honor of sylvain bromberger*, ed. Kenneth Hale and Samuel Keyser. MIT Press.

Hale, Kenneth, and Samuel Keyser. 2001. *Prolegomenon to a theory of argument structure*, volume 39 of *Linguistics Inquiry Monograph*. MIT Press.

Hintikka, Jaakko. 1973. Quantifiers vs quantification theory. *Dialectica* 27:329–358.

Husserl, Edmond. 1900. *Logische untersuchungen. erste teil: Prolegomena zur reinen logic*.

Kamp, Hans, and Uwe Reyle. 1993. *From discourse to logic*. Kluwer.

Keenan, Edward. 1987. Unreducible n-ary quantification in natural language. 109–150. Reidel, Dordrecht.

Keenan, Edward. 1992. Beyond the frege boundary. *Linguistics and Philosophy* 2:199–221.

Krifka, Manfred. 1989. Nominal reference, temporal constitution and quantification in event semantics. In *Semantics and contextual expression*, ed. Renate Bartsch, Johan van Benthem, and P. van Emde Boas, 75–115. Foris Publications.

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

Lambek, J. 1993. From categorial grammar to bilinear logic. In *Substructural logics*, ed. P. Schroeder-Heister K. Dožen, 207–237. Oxford, Clarenton Press.

Lambek, J. 1999. Type grammar revisited. In *Logical aspects of computational linguistics*, ed. G. Perrier A. Lecomte, F. Lamarche, volume 1582 of *Lecture Notes in Computer Science*, 1–27. Springer Berlin Heidelberg.

Lambek, J. 2008. *From word to sentence: A computational algebraic approach to grammar*. Polimetra.

Landman, Fred. 1992. The progressive. *Natural Language Semantics* 1:1–32.

Lasersohn, Peter. 2006. Event-based semantics. In *Encyclopedia of Language and Linguistics*. Amsterdam: Elsevier.

Meinke, Karl, and John Tucker. 1993. *Many-sorted logic and its applications*. Wiley.

Moens, Marc, and Mark Steedman. 1988. Temporal ontology and temporal reference. *Computational Linguistics* 14.

Montague, Richard. 1974. *Formal philosophy: Papers of richard montague*. New Haven, CT: Yale University Press. Richmond H. Thomason, ed.

Moortgat, Michael. 1989. Categorial investigations. Doctoral Dissertation, Universiteit van Amsterdam.

Moot, R., and C. Rétoré. 2012. *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.

Morrill, Glyn. 1994. *Type-logical grammar*. Dordrecht: Kluwer.

Mourelatos, Alexander. 1978. Events, processes, and states. *Linguistics and Philosophy* 2:415–434.

Nouwen, Rick. 2016. Plurality. In *The cambridge handbook of formal semantics*, ed. Maria Aloni and Paul Dekker. Universiteit van Amsterdam.

Parsons, Terence. 1988. Underlying states in the semantical analysis of english. In *Proceedings of the Aristotelian Society*, volume 88, 13–30.

Parsons, Terence. 1989. The progressive in english: Events, states and processes. *Linguistics and Philosophy* 12:213–241.

Parsons, Terence. 1990. *Events in the semantics of english*. The MIT Press.

Partee, Barbara, ed. 1976. *Montague grammar*. New York: Academic Press.

Partee, Barbara. 2000. Building statives. Berkeley Linguistic Society.

Partee, Barbara. 2006. Do we need two basic types? In *40-60 puzzles for manfred krifka*, ed. H-M Gaertner, R.M. Regine Eckardt, and B. Stiebels. Berlin.

Pentus, Mati. 2003. Lambek calculus is np-complete. *Theoretical Computer Science* 357:186–201.

Pietroski, Paul. 2003. Quantification and second-order monadicity. *Philosophical Perspectives* 17:259–298.

Pietroski, Paul. 2005. *Events and semantic architecture*. Oxford University Press.

Pietroski, Paul. 2006. Interpreting concatenation and concatenates. *Philosophical Issues* 16:221–245.

Pietroski, Paul. 2008. Minimalist meaning, internalist interpretation. *Biolinguistics* 4:317–341.

Pollard, Carl, and Ivan Sag. 1994. *Head driven phrase structure grammars*. Stanford, CA: CSLI Publications.

Preller, A. 2005. Category theoretic semantics for pregroup grammars. In *Logical aspects of computational linguistics*, volume 3492 of *Lecture Notes in Computer Science*, 238–254. Springer Berlin Heidelberg.

Preller, A. 2007. Toward discourse representation via pregroup grammars. *Journal of Logic, Language and Information* 16:173–194.

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

Rothstein, Susan. 2008. Verb classes and aspectual classification. In *Structuring events: A study in the seamntics of lexical aspect*. Oxford, UK: Blackwell Publishing Ltd.

Schein, Barry. 1993. *Plurals and events*. MIT Press.

Schein, Barry. 2002. Events and the semantic content of thematic relations. In *Logical form and language*, ed. Gerhard Preyer Georg Peter, 263–344. Oxford University Press.

Stabler, Edward. 1997. Derivational minimalism. In *Logical Aspects of Computational Linguistics*, ed. Christian Retoré, Lecture Notes in Computer Science, vol 1328, 68–95.

Stabler, Edward. 2013. Two models of minimalist, incremental syntactic analysis. *Topics in Cognitive Science* 5:611–633.

Steedman, Mark. 2000. *The syntactic process*. The MIT Press.

Thomas, Wolfgang. 1996. Languages, automata, and logic. In *Handbook of formal languages*, 389–455. Springer.

Vendler, Zeno. 1957. Verbs and times. *The philosophical review* 66:143–160.

Vijay-Shanker, K., and David Weird. 1994. The equivalence of four extensions of context-free grammar. *Mathematical Systems Theory* 27:511–546.

Winter, Yoad, and Joost Zwarts. 2011a. Event semantics and abstract categorial grammar. In *The mathematics of language*, ed. Makoto Kanazawa, Andrs Kornai, Marcus Kracht, and Hiroyuki Seki, volume 6878 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Winter, Yoad, and Joost Zwarts. 2011b. On the event semantics of nominals and adjectives: The one-argument hypothesis. In *Proceedings of Sinn und Bedeutung 16*, ed. Ana Aguilar Guevara, Anna Chernilovskaya, and Rick Nouwen, volume 2.

Zwarts, Joost. 2005. Prepositional aspect and the algebra of paths. *Linguistics and Philosophy* 28:739–779.