Parsing with Categorial Grammars Workshop
ESSLLI 2009
Bordeaux, France
Book of Abstracts

July 20th to 24th, 2009

Schedule

- Day 1

  14:00 - 14:30 Opening Session

  14:30 - 15:30 Invited Speaker: Johan Bos - Semantic Parsing with CCG in Real-World Applications

- Day 2

  14:00 - 14:30 Yury Savateev - Product-free Lambek Calculus is NP-Complete

  14:30 - 15:00 Glyn Morrill - Memoising Proof Net Theorem-Proving

  15:00 - 15:30 Matteo Capelletti - Parsing with Non-associative Lambek Grammars

- Day 3

  14:00 - 14:30 Timothy Fowler - Parsing CCGbank with the Lambek calculus

  14:30 - 15:00 Matteo Capelletti and Fabio Tamburini (UCG) - Polymorphic Categorial Grammars: expressivity and computational properties

  15:00 - 15:30 Ruket Cakici and Mark Steedman - A Wide-Coverage Morphemic CCG Lexicon for Turkish

- Day 4

  14:00 - 14:30 Annie Foret and Denis Bchet - PPQ : a pregroup parser using majority composition

14:30 - 15:00 Pierre Lison - A Method to Improve the Efficiency of Deep Parsers with Incremental Chart Pruning

15:00 - 15:30 Guillaume Bonfante, Bruno Guillaume and Mathieu Morey - Word Order Constraints for Lexical Disambiguation of Interaction Grammars

- Day 5

  14:00 - 15:00 Invited Speaker: Makoto Kanazawa - Datalog as a Uniform Framework for Parsing and Generation

  15:00 - 15:30 Alexandre Dikovsky - Towards Wide Coverage Categorial Dependency Grammars

Invited Speaker: Johan Bos

Semantic Parsing with CCG in Real-World Applications

Formal approaches in semantics have long been restricted to small or medium-sized fragments of natural language grammars. We argue that categorial grammars are very suitable for reaching wide-coverage grammars for semantic interpretation because they are lexicalised and have a transparant mapping between syntactic categories and semantic types. We substantiate this claim by taking Clark & Curran's statistical CCG parser (trained on Hockenmaier's CCGbank for English) and show how to build a principled syntax-semantic interface situated in Discourse Representation Theory. Despite the theoretical beauty of categorial grammars, not all is hunky-dory, and we will illustrate both the attractive and the less attractive aspects that arise in developing practical grammar formalisms. We will illustrate the resulting system for robust text interpretation in applications such as question answering and textual inference.

Invited Speaker: Makoto Kanazawa

Datalog as a Uniform Framework for Parsing and Generation

Various grammar formalisms with "context-free" derivations, including multiple context-free grammars, tree-adjoining grammars, and context-free tree grammars, can be straightforwardly represented by Datalog programs (i.e., logic programs without function symbols), if strings and trees are viewed as first-order structures or "databases". This is a generalization of the well-known definite clause grammar representation of context-free grammars and underlies CYK-style tabular parsing methods for these grammar formalisms. Moreover, when these grammars are coupled with Montague-style semantics, where meanings are represented by typed lambda terms, the problem of "tactical generation" or "surface realization" also reduces to Datalog query evaluation, provided that the lambda terms associated with the grammar rules are "almost linear". (The correctness of this reduction can be rigorously proved using a certain relaxation of second-order abstract categorial grammars, as shown by Kanazawa 2007.)

The Datalog representation allows a unified view of algorithmic and complexity-theoretic issues surrounding parsing and generation, in abstraction from particular grammar formalisms. In this talk, I will look in some detail at the complexity-theoretic consequences of the Datalog representation on parsing and generation.

Word Order Constraints for Lexical Disambiguation of Interaction Grammars

Guillaume Bonfante, Bruno Guillaume and Mathieu Morey
LORIA - Nancy-Université - INRIA Nancy Grand-Est
{guillaume.bonfante, bruno.guillaume, mathieu.morey}@loria.fr

Abstract

We propose a new method to perform lexical disambiguation of Interaction Grammars. It deals with the order constraints on words. Actually, the soundness of the method is due to an invariant property of the parsing of an Interaction Grammar. We show how this invariant can be computed statically from the grammar.

1 Introduction

Interaction Grammars are a lexicalized grammatical formalism. They share with Categorial Grammars the idea that words are composed of syntactic constituents with a notion of polarity. Some constituents are unsaturated: they "wait" for some resources and provide some other ones. Interaction Grammars also have, as Categorial Grammars, a mechanism to cope with the linear order on words.

We show in this paper that ordering constraints can be used to partially disambiguate the words of a sentence. But, first of all, let us state some of our wills. First, the issue considered here is to be thought in the context of syntactic analysis. We do not want to use statistical methods for lexical disambiguation since an error at that point cannot be recovered at the parsing step. Consequently, given a sentence, we accept to have more than one lexical tagging for it, as long as we can ensure to have the good ones (when they exist!).

Now, since we have to consider all possible lexical taggings to find the right ones, there is an immediate problem of complexity. Knowing that a word has typically about 10 corresponding lexical descriptions, for a short sentence of 10 words, we get 10^{10} possible taggings. It is not reasonable to treat them individually.

To avoid it, it is convenient to use an automaton to represent the set of all paths. This automaton has linear size with regard to the initial lexical ambiguity. The idea of using automata is not new. In particular, methods based on Hidden Markov Models (HMM) use such a technique for part-of-speech tagging [3, 4]. Using automata, one may conceive dynamic programming procedures, and consequently benefits from an exponential temporal speed up, together with the space one.

2 Interaction Grammars

We give here a very short and simplified description of IG and then, an example to illustrate them at work; we refer the reader to [2] for a complete and detailed presentation.

The final structure, used as output of the parsing process, is an ordered tree called **parse tree** (PT).

An example of a PT is given in Figure 1, on the right. A PT for a sentence contains the words of the sentence or the empty word ϵ in its leaves (the left-right order of the tree leaves follows the left-right order of words in the input sentence). The internal nodes of a PT represent the constituents of the sentence. The morpho-syntactic properties of these constituents are described with feature structures (only the category is shown in the figure).

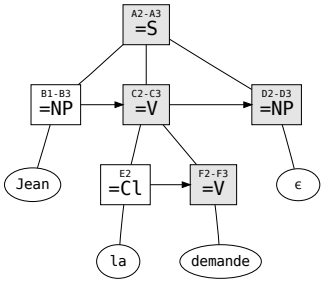


Figure 1: The PT of *"Jean la demande."* [John asks for it.]

As IG use the Model-Theoretic Syntax (MTS) framework, a PT is defined as the model of a set of constraints. Constraints are defined at the word level: words are associated to a set of constraints formally described as a **polarized tree description** (PTD). A PTD is a set of nodes provided with relations between these nodes. In Figure 2, the three PTDs given on the left are used to build the model above. The relations used in the PTDs are: dominance (lines) and immediate sisterhood (arrows). Nodes represent syntactic constituents and relations express structural dependencies between these constituents; moreover, nodes carry a polarity (polarities are $\{+, -, =, \sim\}$) which expresses a saturation constraint.

Now, we define a PT to be a **model** of a set of PTDs if there is a surjective function $\mathcal{I}$ from nodes of the PTDs to nodes of the PT such that:

- relations in the PTDs are realized in the PT: if M is a daughter (resp. immediate sister) of N in some PTD then $\mathcal{I}(M)$ is a daughter (resp. immediate sister) of $\mathcal{I}(N)$;
- each node N in the PT is saturated: the composition of the polarities of the nodes in $\mathcal{I}^{-1}(N)$ with the associative and commutative rule given in Table 3 is $=$;
- the feature structure of a node N in the PT is the unification of the feature structures of the nodes in $\mathcal{I}^{-1}(N)$.

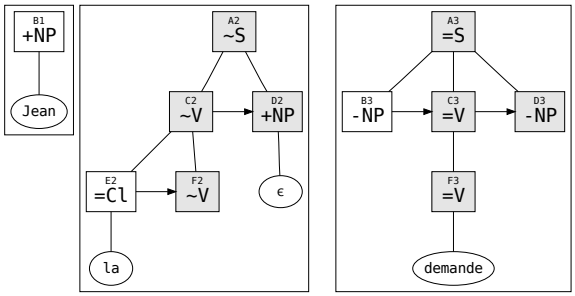


Figure 2: PTDs for the sentence *"Jean la demande."* [John asks for it.]

One of the strong points of IG is the flexibility given by the MTS approach: PTDs can be partially superposed to produce the final tree (superposition is limited in usual CG or in TAG for instance). In our example, the four grey nodes in the PTD which contains *"la"* are superposed to the four grey nodes in the PTD which contains *"demande"* to produce the four grey nodes in the model. In the full IG formalism, relations between nodes can be underspecified, for instance a PTD can impose a node to be an ancestor of another one without constraining the length of the path in the

model. In full IG, polarities are linked to features, not to nodes. A node can contain several polarities. The methods we present here can be straightforwardly extended to full IG (with unessential technical details).

An IG is made of:

- A finite set $\mathcal{W}$ of words;
- A finite set $\mathcal{G}$ of PTDs (without the word attached to them);
- A function $\ell : \mathcal{W} \longrightarrow \mathcal{P}(\mathcal{G})$ which associates words with set of PTDs.

	$\sim$	$-$	$+$	$=$
$\sim$	$\sim$	$-$	$+$	$=$
$-$	$-$		$=$	
$+$	$+$	$=$		
$=$	$=$			

Figure 3: Polarity composition

Now, to parse a sentence $S = w_1 \dots w_n$, we have to:

- first, for each w_i, choose one of the PTDs $d_i \in \ell(w_i)$ (we call **lexical tagging** a choice $\{d_1, \dots, d_n\}$ of one PTD for each word of the sentence);
- find a parse tree which is a model of the set of PTDs of the lexical tagging.

3 The Left-Right Principle

Computing a model of a sentence from a lexical tagging requires to saturate all the polarity constraints of its PTDs. To build a model, each node which is not saturated (with a polarity $+$, $-$ or $\sim$) has to be merged with its "companions", namely nodes from other PTDs that will saturate it.

Now let us take a look at the grammar, independently of any sentence, and try to find the "potential companions" of a given unsaturated node. Actually, as our grammar models word order constraints, it can be the case that using a node M to saturate a node N requires the PTD corresponding to M to be on the left (resp. on the right) of N. Therefore, for each unsaturated node N of the grammar $\mathcal{G}$, we can enumerate all of its potential companions using two possibly overlapping lists: its left potential companions (written $LPC(N)$) and its right potential companions (written $RPC(N)$). By extension, we say that $d \in LPC(N)$ (resp. $d \in RPC(N)$) for some PTD d whenever $\exists M \in d : M \in LPC(N)$ (resp. $\exists M \in d : M \in RPC(N)$).

Observe that constructing the LPC and the RPC sets can be done independently from any sentence, it is a property of the grammar which can be computed from the grammar itself.

Let us consider a sentence $w_1 w_2 \cdots w_n$ and one of its lexical taggings $d_1 d_2 \cdots d_n$. Suppose that there is one node $N \in d_i$ for some i for which there is neither some $d_j \in LPC(N)$ with $j < i$ nor some $d_k \in RPC(N)$ with $k > i$. Then, without performing deep parsing, we can state that such a lexical tagging has no model. So, it is a necessary condition of the success of parsing that there is no such node. We call this the Left-Right Principle.

4 Implementation of the Left-Right Principle with automata

We have seen above that the Left-Right principle applies to lexical taggings. As a matter of fact, in this section we keep the promise we made in the introduction of this paper: we show that it can be computed by means of automata, saving space and time. Actually, we propose two implementations of the Left-Right principle, an exact one and an approximate one. The latter is really fast and can be used as a first step before applying the first one.

4.1 Exact Left-Right disambiguation (ELR)

Given a sentence $w_1 \cdots w_n$, a PTD $d \in \ell(w_i)$ and a node N of d, we can build the *companionship automaton* $\mathcal{A}(w_i, d, N)$ for the sentence. It represents the saturation state of the polarity constraint corresponding to N after each choice of a PTD for a word.

Each state of $\mathcal{A}(w_i, d, N)$ is labelled with a couple (j, x), with j the position of the last considered word and x the saturation state (`Open` or `Close`). A state is labelled with `Close` when all of its incoming paths fulfill the polarity constraint of N. Otherwise the state is labelled with `Open`.

More formally, $\mathcal{A}(w_i, d, N)$ is defined as follows. States are a subset of $\mathbb{N} \times \{\texttt{Open}, \texttt{Close}\}$. Transitions $(j-1, x) \xrightarrow{d'} (j, y)$ are labelled by $d' \in \ell(w_j)$. The value of y is determined in the following way :

- if $(j = i) \wedge (d = d')$ then $y = x$,
- if $(j = i) \wedge (d \neq d')$ then $y = \texttt{Close}$,
- if $(j < i) \wedge (d' \in LPC(N))$ then $y = \texttt{Close}$,
- if $(j > i) \wedge (d' \in RPC(N))$ then $y = \texttt{Close}$,
- otherwise $y = \texttt{Open}$.

The initial state is $(0, \texttt{Open})$ and the unique accepting final state is $(n, \texttt{Close})$.

For every word w_i of the sentence, we construct the companionship automaton $\mathcal{A}(w_i, d, N)$ of every polarized node N in every PTD $d \in \ell(w_i)$. The intersection of these automata represents all the possible lexical taggings of the sentence which respect the Left-Right Principle. That is, we output:

$$\bigcap_{1 \leq i \leq n,\ d \in \ell(w_i),\ N \in d} \mathcal{A}(w_i, d, N)$$

Let us say that for the sentence *"Jean la demande."*, the possible PTDs are those described in Figure 4: $\ell(\text{"}Jean\text{"})$ ={Jean_NP}, $\ell(\text{"}la\text{"})$ ={la_Det, la_Clit, la_CN}, $\ell(\text{"}demande\text{"})$ ={demande_V, demande_CN}. Then Figure 5 represents the automaton $\mathcal{A}(\text{"}demande\text{"}, \text{demande\_V}, \text{D3})$.

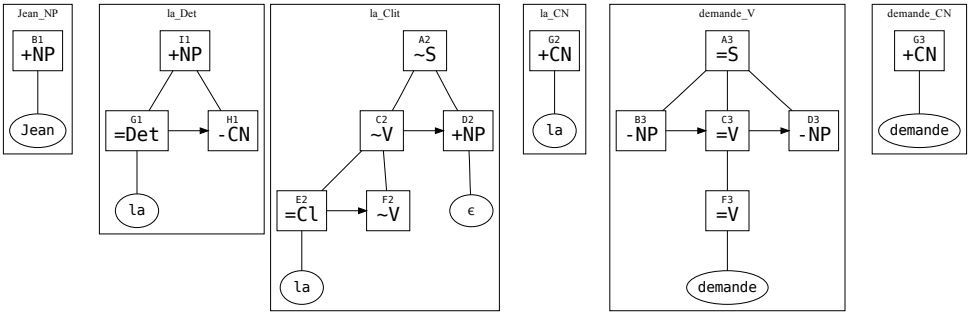


Figure 4: Possible PTDs for the sentence *"Jean la demande."*

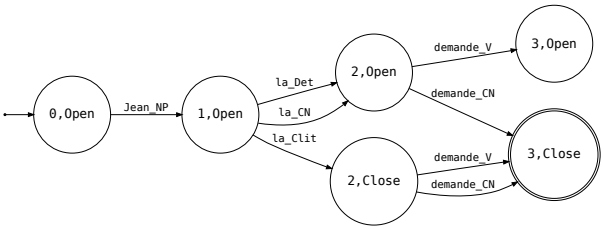


Figure 5: $\mathcal{A}$("*demande*", demande_V, D3) for the sentence *"Jean la demande."*

4.2 Quick and dirty approximation (QLR)

The issue with the previous algorithm is that it involves a large number of automata (actually $O(n)$) where n is the size of the input sentence. Each of these automata has size $O(n)$. The theoretical complexity of the intersection is then $O(n^n)$. Sometimes, we face the exponential. So, let us provide an algorithm which approximates the Principle.

Again, we consider a sentence $w_1 \cdots w_n$. Suppose that for some $d \in \ell(w_i)$ and some $N \in d$, there is no $d' \in LPC(N)$ with $d' \in \ell(w_j), j < i$ nor some $d'' \in RPC(N)$ with $d'' \in \ell(w_k), k > i$. Then lexical taggings containing d at position i have no model, hence, d can be removed.

This can be computed by a double-for loop: for each node N of a PTD $d \in \ell(w_i)$ in the sentence, verify that there is a companion node $M \in d'$, $d' \in \ell(w_j)$ for it. If it is not the case, simply remove the lexical choice d for the word w_i. Observe that the cost of this algorithm is $O(n^2)$.

Note that one must iterate this algorithm until a fix-point is reached. Indeed, removing a PTD which serves as a potential companion breaks the verification. Nevertheless, since for each step before the fix-point is reached, we remove at least one PTD, we iterate the double-for at most $O(n)$ times. The complexity of the whole algorithm is then $O(n^3)$.

Let us see on an example the difference between the two algorithms ELR and QLR. Take a very simple grammar, with a single word w associated to two PTDs d_1, d_2 such that $RPC(d_1) = LPC(d_1) = \{d_2\}, LPC(d_2) = RPC(d_2) = \{d_1\}$[1]. For the sentence ww, the algorithm QLR keeps the four lexical taggings d_1d_1, d_1d_2, d_2d_1 and d_2d_2, whereas ELR only keeps d_1d_2 and d_2d_1.

5 Experimental results

We present here some results obtained[2] with our methods. Our test corpus is composed of 189 sentences (with a mean length of 10 words) extracted from the French newspaper *"Le Monde"*. Our linguistic resources are a French IG [5] and a lexicon built from freely available French resources. In the table below, we give the filtering time for all sentences and filtering ratio for grammatical[3] sentences.

The filtering methods we consider are: POL (as a baseline) which uses a global filter based on polarity counting (described in [1]); QLR is the method described in 4.2 and ELR is described

[1]For instance, take the PTDs made of one node polarized $+N$ for d_1 and $-N$ for d_2.

[2]These results are obtained with a PC (Pentium® D930, 3.0Ghz, 4Go).

[3]Here grammatical means that they are accepted by the grammar.

in 4.1. Note that we have not reported experiments about the ELR method alone because the filtering time is too high for some sentences.

Values in the table are percentiles. For instance the value 1.20s (in the box) means that with the ELR+POL filtering, 75% of the 189 sentences are filtered in 1.20s **at most**. The ratio $3.06 \cdot 10^5$ (also in a box) means that 85% of the 133 sentences have a filtering ratio of $3.06 \cdot 10^5$ **at least**.

		POL	QLR	QLR + POL	ELR + POL
Filtering time	50%	0.38s	0.03s	0.07s	0.11s
for all sentences	75%	2.84s	0.07s	0.38s	1.20s
(189 sentences)	85%	9.83s	0.11s	0.98s	5.92s
Filtering ratio for	50%	$2.86 \cdot 10^5$	$8.06 \cdot 10^3$	$5.76 \cdot 10^7$	$1.70 \cdot 10^8$
grammatical sentences	75%	$3.07 \cdot 10^4$	$9.68 \cdot 10^2$	$1.01 \cdot 10^6$	$3.24 \cdot 10^6$
(133 sentences)	85%	$1.99 \cdot 10^4$	$2.66 \cdot 10^2$	$3.06 \cdot 10^5$	$7.52 \cdot 10^5$

It is clear from the first 2 experiments that the QLR is much more efficient (85% of the sentences can be filtered in less than 0.11s) but the ratio is lower than the baseline. The combination of the two methods (QLR+POL) greatly improves the baseline both in filtering time and ratio. The last experiment is more time consuming (bigger automata are built) but it is still usable in practice and shows the higher impact that can be reached with our methods (number of taggings divided by at least $7.52 \cdot 10^5$ for 85% of the sentences).

References

[1] G. Perrier G. Bonfante, B. Guillaume. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *Proceedings of CoLing 2004*, 2004.

[2] Bruno Guillaume and Guy Perrier. Interaction Grammars. Research Report RR-6621, INRIA, 2008.

[3] Julien Kupiec. Robust Part-of-Speech Tagging Using a Hidden Markov Model. *Computer Speech and Language*, 6(3):225–242, 1992.

[4] Bernard Merialdo. Tagging English Text with a Probabilistic Model. *Computational linguistics*, 20:155–157, 1994.

[5] G. Perrier. A french interaction grammar. In *proceedings od the 6th International Conference on Recent Advances in Natural Language Processing*, pages 463–467, Borovets, Bulgaria, 2007.

A Wide-Coverage Morphemic CCG Lexicon for Turkish

Ruket Çakıcı
School of Informatics
University of Edinburgh, UK
and
Computer Engineering, METU, Ankara, Turkey
r.cakici@sms.ed.ac.uk

Mark Steedman
School of Informatics
University of Edinburgh, UK
steedman@inf.ed.ac.uk

Abstract

In this paper we present a method to induce a wide-coverage morpheme-cluster based CCG lexicon from Turkish dependency treebank. We use the sub-word level dependencies in the Turkish dependency treebank, and show that this lexicon performs better than its lexemic counterpart in explaining morphosyntactic behaviour in Turkish. It also aims to solve the data sparseness by allowing better statistics for morphologically decomposed words when compared to fully inflected counterparts.

1 Introduction

Turkish is a free word order language with rich agglutinating morphology. Not only case, tense and number, but also modality, polarity, voice and even relativisation are achieved through morphology. Agglutinating languages may have very complex word forms. There are 231,818 morphemes including punctuation for 53,796 tokens in the Turkish treebank (Oflazer et al., 2003; Atalay et al., 2003). This corresponds to 4.31 morphemes per word including the stem itself. This means that for a fixed amount of data it is likely that some of the inflected (or derived) forms of the words will never be seen. When building language models, languages with complex morphology require either enormous amounts of data or generalisation of some sort. We show in this paper how we generalise the CCG lexicon in Çakıcı (2005) using morphological clusters as lexical entities in Turkish.

2 Data

We use the Turkish dependency treebank to extract the CCG lexicon. The treebank consists of 5670 sentences [1]. It is annotated with surface dependencies together with the morphological structure of each word. Morphological structure of a word consists of sub-word level entitites called "inflectional groups" (IGs) that are divided by derivational boundaries. Relativisation and subordination are also represented as instances of morphological derivation thus they constitute a different IG than the verb stem they are attached to.

	wb	IGb
tokens	53796	64992
token excl. punc.	43426	54662
Avg. sent. length excl. punc.	7.74	9.72
Avg. sent. length incl. punc.	9.57	11.56
Avg. tags/token	4.31	3.57

Table 1: Numbers from the Turkish treebank

Table 1 gives IG-based and word-based statistics for the treebank. The average sentence length with and without punctuation are given. Punctuation constitutes about 10K of the tokens. The average number of IGs per word is 1.26.

3 Morphemic Lexicon

In the rest of the paper, the terms "IG-based lexicon" and "morphemic lexicon" will be used interchangeably. The Turkish relativiser is considered to be a deverbaliser that makes adjectives out of verbs. An example is shown below.

(1)

Ödev-i	bitir-en	çocuk	uyu-du.
hw-Acc	finish-Rel	child	sleep-Past.

The child who finished the homework slept.

We argue that the relativiser morpheme here should have scope over the whole VP following Bozşahin (2002). For instance in (1) the bracketing should be as shown in (2a) rather than (2b). Our work is motivated by solving such problems.

(2) a. [[[[Ödev-i bitir]-en] çocuk] uyu-du]
hw-Acc finish-Rel child sleep-Past

[1] Note that this is the number after the correction of tokenisation errors

b. [[[[Ödev-i] bitir-en] çocuk] uyu-du]
hw-Acc finish-Rel child sleep-Past

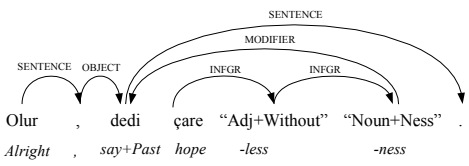
4 Lexicon Induction

The lexicon induction algorithm takes IG-based dependency structures as input and creates CCG categories for every token. The first step is translating the dependency graphs into IG-based dependency graphs. After this step, these graphs go through pre-processing stages to correct annotation errors and separate morphemes that usually have phrasal scope. Finally, CCG categories are assigned to IGs. Details of these stages are explained in the following sections.

4.1 The morphemic dependency structure

The difference between a word-based dependency graph and the IG-based dependency graph arises when there are multiple IGs in a word. When creating the morpheme based dependency graphs if there is more than one IG in a word, we make each of them depend on the IG immediately to the right and make the outward dependency emanate from the last IG to the IG that it depends on in its head word. The dependencies between the internal IGs of a word are labelled INFGR. An example is shown in (3). This method is similar to the CoNNL 2006 shared task conversion (Buchholz and Marsi, 2006). However, instead of replacing IGs with the underscore character, we give them names constructed from the tags they contain. Çakıcı and Baldridge (2006) show that using both the stems and the rest of the word form as features give state-of-the-art results, this is an attempt to simulate this partitioning in order to represent the morphological information.

(3)



T: Olur, dedi çaresizlikle
E: Alright , he said, hopelessly.

The data is preprocessed in order to correct some systematic mistakes about conditionals and zero-morphemes involved in copula constructions. Case morphemes are also represented as separate lexical entities in the dependency structure whenever they have phrasal scope.

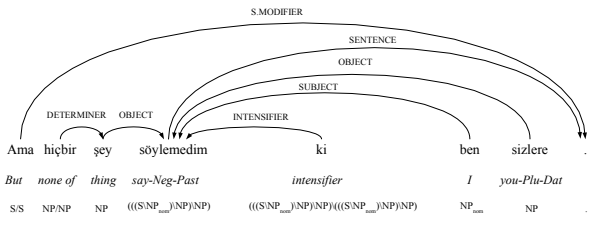
```
if IGs>1
    stemcat = find_stem_cat();
    realcat = find_orig_cat();
    ig[next] = realcat\result(stemcat);
```

Figure 1: Morphemic lexicon induction for words with more than one IG

4.2 Algorithm

After translation of dependency graphs and preprocessing the graphs IGs are assigned CCG categories. The lexicon induction algorithm is based on the one that is described in Çakıcı (2005). The difference is when a word has multiple IGs. This is shown in Figure 1. The dependencies are traversed from the head to the dependents in a recursive manner while constructing the CCG categories.

If head is the stem of the word, we assign the category of the stem depending on the part-of-speech tags and give the (last) IG $X\backslash Cat$ where X is the result category of the stem after taking its arguments, if necessary, and *Cat* is the category it would have been assigned given its label. Note that in Çakıcı (2005) word-based lexical categories assigned to a word are determined by the word's relation to its surface-syntactic head in the treebank i.e. its dependency label. However, in morphemic lexicon induction, the stems are assigned categories dependending on their parts-of-speech, and the rest depending on their labels. Williams (1981) argues that the final IG acts as the head of the whole word or phrase if it has phrasal scope (Right Hand Rule). In a supporting view, particularly for the Turkish treebank, where IGs are mostly representations of segments separated by derivational morphemes derivational morphology changes lexical types of the items they are attached to.



But, I did not say anything to you.

Figure 2: The dependencies and the final CCG categories assigned to a sentence in the treebank

A simple example is given in Figure 2. This example does not contain any multiple IG words. The recursive algorithm applies to the 2 top level elements here, one by one. First it finds the S.MODIFIER and assigns it with category *S/S*, since the sentence it modifies is to the right. After that, it finds the other element at the top level, which is the main verb *söylemedim*. After counting its complements, the algorithm assigns ditransitive category to the verb. The complements receive their categories in a depth-first manner. This means *hiçbir* being *şey*'s determiner is assigned *NP/NP* right after *şey* is assigned NP. Objects are assigned *NP* and subjects are assigned *NP[nom]*. This example has argument scrambling to the right of the verb, as well. We take SOV as the canonical word order and assign categories according to this.

Coordination and extraction cases are handled differently. Examples of different linguistic structures are given in Section 5.

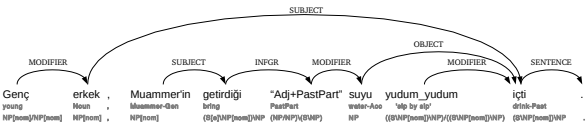
5 Results

We will focus on the outcome of the morphemic CCG lexicon approach for some specific constructions in the Turkish treebank. Taking morphemes as the smallest representational units has some advantages discussed previously. Some of the solutions of the problems discussed earlier are given in this section.

5.1 Relativisation

Object extraction and adjunct extraction examples are given together with the dependencies in the treebank that are used to create the categories in Figures 3 and 4.

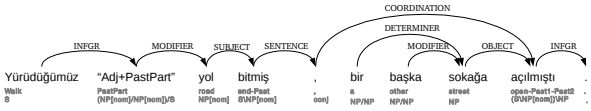
The fact that relative morphemes behave in a similar manner to relative pronouns in English provides the basis for the approach taken here for recovering long-range dependencies in extractions of this type.



The young man drank the water Muammer brought sip by sip.

Figure 3: Categories for object extraction

The object vs. adjunct relativisation ambiguity problem is also solved with the help of the secondary links added to the treebank (Çakıcı, 2005).



The road we walked had ended and lead to another street.

Figure 4: Categories for adjunct extraction

5.2 Long-distance dependencies

Long-distance dependencies as well as surface ones are recovered with the help of CCG categories. Long-distance dependencies caused by coordination and extraction are also not trivial.

The categories assigned to the words with the lexicon induction process are fed into the CCG parser described in Clark and Curran (2007) that is modified for Turkish to demonstrate the usefulness on a very small scale here. 69.98% of the sentences were assigned at least one parse. Errors for the rest of the sentences are mostly caused by wrong CCG catgories and are usually caused by annotation errors in the treebank (Çakıcı, 2008). The cleaning process continues.

The dependencies that the CCG derivation in Figure 5 yields are shown in Figure 6. In addition to the surface dependencies in the original dependency structure, long distance dependencies such as the ones that result from coordination are predicted, too. For instance, dependencies between *kurtulmayacak* and *tutsağım*, and between *olmayacak* and *tutsağım* are not predicted with most dependency parsers, although they are crucial dependencies for semantic interpretation. Most of the gold-standard dependencies are predicted correctly. There are additional ones -the long-distance dependencies that do not exist in the original dependency structure. These dependencies are recovered through coordination and extraction. These are: <kurtul,tutsağ>, <mayacak,tutsağ> and <olma,tutsağ>. The first two are captured by coordination of relativisation and the last one is by relativisation. The dependency between zaman and -ım is the only one predicted wrong. The head of the temporal adverb (zaman) is linked to the furthest verb instead of the one next to it. This evaluation is done with a system configuration where the parser returns the first random parse it finds. With a trained model, these kinds of errors will be less frequent.

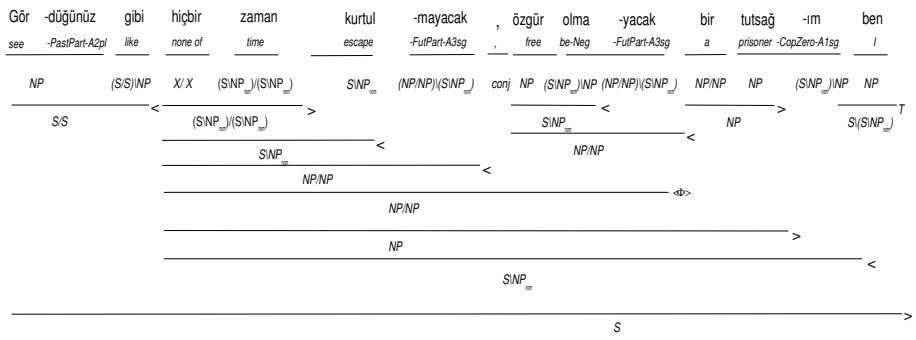


Figure 5: Derivation with morphemic categories

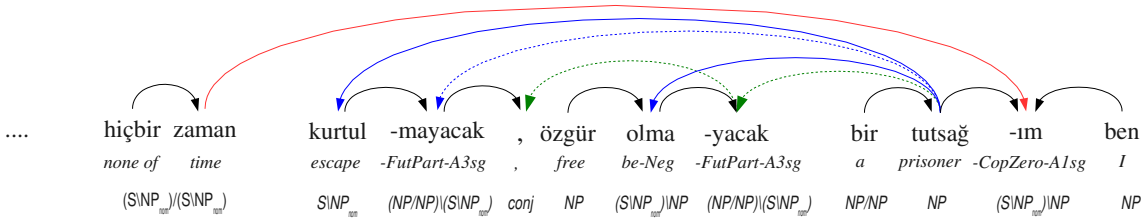


Figure 6: The dependencies recovered from the morphemic lexicon categories with the C&C parser.

f	word	cat
2	oku	S\NP[nom]
4	oku	(S\NP[nom])\NP
20	oku	S
23	oku	S\NP

Table 2: CCG categories (cat) and frequencies (f) of entities of verb *oku (read)* in morphemic lexicon.

6 Evaluation

There are 27895 unique word-category pairs for 19385 distinct tokens in the word-based lexicon that has 54K tokens (Çakıcı, 2008). The morphemic lexicon (69K tokens) has 13016 distinct word-category pairs for 6315 distinct word stems and IG stem names. The average word-category pair frequency goes up from 1.97 to 5.32. The verb *oku (read)* appears 49 times in the dependency treebank. However, only 5 of these inflected forms occur more than once. The unique word-category pairs involving this verb in some inflected or derived form is 45. Table 2 shows the category distribution with the morphemic approach for comparison purposes.

There are 311 morphemic category types as compared to 450 lexemic category types (Çakıcı, 2005). Although the number of category types is less in the morphemic lexicon, we believe that we have a more complete set of morphemic category types than lexemic category types. Figure 7 shows the distribution of morphemic categories with the data.

	words%	cat%|word	pairs%
mean	70.1	94	58.5
std. dev.	1.34	1.70	0.92

Table 3: Coverage results on the 10-fold evaluation of the morphemic lexicon

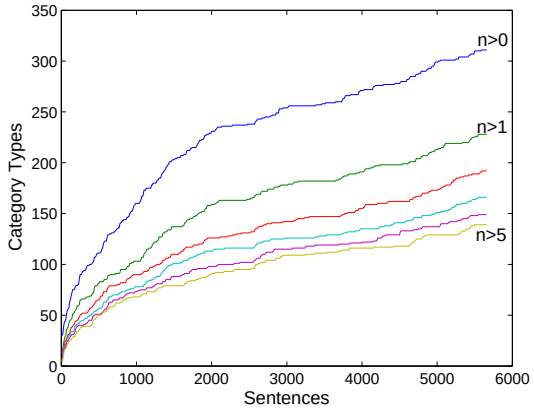


Figure 7: The growth of morphemic category types

Table 3 gives the numbers of a 10-fold evaluation process to test the coverage of the lexicon.

We take out the 1/10 of sentences for test and use the rest as a control set. We test the coverage by checking if the unique lexical entities and unique word-category pairs exist in the control set for each test set.

We also used an alternative evaluation method. The morphemic CCG categories were used as features to a dependency parser described in McDonald et al. (2005). The results are discussed in detail in Çakıcı (2008). To summarise, the boost in accuracy of the MST parser with the use of CCG categories was encouragingly high. Unlabelled and labelled dependency accuracies were 95.08% and 88.96% respectively. When this is compared with the results for Turkish dependency parsing without the CCG categories which are 81.08 unlabelled and 72.0 labelled reported in Çakıcı (2008)[2] it is seen that the use of gold-standard CCG categories boosts the performance of a dependency parser even when they are used as features. We know that a supertagger is crucial in getting realistic results but we include this information as a means of evaluation of the gold-standard CCG categories in the morphemic lexicon induced.

6.1 Evaluation by sampling

We perform a small evaluation by randomly choosing 25 sentences in the morphemic lexicon. 2 out of 202 words were not assigned a category by the lexicon induction algorithm. All of the sentences that were not parsed had at least one category error. On the other hand 3 sentences *with* errors (1 with a minor category feature error) were parsed with the C&C parser. The category accuracy of the morphemic lexicon sample set is more than 6 points higher than the accuracy of the sample set from the lexemic lexicon described in Çakıcı (2005; Çakıcı (2008). There were not any unseen categories in the morphemic evaluation set.

A comparison of lexemic and morphemic lexicon evaluations is given in Table 4. The morphemic lexicon is seen to outperform the lexemic lexicon in both category accuracy and parsing results.

lex	# sent.	# tok	cor	acc%	scor	cov	uns
morp	25	202	188	93.1	17	20	0
lex	25	166	144	86.7	16	14	5

Table 4: Sample evaluation for the morphemic and the lexemic lexicons
cor = tokens with correct category
acc = % category accuracy
scor = sentences that are completely correct
cov = parsed sentences, **uns** = unseen categories

7 Conclusion

We aimed to create an automatic lexicon induction method for Turkish that solves not only the problems associated with a lexicon account disregarding the morphosyntactic dependencies in Turkish but also the problem of data sparsity caused by rich inflectional morphology of Turkish. The outcome may be used in parsing applications and for training supertaggers which the present work and Clark and Curran (2006) show, provide almost all the information that is required for full parsing.

[2]These are also state-of-the art results for Turkish to our knowledge and the same data is used in both experiments.

References

Nart B. Atalay, Kemal Oflazer, and Bilge Say. 2003. The annotation process in the Turkish Treebank. In *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora*, Budapest, Hungary.

Cem Bozşahin. 2002. The combinatory morphemic lexicon. *Computational Linguistics*, 28(2):145–186.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of the 10th Conf. on Computational Natural Language Learning (CoNLL-X)*. SIGNLL.

Ruket Çakıcı and Jason Baldridge. 2006. Projective and non-projective Turkish parsing. In *Proceedings of the TLT 2006*, pages 19–30, Prague, Czech Republic.

Ruken Çakıcı. 2005. Automatic induction of a CCG grammar for Turkish. In *ACL Student Research Workshop*, pages 73–78, Ann Arbor, MI, USA.

Ruket Çakıcı. 2008. *Wide-Coverage Parsing for Turkish*. Ph.D. thesis, University of Edinburgh.

Stephen Clark and James R. Curran. 2006. Partial training for a lexicalized-grammar parser. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 144–151, New York City, USA.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4).

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *HLT-EMNLP 2005*, Vancouver, B.C.

Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gokhan Tür. 2003. Building a Turkish Treebank. In A. Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*, pages 261–277. Kluwer Academic Publishers, Dordrecht.

E. Williams. 1981. On the notions 'lexically related' and 'head of a word'. *Linguistic Inquiry*, 12:245–274.

Parsing with Non-associative Lambek Grammars

Matteo Capelletti
matteo.capelletti@gmail.com

May 8, 2009

Abstract

I present a parsing method for non-associative Lambek grammars. These grammars differ from basic Ajdukiewicz–Bar-Hillel grammars for the presence of introduction rules for the implications. This difference has two important consequences. On one side, it makes Lambek calculi full fledged logical systems, and enhances the grammar system with hypothetical reasoning and type-change rules. On the other, it raises a series of issues for computational applications, such as how to deal automatically with hypotheses, how to handle ambiguity, and finally how to perform parsing, as distinguished from theorem proving. We make clear the distinction between parsing and theorem proving in NL, and propose elegant and efficient solutions to both problems.

1 Parsing with Ajdukiewicz–Bar-Hillel grammars

The categorial grammar model we start with is based on what we call Ajdukiewicz–Bar-Hillel calculus with products. A *sequent* is a pair whose first element, the antecedent, is a list of categories, while the second, the succedent, consists of a single category. We write sequents as $\Gamma \to A$. The calculus consists of the following rules.

$$\frac{}{A \to A}\ Ax \qquad \frac{\Gamma \to A \quad \Delta \to B}{\Gamma, \Delta \to A \otimes B}\ \otimes\mathrm{I}$$

$$\frac{\Gamma \to A/B \quad \Delta \to B}{\Gamma, \Delta \to A}\ /\mathrm{E} \qquad \frac{\Gamma \to B \quad \Delta \to B\backslash A}{\Gamma, \Delta \to A}\ \backslash\mathrm{E}$$

Figure 1: The Ajdukiewicz–Bar-Hillel calculus.

A categorial grammar is a triple (Lex, s, D) such that Lex is a set of word-category pairs, which we write $w :: A$, s is a designated start symbol and D a deductive system, like the one given in Figure 1. A categorial grammar G generates a string $w_1 \dots w_n$, if there is a deduction of $\Gamma \to s$ and $w_1 \dots w_n \Rightarrow^* \Gamma$ by means of lexical assignments.

Parsing with $\mathrm{AB}^{\otimes}$ grammars can be done efficiently by implementing, for instance, the following deductive parser. [1] A deductive parser is a triple $(\mathcal{I}, \mathcal{A}, \mathcal{R})$ of items, axioms and inference rules. The parser operates on two kinds of items which we represent as $(i, \Delta \triangleright \Gamma \to A, j)$ and $(i, \Gamma \triangleleft \Delta \to A, j)$. Here i and j are integers and $\Delta \triangleright \Gamma \to A$ and $\Gamma \triangleleft \Delta \to A$ are sequents in which exactly one occurrence of an auxiliary symbols (either $\triangleright$ or $\triangleleft$) appear. These symbols play a role similar to the dot in the Earley items. An item of the form $(i, \Delta \triangleright \Gamma \to A, j)$ asserts that $\Delta\Gamma \to A \in \mathrm{AB}^{\otimes}$ and that $w_{i+1} \dots w_j \Rightarrow^* \Delta$. Dually, an item of the form $(i, \Gamma \triangleleft \Delta \to A, j)$ asserts that $\Gamma\Delta \to A \in \mathrm{AB}^{\otimes}$ and that $w_{i+1} \dots w_j \Rightarrow^* \Delta$.[2] For simplicity, we write items of the form $(i, \triangleleft \Delta \to A, j)$ and $(i, \Delta \triangleright \to A, j)$ as $(i, \Delta \to A, j)$.

Definition 1 *Let an $AB^{\otimes}$ grammar G and a string $w_1 \dots w_n$ be given. The $\mathcal{AB}^{\otimes}_{Mix}$ deductive parser is the triple $(\mathcal{I}, \mathcal{A}, \mathcal{R})$ presented in Figure 2.*

[1] Other options, based on simpler adaptations of CF-grammar parsing algorithms are also possible. The system presented below has however some computational advantages over them. These advantages are discussed in detail in Capelletti and Tamburini [2009].

[2] For simplicity, we omit the description of the predictive component of the parser.

$$\mathcal{I} = \begin{array}{l} \{\ (i, \Gamma \triangleright \Delta \to A, j) \mid \Gamma\Delta \to A \in \mathrm{AB}^{\otimes},\ 0 \leqslant i \leqslant j \leqslant n\ \} \\ \cup \\ \{\ (i, \Gamma \triangleleft \Delta \to A, j) \mid \Gamma\Delta \to A \in \mathrm{AB}^{\otimes},\ 0 \leqslant i \leqslant j \leqslant n\ \} \end{array}$$

$$\mathcal{A} = \{\ (i-1, A \to A, i) \mid w_i :: A \in \mathit{Lex}\ \}$$

$$\mathcal{R} = \left\{ \begin{array}{ll} \left.\begin{array}{l} \dfrac{(i, \Delta \triangleright A\,\Gamma \to C,\ j)}{(i, \Delta\ A \triangleright \Gamma \to C,\ j)}\ \epsilon \Rightarrow^{+} A \\[2ex] \dfrac{(i, \Gamma\ A \triangleleft \Delta \to C,\ j)}{(i, \Gamma \triangleleft A\,\Delta \to C,\ j)}\ \epsilon \Rightarrow^{+} A \end{array}\right\} & \epsilon\text{-}\mathit{Scanning} \\[2ex] \left\{ \dfrac{(i, \Delta \to C/B,\ j)}{(i, \Delta \triangleright B \to C,\ j)} \quad \dfrac{(i, \Delta \to B\backslash C,\ j)}{(i, B \triangleleft \Delta \to C,\ j)} \right\} & \mathit{Shifting} \\[2ex] \left\{ \dfrac{(i, \Delta \triangleright A \otimes B\,\Gamma \to C,\ j)}{(j, \triangleright A\,B\ \to A \otimes B,\ j)} \quad \dfrac{(i, \Gamma\ A \otimes B \triangleleft \Delta \to C,\ j)}{(i,\ A\,B \triangleleft \to A \otimes B,\ i)} \right\} & \otimes\text{-}\mathit{Prediction} \\[2ex] \left.\begin{array}{l} \dfrac{(i, \Delta \triangleright A\,\Gamma \to C, k) \quad (k, \Lambda \to A, j)}{(i, \Delta\ A \triangleright \Gamma \to C, j)} \\[2ex] \dfrac{(i, \Lambda \to A, k) \quad (k, \Delta\ A \triangleleft \Gamma \to C, j)}{(i, \Delta \triangleleft A\,\Gamma \to C, j)} \end{array}\right\} & \mathit{Completion} \end{array} \right.$$

Figure 2: The system $\mathcal{AB}^{\otimes}_{\mathrm{Mix}}$.

The correctness, complexity and linguistic application of this parsing system are studied in Capelletti [2007]; Capelletti and Tamburini [2009]. We remark here that it can handle straightforwardly product categories and null assignments, what is not possible in CYK style categorial parsing. As for the complexity, it is cubic on the length of the input string. However, it has better average performances than CYK and Early style categorial parsers.

2 Non-associative Lambek calculus

The AB calculus of the previous section can be extended in several ways. Here we examine the system presented in Lambek [1961], the so called non-associative Lambek calculus.

The most remarkable difference with $\mathrm{AB}^{\otimes}$ is the presence in NL of the slash introduction rules, see 1.[3]

$$\frac{(\Gamma,\ B) \to A}{\Gamma \to A/B}\ /\mathrm{I} \qquad\qquad \frac{(B,\ \Delta) \to A}{\Delta \to B\backslash A}\ \backslash\mathrm{I} \tag{1}$$

Given these rules, a number of new patterns of type transformation becomes available. For instance, given an assignment $w :: A$, it is always possible to infer that w has also a type $B/(A\backslash B)$ for any formula B. Consider for instance the following deductions of the sequent $s/(n\backslash s), (s/(n\backslash s))\backslash s \to s$, only the first of which can be obtained in the $\mathrm{AB}^{\otimes}$ calculus.

[3] We remark that, in NL, Γ and Δ are metavariable ranging over bracketed strings of formulas, instead of strings as in $\mathrm{AB}^{\otimes}$. In the rules /I and \I, Γ and Δ are assumed to be non-empty. We refer to Moortgat [1997] for the detailed axiomatization of NL on which the examples below are based.

(2) Verb wide scope reading:

$$\frac{s/(n\backslash s) \to s/(n\backslash s) \quad (s/(n\backslash s))\backslash s \to (s/(n\backslash s))\backslash s}{s/(n\backslash s),\ (s/(n\backslash s))\backslash s \to s}\ \backslash\mathrm{E}$$

(3) Subject wide scope reading:

$$\frac{s/(n\backslash s) \to s/(n\backslash s) \qquad \dfrac{\dfrac{\dfrac{n \to n \quad n\backslash s \to n\backslash s}{n,\ n\backslash s \to s}\ \backslash\mathrm{E}}{n \to s/(n\backslash s)}\ /\mathrm{I} \qquad (s/(n\backslash s))\backslash s \to (s/(n\backslash s))\backslash s}{\dfrac{n,\ (s/(n\backslash s))\backslash s \to s}{(s/(n\backslash s))\backslash s \to n\backslash s}\ \backslash\mathrm{I}}\ \backslash\mathrm{E}}{s/(n\backslash s),\ (s/(n\backslash s))\backslash s \to s}\ /\mathrm{E}$$

As a further simple example, consider also the sequent $n, (s/(n\backslash s))\backslash s \to s$ which is derivable in NL (but not in $\mathrm{AB}^{\otimes}$), in virtue of type raising. Type changing rules in Lambek calculi have been studied in detail in Hendriks [1993].

However, if from $w :: A$, it is always possible to infer $w :: B/(A\backslash B)$ for any B, we may run into problems in an automated deduction, as the pattern can iterate endlessly. Below we will see that there is however a canonical form that can be imposed to NL derivations that avoid such loops.

2.1 Parsing and theorem proving

Looking at the previous example, it is not clear how hypothetical reasoning could be controlled in the *parsing* process. As a logical system, NL can make use of the theorem proving methodologies developed for other logical systems, such as those based on forward chaining in sequent calculus or on the inverse method. However, NL theorems are *structured* sequents, while *parsing* should, by definition, infer the structure. The two problems could be roughly distinguished as follows.

- *Theorem proving*: find a proof of a sequent $\Gamma \to C$ where Γ is a bracketed list of types.
- *Parsing*: given a grammar Lex, a string $w_1, \ldots, w_n$ and a formula C, find a proof of $\Gamma \to C$, where Γ is a bracketed list of types living on $A_1, \ldots, A_n$ and $w_i :: A_i \in Lex$[4].

These two tasks have a very different status depending on whether they refer to $\mathrm{AB}^{\otimes}$ grammars or to NL grammars. Theorem proving is trivial for $\mathrm{AB}^{\otimes}$ systems, while it is an interesting problem for NL for which a polynomial solution has been found by Aarts and Trautwein [1995] and by de Groote [1999b]. On the other hand, while parsing for Ajdukiewicz–Bar-Hillel grammars is a well developed and active research field, see for instance Vijay-Shanker and Weir [1990]; Hockenmaier and Steedman [2001]; Clark et al. [2001]; Clark and Curran [2007], not so much has been done for parsing with NL or in general with Lambek grammars. Some attempts in the latter direction can be found in Finkel and Tellier [1996]; Hepple [1992]; König [1994], though for the associative Lambek calculus without product. Other works, like Morrill [1996]; de Groote [1999a]; Moot [2002] address theorem proving with the proof-net formalism.

2.2 Normal derivations

The approach we follow here is based on the conversion of a NL grammar into an equivalent $\mathrm{AB}^{\otimes}$ grammar. While we are primarily concerned with parsing, our method also provides us with a *normal form* automatic theorem proved for NL. By normal form we mean that our procedure is a solution to the problem of *spurious ambiguity* for NL, that is the problem arising when different derivations receive the same semantic interpretation.

Our method is a refinement and a computational interpretation of the procedure used in Buszkowski [1986]; Kandulski [1988] to prove the context-freeness of NL grammars. Our procedure effectively converts a non-associative Lambek grammar into an equivalent Ajdukiewicz–Bar-Hillel grammar with product.

Let us write $|A|$ for the length of a formula A and $B \equiv A$ if A and B are syntactically identical. We will use the following notation for developing a normal form axiomatization for NL.

- We write $A \xrightarrow{r} B$ for a sequent $A \to B$ such that $|B| < |A|$ or $B \equiv A$ and we call it a *reducing* sequent.

[4]For simplicity, we are assuming here that the lexicon does not contain null assignments.

- We write $A \xrightarrow{e} B$ for a sequent $A \to B$ such that $|B| > |A|$ or $B \equiv A$ and we call it an *expanding* sequent.
- We define $\overline{r} = e$ and $\overline{e} = r$.
- Finally we use p as a variable ranging over $\{e, r\}$.

Below we define the normal form sequent calculus which we call NL*.

Definition 2 *The set Ax of the* axioms *of the normal form calculus NL* is given by the following inference rules.*

$$\textbf{Axioms:} \quad \frac{}{A \xrightarrow{p} A, \text{ with } A \text{ atomic}}$$

$$\textbf{Product Rule:} \quad \frac{A \xrightarrow{p} A' \quad B \xrightarrow{p} B'}{A \otimes B \xrightarrow{p} A' \otimes B'}$$

$$\textbf{Application:} \quad \frac{A \xrightarrow{r} A' \quad B \xrightarrow{r} A'\backslash C}{A \otimes B \xrightarrow{r} C} \qquad \frac{A \xrightarrow{r} C/B' \quad B \xrightarrow{r} B'}{A \otimes B \xrightarrow{r} C}$$

$$\textbf{Monotonicity:} \quad \frac{A' \xrightarrow{p} A \quad B \xrightarrow{\overline{p}} B'}{B'\backslash A' \xrightarrow{p} B\backslash A} \qquad \frac{A' \xrightarrow{p} A \quad B \xrightarrow{\overline{p}} B'}{A'/B' \xrightarrow{p} A/B}$$

$$\textbf{Lifting:} \quad \frac{B \xrightarrow{r} A'/C \quad A' \xrightarrow{e} A}{C \xrightarrow{e} B\backslash A} \qquad \frac{B \xrightarrow{r} C\backslash A' \quad A' \xrightarrow{e} A}{C \xrightarrow{e} A/B}$$

$$\textbf{Coapplication:} \quad \frac{B \xrightarrow{r} B' \quad B' \otimes C \xrightarrow{e} A}{C \xrightarrow{e} B\backslash A} \qquad \frac{B \xrightarrow{r} B' \quad C \otimes B' \xrightarrow{e} A}{C \xrightarrow{e} A/B}$$

The system NL* consists of all sequents $A \to C$ which are the conclusion of the following rule.

$$\frac{A \xrightarrow{r} B \in Ax \quad B \xrightarrow{e} C \in Ax}{A \to C} \tag{4}$$

Every NL sequent of the form $A \to C$ (and in fact, by replacing commas with product, every NL sequent) can be proven by Rule 4. Furthermore, the system defined by Rule 4 is a solution to the spurious ambiguity problem for NL, in the sense that derivations in this system are in one-one correspondence with cut-free proof-nets for NL. The proof of these statements is given in Capelletti [2009].

2.3 Automated theorem proving

Observe that the rules of the Ax system allow an easy computational interpretation in terms of type contraction rules. The 'polarity' p appearing on top of the arrow symbol is interpreted as a function from a formula to a set of formulas. If p is r the function is a *contraction* of the antecedent to the succedent, if it is e it is a contraction of the succedent to the antecedent. In Figure 3, we give a recursive procedure implementing such type contraction (we omit the symmetric case).

$$\begin{array}{lcl}
e(A) & = & \{A\}, \text{ if } A \text{ is an atom} \\
e(A \otimes B) & = & \{\ A' \otimes B' \mid A' \in e(A)\ \&\ B' \in e(B)\ \} \\
e(A/B) & = & \text{let } \mathcal{A} \text{ be } e(A) \text{ and } \mathcal{B} \text{ be } r(B) \text{ in} \\
 & & \{\ A'/B' \mid A' \in \mathcal{A}\ \&\ B' \in \mathcal{A}\ \} \cup \\
 & & \{\ C \mid C \otimes B' \in \mathcal{A}\ \&\ B' \in \mathcal{B}\ \} \cup \\
 & & \{\ C \mid A' \in \mathcal{A}\ \&\ C\backslash A' \in \mathcal{B}\ \}
\end{array}$$

$$\begin{array}{lcl}
r(A) & = & \{A\}, \text{ if } A \text{ is an atom} \\
r(A \otimes B) & = & \text{let } \mathcal{A} \text{ be } r(A) \text{ and } \mathcal{B} \text{ be } r(B) \text{ in} \\
 & & \{\ A' \otimes B' \mid A' \in \mathcal{A}\ \&\ B' \in \mathcal{B}\ \} \cup \\
 & & \{\ C \mid C/B' \in \mathcal{A}\ \&\ B' \in \mathcal{B}\ \} \cup \\
 & & \{\ C \mid A' \in \mathcal{A}\ \&\ A'\backslash C \in \mathcal{B}\ \} \\
r(A/B) & = & \{\ A'/B' \mid A' \in r(A)\ \&\ B' \in e(B)\ \}
\end{array}$$

Figure 3: The functions *expand*, e, and *reduce*, r.

The functions e and r return sets of formulas which are shorter than the input formula. Hence the procedures are bound to terminate. The following examples may help to clarify how the procedure works.

(5) $r((s/(n\backslash s))\backslash s)$:

$$\dfrac{r(s)=\{s\} \quad \dfrac{e(s)=\{s\} \quad \dfrac{r(s)=\{s\} \quad e(n)=\{n\}}{r(n\backslash s)=\{n\backslash s\}}}{e(s/(n\backslash s))=\{s/(n\backslash s),\ n\}}}{r((s/(n\backslash s))\backslash s)=\{(s/(n\backslash s))\backslash s,\ n\backslash s\}}$$

(6) $e((s/(n\backslash s))\backslash s)$:

$$\dfrac{e(s)=\{s\} \quad \dfrac{r(s)=\{s\} \quad \dfrac{e(s)=\{s\} \quad r(n)=\{n\}}{e(n\backslash s)=\{n\backslash s\}}}{r(s/(n\backslash s))=\{s/(n\backslash s)\}}}{e((s/(n\backslash s))\backslash s)=\{(s/(n\backslash s))\backslash s,\ n\backslash s\}}$$

With the procedure defined in Figure 3, provability in NL amounts to the following:

$$\vdash_{\mathrm{NL}} A \to C \quad \text{iff} \quad \exists B.[B \in r(A)\ \&\ B \in e(C)] \tag{7}$$

2.4 Parsing

The procedure in 7 defines a normal form *theorem prover* for NL. However, we are interested in parsing and the following definition will clarify the connection between the calculus and procedure that we have just defined and the parsing problem for NL grammars.

Given an NL grammar G with lexicon Lex we generate an equivalent $\mathrm{AB}^{\otimes}$ grammar G' with lexicon Lex' by the following operation

$$Lex' = \{\ w :: A' \mid w :: A \in Lex\ \&\ A' \in r(A)\ \} \tag{8}$$

2.5 Conclusion

The *lexical expansion* operation in 8 converts an NL grammar into a strongly equivalent $\mathrm{AB}^{\otimes}$ grammar. After conversion, the parsing method presented in Figure 2 can be applied.

The expansion should be executed once and for all. Parsing on the resulting grammar is cubic, as standard context-free grammar parsing. Furthermore, if the original NL lexicon was semantically annotated, the resulting $\mathrm{AB}^{\otimes}$ lexicon is too, as the rules applied for the conversion have a straightforward Curry-Howard interpretation.

We conclude by observing that the spurious ambiguity problem affects neither the lexical conversion procedure nor the parsing algorithms. Thus we have defined a redundancy-free parsing method for NL grammars.

References

Aarts, E. and Trautwein, K. (1995). Non-associative Lambek categorial grammar in polynomial time. *Mathematical Logic Quarterly*, 41:476–484.

Buszkowski, W. (1986). Generative capacity of the nonassociative Lambek calculus. *Bulletin of the Polish Academy of Sciences, Mathematics*, 34:507–516.

Capelletti, M. (2007). *Parsing with Structure-Preserving Categorial Grammars*. PhD thesis, UiL OTS, Utrecht.

Capelletti, M. (2009). Normal derivations for the non-associative Lambek calculus. Under review: Theoretical Computer Science.

Capelletti, M. and Tamburini, F. (2009). Parsing with three models of categorial grammar. Under review: Computational Linguistics.

Clark, S. and Curran, J. R. (2007). Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistic*, 33(4):493–552.

Clark, S., Hockenmaier, J., and Steedman, M. (2001). Building deep dependency structures with a wide-coverage ccg parser. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 327–334, Morristown, NJ, USA. Association for Computational Linguistics.

de Groote, P. (1999a). A dynamic programming approach to categorial deduction. In *CADE-16: Proceedings of the 16th International Conference on Automated Deduction*, pages 1–15, London, UK. Springer-Verlag.

de Groote, P. (1999b). The non-associative Lambek calculus with product in polynomial time. In Murray, N. V., editor, *Lecture Notes in Artificial Intelligence*, volume 1617. Springer-Verlag.

de Groote, P. and Lamarche, F. (2002). Classical non-associative Lambek calculus. *Studia Logica*, 71(3):355–388.

Finkel, A. and Tellier, I. (1996). A polynomial algorithm for the membership problem with categorial grammar. *Theoretical Computer Science*, 164:207–221.

Hendriks, H. (1993). *Studied Flexibility: Categories and Types in Syntax and Semantics*. PhD thesis, ILLC, Amsterdam.

Hepple, M. (1992). Chart parsing Lambek grammars: Modal extensions and incrementality. In *Proceedings of COLING-92*, pages 134–140.

Hockenmaier, J. and Steedman, M. (2001). Generative models for statistical parsing with combinatory categorial grammar. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 335–342, Morristown, NJ, USA. Association for Computational Linguistics.

Kandulski, M. (1988). The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 34:41–52.

König, E. (1994). A hypothetical reasoning algorithm for linguistic analysis. *Journal of Logic and Computation*, 4(1):1–19.

Lambek, J. (1961). On the calculus of syntactic types. In Jacobson, R., editor, *Proceedings of the Twelfth Symposium in Applied Mathematics*, volume XII, pages 166–178.

Moortgat, M. (1997). Categorial type logics. In van Benthem, J. and ter Meulen, A., editors, *Handbook of Logic and Language*, pages 93–177. Elsevier, Amsterdam.

Moot, R. (2002). *Proof Nets for Linguistic Analysis*. PhD thesis, UiL-OTS, Utrecht.

Morrill, G. (1996). Memoisation of categorial proof nets: Parallelism in categorial processing. In Abrusci, V. M. and Casadio, C., editors, *Proofs and Linguistic Categories,* Proceedings 1996 Roma Workshop, pages 157–169, Bologna. Cooperativa Libraria Universitaria Editrice.

Vijay-Shanker, K. and Weir, D. J. (1990). Polynomial time parsing of combinatory categorial grammars. In *ACL*, pages 1–8.

Polymorphic Categorial Grammars: expressivity and computational properties

Matteo Capelletti
matteo.capelletti@gmail.com

Fabio Tamburini
DSLO, University of Bologna - Italy
fabio.tamburini@unibo.it

Abstract

We investigate the use of polymorphic categorial grammars as a model for parsing natural language. We will show that, despite the undecidability of the general model, a subclass of polymorphic categorial grammars, which we call *linear*, is mildly context-sensitive and we propose a polynomial parsing algorithm for them. An interesting aspect of the resulting system is the absence of spurious ambiguity.

1 Introduction

The simplest model of a categorial grammar is based on the so called Ajdukiewicz–Bar-Hillel calculus of Ajdukiewicz [1935] and Bar-Hillel [1953], with only elimination rules for the slashes. Contemporary categorial grammars in the style of Ajdukiewicz–Bar-Hillel grammars are called *combinatory categorial grammars*, see Steedman [2000]. Such systems adopt other forms of composition rules which enable them to generate non–context-free languages, see Weir and Joshi [1988]; Vijay-Shanker and Weir [1994]. The other main tradition of categorial grammar, the type-logical grammars of Morrill [1994]; Moortgat [1997], stemming from the work of Lambek [1958], adopt special kinds of structural rules, that enable the system to generate non–context-free languages. Both approaches increase the generative power of the basic system by adding special kinds of rules.

Here we adopt a different strategy, which consists in keeping the elementary rule component of AB grammars and in introducing *polymorphic* categories, that is syntactic categories containing atomic variables ranging over categories. The inference process will be driven by unification, rather than by simple identity of formulas. We will see two kinds of polymorphic categorial grammars, one that is Turing complete and another, resulting from a restriction on the first, which is mildly context-sensitive. This second system, which is obviously the most interesting one for linguistics, has some important advantages with respect to the aforementioned categorial settings. In respect to TLG, the polymorphic system we define is *polynomial*, as we will prove by providing a parsing algorithm. In respect to CCG, our system is not affected by the so called *spurious ambiguity* problem, that is the problem of generating multiple, semantically equivalent, derivations.

The deductive system given in Figure 1, which we call $\mathrm{AB}^{\otimes}$, is a simple modification of the calculus of Kandulski [1988] to which it can easily be proved equivalent.

Identity Axioms:	$A \to A$	
Product Axioms:	$A,\, B \to A \otimes B$	
Shifting Rules:	$\dfrac{\Gamma \to C/A}{\Gamma, A \to C}\ (S_1)$	$\dfrac{\Gamma \to A\backslash C}{A, \Gamma \to C}\ (S_2)$
Cut Rules:	$\dfrac{\Gamma, A \to C \quad \Delta \to A}{\Gamma, \Delta \to C}\ (C_1)$	$\dfrac{\Gamma \to A \quad A, \Delta \to C}{\Gamma, \Delta \to C}\ (C_2)$

Figure 1: Ajdukiewicz–Bar-Hillel calculus with product, $\mathrm{AB}^{\otimes}$.

This basic CG models can be extended to generate non context-free languages in at least two ways. The first uses structural rules, introduction rules and other types of composition schemes. These approaches are characteristic of TLG, see Morrill [1994]; Moortgat [1997]; Moot [2002], and CCG, see Steedman [2000];

Baldridge [2002], and have been widely explored in the past. The second is based on the introduction of *polymorphism*. Here, we study this second approach.

The formalism of polymorphic categorial grammar that we are going to present is inspired by the polymorphic theory of types, see Girard et al. [1989]; Barendregt [1992]. Types may contain type variables together with constants, and these variables may be (implicitly or explicitly) quantified over. The idea of polymorphism is very simple and natural. Rather than defining a class of `id` functions $\mathtt{id}_{Int} :: Int \to Int$, $\mathtt{id}_{Char} :: Char \to Char$ and so forth, the function `id` is defined for any type α, as $\mathtt{id} :: \forall\alpha.\alpha \to \alpha$ or $\mathtt{id} :: \alpha \to \alpha$ where α is implicitly universally quantified.

The same idea is very natural also in linguistics, where, for example, coordination particles such as 'and' and 'or' are typically polymorphic, as they coordinate expressions of *almost* any syntactic category. Thus one can find in the categorial grammar literature several examples of polymorphic assignments for these expressions Lambek [1958]; Steedman [1985]; Emms [1993]; Clark and Curran [2007].

Another example of Ajdukiewicz–Bar-Hillel style categorial grammars adopting a form of polymorphism are the unification categorial grammars Uszkoreit [1986]; Zeevat [1988]; Heylen [1999], where polymorphism is used at the level of feature structures.

1.1 Unification Ajdukiewicz–Bar-Hillel grammars

Syntactic categories of UAB$^{\otimes}$ are defined as follows.

$$\begin{array}{lrcl} \textbf{Atoms:} & \mathcal{A} & ::= & a, b, c, n, s, i \ldots \\ \textbf{Variables:} & \mathcal{V} & ::= & \alpha, \beta, \gamma \ldots \\ \textbf{Categories:} & \mathcal{F} & ::= & \mathcal{A} \mid \mathcal{V} \mid \mathcal{F} \otimes \mathcal{F} \mid \mathcal{F}\backslash\mathcal{F} \mid \mathcal{F}/\mathcal{F} \end{array}$$

Unification of two categories A and B is defined in the obvious way and and the resulting substitution is denoted $A \approx B$.[1] The unification Ajdukiewicz–Bar-Hillel calculus, UAB$^{\otimes}$ is defined in Figure 2.[2]

Identity Axioms:	$A \to A$	
Product Axioms:	$A,\ B \to A \otimes B$	
Shifting Rules:	$\dfrac{\Gamma \to C/A}{\Gamma, A \to C}\ (S_1)$	$\dfrac{\Gamma \to A\backslash C}{A, \Gamma \to C}\ (S_2)$
Cut Rules:	$\dfrac{\Gamma, A \to C \quad \Delta \to B}{\Gamma, \Delta \to C(A \approx B)}\ (C'_1)$	$\dfrac{\Gamma \to B \quad A, \Delta \to C}{\Gamma, \Delta \to C(A \approx B)}\ (C'_2)$

Figure 2: Unification Ajdukiewicz–Bar-Hillel calculus, UAB$^{\otimes}$

We give here some examples of non context-free languages generated by UAB$^{\otimes}$ grammars.

Example 1 *We define the UAB$^{\otimes}$ grammar for the language $a^n b^n c^n$, $n \geqslant 1$. Let grammar G_1 consist of the following assignments:*

$$\begin{array}{ll} a :: s/(b \otimes c) & b :: b \\ a :: (s/\alpha)\backslash(s/(b \otimes (\alpha \otimes c))) & c :: c \end{array}$$

We derive the string 'aabbcc'. We write A for the formula $(s/\alpha)\backslash(s/(b\otimes(\alpha\otimes c)))$. For readability, boxes are drawn around the words that anchor the axioms to the lexicon.

$$\dfrac{\dfrac{\dfrac{\boxed{a}}{s/(b\otimes c) \to s/(b\otimes c)} \quad \dfrac{\dfrac{\boxed{a}}{A \to A}}{s/\alpha, A \to s/(b\otimes(\alpha\otimes c))}}{s/(b\otimes c), A \to s/(b\otimes((b\otimes c)\otimes c))} \quad \dfrac{\boxed{b}}{b \to b}}{s/(b\otimes c), A, (b\otimes((b\otimes c)\otimes c)) \to s} \quad \dfrac{\dfrac{\boxed{b}}{b \to b} \quad \dfrac{\boxed{c}}{c \to c}}{b, c \to b \otimes c} \quad \dfrac{\boxed{c}}{c \to c}}{b, c, c \to (b\otimes c)\otimes c}}{b, b, c, c \to b \otimes ((b\otimes c)\otimes c)}}{s/(b\otimes c), A, b, b, c, c \to s}$$

Example 2 *We define a UAB$^{\otimes}$ grammar for 'ww', $w \in \{a, b\}^+$.*

[1] We use postfix notation for application of a substitution to a formula.

[2] Obviously, the rules involving unification are only defined if unification is defined.

Let grammar G_2 consist of the following assignments:

$$\begin{array}{ll} a :: a & b :: b \\ a :: s/a & b :: s/b \\ a :: (s/\alpha)\backslash(s/(\alpha \otimes a)) & b :: (s/\alpha)\backslash(s/(\alpha \otimes b)) \end{array}$$

It is easy to see that grammar G_2 generates exactly the language 'ww' with $w \in \{a, b\}^+$. As in the case of G_1, type variables are used as accumulators for long-distance dependencies.

A typical example of non context-freeness of natural language are the so called cross serial dependencies, which can be found, for instance, in Dutch subordinate clauses.

Example 3 *We define a UAB$^\otimes$ grammar for Dutch cross-serial dependencies. An example is the following subordinate clause, from Steedman [2000]:*

Ik	*Cecilia*	*Henk*	*de*	*nijlpaarden*	*zag*	*helpen*	*voeren.*
I	Cecilia	Henk	the	hippopotamuses	saw	help	feed.

I saw Cecilia help Henk feed the hippopotamuses.

These constructs exhibit dependencies of the form 'ww', where the ith words in the two halves are matched. An example lexicon generating the sentence in this example is the following[3]*:*

$$\begin{array}{ll} \textit{Ik, Cecilia, Henk, de nijlpaarden} & :: n \\ \textit{zag} & :: ((n \otimes (n \otimes \alpha))\backslash c)/(\alpha\backslash i) \\ \textit{helpen} & :: ((n \otimes \alpha)\backslash i)/(\alpha\backslash i) \\ \textit{voeren} & :: n\backslash i \end{array}$$

$$\dfrac{\boxed{N} \qquad \dfrac{\dfrac{\boxed{zag}}{Z \to Z} \qquad \dfrac{\dfrac{\boxed{helpen}}{H \to H} \qquad \boxed{voeren}}{H, \alpha\backslash i \to (n \otimes \alpha)\backslash i \qquad n\backslash i \to n\backslash i}}{Z, \alpha\backslash i \to (n \otimes (n \otimes \alpha))\backslash c \qquad H, n\backslash i \to (n \otimes n)\backslash i}}{\dfrac{Z, (H, n\backslash i) \to (n \otimes (n \otimes (n \otimes n)))\backslash c}{(n \otimes (n \otimes (n \otimes n))), (Z, (H, n\backslash i)) \to c}}}{n \otimes (n \otimes (n \otimes n)) \qquad (n \otimes (n \otimes (n \otimes n))), (Z, (H, n\backslash i)) \to c}$$
$$n \otimes (n \otimes (n \otimes n)), (Z, (H, n\backslash i)) \to c$$

These examples show that the languages generated by UAB$^\otimes$ grammars properly include the context-free languages (since AB$^\otimes$ grammars are properly included in UAB$^\otimes$ grammars). It is also easy to show that if we allow null assignments, that is assignments of the form $\epsilon :: A$, where ϵ is the empty string, the UAB$^\otimes$ formalism becomes undecidable[4].

1.2 Constraining UAB$^\otimes$ grammars

One constraint that we can impose on UAB$^\otimes$ grammars to avoid undecidability is *linearity*. Roughly, we impose the restriction that any lexical type may contain at most one variable, occurring once in an argument position and once in value position. Thus, for instance, α/α, $(s/\alpha)\backslash(s/(\alpha \otimes a))$ are licit types, while $(\alpha\backslash\alpha)/\alpha$, $(s/(\alpha \otimes \beta))\backslash(s/((\alpha \otimes \beta) \otimes a))$ and $(s/(\alpha \otimes \alpha))\backslash(s/((\alpha \otimes \alpha) \otimes a))$ are not. More precisely we define *linear* categories as the types F_2 generated by the following context-free grammar.

$$\begin{array}{rcl@{\qquad}rcl} \sharp & ::= & \otimes \mid / \mid \backslash & \sharp' & ::= & / \mid \backslash \\ F_0 & ::= & A \mid F_0 \sharp F_0 & F_1 & ::= & F_1 \sharp F_0 \mid F_0 \sharp F_1 \mid \alpha \\ F_2 & ::= & F_1 \sharp' F_1 \mid F_0 \mid F_2 \sharp' F_0 \mid F_0 \sharp' F_2 & & & \end{array} \tag{1}$$

The interesting case in this definition are the F_2 formulas of the form A/B or $B\backslash A$, with A and B in F_1, the others being meant essentially to put these in context. Consider the case of A/B, then α occurs exactly once in A and in B, since a F_1 category contains the variable α by construction. By analogy with lambda terms,

[3] In the deduction, we write Z for the type of 'zag', H for that of 'helpen' and N for the string 'Ik, Cecilia, Henk, de nijlpaarden'.

[4] One can easily adapt the construction of Johnson [1988] for proving the undecidability unification cased phrase-structure grammar formalisms, see Capelletti and Tamburini [2009b].

we can think of the occurrence of α in B as a *binder* (possibly a pattern-binder), and of the occurrence in A as the *bound* variable.

An UAB$^{\otimes}$ grammar is linear if all its lexical assignments are linear. Furthermore, in linear UAB$^{\otimes}$ grammar, we work by simple *variable instantiation*, rather than by a full-fledged unification algorithm. More precisely let us denote A^B a formula A with a distinguished occurrence of a subformula B. A^C is the formula obtained from A^B by replacing the occurrence of the subformula B with the formula C. The linear UAB$^{\otimes}$ calculus consists of all the rules of the UAB$^{\otimes}$ calculus in Figure 2 replacing the Cut rules with the following *instantiation* rules.

$$\frac{\Delta \to A^B \quad \Gamma, A^\alpha \to C}{\Gamma, \Delta \to C[\alpha := B]} \qquad \frac{\Gamma \to A^B \quad A^\alpha, \Delta \to C}{\Gamma, \Delta \to C[\alpha := B]} \tag{2}$$

Observe that given a linear UAB$^{\otimes}$ grammar adopting the rules in 2, only linear types can occur in any of its deductions.

Observe also that the UAB$^{\otimes}$ grammars for $\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n$ and ww languages as well as that for the Dutch cross serial patterns, are all linear. On the other hand, no linear UAB$^{\otimes}$ grammar can be given for the so called MIX or Bach language that is the language of the strings containing an equal number of a's, b's and c's[5].

As we have the proper inclusion of context-free languages and the realization of limited cross-serial dependencies, in order to have a *mildly context-sensitive* grammar formalism we shall prove that linear UAB$^{\otimes}$ grammars can be parsed in polynomial time. We do this in the next section by providing a parsing algorithm for linear UAB$^{\otimes}$ grammars.

2 Polynomial parsing with linear UAB$^{\otimes}$ grammars

Linear UAB$^{\otimes}$ grammars can be parsed in polynomial time by means of a simple extension of parsing algorithm for AB$^{\otimes}$ grammars given in Capelletti [2009], see Appendix A. Attention has to be paid to the way we implement the completion rules based on the cut rules in 2. Clearly the direct instantiation and substitution of the variable in the conclusion sequent will give an exponential growth of the number of items generated (in a similar way is it would happen by implementing naively the CCG composition rules, see Vijay-Shanker and Weir [1990]). Therefore we make use of an extra table to keep track of partial variable instantiations and postpone substitution as far as possible. This table, which we call *instantiation table*, is used for storing the 'partial' instantiations of variables. Let n be the length of the input string and Lex the input lexicon. Cells of the instantiation table are denoted $t_{(i,k,j)}$, where $0 \leqslant i < j \leqslant n$ and $0 \leqslant k \leqslant |Lex|$. We extend the construction of formulas with two kinds of variables, α_k and $\alpha_{(i,k,j)}$ where i, k and j are as before. The difference between the two kinds of variables is that α_k is an uninstantiated variable while $\alpha_{(i,k,j)}$ is a variable α_k which has been instantiated when an item $(i, \Lambda \to C, j)$ was generated, by the new instantiation rule given below. The algorithm assumes that different lexical entries contain different variables, that is for no k the variable α_k occurs in two distinct lexical assignments. The algorithm uses the following two new rules (of which we give only one oriented variant) together with those given for parsing with $\mathcal{AB}^{\otimes}$ in Appendix A.

$$\begin{array}{ll}
\text{Given items} & (i, \Delta \triangleright A^{\alpha_l}\, \Gamma \to C, k) \text{ and } (k, \Lambda \to A^B, j), \\
\text{generate the item} & (i, \Delta\, A^{\alpha_l} \triangleright \Gamma \to C[\alpha_l := \alpha_{(i,l,j)}], j) \\
\text{update the table} & t_{(i,l,j)} = t_{(i,l,j)} \cup \{B\} \\
 & \\
\text{Given item} & (i, \Delta \triangleright \alpha_{(k,l,m)}\, \Gamma \to C, j) \text{ and } A \in t_{(k,l,m)} \\
\text{generate item} & (j, \triangleright A \to \alpha_{(k,l,m)}, j)
\end{array} \tag{3}$$

In Capelletti and Tamburini [2009b], we presented the detailed implementation of the parsing algorithm and proved its correctness. The complexity of the implementation given there is $O(|Lex||\Sigma|n^5)$, where $|Lex|$ and $|\Sigma|$ are the sizes of the lexicon and of the set of subformulas of the lexicon, respectively.

[5] To see this, we observe that the context-free language of the strings containing an equal number of a's and b's is not *linear*, in the sense of Hopcroft and Ullman [1979], see Linz [1990]. Hence for the MIX language, a UAB$^{\otimes}$ grammar needs to bind two distinct variables for each symbol, what violates linearity.

3 Conclusion

We have investigated some linguistic and computational properties of unification based categorial grammars. We have seen that, like other unification based grammar formalisms, unrestricted UAB$^{\otimes}$ grammars are Turing complete. However, we have also seen that the constraint of *linearity* locates the system among the mildly context-sensitive formalisms. A pleasant aspect of the resulting system, particularly with respect to others CG-based mildly context-sensitive categorial formalisms is the absence of spurious ambiguity. This is a pleasant property that results from the simple non-associative composition schemes adopted in the *parsing* system (see Appendix A and Capelletti and Tamburini [2009a]), and not from special constraints imposed to the derivations, as in Eisner [1996].

We conclude by observing that the linearity constraint can also be relaxed. For instance, while preserving the condition that only one variable occurs in a formula, we can admit more than two occurrences of this variable. In this way, we can include the standard types for coordination, $(\alpha\backslash\alpha)/\alpha$.

This condition enlarges the class of generated languages, producing for instance w^i or $\mathrm{a}_1^i\mathrm{a}_2^i\ldots\mathrm{a}_n^i$. To what extent and with what consequences from the computational point of view, is an open subject of investigation.

A Parser

Let an AB$^{\otimes}$ grammar G and a string $w_1 \ldots w_n$ be given. The $\mathcal{AB}^{\otimes}_{\mathrm{Mix}}$ deductive parser is the triple $(\mathcal{I}, \mathcal{A}, \mathcal{R})$ presented in Figure 3. See Capelletti and Tamburini [2009b] for the $O(|Lex||\Sigma|n^5)$ implementation of this parsing algorithm.

$$\mathcal{I} = \begin{array}{l} \{\ (i, \Gamma \triangleright \Delta \rightarrow A, j) \mid \Gamma\Delta \rightarrow A \in \mathrm{AB}^{\otimes},\ 0 \leqslant i \leqslant j \leqslant n\ \} \\ \qquad\qquad \cup \\ \{\ (i, \Gamma \triangleleft \Delta \rightarrow A, j) \mid \Gamma\Delta \rightarrow A \in \mathrm{AB}^{\otimes},\ 0 \leqslant i \leqslant j \leqslant n\ \} \end{array}$$

$$\mathcal{A} = \{\ (i-1, A \rightarrow A, i) \mid w_i :: A \in Lex\ \}$$

$$\mathcal{R} = \left\{ \begin{array}{ll} \left.\begin{array}{l} \dfrac{(i, \Delta \triangleright A\,\Gamma \rightarrow C,\ j)}{(i, \Delta\ A \triangleright \Gamma \rightarrow C,\ j)}\ \epsilon{:}^{+}A \\[2ex] \dfrac{(i, \Gamma\ A \triangleleft \Delta \rightarrow C,\ j)}{(i, \Gamma \triangleleft A\,\Delta \rightarrow C,\ j)}\ \epsilon{:}^{+}A \end{array}\right\} & \epsilon\textit{-Scanning} \\[2ex] \left\{ \dfrac{(i, \Delta \rightarrow C/B,\ j)}{(i, \Delta \triangleright B \rightarrow C,\ j)}\quad \dfrac{(i, \Delta \rightarrow B\backslash C,\ j)}{(i, B \triangleleft \Delta \rightarrow C,\ j)} \right\} & \textit{Shifting} \\[2ex] \left\{ \dfrac{(i, \Delta \triangleright A \otimes B\,\Gamma \rightarrow C,\ j)}{(j, \triangleright A\,B\ \rightarrow A \otimes B,\ j)}\quad \dfrac{(i, \Gamma\ A \otimes B \triangleleft \Delta \rightarrow C,\ j)}{(i,\ A\,B \triangleleft \rightarrow A \otimes B,\ i)} \right\} & \otimes\textit{-Prediction} \\[2ex] \left.\begin{array}{l} \dfrac{(i, \Delta \triangleright A\,\Gamma \rightarrow C, k)\quad (k, \Lambda \rightarrow A, j)}{(i, \Delta\ A \triangleright \Gamma \rightarrow C, j)} \\[2ex] \dfrac{(i, \Lambda \rightarrow A, k)\quad (k, \Delta\ A \triangleleft \Gamma \rightarrow C, j)}{(i, \Delta \triangleleft A\,\Gamma \rightarrow C, j)} \end{array}\right\} & \textit{Completion} \end{array} \right.$$

Figure 3: The system $\mathcal{AB}^{\otimes}_{\mathrm{Mix}}$.

References

Ajdukiewicz, K. (1935). Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27.

Baldridge, J. (2002). *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh.

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

Barendregt, H. (1992). Lambda calculus with types. In Abramsky, S., Gabbay, D. M., and Maibaum, T. S. E., editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press.

Capelletti, M. (2009). Parsing with non-associative Lambek grammars. In Penn, G. and Fowler, T., editors, *Parsing with Categorial Grammars*. Esslli proceedings.

Capelletti, M. and Tamburini, F. (2009a). Parsing with three models of categorial grammar. Under review: Computational Linguistics.

Capelletti, M. and Tamburini, F. (2009b). Polymorphic categorial grammars: expressivity and computational properties. In Penn, G. and Fowler, T., editors, *Parsing with Categorial Grammars*. Esslli proceedings.

Clark, S. and Curran, J. R. (2007). Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistic*, 33(4):493–552.

Eisner, J. (1996). Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 79–86, Morristown, NJ, USA. Association for Computational Linguistics.

Emms, M. (1993). Parsing with polymorphism. In *EACL*, pages 120–129.

Girard, J.-Y., Taylor, P., and Lafont, Y. (1989). *Proofs and Types*. Cambridge University Press.

Heylen, D. (1999). *Types and Sorts. Resource logic for feature checking*. PhD thesis, UiL-OTS, Utrecht.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes*. CSLI, Stanford, California.

Kandulski, M. (1988). The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 34:41–52.

Lambek, J. (1958). The mathematic of sentence structure. *American Mathematical Monthly*, 65(3):154–170.

Linz, P. (1990). *An introduction to formal languages and automata*. D. C. Heath and Company, Lexington, MA, USA.

Moortgat, M. (1997). Categorial type logics. In van Benthem, J. and ter Meulen, A., editors, *Handbook of Logic and Language*, pages 93–177. Elsevier, Amsterdam.

Moot, R. (2002). *Proof Nets for Linguistic Analysis*. PhD thesis, UiL-OTS, Utrecht.

Morrill, G. (1994). *Type Logical Grammar: Categorial Logic of Signs*. Kluwer, Dordrecht.

Steedman, M. (1985). Dependency and coordination in the grammar of dutch and english. *Language*, 61(3):523–568.

Steedman, M. (2000). *The Syntactic Process*. The MIT Press.

Uszkoreit, H. (1986). Categorial unification grammars. In *COLING*, pages 187–194.

Vijay-Shanker, K. and Weir, D. J. (1990). Polynomial time parsing of combinatory categorial grammars. In *ACL*, pages 1–8.

Vijay-Shanker, K. and Weir, D. J. (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27.

Weir, D. J. and Joshi, A. K. (1988). Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In *Meeting of the Association for Computational Linguistics*, pages 278–285.

Zeevat, H. (1988). Combining categorial grammar and unification. In Reyle, U. and Rohrer, C., editors, *Natural Language Parsing and Linguistic Theories*, pages 202–229. D. Reidel, Dordrecht.

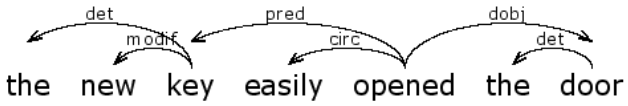
Towards Wide Coverage Categorial Dependency Grammars

Alexander Dikovsky

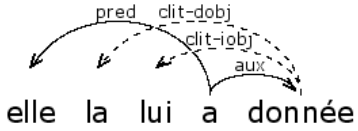
LINA CNRS UMR 6241, Université de Nantes
alexandre.dikovsky@univ-nantes.fr

1 Introduction

Categorial Dependency Grammars (CDG) [5, 4, 2] are categorial grammars generating sentences together with their *dependency structures* (DS). The DS may be trees (DT) as in Fig. 1 or not (cf. Fig . 2). Whatever is the case, they are linearly ordered and this order may be *projective*, as in Fig. 1(a), or not, as in Fig. 1(b). The non-projective order is due to *discontinuous dependencies* in which a governor g is separated from its subordinate s by a word not dominated by g (as it is the case of the auxiliary verb a in Fig. 1(b) separating the participle *donnée* from its pronominalized objects).



(a) A projective DT



(b) A non-projective DT (French: **she* it_{FEM} *to-him has given*)

Figure 1

The main advantage of CDG is that they allow to treat discontinuous non-projective dependencies strictly locally (i.e. within the lexicon entries of the governor and of the subordinate) and to parse the generated languages in a practical polynomial time.

To adapt the CDG for practical use, i.e. for elaboration of wide-coverage grammars, one should resolve the problem of grammar size explosion. Like the classical categorial grammars, CDG are lexicalized, i.e. are defined by a *lexicon*: a function λ assigning to each word a finite set of its dependency types. The primary source of lexicon size explosion is that the number of types per entry is close to the product of the numbers of possible argument subtypes. Another considerable source is a loose word order peculiar to many languages with reach

inflectional morphology (cf. Russian, German, Arabic, etc.). Finally, an excessive lexicon size may be due to massive type sharing between entries. There are various means aiming at a practical solution to this problem: regular expressions in types, supertagging, lexicon structuring. Below we show a way to reduce the lexicon size using *flexible types introduced through specific calculus rules.*

2 Generalized Categorial Dependency Grammars

We will consider the *Generalized CDG* (gCDG) ([5, 2]), which can generate non-tree dependency structures (for instance, a union of superposed trees).

A gCDG type has the form $[l_m\backslash\ldots\backslash l_1\backslash v/r_n/\ldots/r_1]^P$, where v is the (continuous) dependency on the governor, $l_m,\ldots,l_1$ and $r_n,\ldots,r_1$ are the (continuous) dependencies of left and right subordinates in the reverse order, and P is a sequence of *polarized valencies* (the *potential*) determining discontinuous dependencies. Types with empty potentials determine projective DT. E.g., the DT in Fig. 1(a) is determined by the type assignments:

$$\text{the} \mapsto det \quad \text{key} \mapsto [modif*\backslash det\backslash pred] \quad \text{new} \mapsto modif$$
$$\text{easily} \mapsto circ \; \text{opened} \mapsto [circ*\backslash pred\backslash S/dobj] \; \text{door} \mapsto [det\backslash dobj]$$

where the types $modif$ and $circ$ are *iterated* and S is the sentence type. Potentials are sequences of polarized valencies of four kinds: $\swarrow d$ (negative: that of the distant right governor through dependency d), $\searrow d$ (same on the left), $\nwarrow d$ (positive: that of a distant left subordinate through dependency d), $\nearrow d$ (same on the right). A pair of *dual* valencies $\swarrow d$ and $\nwarrow d$ ($\nearrow d$ and $\searrow d$) define a discontinuous dependency d. A negative valency can be anchored on a host word using anchor types $\#(\swarrow d)$, $\#(\searrow d)$. E.g., the lexicon below uniquely determines the DT in Fig. 1(b) ($\swarrow clit{-}dobj, \swarrow clit{-}iobj$ are anchored on the auxiliary a):

$$\text{elle} \mapsto pred \quad \text{la} \mapsto [\#(\swarrow clit{-}dobj)]^{\swarrow clit\text{–}dobj} \quad \text{lui} \mapsto [\#(\swarrow clit{-}iobj)]^{\swarrow clit\text{–}iobj}$$
$$\text{a} \mapsto [\#(\swarrow clit{-}iobj)\backslash\#(\swarrow clit{-}dobj)\backslash pred\backslash S/aux] \quad \text{donnée} \mapsto [aux]^{\nwarrow clit\text{–}dobj\nwarrow clit\text{–}iobj}$$

Such dependency types are formalized with the following calculus [1]

L$^{\mathbf{l}}$. $C^{P_1}[C\backslash\beta]^{P_2} \vdash [\beta]^{P_1P_2}$

I$^{\mathbf{l}}$. $C^{P_1}[C^*\backslash\beta]^{P_2} \vdash [C^*\backslash\beta]^{P_1P_2}$

Ω$^{\mathbf{l}}$. $[C^*\backslash\beta]^P \vdash [\beta]^P$

D$^{\mathbf{l}}$. $\alpha^{P_1(\swarrow C)P(\nwarrow C)P_2} \vdash \alpha^{P_1PP_2}$, if $(\swarrow C)P(\nwarrow C)$ satisfies the pairing rule **FA** (first available): P has no occurrences of $\swarrow C, \nwarrow C$.

L$^{\mathbf{l}}$ is the classical elimination rule. Eliminating the argument subtype $C \neq \#(\alpha)$ it constructs the (projective) dependency C and concatenates the potentials. $C = \#(\alpha)$ creates no dependency. **I**$^{\mathbf{l}}$ derives $k > 0$ instances of C. **Ω**$^{\mathbf{l}}$ serves for the case $k = 0$. **D**$^{\mathbf{l}}$ derives discontinuous dependencies. It pairs and eliminates dual valencies satisfying the rule **FA** to create the discontinuous dependency C.

For a DS D and a string x, let $G(D, x)$ denote the relation: D is constructed in a proof $\Gamma \vdash S$ for some $\Gamma \in \lambda(x)$. Then the *language* generated by G is the set $L(G) =_{df} \{w \mid \exists D\ G(D, w)\}$.

[1] We show left-oriented rules. The right-oriented are symmetric.

The gCDG are very expressive. Evidently, they generate all CF-languages. They can also generate non-CF languages.

Example 1 *The gCDG*

$G_{abc} : a \mapsto A^{\swarrow A}, [A\backslash A]^{\swarrow A},\ b \mapsto [B/C]^{\nwarrow A}, [A\backslash S/C]^{\nwarrow A},\ c \mapsto C, [B\backslash C]$

generates the language $\{a^n b^n c^n \mid n > 0\}$ *[6]. For instance,* $G_{abc}(D^{(3)}, a^3b^3c^3)$ *holds for the dependency structure (DS) in Fig. 2 and the string* $a^3b^3c^3$ *due to the proof in Fig. 3.*

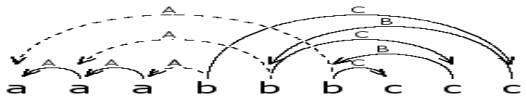


Fig. 2. Dependency structure for $a^3b^3c^3$

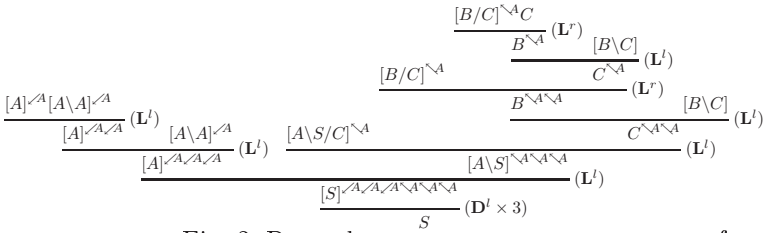


Fig. 3. Dependency structure correctness proof

Seemingly, it is incomparable with mildly CS languages ([8]) generated by multi-component TAG, linear CF rewrite systems and some other grammars. $\mathcal{L}(gCDG)$ contains non-TAG languages, e.g. $L^{(m)} = \{a_1^n a_2^n ... a_m^n \parallel n \geq 1\}$ for all $m > 0$. In particular, it contains the language $MIX = \{w \in \{a,b,c\}^+ \mid |w|_a = |w|_b = |w|_c\}$, for which E. Bach conjectures that it is not mildly CS. On the other hand, we conjecture [3] that this family does not contain the copy language $L_{copy} = \{xx \mid x \in \{a,b\}^*\}$, which is TAG [2]. gCDG have an efficient polynomial time parsing algorithm [3, 2] which resides upon a property of independence of basic types and polarized valencies expressed in terms of *projections* of types and *"well-bracketing"* criteria for potentials.

For a sequence of types γ, its *local projection* $\|\gamma\|_l$ and *de valency projection* $\|\gamma\|_v$ are defined as follows:

1. $\|\varepsilon\|_l = \|\varepsilon\|_v = \varepsilon$; $\|\alpha\gamma\|_l = \|\alpha\|_l\|\gamma\|_l$ and $\|\alpha\gamma\|_v = \|\alpha\|_v\|\gamma\|_v$ for a type α.

2. $\|C^P\|_l = C$ et $\|C^P\|_v = P$ for every type C^P.

To speak about "well-bracketing" of potentials, we interpret $\swarrow d$ and $\nearrow d$ as *left brackets* and $\nwarrow d$ and $\searrow d$ as *right brackets*. A potential is *balanced* if it is well bracketed in the usual sense.

Let $\mathbf{c}$ be the projective core of the dependency calculus, consisting of the rules $\mathbf{L}$, $\mathbf{I}$ and $\mathbf{\Omega}$ and $\vdash_{\mathbf{c}}$ denote the provability relation in this sub-calculus. Then the *projections independence* property of gCDG [3, 2] is formulated as follows.

Theorem 1 *For a gCDG G with lexicon* λ *and a string* x, $x \in L(G)$ *iff there is* $\Gamma \in \lambda(x)$ *such that* $\|\Gamma\|_l \vdash_{\mathbf{c}} S$ *and* $\|\Gamma\|_v$ *is balanced.*

[2] It doesn't prevent that gCDG express the cross-serial dependencies [5].

3 Extension of gCDG with Flexible Type Rules

1. Alternative rules.
The first extension allows to use alternative choice of a subtype in a type: $(C_1|\dots|C_k)$ meaning intuitively "one of types C_i". E.g., assigning to *are* the type $[pred\backslash S/(n-copul|c-copul|q-copul)]$ one can replace three entries: $\lambda : are \mapsto [pred\backslash S/n-copul])$, $\lambda : are \mapsto [pred\backslash S/c-copul])$, $\lambda : are \mapsto [pred\backslash S/q\text{–}copul])$ by one entry: $\lambda : are \mapsto [pred\backslash S/(n\text{–}copul|c\text{–}copul|q\text{–}copul)])$. Here are the calculus rules allowing such types:

$\mathbf{LA^l_{gov}}$. $C^{P_1}[(C|\alpha)\backslash\beta]^{P_2} \vdash [\beta]^{P_1P_2}$
$\mathbf{LA^l_{sub}}$. $(C|\alpha)^{P_1}[C\backslash\beta]^{P_2} \vdash [\beta]^{P_1P_2}$
$\mathbf{LA^l_{gs}}$. $(C|\alpha_1)^{P_1}[(C|\alpha_2)\backslash\beta]^{P_2} \vdash [\beta]^{P_1P_2}$

2. Flexible order rules.
The next extension allows to eliminate left (right) subordinates whose position with respect to other subordinates is not fixed. It is supported by two rules treating the set of such not restrained subordinates as a bag. The first rule eliminates the subordinates belonging to the bag. The second rule eliminates fixed-position subordinates irrelative to presence and position of the bag subordinates.

2.1. X-sided bag rules. We show inside and outside left-sided bag rules (X=l):

$\mathbf{LB^l_i}$. $C^{P_1}[\{C,\alpha\}\backslash\beta]^{P_2} \vdash [\{\alpha\}\backslash\beta]^{P_1P_2}$, $C \notin \beta$.
$\mathbf{LB^l_o}$. $C^{P_1}[\{\alpha\}\backslash C\backslash\beta]^{P_2} \vdash [\{\alpha\}\backslash\beta]^{P_1P_2}$, $C \notin \alpha$.

The next two rules support elimination of subordinates whose position with respect to the governor is also not fixed.

2.2. Unoriented bag rules. We show inside and outside unoriented bag rules:

$\mathbf{LB^u_i}$. (a) $C^{P_1}[\beta]^{P_2}_{\{C,\alpha\}} \vdash [\beta]^{P_1P_2}_{\{\alpha\}}$, (b) $[\beta]^{P_2}_{\{C,\alpha\}}C^{P_1} \vdash [\beta]^{P_2P_1}_{\{\alpha\}}$ $(C \notin \beta)$.
$\mathbf{LB^u_o}$. $C^{P_1}[C\backslash\beta]^{P_2}_{\{\alpha\}} \vdash [\beta]^{P_1P_2}_{\{\alpha\}}$, $C \notin \alpha$.

Example 2 *Let us consider the following fragment of a lexicon for German:*

$$\begin{cases} deshalb \mapsto [circ] & gab \mapsto [circ^*\backslash S/\{pred, dobj, iobj\}] \\ Frau \mapsto [det\text{–}df\backslash iobj] & das \mapsto [det\text{–}a] \\ Mann \mapsto [det\text{–}n\backslash pred] & der \mapsto [(det\text{–}n|det\text{–}df)] \\ Buch \mapsto [det\text{–}a\backslash dobj] \end{cases}$$

The proof in Fig. 4 shows correctness of this assignment for the sentence deshalb gab der Mann der Frau das Buch *and for its DS. The proof in Fig. 5 shows correctness of the same assignment for the sentence* deshalb gab das Buch der Mann der Frau *with a different word order and a different DS.*

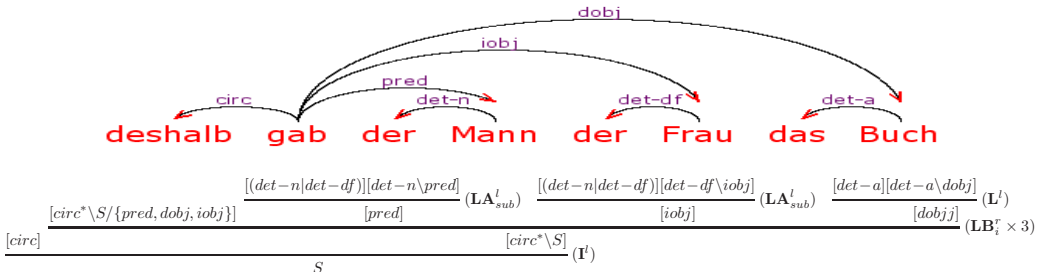


Fig. 4. Correctness proof for the DS of *deshalb gab der Mann der Frau das Buch*

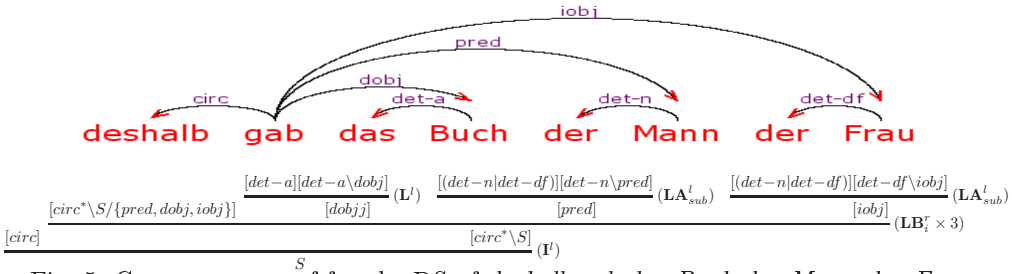


Fig. 5. Correctness proof for the DS of *deshalb gab das Buch der Mann der Frau*

Let us denote by $\mathcal{L}(fgCDG)$ the set of all languages generated by the *extended flexible type gCDG*. It is easy to prove that:
1. $\mathcal{L}(fgCDG) = \mathcal{L}(gCDG)$,
2. Theorem 1 holds for the extended gCDG as well, and
3. Parsing of $fgCDG$ has the same complexity as for $gCDG$.

Remark. One should distinguish three, in principle different, but in practice mixing together techniques: *meta-expressions for rules*, *grammar factorization*, and *conservative calculus extensions*. Early techniques (cf. link grammars [7]) used meta-expressions. Nowadays, they are accompanied by grammar factorization through classifications, inheritance etc. (cf. successful combination of the two applied to TAGs, e.g. [1]). The two techniques require compilation into a target grammar. The third techniques, applied to gCDG in this paper, needs no compilation. When it applies to an original calculus, it gives an equivalent calculus of the same complexity applied to more compact grammars without changing them. Factorization of flexible gCDG is now under study.

References

1. E. de la Clergerie. From metagrammars to factorized tag/tig parsers. In *In Proc. of IWPT'2005*, pages 190–191, Vancouver, Canada, 2005.
2. M. Dekhtyar and A. Dikovsky. Generalized categorial dependency grammars. In A. Aviron et al., editor, *Trakhtenbrot/Festschrift*, LNCS 4800, pages 230–255. Springer Verlag, 2008.
3. Michael Dekhtyar and Alexander Dikovsky. Categorial dependency grammars. In M. Moortgat and V. Prince, editors, *Proc. of Intern. Conf. on Categorial Grammars*, pages 76–91, Montpellier, 2004.
4. A. Dikovsky. A finite-state functional grammar architecture. In D. Leivant and Ruy de Queiroz, editors, *Logic, Language, Information and Computation. Proc. of the 14th Intern. Workshop WoLLIC 2007*, LNCS 4576, pages 131–146, Rio de Janeiro, Bresil, 2007. Springer Verlag.
5. A. Dikovsky. Multimodal categorial dependency grammars. In *Proc. of the 12th Conference on Formal Grammar*, pages 1–12, Dublin, Ireland, 2007.
6. Alexander Dikovsky. Dependencies as categories. In *"Recent Advances in Dependency Grammars". COLING'04 Workshop*, pages 90–97, 2004.
7. Daniel Sleator and Davy Temperly. Parsing English with a Link Grammar. In *Proc. IWPT'93*, pages 277–291, 1993.
8. K. Vijay-Shanker and D.J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–545, 1994.

PPQ : a pregroup parser using majority composition

Denis Béchet[1] and Annie Foret[2]

[1] LINA CNRS – UMR 6241 – Université de Nantes
2, rue de la Houssiniére – BP 92208
44322 Nantes Cedex 03 – France
Denis.Bechet@univ-nantes.fr
[2] IRISA – Université de Rennes 1
Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 Rennes Cedex – France
Annie.Foret@irisa.fr

Abstract. Pregroup grammars are a mathematical formalism in the spirit of categorial grammars. They are close to logical formalism like Lambek calculus but have a polynomial parsing algorithm. The paper presents a parser based on pregroup grammar that uses a tabular approach based on *majority partial composition*.
Keywords: parser, pregroups, Lambek categorial grammars, parsing software, XML data.

1 Introduction

Pregroup grammars (PG) [1] have been introduced as a simplification of Lambek calculus [2]. They have been used to model fragments of syntax of several natural languages. They belong to categorial and lexicalized grammatical frameworks : categorial grammars have nice links to semantical interpretation while lexicalism has many advantages for constructing grammars and for parsing.

Another interest of PG is their order on basic types, that helps grammar design with natural and compact types (less types) ; this point can also be generalized to combine calculi, both formally and in software [3]. In contrast to some other categorial variants, PG parsing is polynomial ($O(n^3)$).

Based on the PG formalism, and some extensions of it, we have programmed a pregroup toolbox, including a specific parser based on partial composition (in contrast to [4, 5]), and a grammar definition tool. Data are stored in XML format (with DTD), to allow better interconnections with other tools. A web version is also provided for parsing with a grammar, either from raw text, from (partially) parenthesized text or from analyzed text (as in XML treebanks). This article explains the tool characteristics in connection with the underlying formalism, and gives an overview of the toolbox.

2 Pregroups

2.1 Pregroup Grammars

A *pregroup* is a structure $(P, \leq, \cdot, l, r, 1)$ such that $(P, \leq, \cdot, 1)$ is a partially ordered monoid and l, r are two unary operations on P that satisfy for every element $x \in P$, $x^l x \leq 1 \leq x x^l$ and $x x^r \leq 1 \leq x^r x$. We write $p^{(0)} = p$, and $p^{(n)}$ for $(p^{(n-1)})^r$ if $n > 0$ and $p^{(n)}$ for $(p^{(n+1)})^l$ if $n < 0$; these are called simple types. From now on, let P denote a free pregroup based on a poset of basic types written $(Pr, \leq_{Pr})$.

A *Pregoup Grammar* G is $G \subset \Sigma \times P$ (Σ words, G finite). Its language $L(G) \subseteq \Sigma^+$ is the set of sequence of words such that the concatenation of types entails ($\leq$) the distinguished type s.

Parsing can be based on rewrite rules such as: $Xp^{(n)}q^{(n+1)}Y \overset{(GCON)}{\longrightarrow} XY$
if $p \leq_{Pr} q$ and n is even or if $q \leq_{Pr} p$ and n is odd

Parsing using partial and majority composition Rules below proceed by pairs of words (their types are separated by a comma) ; thus parsing also provides a binary tree on words.

- [**C**] **(partial composition)** : for $k \in \mathbb{N}$, $X' = p_1^{(n_1)} \cdots p_k^{(n_k)}, Y' = q_k^{(n_k+1)} \cdots q_1^{(n_1+1)}$

$$\Gamma, Xp_1^{(n_1)} \cdots p_k^{(n_k)}, q_k^{(n_k+1)} \cdots q_1^{(n_1+1)} Y, \Delta \overset{C}{\longrightarrow} \Gamma, XY, \Delta$$

  if $p_i \leq_{Pr} q_i$ and n_i is even or if $q_i \leq_{Pr} p_i$ and n_i is odd , for $1 \leq i \leq k$.
- [$\overset{@}{\longrightarrow}$] **(majority composition)** : if (moreover) the result's width (of $|XY|$) is not greater than the maximum argument width (of XX' and $Y'Y$)

2.2 Pregroup extended with iteration types

For iteration types p^*, the parser is also based on partial composition rules:

- [**C**] **(partial composition)** : for $X'Y' \leq Z'$, with Z' empty (as 1) or $a^{*(2k+1)}$:

$$\Gamma, XX', Y'Y, \Delta \overset{C}{\longrightarrow} \Gamma, XZ'Y, \Delta$$

- [$\overset{@}{\longrightarrow}$] **(majority composition)** : if (moreover) at most a half of the argument is in the result ($|X'| \geq |X|$ or $|Y'| \geq |Y|$).

Example 1. Let us see the following sentence taken from "Un amour de Swann" by M. Proust: *Maintenant, tous les soirs, quand il l'avait ramenée chez elle, il fallait qu'il entrât.* [3] In Fig. 1 we show a proof of correctness of assignment of types to its fragment. The primitive types used in this proof are: π_3 and $\overline{\pi_3}$ = third person (subject) with $\pi_3 \leq \overline{\pi_3}$, p_2 = past participle, ω = object, s = sentence, s_5 = subjunctive clause, with $s_5 \leq s$, σ = complete subjunctive clause, τ = adverbial phrase. This grammar assigns s to the following sentence:

quand il l' avait ramenée chez-elle il fallait qu' il entrât

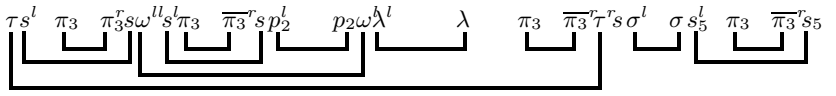
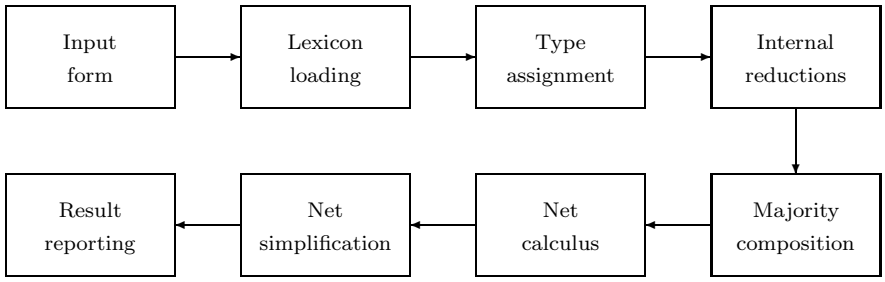


Figure 1

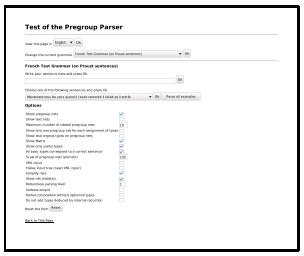
3 Parsing using majority partial composition

A Cocke-Younger-Kasami (CYK) algorithm for pregroup grammar can be developed. The granularity of this algorithm is words (or entries if the lexicon assigns also types to a sequence of words like "pomme de terre" (potato in French)). This method presented in [6] has been implemented into a tabular parser together with other components:



[3] [FR: *Now, every evening when he took back her to her home, he ought to enter*]

Input form.



This form selects one of the grammar, asks for the input string (alternatively, one may choose one or all samples that are associated to the selected lexicon). Several options are also offered. For instance, the parser can show only a limited number of analyses (ten by default). The user can enter a different limit. An option selects if the input is a string or an XML tree. In this case, the parser can also follow the XML structure when computing majority partial composition.

Lexicon loading.

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar>
 <pregroup>
  <order inf="n" sup="n-bar"/>
...
 </pregroup>
 <sentence type="s"/>
 <lexicon>
  <w><mot>whom</mot>
     <type><simple atom="q'"/>
           <simple atom="o" exponent="-2"/>
           <simple atom="q" exponent="-1"/>
     </type>
  </w>
...
 </lexicon>
</grammar>
```

Grammars are described in XML files. A grammar defines a partial order on basic types, a set of basic types that are considered to form the correct sentences and a lexicon that associated to an entry (a list of tokens) a set of types. It is also possible to describe special entries with a regular expression that is useful for instance for the class of numerical number or the class of proper noun (that starts with an upper case letter). To improve the efficiency of this step that may be very long if the lexicon is big (Lefff 2.5.5[7] has 534753 entries – The PPQ XML corresponding lexicon is a file whose size is 31,367,146 bytes), a compressed text format or a SQLite database can be used.

With an indexed table, even a big lexicon is accessed very quickly (less than a second rather than several tens of seconds). In fact, the parser does not load all the lexicon. It selects the entries that correspond to the input string.

Type assignment to words/entries. This step assigns to each list of tokens of the input string a set a pregroup types. The input string is split into tokens using spaces and several regular expressions For instance *l'homme* (the man in French) is segmented into two tokens: *l'* and *homme*. If the string is split in n tokens, there are $n \times (n+1)/2$ possible entries. Each one is searched in the lexicon and defines the initial value of the parsing matrix that computes the types associated with each segment of tokens of the input string.

Internal reduction of types. Because the main step of the parser is based on majority partial composition, a completion using internal reduction must be performed on the types computed during the previous step (see [6]). Usually this step is not useful because the types in the grammar cannot be reduced or the grammar already includes all the derived types.

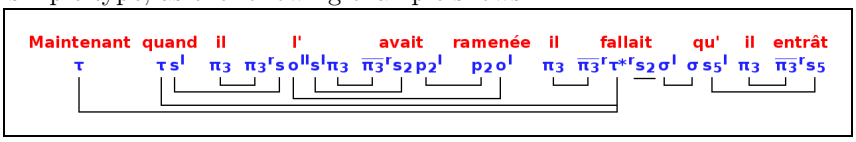
Majority partial composition of sequence of entry.

The Matrix Content

cell 1-4 q'			
cell 1-3 q' o^{ll} p_2^{l}	cell 2-4 q o^{l}		
cell 1-2	cell 2-3 q p_2^{l}	cell 3-4	
cell 1-1 q' o^{ll} q^{l}	cell 2-2 q p_2^{l} π_2^{l}	cell 3-3 π_2	cell 4-4 p_2 o^{l}
whom	have	you	seen

This step computes the parsing matrix with the result of majority partial composition (rather than using production rules of the Chomsky normal form of a context-free grammar for CYK algorithm). Of course, because we also want to describe the resulting analyses as pregroup nets, the matrix is in fact a complex directed acyclic graph. This matrix may be displayed by the parser at the end of the report. The matrix of "whom have you seen" as input string for Test grammar is displayed on the left.

Net calculus. This step computes representation of the analyses of the parser. They are called pregroup nets. In a net, each entry is associated to a pregroup type and the link represents the different axioms that associate two by two the simple types. This representation is close to a dependency tree except that the structure is a graph rather than a tree. Moreover, with the introduction of iterative simple types[8], a simple type can be connected to more than one other simple type, as the following example shows.



Net simplifications. This step simplifies nets by suppressing the iterated simple types that are not used and by taking into account cut annotations (see Section 4).

Result reporting. This step puts together all the results and presents it using different formats. Actually, there are three possible output formats: an HTML format useful for a web server, a text output that is suitable for a terminal and an XML format that may be used if the output needs to be processed by another program.

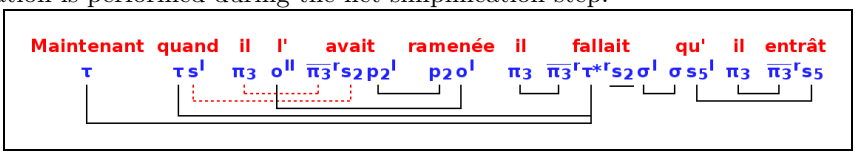
4 Parsing with added cuts

Grammars based on free pregroups even with iterative simple types are context free. Thus, for several complex syntactical constructions some types must include a way to cross part of the environment. This is particularly the case for non projective dependencies. For instance, the French clitics are placed between the verb and the subject: In "il la mange" (he is eating it) "la" is between "il" and "mange". The previous example shows such a construction. The clitic "l'" is assigned $\pi_3^r s o^{ll} s^l \pi_3$. The main simple type is o^{ll}. The rest enables the crossing of two axioms (one corresponding to π_3^r and π_3, the other to s and s^l).

To solve this problem, PPQ uses *cut* annotations on the types assigned to special words like clitics or adverb. These annotations that can be seen as a limited form of semantical interpretation replace two normal axiom links by a long distance axiom link.

The previous French example has such annotations for the type associated to the clitic *l'*. Here two cuts have been added, one between π_3^r and π_3 and one between s and s^l. Thus on the net, the axiom between π_3 of *il* and π_3^r of *l'* and the axiom between π_3 of *l'* and π_3^r of *avait* are replaced by a single long distance axiom between π_3 of *il* and π_3^r of *avait*. Another long distance axiom is created for the other cut that links s^l of *quand* and s^2 of *avait*.

On the final picture, these special long distance axioms are shown as dashed red lines. Such lines can cross other axioms. The cut simple types are also erased from the picture. This interpretation is performed during the net simplification step.



5 Grammar construction

Other packages concern the construction of XML pregroup grammars : xslt programs have been developed for this task, including a specific mode for the French Paris7 Treebank. Another set of programs (XML2CTX, LIS2XML) provides an interface with Camelis/Glis (http://www.irisa.fr/LIS/ferre/camelis/index.html) an implementation of Logical Information Systems (LIS), allowing navigation. A user can define a lexicon with Glis, then save it

as a LIS context, where objects are words ; this context is then transfered to the pregroup XML format (using LIS2XML) conversely, a pregroup grammar in XML format, can be transfered to a LIS context (using XML2CTX). This mode has been used for several prototype languages.

6 Conclusion

The pregroup parser PPQ implements majority partial composition. This program, that can be used inline through a PHP webserver or as a command line program, uses XML files for describing a pregroup grammar. An optional indexed database can speed up the lookup in the lexicon. The result is a HTML or text page with pregroup nets as syntactical analysis that is convenient for human reading. The command line program can also produce a XML output if the result must be used by another program. This model also enables a form of semantical interpretation limited to the reduction of annotated "cuts".

This program which is rather a test platform than a finished software has at present a large cover of the French language (however with a rough pregroup type system) and several toy lexicons for English, Breton (a Celtic language) and Bambara (an African language).

References

1. Lambek, J.: Type grammars revisited. In Lecomte, A., Lamarche, F., Perrier, G., eds.: Logical aspects of computational linguistics: Second International Conference, LACL '97, Nancy, France, September 22–24, 1997; selected papers. Volume 1582., Springer-Verlag (1999)
2. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly **65** (1958)
3. Foret, A.: Pregroup calculus as a logical functor. In: Proceedings of WOLLIC 2007. Volume LNCS 4576., Springer (2007)
4. Degeilh, S., Preller, A.: Efficiency of pregroup and the french noun phrase. Journal of Language, Logic and Information **14**(4) (2005) 423–444
5. Oehrle, R.: A parsing algorithm for pregroup grammars. In Moortgat, M., Prince, V., eds.: Proc. of Intern. Conf. on Categorial Grammars, Montpellier (2004)
6. Béchet, D.: Parsing pregroup grammars and Lambek calculus using partial composition. Studia logica **87**(2/3) (2007)
7. Sagot, B., Clément, L., de la Clergerie, E.V., Boullier, P.: The lefff 2 syntactic lexicon for french: architecture, acquisition. In: LREC'06. (2006)
8. Béchet, D., Dikovsky, A., Foret, A., Garel, E.: Optional and iterated types for pregroup grammars. In: Proceedings of the 2nd International Conference on Language and Automata Theory and Applications (LATA 2008), March 2008, Tarragona, Spain. Lecture Notes in Computer Science (LNCS), Springer (2008) 88–100
9. Došen, K.: Cut Elimination in Categories. Kluwer Academic publishers (1999)
10. Buszkowski, W.: Cut elimination for the lambek calculus of adjoints. In Abrusci, V., Casadio, C., eds.: New Perspectives in Logic and Formal Linguisitics, Proceedings Vth ROMA Workshop, Bulzoni Editore (2001)

Parsing CCGbank with the Lambek Calculus

Timothy A. D. Fowler
Department of Computer Science
University of Toronto
10 King's College Road, Toronto, ON, M5S 3G4, Canada
tfowler@cs.toronto.edu

May 1st, 2009

Abstract

This paper will analyze CCGbank, a corpus of CCG derivations, for use with the Lambek calculus. We also present a Java implementation of the parsing algorithm for the Lambek calculus presented in Fowler (2009) and the results of experiments using that algorithm to parse the categories in CCGbank. We conclude that the Lambek calculus is computationally tractable for this task and provide insight into a full conversion of CCGbank to a bank of Lambek derivations.

1 Introduction

The Lambek calculus (Lambek, 1958) and Combinatory Categorial Grammar (CCG) (Steedman, 2000) are two related variants of categorial grammar that have received a lot of attention from the computational linguistics community.

The attention that the Lambek calculus has received has been mostly theoretical, with particularly important results including its weak equivalence to context-free grammars (CFGs) (Pentus, 1997) and the NP-completeness of its parsing problem (Savateev, 2008). In response to these results, Tiede (1999) proves that the Lambek calculus and CFGs are not strongly equivalent and Fowler (2009) provides a method for restricting the Lambek calculus to obtain a polynomial time parsing algorithm. CCG research, on the other hand, has been more practically focused. Hockenmaier and Steedman (2007) introduces CCGbank, a corpus of CCG derivations translated from the WSJ section of the Penn treebank. Then, Clark and Curran (2004) describe a wide-coverage statistical parser that using CCGbank for training and that is competitive with other state of the art parsers.

The aim of this paper is to begin to bring these two research streams together. That is, we will investigate the practicality of using the algorithm of Fowler (2009) to parse sentences using the categories of CCGbank. First, we analyze CCGbank for use with the Lambek calculus and present the results of some experiments using a Lambek calculus grammar based on the categories in CCGbank. Then, we provide methods for converting the most problematic derivations in CCGbank to Lambek calculus derivations for use in future research.

One might be tempted to ignore the Lambek calculus in favour of extending standard CCG by introducing rules such as the D-combinator of Hoyt and Baldridge (2008) as they become necessary for linguistic analysis. However, Zielonka (1981) proves that the introduction rules of the Lambek calculus cannot be replaced by any finite number of such rules. Thus, in one step the Lambek calculus gives us a rule system that would require the addition of an infinite number of logically sound combinatory rules.

2 The Lambek Calculus

The set of *categories* $\mathcal{C}$ is built up from a set of *atoms* (e.g. $\{S, NP, N, PP\}$) and the two binary connectives $/$ and $\backslash$. A *Lambek grammar* G is a 4-tuple $\langle \Sigma, A, R, S \rangle$ where Σ is an *alphabet*, A is a set of atoms, R is a relation between symbols in Σ and categories in $\mathcal{C}$ and S is the set of *sentence categories*.

The order of a category is a measure of its complexity, which measures the depth of the nesting of arguments and is formally defined as follows:

$$\begin{aligned} o(\alpha) &= 0 \text{ for } \alpha \text{ a basic category} \\ o(\alpha/\beta) &= o(\beta\backslash\alpha) = \max(o(\alpha), o(\beta) + 1) \end{aligned}$$

For example, the order of $(S/NP)\backslash(S/NP)$ is 2. A *Lambek grammar with order bounded by* k is a Lambek grammar such that R is a relation between symbols in Σ and categories in $\mathcal{C}$ of order $\leq k$.

Parsing with the Lambek calculus is equivalent to logical deduction from the categories of the sentence. The inference rules for the Lambek calculus are shown in figure 1. The $/E$ and $\backslash E$ rules are referred to as *elimination rules*. The $/I_i$ and $\backslash I_i$ are referred to as *introduction rules* and allow the introduction of hypothetical categories (shown in square brackets) with the requirement that they are *discharged* at a point in the proof where they are the rightmost (in the case of $/I_i$) or leftmost (in the case of $\backslash I_i$) category in the proof tree.

$$\frac{X/Y \qquad Y}{X}\,/E \qquad \frac{Y \qquad Y\backslash X}{X}\,\backslash E$$

$$\frac{\begin{array}{c}[Y]_i \\ \vdots \\ X\end{array}}{X/Y}\,/I_i \qquad \frac{\begin{array}{c}[Y]_i \\ \vdots \\ X\end{array}}{Y\backslash X}\,\backslash I_i$$

Figure 1: Inference rules in the Lambek Calculus.

The parsing problem for a string of symbols $s_1 \ldots s_k$ can be characterized as follows: $s_1 \ldots s_k \in \Sigma^*$ is parseable in a Lambek grammar $\langle \Sigma, A, R, S \rangle$ if there exists $c_1, \ldots, c_k \in \mathcal{C}$ such that $c_i \in R(s_i)$ for $1 \leq i \leq k$ and the categories $c_1, \ldots, c_k$ logically derive some category in S via the rules in figure 1. Fowler (2009) provides a chart parsing algorithm that uses abstract term graphs (ATGs) as entries in the chart that is polynomial time if the order of categories is bounded. It is this algorithm that we will be evaluating for practical use with CCGbank.

3 CCG

Combinatory Categorial Grammar (CCG) (Steedman, 2000) is a well-known variant of categorial grammar that incorporates a number of combinatory rules in addition to the *elimination rules* of the Lambek calculus. We will divide the combinatory rules into two groups. Those which are derivable in the Lambek calculus:

- **C**omposition
  1. $X/Y, Y/Z \Rightarrow X/Z$ (Forward)
  2. $Z\backslash Y, Y\backslash X \Rightarrow Z\backslash X$[1] (Backward)
- **T**ype-Raising
  1. $X \Rightarrow T/(X\backslash T)$ (Forward)
  2. $X \Rightarrow (T/X)\backslash T$ (Backward)

And those rules which are not derivable in the Lambek calculus (which we will refer to as non-Lambek rules):

- **C**rossed Composition
  1. $X/Y, Z\backslash Y \Rightarrow Z\backslash X$ (Forward Crossing)
  2. $Y/Z, Y\backslash X \Rightarrow X/Z$ (Backward Crossing)
- **S**ubstitution
  1. $(X/Y)/Z, Y/Z \Rightarrow X/Z$
  2. $Z\backslash(X/Y), Z\backslash Y \Rightarrow Z\backslash X$
  3. $Z\backslash Y, Z\backslash(Y\backslash X) \Rightarrow Z\backslash X$
  4. $Y/Z, (Y\backslash X)/Z \Rightarrow X/Z$

[1] For consistency we will be using the Ajdukiewicz notation, rather than the Steedman notation usually used for CCG.

Any linguistic analysis made within CCG which uses any of the non-Lambek rules will need to be modified to receive an analysis within the Lambek calculus.

4 The Lambek Calculus and CCGbank

Section 4.1 discusses the order of the categories in CCGbank. Section 4.2 discusses an implementation of the algorithm of Fowler (2009) run on the categories of CCGbank. Section 4.3 discusses the necessary major modifications to convert the CCG derivations of CCGbank to Lambek derivations.

4.1 The order of CCGbank

The parsing algorithm described in Fowler (2009) is only polynomial time if the order of categories are bounded by a constant. Furthermore, the algorithm is exponential in that constant. So, the practicality of parsing CCGbank using this algorithm is very dependent on the order of categories in CCGbank. Figure 2 shows the distribution of order across the categories occurring in sections 02 to 21 of CCGbank. The maximum order of any category is 5 and the average is 0.78. These results indicate that the parsing algorithm will be efficient on CCGbank categories, if not competitive with other categorial grammar parsing strategies.

4.2 Parsing CCGbank

We implemented the parsing algorithm of Fowler (2009) in Java and performed our experiments on a machine with 4 Intel Xeon CPUs at 3.0 Ghz and 16Gb of RAM. Our training data was taken from sections 02-21 of CCGbank and our test data was taken from section 00. We have reserved section 23 for future use.

We define Lambek grammars $\langle \Sigma, A, R_t, S \rangle$ where A is the set of atoms appearing in sections 02-21.[2]

[2]We ignore the feature system of CCGbank because the parsing algorithm cannot currently handle the unification of features.

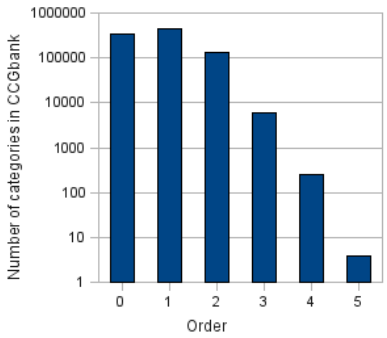


Figure 2: The order of categories in CCGbank

t	Category-Word pairs	Categories per word
0.1	59683	1.350
0.01	69382	1.569
0.001	73265	1.657
0	74667	1.689

Table 1: The effect of the threshold on the lexicon.

Σ is the set of words appearing in both sections 00 and 02-21 except that we will disregard punctuation entirely. R_t will be defined as follows:

- $R_t(s) = \{c | c$ has frequency at least t among the categories for $s\}$ for s in sections 02-21
- $R_t(s) = \{NP, N\}$ for s not in sections 02-21

NP and N were chosen because nouns and proper nouns make up a large portion of unknown words in corpora. The sentence categories are defined as $S = \{S, NP, PP, S/NP, NP\backslash S\}$ which are the root categories of trees in sections 02-21. We will be using the following values for t: 0.1, 0.01 and 0.001. Their effect on the lexicon is shown in table 1.

The results of the experiments are shown in table 2. We can see that only a very small number of sentences receive parses even with a value of $t = 0.001$. The reason for this is the very large number of unary type changing rules from N to NP in CCGbank. Without solving the unary type changing problem in CCGbank, we can increase the number of derivable sentences (and consequently get a more realistic

Grammar Type	Sentences with Parses	Seconds per Sentence	Seconds Total	Mean # of Atoms	Mean # of ATGs
Basic, $t = 0.1$	47/1913	0.365	700	104.4	73.4
Basic, $t = 0.01$	112/1913	0.381	729	241.3	409.7
Basic, $t = 0.001$	247/1913	4.034	7717	611.4	5280.1
$N = NP$, $t = 0.1$	289/1913	0.369	706	104.4	183.8
$N = NP$, $t = 0.01$	678/1913	0.578	1106	241.3	1275.3
$N = NP$, $t = 0.001$	1005/1913	25.044	47908	611.4	14995.5

Table 2: Results of experiments

picture of practical parsing) by converting all occurrences of N to NP (denoted $N = NP$ in figure 2).

Without converting N to NP, parsing is efficient, but this is largely due to a small number of entries in the chart. However, when we equate N and NP, parsing remains efficient for t values of 0.1 and 0.01 and the higher numbers of sentences receiving parses makes these cases a more realistic picture of parsing. The last case ($N = NP$, $t = 0.001$) sees a huge increase in parse time, but also a significant increase in coverage. It is likely that the parse time here would be reduced if we used the supertagging techniques of Clark and Curran (2004). In addition, the low coverage of even our best method is likely due to the categories in CCGbank being designed to participate in non-Lambek rules. This will be discussed in the next section.

4.3 Converting CCGbank

In addition to the usual non-Lambek rules of CCG there are a number of additional rules used in derivations in CCGbank which can be characterized as general phrase structure rules. Due to the nature of parsing algorithms for CCG, these rules do not introduce any additional difficulties for parsing CCGbank using CCG but they cannot be easily accommodated into a logical approach such as the Lambek calculus. In this section, we analyze the non-Lambek rules of CCGbank and discuss methods for converting these analyses to ones using Lambek's rules.

Table 3 categorizes the binary rules in CCGbank according to their type. The first 6 rule types are the rules from CCG and the last 3 are general phrase structure rules. The group of Crossing Composition rules are of concern because they are large in number and because they are a fundamental part of CCG. The vast majority of such rules attach lexical adjuncts to items that they modify. The directions of the slashes in these lexical items can be modified so that this is no longer crossing composition. In some cases, this will increase lexical ambiguity, but this lexical ambiguity appears necessary unless we adopt some kind of crossing rules into the Lambek calculus. The "Punctuation and Conjunction" rules are rules for handling various kinds of punctuation and conjunctions such as "and" and "or". One way that these rules can be handled is to assign certain categories to punctuation marks (Hockenmaier, 2003, pg. 34) and assigning coordination categories such as $(NP\backslash NP)/NP$ to the conjunctions coordinating NPs. The small number of rules under the "Merger" category are rules that take two identical atoms and produce one instance of the same atom. These can be altered on a case by case basis since there are so few and most can be modified so that the rule becomes an application rule. Similarly, for the "Other" rules.

Table 4 groups the unary rules of CCGbank into categories. The Type-Raising rules are Lambek rules, but the remainder must be altered to be considered in a Lambek framework. Of these unary rules, the vast majority are of the type $N \Rightarrow NP$ meaning that within the derivation we need to convert some category's instance of an N atom into an NP atom. These can easily be changed in the lexicon, increasing lexical ambiguity but allowing for a more

Rule Type	# in CCGbank
Forward Application	674513
Backward Application	227663
Forward Composition	7947
Backward Composition	1844
Crossing Composition	14468
Substitution	4
Punctuation and Conjunction	172987
Mergers	32
Other	6
Total	1099464

Table 3: Counts of Binary Rules

Rule Type	# in CCGbank
Type-Raising	4016
$N \Rightarrow NP$	142525
$NP\backslash S \Rightarrow NP\backslash NP$	8986
$NP\backslash S \Rightarrow (NP\backslash S)\backslash(NP\backslash S)$	3421
Other	4690
Total	163638

Table 4: Counts of Unary Rules

classical categorial analysis. In addition, there are rules for handling relative clauses of the form $S/NP \Rightarrow NP\backslash NP$. These rules can be handled in a similar way, but the effect on the size of the lexicon is an open question.

5 Conclusion

Our experiments indicate that the NP-completeness of parsing with the Lambek calculus is not the barrier that it seems to be, given the fact that our implementation is basic and can be improved in a number of ways. The low coverage of the parser is problematic, but is likely due to the fact that we have not yet fully converted CCGbank for use with the Lambek calculus. To address this, we have outlined methods for doing such a conversion.

However, the benefits of using the Lambek calculus as a grammar over a combinatory categorial grammar cannot be fully realized until the derivations in the corpus take advantage of the logical basis of Lambek's introduction rules. This can be accomplished by revisiting the methods used to convert the Penn treebank to CCG by Hockenmaier and Steedman (2007).

References

S. Clark and J. R. Curran. Parsing the WSJ using CCG and log-linear models. *Proceedings of ACL '04*, pages 104–111, 2004.

T.A.D. Fowler. A Polynomial Time Algorithm for Parsing with the Bounded Order Lambek Calculus. In *Proceedings of MOL '09*, 2009.

J. Hockenmaier. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh., 2003.

J. Hockenmaier and M. Steedman. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007.

F. Hoyt and J. Baldridge. A Logical Basis for the D Combinator and Normal Form in CCG. *Proceedings of ACL '08*, 2008.

J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.

M. Pentus. Product-Free Lambek Calculus and Context-Free Grammars. *The Journal of Symbolic Logic*, 62(2):648–660, 1997.

Y. Savateev. Product-free Lambek Calculus is NP-complete. *CUNY CS Tech Report*, 2008.

M. Steedman. *The Syntactic Process*. Bradford, 2000.

H.J. Tiede. *Deductive Systems and Grammars: Proofs as Grammatical Structures*. PhD thesis, Indiana University, 1999.

W. Zielonka. Axiomatizability of Ajdukiewicz-Lambek calculus by means of cancellation schemes. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 27, 1981.

A Method to Improve the Efficiency of Deep Parsers with Incremental Chart Pruning

Pierre Lison
Language Technology Lab,
DFKI GmbH,
Saarbrücken, Germany

Abstract

The use of deep parsers in spoken dialogue systems is usually subject to strong performance requirements. Real-time dialogue applications must be capable of responding quickly to any given utterance, even in the presence of noisy, ambiguous or distorted input. The parser must therefore ensure that the number of analyses remains bounded at every processing step.

The paper presents a practical approach to address this issue in the context of deep parsers designed for spoken dialogue. The approach is based on a word lattice parser for Combinatory Categorial Grammar combined with a discriminative model for parse selection. Each word lattice is parsed incrementally, word by word, and a discriminative model is applied at each incremental step to prune the set of resulting partial analyses. The model incorporates a wide range of linguistic and contextual features and can be trained with a simple perceptron. The approach is fully implemented as part of a spoken dialogue system for human-robot interaction. Evaluation results on a Wizard-of-Oz test suite demonstrate significant improvements in parsing time.

1 Introduction

Developing robust and efficient parsers for spoken dialogue is a difficult and demanding enterprise. This is due to several interconnected reasons.

The first reason is the pervasiveness of *speech recognition errors* in natural (i.e. noisy) environments, especially for open, non-trivial discourse domains. Automatic speech recognition (ASR) is indeed a highly error-prone task, and parsers designed to process spoken input must therefore find ways to accomodate the various ASR errors that may (and will) arise.

Next to speech recognition, the second issue we need to address is the *relaxed grammaticality* of spoken language. Dialogue utterances are often incomplete or ungrammatical, and may contain numerous disfluencies like fillers (err, uh, mm), repetitions, self-corrections, etc.

Finally, the vast majority of spoken dialogue systems are designed to operate in *real-time*. This has two important consequences. First, the parser should not wait for the utterance to be complete to start processing it – instead, the set of possible semantic interpretations should be gradually built and extended as the utterance unfolds. Second, each incremental parsing step should operate under strict time constraints. The main obstacle here is the high level of ambiguity arising in natural language, which can lead to a combinatorial explosion in the number of possible readings.

The remaining of this paper is devoted to addressing this last issue, building on an integrated approach to situated spoken dialogue processing previously outlined in (Lison, 2008; Lison and Kruijff, 2009). The approach we present here is similar to (Collins and Roark, 2004), with some notable differences concerning the parser (our parser being specifically tailored for robust spoken dialogue processing), and the features included in the discriminative model.

An overview of the paper is as follows. We first describe in Section 2 the cognitive architecture in which our system has been integrated. We then discuss the approach in detail in Section 3. Finally, we present in Section 4 the quantitative evaluations on a WOZ test suite, and conclude.

2 Architecture

The approach we present in this paper is fully implemented and integrated into a cognitive architecture for autonomous robots (Hawes et al.,

2007). It is capable of building up visuo-spatial models of a dynamic local scene, and continuously plan and execute manipulation actions on objects within that scene. The robot can discuss objects and their material- and spatial properties for the purpose of visual learning and manipulation tasks. Figure 1 illustrates the architecture schema for the communication subsystem.

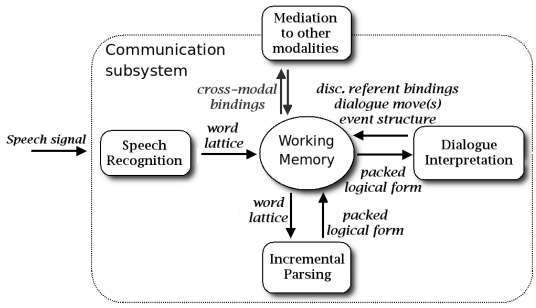


Figure 1: Architecture schema of the communication subsystem (only for comprehension).

Starting with ASR, we process the audio signal to establish a *word lattice* containing statistically ranked hypotheses about word sequences. Subsequently, parsing constructs grammatical analyses for the given (partial) word lattice. A grammatical analysis constructs both a syntactic analysis of the utterance, and a representation of its meaning. The analysis is based on an incremental chart parser[1] for Combinatory Categorial Grammar (Steedman and Baldridge, 2009). These meaning representations are ontologically richly sorted, relational structures, formulated in a (propositional) description logic – more precisely in the HLDS formalism (Baldridge and Kruijff, 2002). The incremental build of derivational structures is realised within the parser via type-raising and composition rules.

Once all the possible (partial) parses for a given (partial) utterance are computed, they are filtered in order to retain only the most likely interpretation(s). This ensures that the number of parses at each incremental step remains bounded and avoid a combinatorial explosion of the search space. The task of selecting the most likely parse(s) among a set of possible ones is called *parse selection*. We describe it in detail in the next section.

At the level of dialogue interpretation, the logical forms are resolved against a dialogue model to establish co-reference and dialogue moves.

Finally, linguistic interpretations must be associated with extra-linguistic knowledge about the environment – dialogue comprehension hence needs to connect with other subarchitectures like vision, spatial reasoning or planning.

[1]Built using the OpenCCG API: http://openccg.sf.net

3 Approach

3.1 Parse selection

As we just explained, the parse selection module is responsible for selecting at each incremental step a subset of "good" parses. Once the selection is made, the best analyses are kept in the (CKY-like) parse chart, while the others are discarded and pruned from the chart.

To achieve this selection, we need a mechanism to discriminate among the possible parses. This is done via a (discriminative) statistical model covering a large number of features.

Formally, the task is defined as a function $F : \mathcal{X} \rightarrow \mathcal{Y}$ where the domain $\mathcal{X}$ is the set of possible inputs (in our case, $\mathcal{X}$ is the set of possible *word lattices*), and $\mathcal{Y}$ the set of parses. We assume:

1. A function $\mathbf{GEN}(x)$ which enumerates all possible parses for an input x. In our case, the function represents the admissibles parses according to the CCG grammar.

2. A d-dimensional feature vector $\mathbf{f}(x, y) \in \Re^d$, representing specific features of the pair (x, y). It can include various acoustic, syntactic, semantic or contextual features.

3. A parameter vector $\mathbf{w} \in \Re^d$.

The function F, mapping a word lattice to its most likely parse, is then defined as:

$$F(x) = \underset{y \in \mathbf{GEN}(x)}{\operatorname{argmax}} \ \mathbf{w}^T \cdot \mathbf{f}(x, y) \qquad (1)$$

Given the parameters $\mathbf{w}$, the optimal parse of a given word lattice x can be therefore easily determined by enumerating all the parses generated by the grammar, extracting their features, computing the inner product $\mathbf{w}^T \cdot \mathbf{f}(x, y)$, and selecting the parse with the highest score.

3.2 Learning

3.2.1 Training data

To estimate the parameters $\mathbf{w}$, we need a set of training examples. Since no corpus of situated dialogue adapted to our task domain is available to this day – let alone semantically annotated – we

followed the approach advocated in (Weilhammer et al., 2006) and *generated* a corpus from a hand-written task grammar.

We first designed a small grammar covering our task domain, each rule being associated to a HLDS representation and a weight. Once specified, the grammar is then randomly traversed a large number of times, resulting in a large set of utterances along with their semantic representations.

3.2.2 Perceptron learning

The algorithm we use to estimate the parameters **w** using the training data is a **perceptron**. The algorithm is fully online - it visits each example in turn, in an incremental fashion, and updates **w** if necessary. Albeit simple, the algorithm has proven to be very efficient and accurate for the task of parse selection (Collins and Roark, 2004; Zettlemoyer and Collins, 2007).

The pseudo-code for the online learning algorithm is detailed in [**Algorithm 1**].

3.3 Features

As we have seen, the parse selection operates by enumerating the possible parses and selecting the one with the highest score according to the linear model parametrised by $\mathbf{w}$.

The accuracy of our method crucially relies on the selection of "good" features $\mathbf{f}(x, y)$ for our model - that is, features which help *discriminating* the parses. In our model, the features are of four types: semantic features, syntactic features, contextual features, and speech recognition features.

3.3.1 Semantic features

What are the substructures of a logical form which may be relevant to discriminate the parses? We define features on the following information sources: the nominals, the ontological sorts of the nominals, and the dependency relations (following (Clark and Curran, 2003)).

These features therefore help us handle various forms of lexical and syntactic ambiguities.

3.3.2 Syntactic features

Syntactic features are features associated to the *derivational history* of a specific parse. Alongside the usual CCG rules (application, composition and type raising), our parser also uses a set of non-standard (type-changing) rules designed to handle disfluencies, speech recognition errors, and combinations of discourse units by selectively relaxing the grammatical constraints (see (Lison and

Algorithm 1 Online perceptron learning

Require: - Set of n training examples $\{(x_i, z_i) : i = 1...n\}$
- For each incremental step j with $0 \leq j \leq |x_i|$, we define the partially parsed word lattice x_i^j and its gold standard semantics z_i^j
- T: number of iterations over the training set
- $\texttt{GEN}(x)$: function enumerating possible parses for an input x, according to the CCG grammar.
- $\texttt{GEN}(x, z)$: function enumerating possible parses for an input x and which have semantics z, according to the CCG grammar.
- $L(y)$ maps a parse tree y to its logical form.
- Initial parameter vector $\mathbf{w_0}$

% Initialise
$\mathbf{w} \leftarrow \mathbf{w_0}$
% Loop T times on the training examples
for $t = 1 \ldots T$ **do**
  for $i = 1 \ldots n$ **do**
    % Loop on the incremental parsing steps
    for $j = 0...|x_i|$ and if x_i not already updated **do**
      % Compute best parse according to model
      Let $y' = \text{argmax}_{y \in \mathbf{GEN}(x_i^j)} \mathbf{w}^T \cdot \mathbf{f}(x_i^j, y)$
      % If the decoded parse $\neq$ expected parse, update the parameters of the model
      if $L(y') \neq z_i^j$ **then**
        % Search the best parse for the partial word lattice x_i^j with semantics z_i^j
        Let $y^* = \text{argmax}_{y \in \mathbf{GEN}(x_i^j, z_i^j)} \mathbf{w}^T \cdot \mathbf{f}(x_i^j, y)$
        % Update parameter vector $\mathbf{w}$
        Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i^j, y^*) - \mathbf{f}(x_i^j, y')$
      end if
    end for
  end for
end for
return parameter vector **w**

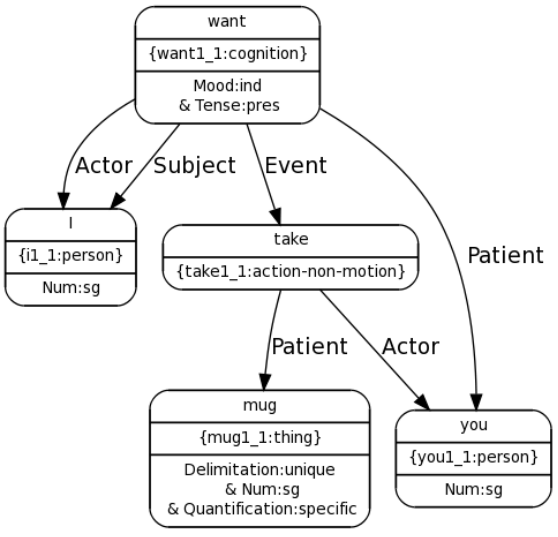


Figure 2: graphical representation of the HLDS logical form for "*I want you to take the mug*".

Kruijff, 2009) for details). In order to "*penalise*" to a correct extent the application of these non-standard rules, we include in the feature vector $\mathbf{f}(x, y)$ new features counting the number of times these rules are applied in the parse. In the derivation shown in the Figure 3, the rule *corr* (correction of a speech recognition error) is for instance applied once.

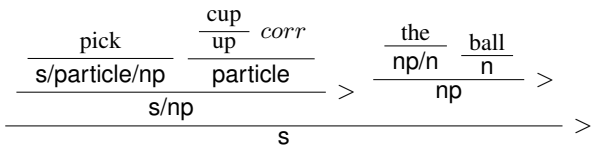


Figure 3: CCG derivation of *"pick cup the ball"*.

In the usual case, the perceptron will learn to assign negative weights to the syntactic features during the training process. In other words, these features can be seen as a *penalty* given to the parses using these non-standard rules, thereby giving a preference to the "normal" parses over them. This ensures that the grammar relaxation is only applied "as a last resort" when the usual grammatical analysis fails to provide a parse.

3.3.3 Contextual features

One striking characteristic of spoken dialogue is the importance of *context*. Understanding the visual and discourse contexts is crucial to resolve potential ambiguities and compute the most likely interpretation(s) of a given utterance.

The feature vector $\mathbf{f}(x, y)$ therefore includes various contextual features. Our dialogue system notably maintains in its working memory a list of contextually activated words (Lison and Kruijff, 2008). This list is continuously updated as the dialogue and the environment evolves. For each context-dependent word, we include one feature counting its occurrence in the utterance.

3.3.4 Speech recognition features

Finally, the feature vector $\mathbf{f}(x, y)$ also includes features related to the *speech recognition*. The ASR module outputs a set of (partial) recognition hypotheses, packed in a word lattice (Figure 4).

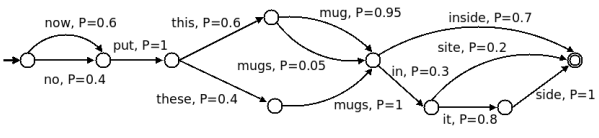


Figure 4: Example of word lattice

We want to favour the hypotheses with high confidence scores, which are, according to the statistical models incorporated in the ASR, more likely to reflect what was uttered. To this end, we introduce in the feature vector several acoustic features measuring the likelihood of each recognition hypothesis.

3.4 Incremental chart pruning

In the previous subsections, we explained how the parse selection was performed, and on basis of which features. We now briefly describe how it can be used for incremental chart pruning.

The main idea is to specify a *beam width* parameter in the parser. This beam width defines the maximal number of analyses which can be kept in the chart at each incremental step. If the number of possible readings exceeds the beam width, the analyses with a lower parse selection score are removed from the chart. Practically, this is realised by removing the top signs associated in the chart with the set of analyses to prune, as well as all the intermediate signs which are included in these top signs *and* are not used in any of the analyses retained by the parse selection module.

The combination of incremental parsing and incremental chart pruning provides two decisive advantages over classical, non-incremental parsers: first, we can start processing the spoken inputs as soon as a partial analysis can be outputted by the ASR. Second, the pruning mechanism ensures that each parsing step remains time-bounded. It is therefore ideally suited for spoken dialogue systems used in human-robot interaction.

4 Experimental evaluation

We performed a quantitative evaluation of our approach, using its implementation in a fully integrated system (cf. Section 2). To set up the experiments, we gathered a Wizard-of-Oz corpus of human-robot spoken dialogue for our task-domain, segmented and annotated manually with their expected semantic interpretation. The data set contains 195 individual utterances [2] along with their complete logical forms.

The results are shown in the Table 1. We tested our approach for five different values of the beam width parameter. The results are compared against a baseline, which is the performance of our parser

[2] More precisely, word lattices provided by the ASR, containing up to 10 alternative recognition hypotheses.

	Beam width	Average parsing time (in s.)	Exact-match			Partial-match		
			Precision	Recall	F_1-value	Precision	Recall	F_1-value
(Baseline)	*(none)*	*10.1*	*40.4*	*100.0*	*57.5*	*81.4*	*100.0*	*89.8*
	120	5.78	40.9	96.9	57.5	81.9	98.0	89.2
	60	**4.82**	41.1	92.5	56.9	81.7	94.1	87.4
	40	4.66	39.9	88.1	54.9	79.6	91.9	85.3
	30	4.21	41.0	83.0	54.9	80.2	88.6	84.2
	20	4.30	40.1	80.3	53.5	78.9	86.5	82.5

Table 1: Evaluation results (in seconds for the parsing time, in % for the exact- and partial-match).

without chart pruning. For each configuration, we give the average parsing time, as well as the exact-match and partial-match results (in order to verify that the performance increase is not cancelled by a drop in accuracy). We observe that the choice of the beam width parameter is crucial. Above 30, the chart pruning mechanism works very efficiently – we observe a notable decrease in the parsing time without significantly affecting the accuracy performance. Below 30, the beam width is too small to retain all the necessary information in the chart, and the recall quickly drops.

5 Conclusions

In this paper, we presented an original method to improve the efficiency of deep parsers (in particular, parsers for categorial grammars) with an incremental chart pruning mechanism used to limit at every processing step the number of analyses retained in the parse chart.

The incremental chart pruning mechanism is based on a discriminative model exploring a set of relevant semantic, syntactic, contextual and acoustic features extracted for each parse. At each incremental step, the discriminative model yields a score for each resulting parse. The parser then only retains in its chart the set of parses associated with a high score, the others being pruned.

As forthcoming work, we shall examine the extension of our approach in new directions, such as the introduction of more refined contextual features or the use of more sophisticated learning algorithms such as Support Vector Machines.

6 Acknowledgements

This work was supported by the EU FP7 ICT Integrated Project "CogX" (FP7-ICT- 215181).

References

J. Baldridge and G.-J. M. Kruijff. 2002. Coupling CCG and hybrid logic dependency semantics. In *ACL'02: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 319–326, Philadelphia, PA. Association for Computational Linguistics.

S. Clark and J. R. Curran. 2003. Log-linear models for wide-coverage ccg parsing. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 97–104, Morristown, NJ, USA. Association for Computational Linguistics.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL '04: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, page 111, Morristown, NJ, USA. Association for Computational Linguistics.

N. A. Hawes, A. Sloman, J. Wyatt, M. Zillich, H. Jacobsson, G.-J. M. Kruijff, M. Brenner, G. Berginc, and D. Skocaj. 2007. Towards an integrated robot with multiple cognitive functions. In *Proc. AAAI'07*, pages 1548–1553. AAAI Press.

P. Lison and G.-J. M. Kruijff. 2008. Salience-driven contextual priming of speech recognition for human-robot interaction. In *Proceedings of the 18th European Conference on Artificial Intelligence*, Patras (Greece).

P. Lison and G.-J. M. Kruijff. 2009. An integrated approach to robust processing of situated spoken dialogue. In *Proceedings of the International Workshop on Semantic Representation of Spoken Language (SRSL'09)*, Athens, Greece. (to appear).

P. Lison. 2008. Robust processing of situated spoken dialogue. Master's thesis, Universität des Saarlandes, Saarbrücken. http://www.dfki.de/∼plison/pubs/thesis/main.thesis.plison2008.pdf.

M. Steedman and J. Baldridge. 2009. Combinatory categorial grammar. In Robert Borsley and Kersti Börjars, editors, *Nontransformational Syntax: A Guide to Current Models*. Blackwell, Oxford.

K. Weilhammer, M. N. Stuttle, and S. Young. 2006. Bootstrapping language models for dialogue systems. In *Proceedings of INTERSPEECH 2006*, Pittsburgh, PA.

L. S. Zettlemoyer and M. Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687.

Memoising Proof Net Theorem-Proving

Glyn Morrill

5th March 2009

We observe that in proof net theorem proving, only a subset of the axiom links of a partial proof structure are *essential* to the extendibility of the partial proof structure to a proof net. Therefore, when the essential axiom links of a partial proof structure are a subset of those of another with the same span, the latter is subsumed and can be compressed out of a chart. We illustrate these ideas with a CKY proof net continuous partial proof structure memoising algorithm for chart Lambek theorem proving.

1 Background

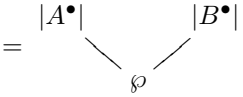
The following theory of proof nets for the calculus of Lambek (1958)[4] is standard; see principally Roorda (1991)[7].

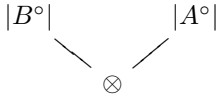
Given a set $\mathcal{A}$ of *atomic types*, we define the set $\mathcal{F}$ of *types* by:

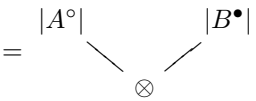
(1) $\mathcal{F} ::= \mathcal{A} \mid \mathcal{F}\bullet\mathcal{F} \mid \mathcal{F}\backslash\mathcal{F} \mid \mathcal{F}/\mathcal{F}$

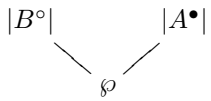
A *polar type* A^p comprises a type A together with a *polarity* $p = \bullet$ ("input") or $\circ$ ("output"). The *polar type tree* $|A^p|$ of a polar type A^p is the ordered tree defined by:

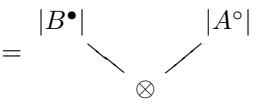
(2) $|P^p| = P^p$ if P is atomic

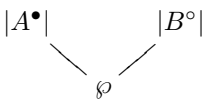
$|A\bullet B^\bullet| =$ 

$|A\bullet B^\circ| =$ 

$|A\backslash B^\bullet| =$ 

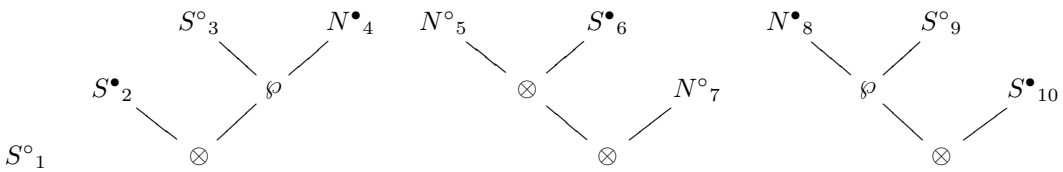
$|A\backslash B^\circ| =$ 

$|B/A^\bullet| =$ 

$|B/A^\circ| =$ 

A *sequent* $A_0, \ldots, A_n \Rightarrow A$ comprises a finite non-empty sequence $A_0, \ldots, A_n$ of *antecedent* types and a *succedent* type A. The *frame* of a sequent $A_0, \ldots, A_n \Rightarrow A$ is the sequence $\langle |A^\circ|, |A_0{}^\bullet|, \ldots, |A_n{}^\bullet| \rangle$. For example, the frame of the sequent (3) is (4), where we have numbered the leaves.

(3) $(S/(N\backslash S), (N\backslash S)/N, (S/N)\backslash S \Rightarrow S$

(4) 

We define the *complement* $\overline{X}$ of a polar type X by $\overline{A^\bullet} = A^\circ$ and $\overline{A^\circ} = A^\bullet$. Two polar types are *complementary* if and only if they are the complements of each other. An *axiom link* on a proof frame is a pair of complementary leaves. An *axiom linking* for a proof

frame is a set of axiom links with at most one axiom link per leaf and which is *planar*, i.e. there are no two axiom links (i,k) and (j,l) such that $i < j < k < l$. That an axiom linking is planar means that it can be drawn in the half-plane without crossing lines. A *partial proof structure* (PPS) is a frame together with an axiom linking. A *proof structure* is a frame together with an axiom linking that links every leaf. A *switching* of a PPS is a graph resulting from removing one of the immediate descendent edges of each $\wp$-node. A *proof net* is a proof structure in which i) every switching is a connected and acyclic graph (Danos-Regnier acyclicity and connectedness; see Danos and Regnier, 1989[2]), and ii) no axiom link connects the leftmost and rightmost descendent leaves of an output division (we call this Retoré no subtending). No subtending prohibits empty antecedents.

(5) **Theorem**. A sequent is a theorem of the Lambek calculus if and only there is an axiom linking which forms a proof net on its frame.

Fadda and Morrill (2005)[3] show that in view of the intuitionistic nature of Lambek sequents (that there is exactly one root of output polarity), every proof structure which satifies Danos-Regnier (DR) acyclicity also satifies DR connectedness. Therefore we need only check for DR acyclicity (and no subtending). We call a partial proof structure *correct* if and only if it satifies DR acyclicity and no subtending. Correctness is decidable in polynomial time, e.g. by the red-blue graphs of Retoré and Lecomte (1995)[5].

(6) **Corollary**. A sequent is a theorem of the Lambek calculus if and only there is an axiom linking which forms a correct proof structure on its frame.

Therefore we can carry out Lambek theorem-proving by building up proof nets incrementally, checking for correctness (DR acyclicity and Retoré no subtending) at each step. We have a Lambek theorem iff we succeed in linking all the leaves while satisfying these criteria.

2 Chart theorem-proving without compression

Planar linking is a Dyck language. Its grammar can be given as follows:

(7) $S \rightarrow A\ \overline{A} \mid A\ S\ \overline{A} \mid S\ S$

Given a frame with $n = \dot{2}$ leaves $L_1, \ldots, L_n$,[1] there is the CKY chart theorem-proving algorithm without compression defined in Figure 1 (cf. Morrill 1996)[6]. We memorize continuous planar linkings, where a continuous planar linking is a set of axiom links connecting contiguous leaves. The *span* of a continuous planar linking is identified with the pair comprising the positions of the start of its first axiom link and the end of its last axiom link. Here we notate a continuous planar linking as a well-parenthesized strings of brackets encoding textually the planar axiom linking over the span.

Then for example the chart computed for (4) is as follows:

(8)

9									$\emptyset$
8								$\emptyset$	
7							$\{[]\}$		$\emptyset$
6						$\emptyset$		$\{[[]]\}$	
5					$\emptyset$		$\emptyset$		$\emptyset$
4				$\{[]\}$		$\emptyset$		$\{[][]\}$	
3			$\emptyset$		$\{[[]]\}$		$\{[[]][]\}$		$\{[][[]]\}$
2		$\emptyset$		$\emptyset$		$\emptyset$		$\{[[[]][]]\}$	
1	$\{[]\}$		$\emptyset$		$\{[][[]]\}$		$\{[][[]][]\}$		$\{[][[][]], [[[]][]]\}$
	2	3	4	5	6	7	8	9	10

[1] Observe that the leaves cannot be pairwise matched if $n \neq \dot{2}$.

```
for i := 1 to n − 1 do
    if L_i = overline(L_{i+1}) and [] with span (i, i + 1) is a correct linking
        then C(i, i + 1) := {[]}
        else C(i, i + 1) := ∅;
for l := 4 to n in steps of 2 do
    for i := 1 to n + 1 − l do
        begin
        k := i + l − 1;
        C(i, k) := ∅;
        if L_i = overline(L_k)
            then for each K ∈ C(i + 1, k − 1) do
                if [K] with span (i, k) is a correct linking
                    then C(i, k) := C(i, k) ∪ {[K]};
        for j := i + 2 to k − 1 in steps of 2 do
            for each K_1 ∈ C(i, j − 1) and K_2 ∈ C(j, k) do
                if K_1K_2 with span (i, k) is a correct linking
                    then C(i, k) := C(i, k) ∪ {K_1K_2}
        end;
if C(0, n) ≠ ∅
    then print "theorem"
    else print "non-theorem".
```

Figure 1: Lambek chart theorem-proving without compression

3 Subsumption of linkings

In a graph we define a *terminal* node as one with only one incident edge. Let G be a subgraph of a correct partial proof structure. We call a node in G *open* iff in the partial proof structure it is a leaf not connected by an axiom link. We call a node in G *closed* iff it is not open, i.e. iff in the partial proof structure it is either a leaf connected by an axiom link or a mother. If a terminal node N in G is open, it might be extended with an axiom link in extensions of the partial proof structure and might come to form part of a (DR) cycle. But if N is closed, it will never have another incident edge in any extension of the partial proof structure and its incident edge is irrelevant to whether or not there will ever be cycles. We call *pruning* the removal from G of the incident edge of a closed terminal node. Likewise (cf. Béchet 2003)[1], consider a root $\wp$-node in a graph. Every switching breaks the path between its daughters through their mother. Therefore the edges to its daughters are irrelevant to the detection of cycles. We call *harrowing* the removal from G of the incident edges of a root $\wp$-node.

We call the closure of the graph of a correct PPS under pruning and harrowing its *reduct*. Where M is the axiom linking of a correct PPS, we define its *essential* axiom links $M{\downarrow}$ to be the subset of axiom links of M which still remain in the reduct of the PPS. This is the key here to subsumption in memoisation of distinct PPSs with the same span. Two linkings of the same leaves can be different, but all we need to remember to check DR acyclicity of extensions are the subsets of *essential* links, and if the essential axiom links of one PPS are a subset of those of another, the latter is subsumed and can compressed out in the chart.[2]

Thus there is the CKY chart theorem-proving algorithm with compaction in Figure 2, which stores the essential axiom links of continuous linkings as ordered pairs of numbers. Then the chart computed for (4) is as shown in Figure 3.

[2]Note the direction of subsumption: any DR-cycle free extension of the latter would also be a DR-cycle free extension of the former, therefore it is only the former which needs to be remembered.

```
for i := 1 to n − 1 do
    if L_i = \overline{L_{i+1}} and {(i, i + 1)} is a correct linking
        then C(i, i + 1) := {[{(i, i + 1)}]}
        else C(i, i + 1) := ∅;
for l := 4 to n in steps of 2 do
    for i := 1 to n + 1 − l do
        begin
        k := i + l − 1;
        C(i, k) := ∅;
        if L_i = \overline{L_k}
            then for each K ∈ C(i + 1, k − 1) do
                if {(i, k)} ∪ K is a correct linking
                    then C(i, k) := C(i, k) ∪ {({(i, k)} ∪ K)↓};
        for j := i + 2 to k − 1 in steps of 2 do
            for each K_1 ∈ C(i, j − 1) and K_2 ∈ C(j, k) do
                if K_1 ∪ K_2 is a correct linking
                    then C(i, k) := C(i, k) ∪ {(K_1 ∪ K_2)↓}
        end;
if C(0, n) ≠ ∅
    then print "theorem"
    else print "non-theorem".
```

Figure 2: Lambek chart theorem-proving with compression

References

[1] Denis Béchet. Incremental Parsing of Lambek Calculus Using Proof-Net Interfaces. In *Proceedings of IWPT 2003*, 2003.

[2] V. Danos and L. Regnier. The Structure of Multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[3] Mario Fadda and Glyn Morrill. The Lambek Calculus with Brackets. In C. Casadio, P.J. Scott, and R.A.G. Seely, editors, *Language and Grammar: Studies in Mathematical Linguistics and Natural Language*, number 168 in CSLI Lecture Notes, pages 113–128. CSLI Publications, Stanford, 2005.

[4] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958. Reprinted in Buszkowski, Wojciech, Wojciech Marciszewski, and Johan van Benthem, editors, 1988, *Categorial Grammar*, Linguistic & Literary Studies in Eastern Europe volume 25, John Benjamins, Amsterdam, 153–172.

[5] Alain Lecomte and Christian Rétoré. An Alternative Categorial Grammar. In G. Morrill and R.T.E. Oehrle, editors, *Proceedings of Formal Grammar 1995*, Barcelona, 1995. ESSLLI95.

[6] Glyn Morrill. Memoisation of Categorial Proof Nets: Parallelism in Categorial Processing. In V. Michele Abrusci and Claudia Casadio, editors, *Proofs and Linguistic Categories, Proceedings 1996 Roma Workshop*, pages 157–169, 1996.

[7] Dirk Roorda. *Resource Logics: Proof-Theoretical Investigations*. PhD thesis, Universiteit van Amsterdam, 1991.

	2	3	4	5	6	7	8	9	10
9									
8									
7							$\{\{(7,8)\}\}$	$\emptyset$	$\emptyset$
6						$\emptyset$		$\{\{(6,9),(7,8)\}\}$	
5					$\emptyset$		$\emptyset$		$\emptyset$
4				$\{\{(4,5)\}\}$		$\emptyset$		$\{\{(4,5),(6,9),(7,8)\}\}$	
3			$\emptyset$		$\{\{(3,6),(4,5)\}\}$		$\{\{(3,6),(4,5),(7,8)\}\}$		$\{\{(3,10),(4,5),(6,9),(7,8)\}\}$
2		$\emptyset$		$\emptyset$		$\emptyset$		$\{\{(2,9),(3,6),(4,5),(7,8)\}\}$	
1	$\{\emptyset\}$		$\emptyset$		$\{\emptyset\}$		$\{\emptyset\}$		$\{\emptyset\}$

Figure 3: Chart for (4) with only essential axiom links

Product-free Lambek Calculus is NP-complete

Yury Savateev*
Department of Mathematical Logic and Theory of Algorithms
Moscow State University

Abstract

In this paper we prove that the derivability problems for product-free Lambek calculus and product-free Lambek calculus allowing empty premises are NP-complete. Also we introduce a new derivability characterization for these calculi.

Introduction

Lambek calculus L was first introduced in [3]. Lambek calculus uses syntactic types that are built from primitive types using three binary connectives: multiplication, left division, and right division. Natural fragments of Lambek calculus are the product-free Lambek calculus L($\backslash, /$), which does not use multiplication, and the unidirectional Lambek calculi, which have only one connective left: a division (left or right).

For the non-associative variant of Lambek calculus the derivability can be checked in polynomial time as shown in [2] (for the product-free fragment of the non-associative Lambek calculus this was proved already in [1]).

In [5] NP-completeness was proved for the derivability problem for full associative Lambek calculus. In [6] there was presented a polynomial algorithm for its unidirectional fragments.

We show that the classical satisfiability problem *SAT* is polynomial time reducible to the L($\backslash, /$)-derivability problem and thus L($\backslash, /$) is NP-complete.

After first presenting this result author was pointed to [4] where a very similar (but more complex) technique to explore the derivability for product-free Lambek calculus was presented, though without proving any complexity results.

1 Product-free Lambek Calculus

Product-free Lambek calculus L($\backslash, /$) can be constructed as follows. Let $\mathbf{P} = \{p_0, p_1, \ldots\}$ be a countable set of what we call *primitive types*. Let Tp be the set of *types* constructed from primitive types with two binary connectives $/$, $\backslash$. We will denote primitive types by small letters (p, q, $r, \ldots$) and types by capital letters (A, B, $C, \ldots$). By capital greek letters (Π, Γ, $\Delta, \ldots$) we will denote finite

*This research was supported in part by the Russian Foundation for Basic Research grant 08-01-00399.

(possibly empty) sequences of types. Expressions like $\Pi \to A$, where Π is not empty, are called *sequents*.

Axioms and rules of L($\backslash, /$):

$$A \to A, \qquad \frac{\Phi \to B \quad \Gamma B \Delta \to A}{\Gamma \Phi \Delta \to A}\ (\mathrm{CUT}),$$

$$\frac{\Pi A \to B}{\Pi \to (B/A)}\ (\to /), \qquad \frac{\Phi \to A \quad \Gamma B \Delta \to C}{\Gamma (B/A) \Phi \Delta \to C}\ (/ \to),$$

$$\frac{A \Pi \to B}{\Pi \to (A \backslash B)}\ (\to \backslash), \qquad \frac{\Phi \to A \quad \Gamma B \Delta \to C}{\Gamma \Phi (A \backslash B) \Delta \to C}\ (\backslash \to),$$

(Here Γ and Δ can be empty.)

In this paper we will consider two calculi — L($\backslash, /$) and L*($\backslash, /$), called product-free Lambek calculus allowing empty premises. In L*($\backslash, /$) we allow the antecedent of a sequent to be empty.

2 Reduction from *SAT*

Let $c_1 \wedge \ldots \wedge c_m$ be a Boolean formula in conjunctive normal form with clauses $c_1 \ldots c_m$ and variables $x_1 \ldots x_n$. The reduction maps the formula to a sequent, which is derivable in L($\backslash, /$) (and in L*($\backslash, /$)) if and only if the formula $c_1 \wedge \ldots \wedge c_m$ is satisfiable.

For any Boolean variable x_i let $\neg_0 x_i$ stand for the literal $\neg x_i$ and $\neg_1 x_i$ stand for the literal x_i.

Let $p_i^j, q_i^j, a_i^j, b_i^j; 0 \le i \le n, 0 \le j \le m$ be distinct primitive types from $\mathbf{P}$.

We define the following families of types:

$$\begin{aligned}
G^0 &\rightleftharpoons (p_0^0 \backslash p_n^0)\\
G^j &\rightleftharpoons (q_n^j / ((q_0^j \backslash p_0^j) \backslash G^{j-1})) \backslash p_n^j\\
G &\rightleftharpoons G^m\\
A_i^0 &\rightleftharpoons (a_i^0 \backslash p_i^0)\\
A_i^j &\rightleftharpoons (q_i^j / ((b_i^j \backslash a_i^j) \backslash A_i^{j-1})) \backslash p_i^j\\
A_i &\rightleftharpoons A_i^m\\
E_i^0(t) &\rightleftharpoons p_{i-1}^0\\
E_i^j(t) &\rightleftharpoons \begin{cases} q_i^j / (((q_{i-1}^j / E_i^{j-1}(t)) \backslash p_{i-1}^j) \backslash p_i^{j-1}), & \text{if } \neg_t x_i \text{ appears in } c_j\\ (q_{i-1}^j / (q_i^j / (E_i^{j-1}(t) \backslash p_i^{j-1}))) \backslash p_{i-1}^j, & \text{if } \neg_t x_i \text{ does not appear in } c_j \end{cases}\\
F_i^j(t) &\rightleftharpoons (E_i^j(t) \backslash p_i^j)\\
F_i(t) &\rightleftharpoons F_i^m(t)\\
B_i^0 &\rightleftharpoons a_i^0\\
B_i^j &\rightleftharpoons q_{i-1}^j / (((b_i^j / B_i^{j-1}) \backslash a_i^j) \backslash p_{i-1}^{j-1})\\
B_i &\rightleftharpoons B_i^m \backslash p_{i-1}^m
\end{aligned}$$

Let Π_i denote the following sequences of types:

$$(F_i(0)/(B_i\backslash A_i))\ F_i(0)\ (F_i(0)\backslash F_i(1)).$$

Theorem 2.1. *The following statements are equivalent:*

1. $c_1 \wedge \ldots \wedge c_m$ *is satisfiable.*
2. $\mathrm{L}(\backslash, /) \vdash \Pi_1 \ldots \Pi_n \to G$
3. $\mathrm{L}^*(\backslash, /) \vdash \Pi_1 \ldots \Pi_n \to G$.

This sketch of the proof of this theorem is presented in section 5. The lemmas are proved using the derivability characterization presented in the next section.

3 Derivability Characterization

Let At be the set of *atoms* or *primitive types with superscripts*, $\{p^{(i)} | p \in \mathbf{P}, i \in \mathbb{Z}\}$. Let FS be the free monoid (the set of all finite strings) generated by elements of At. We will denote elements of FS by $\mathbb{A}$, $\mathbb{B}$, $\mathbb{C}$ and so on, by ε we will denote the empty string.

Consider two mappings:

$$t : \mathrm{FS} \to \mathbf{P}, t(\mathbb{A}p^{(i)}) = p; \quad d : \mathrm{FS} \to \mathbb{Z}, d(\mathbb{A}p^{(i)}) = i.$$

Let $\mathbb{A} \sqsubseteq \mathbb{B} (\mathbb{A} \sqsubset \mathbb{B})$ denote that $\mathbb{A}$ is a (strict) prefix of $\mathbb{B}$.

For $\mathbb{A} \in \mathrm{FS}, \mathbb{A} \neq \varepsilon$ let $\mathcal{P}_\mathbb{A} = \{\mathbb{B} \mid \mathbb{B} \sqsubseteq \mathbb{A}, \mathbb{B} \neq \varepsilon\}$. The relation $\sqsubseteq$ is a total order on $\mathcal{P}_\mathbb{A}$.

Let α be a partial function on $\mathcal{P}_\mathbb{A}$. For each such function we can define the following:

$$\mathbb{B} <_\alpha \mathbb{C} \Leftrightarrow \exists n \geq 1, \alpha^n(\mathbb{B}) = \mathbb{C},$$
$$\mu^-_\alpha(\mathbb{B}) = \min_\sqsubset(\mathbb{B}, \alpha(\mathbb{B})),$$
$$\mu^+_\alpha(\mathbb{B}) = \max_\sqsubset(\mathbb{B}, \alpha(\mathbb{B})).$$

A function $f\colon X \to X$ is an antiendomorphism if $\forall a, b \in X, f(ab) = f(b)f(a)$. In a free monoid it can be defined by its actions on the generators.

Consider two antiendomorphisms $(\cdot)^\leftarrow$ and $(\cdot)^\rightarrow$ on FS defined by

$$(p^{(0)})^\leftarrow = p^{(-1)}, \quad (p^{(0)})^\rightarrow = p^{(1)},$$
$$(p^{(i)})^\leftarrow = (p^{(i)})^\rightarrow = p^{(-i-sgn(i))}, \text{for } i \neq 0.$$

Consider $[\![\cdot]\!]\colon \mathrm{Tp} \to \mathrm{FS}$, a mapping from Lambek types to elements of the free monoid defined by

$$[\![p]\!] = p^{(0)}, \quad [\![(A/B)]\!] = [\![B]\!]^\rightarrow [\![A]\!], \quad [\![(A\backslash B)]\!] = [\![B]\!][\![A]\!]^\leftarrow.$$

Let us define φ — a partial function on $\mathcal{P}_{[\![A]\!]}$ that reflects the structure of the Lambek type A:

$$\varphi(\mathbb{A}) = \begin{cases} \inf_\sqsubset\{\mathbb{B} \mid \mathbb{A} \sqsubset \mathbb{B}, |d(\mathbb{B})| = |d(\mathbb{A})| - 1\}, & \text{if } d(\mathbb{A}) > 0; \\ \sup_\sqsubset\{\mathbb{B} \mid \mathbb{B} \sqsubset \mathbb{A}, |d(\mathbb{B})| = |d(\mathbb{A})| - 1\}, & \text{if } d(\mathbb{A}) < 0. \end{cases}$$

Suppose we have a Lambek sequent $A_1 \dots A_n \to B$. Let

$$\mathbb{W} = [\![(\dots(B/A_n)/\dots)/A_1]\!] = [\![A_1]\!]^{\rightarrow} \dots [\![A_n]\!]^{\rightarrow}[\![B]\!].$$

Let $\mathcal{N}_{\mathbb{W}} = \{\mathbb{A} \in \mathcal{P}_{\mathbb{W}} \mid d(\mathbb{A}) = 2i+1, i \in \mathbb{Z}\}$.

Let π be a function from $\mathcal{N}_{\mathbb{W}}$ to $\mathcal{P}_{\mathbb{W}}$, and ψ be a partial function on $\mathcal{P}_{\mathbb{W}}$ defined by

$$\psi(\mathbb{A}) = \begin{cases} \pi(\mathbb{A}), & \text{if } \mathbb{A} \in \mathcal{N}_{\mathbb{W}}; \\ \varphi(\mathbb{A}), & \text{if } \mathbb{A} \notin \mathcal{N}_{\mathbb{W}} \text{ and } d(\mathbb{A}) \neq 0. \end{cases}$$

To characterize derivability of the sequent $A_1 \dots A_n \to B$ we shall use the following conditions, which we call proof conditions.

1. Function π is a bijection between $\mathcal{N}_{\mathbb{W}}$ and $\mathcal{P}_{\mathbb{W}} \setminus \mathcal{N}_{\mathbb{W}}$.
2. $t(\pi(\mathbb{A})) = t(\mathbb{A})$.
3. $\mu^-_\pi(\mathbb{A}) \sqsubset \mu^-_\pi(\mathbb{B}) \Rightarrow \mu^+_\pi(\mathbb{A}) \sqsubset \mu^-_\pi(\mathbb{B}) \vee \mu^+_\pi(\mathbb{B}) \sqsubset \mu^+_\pi(\mathbb{A})$.
4. $\mathbb{A} \in \mathcal{N}_{\mathbb{W}} \Longrightarrow \mathbb{A} <_\psi \varphi(\mathbb{A})$.
5. $\mathbb{A} \notin \mathcal{N}_{\mathbb{W}} \wedge \mathbb{A} \neq \mathbb{A}_0 \Longrightarrow \exists \mathbb{B}(\mathbb{B} <_\psi \mathbb{A} \wedge \mathbb{B} \not<_\varphi \mathbb{A})$.

Lemma 3.1. *$\mathrm{L}^*(\backslash, /) \vdash A_1 \dots A_n \to B$ if and only if there exists π satisfying proof conditions (1)-(4).*

$\mathrm{L}(\backslash, /) \vdash A_1 \dots A_n \to B$ if and only if there exists π satisfying proof conditions (1)-(5).

4 Graphic Representation

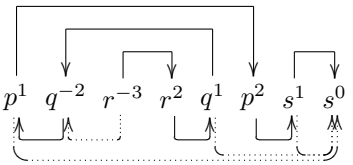
Consider the following Lambek sequence:

$$(p/(r\backslash q))\ (r\backslash q)\ (p\backslash s) \to s$$

The corresponding element of FS:

$$p^1 q^{-2} r^{-3} r^2 q^1 p^2 s^1 s^0$$

Elements of $\mathcal{P}_{\mathbb{W}}$ correspond to occurences of atoms in the string. So we can draw arrows between such occurences to represent functions φ and ψ. We draw arrows for π for members of $\mathcal{N}_{\mathbb{W}}$ in the upper semiplane of the string and arrows for φ in the lower. Dotted arrows denote parts of φ that are not part of ψ. Consider the following diagram:



Such diagrams are called proofnets.

Proofnets provide useful intuition about proof conditions. For example proof condition (3) is equivalent to statement "arrows in the upper semiplane can be drawn without intersections". Proof condition (4) states that for every dotted arrow if we start at its origin and follow solid arrows we will reach its destination.

It is readily seen that this proofnet satisfies proof conditions (1)-(5) and thus $\mathrm{L}(\backslash, /) \vdash (p/(r\backslash q))(r\backslash q)(p\backslash s) \to s$.

5 Sketch of the Proof of the Main Theorem

Lemma 5.1. *$\langle t_1, \ldots, t_n \rangle$ is a satisfying assignment for $c_1 \wedge \ldots \wedge c_m$ if and only if $\mathrm{L}^*(\backslash, /) \vdash F_1(t_1) \ldots F_n(t_n) \to G$ and if and only if*

$$\mathrm{L}(\backslash, /) \vdash F_1(t_1) \ldots F_n(t_n) \to G.$$

Lemma 5.2. *$\mathrm{L}(\backslash, /) \vdash \Pi_i \to F_i(t_i)$, where $t_i \in \{0, 1\}$.*

Lemma 5.3. *If the formula $c_1 \wedge \ldots \wedge c_m$ is satifiable, then $\mathrm{L}(\backslash, /) \vdash \Pi_1 \ldots \Pi_n \to G$.*

Proof. Suppose $\langle t_1, \ldots, t_n \rangle$ is a satisfying assignment for $c_1 \wedge \ldots \wedge c_m$. According to Lemma 5.1 $\mathrm{L}(\backslash, /) \vdash F_1(t_1) \ldots F_n(t_n) \to G$. Now we apply Lemma 5.2 and the cut rule n times. □

Lemma 5.4. *If $\mathrm{L}^*(\backslash, /) \vdash \Pi_1 \ldots \Pi_i F_{i+1}(t_{i+1}) \ldots F_n(t_n) \to G$, then there is $t_i \in \{0, 1\}$ such that $\mathrm{L}^*(\backslash, /) \vdash \Pi_1 \ldots \Pi_{i-1} F_i(t_i) \ldots F_n(t_n) \to G$*

Lemma 5.5. *If $\mathrm{L}^*(\backslash, /) \vdash \Pi_1 \ldots \Pi_n \to G$, then the formula $c_1 \wedge \ldots \wedge c_m$ is satisfiable.*

Proof. Applying n times Lemma 5.4, we get that there exists $\langle t_1, \ldots, t_n \rangle \in \{0, 1\}^n$ such that $\mathrm{L}^*(\backslash, /) \vdash F_1(t_1) \ldots F_n(t_n) \to G$. By Lemma 5.1 this means that $\langle t_1, \ldots, t_n \rangle$ is a satisfying assignment for $c_1 \wedge \ldots \wedge c_m$. □

Since for all sequents $\mathrm{L}(\backslash, /) \vdash \Pi \to A \Rightarrow \mathrm{L}^*(\backslash, /) \vdash \Pi \to A$, Lemma 5.3 and Lemma 5.5 together give us Theorem 2.1.

References

[1] E. Aarts and K. Trautwein, Non-associative Lambek categorial grammar in polynomial time, *Mathematical logic Quarterly* **41** (1995) pp. 476–484.

[2] Ph. de Groote, The non-associative Lambek calculus with product in polynomial time, in: *Automated Reasoning with Analytic Tableaux and Related Methods*, (N. V. Murray, ed.), LLNC vol. **1617**, Springer (1999), pp. 128–139.

[3] J. Lambek, The mathematics of sentence structure, *American Mathematical Monthly* **65** (3) (1958) pp. 154–170.

[4] G. Penn, A Graph-Theoretic Approach to Polynomial-Time Recognition with the Lambek Calculus, *Electronic Notes in Theoretical Computer Science* vol. **53**, Elsevier (2005)

[5] M. Pentus, Lambek calculus is NP-complete, *Theoretical Computer Science* **357**, no. 1–3 (2006) pp. 186–201.

[6] Y. Savateev, Lambek grammars with one division are decidable in polynomial time, in: *Computer Science — Theory and Applications*, (E.A. Hirsch et al. eds.), LLNC vol. **5010**, Springer (2008), pp. 273–282.