Combining logical and distributional methods in type-logical grammars

Richard Moot
LIRMM, Montpellier University, CNRS

ABSTRACT

We propose a low-level way of combining distributional and logical ideas into a single formal system. This will be an instantiation of a more general system, adding weights to proof rules. These weights will not measure some sort of "confidence the proof is valid", but rather act as a way to *prefer* some proofs over others, where preference can mean "easier to process (for humans)" or "more coherent (combining words that make sense together)". The resulting system of weighted theorem proving can be implemented either as a best-first proof search strategy or as a polynomial-time approximation of proof search for NP-complete parsing problems.

Keywords: type-logical grammar, Lambek calculus, theorem proving

1 INTRODUCTION

Type-logical grammars (and formal semantics in general) are agnostic about the meaning of atomic terms, such as those corresponding to nouns and verbs (though not about the meaning corresponding to words with logical content such as "not", "and", "all", "which"). Another way to see this is that in standard formal semantics, entailment only holds under strict identity of predicates. As a consequence, practical use of the output of a system computing such formal semantics depends to a large extent on the available world knowledge (Bos and Markert 2005), possibly stated in the form of additional axioms or meaning postulates, stating that "pub" and "bar" (in one meaning of the word) are synonyms, and that "good" and "bad" are antonyms, i.e. "bad" entails "not good" and inversely.

In contrast to formal semantics in the tradition of Montague, distributional or vector-based semantics take semantic similarity, as measured by word cooccurrences, as their basic notion. Systems using only semantic similarity are agnostic about argument structure and agnostic about the meaning of words with logical content. Given a vector of a sequence of words, it is not a priori clear how to combine these into the meaning of a phrase. In other words, vector space models are not compositional by nature, although many ways of computing vector space semantics for texts exist, and even the simplest models (adding or averaging all vectors for the words in a larger text) can perform well on a number of tasks (see Mitchell and Lapata 2010; Pham 2016, for discussion).

Whereas compositional formal semantics, unless augmented by specific lexical meanings or meaning postulates, concludes that "good" and "bad" are unrelated unary predicates, vector semantics concludes that "good" and "bad" are very similar. Other semantically similar words are "animal" and "veterinarian", and "sweater" and "warm". The absence of argument structure (who does what to whom) from the vectors makes "animal" and "veterinarian" similar: even though many sentences contain both words, these tend to be sentences where the veterinarian treats or examines the animal, or where the owner takes an animal to the veterinarian.

There appears to be some complementarity to these two approaches: formal semantics takes compositionality as its basic principle but has little to say about the meaning of individual predicates; vector semantics takes the similarity of predicates as its basic concept but then has little to say about compositionality and the words with logical content.

In this paper, we will look at a low-level way of combining distributional and logical ideas into a single formal system. This will be an instantiation of a more general system, adding weights to proof rules. These weights will not measure some sort of "confidence the proof is valid", but rather act as a way to *prefer* some proofs over others, where preference can mean "easier to process (for humans)" or "more coherent (combining words that make sense together)". The resulting system of weighted theorem proving can be implemented either as a best-first proof search strategy or as a polynomial-time approximation of proof search for NP-complete parsing problems.

2 TYPE-LOGICAL GRAMMAR AND FORMAL SEMANTICS

Definition 2.1 (Type-logical grammar) *Given a logic $\mathscr{L}$ with formulas $\mathscr{F}$, a type-logical grammar over $\mathscr{L}$ is a tuple $\langle \Sigma, Lex, goal, yield, h \rangle$, where*

1. *Σ is a set of words (the vocabulary of the language),*
2. *the lexicon* Lex*, is a function from $w \in \Sigma$ to a (non-empty) subset of $\mathscr{F}$,*
3. goal*, the set of goal formulas is a (non-empty) subset of $\mathscr{F}$,*
4. yield *is a function from antecedents of $\mathscr{L}$ to sequences of formulas,*
5. *h is a homomorphism from proofs in $\mathscr{L}$ to proofs in multiplicative intuitionistic linear logic representing their "deep structure".*

Informally, a sentence is grammatical whenever the lexicon assigns each word in the sentence a formula, and these formulas produce a derivable statement in the logic. More formally, we say a sentence $w_1, \ldots, w_n$ is *grammatical* if for all i, $w_i \in \Sigma$ (each word is in the vocabulary) and there is an $A_i \in Lex(w_i)$ (we choose, for each word, one of the formulas assigned to it by the lexicon), there is a structure Γ with $yield(\Gamma) = A_1, \ldots, A_n$, and there is a $C \in goal$ such that the statement $\Gamma \vdash C$ is a theorem of the logic $\mathscr{L}$. A sentence is *ungrammatical* otherwise.

Many authors choose the set $\{s\}$ for *goal* (that is, the only valid goal category is s, for *sentence*). However, for more elaborate grammars, we may be interested not only in declarative sentences, but also in yes-no questions, *wh* questions, imperatives, etc., and it seems reasonable to allow such sentences to have a different type of meaning from declarative sentences.

For the Lambek calculus (Lambek 1958), the logic is **L**, the yield function is the identity function (since antecedents Γ of **L** are already sequences of formulas), and h translates the Lambek calculus slashes "$/$" and "$\backslash$" to the multiplicative linear logic implication "$\multimap$" (and the product "$\bullet$" to multiplicative conjunction "$\otimes$").

For multimodal type-logical grammars (Moortgat 1997), sequents are of the form $\Gamma \vdash C$ where the antecedent Γ is a labelled tree with unary and binary branches and with formulas as its leaves. The yield

Figure 1: Proof rules and corresponding lambda term operations

$$\frac{A/B : M^{U\to T} \quad B : N^{U}}{A : (MN)^{T}}\ /E \qquad \frac{B : N^{U} \quad B\backslash A : M^{U\to T}}{A : (MN)^{T}}\ \backslash E$$

$$\begin{array}{c} \ldots [B : x^{U}]_i \\ \vdots \\ \dfrac{A : M^{T}}{A/B : (\lambda x.M)^{U\to T}}\ /I_i \end{array} \qquad \begin{array}{c} [B : x^{U}]_i \ldots \\ \vdots \\ \dfrac{A : M^{T}}{B\backslash A : (\lambda x.M)^{U\to T}}\ \backslash I_i \end{array}$$

function is simply the left-to-right sequence of formulas occurring as its leaves (i.e. we use the standard definition of the yield of a tree).

2.1 *The Lambek calculus*

To makes this more concrete, we'll instantiate the general type-logical grammar framework to Lambek's Syntactic Calculus, **L** (Lambek 1958). Formulas of the Lambek calculus are inductively defined from a set of atomic formulas, including np (noun phrase), n (common noun), s (sentence) and pp (prepositional phrase). A formula in the Lambek calculus is:

- an atomic formula,
- if A and B are formulas, then A/B (pronounced "A over B", it looks for a B formula to its right to produce an A), $B\backslash A$ (pronounced "B under A", it looks for a B formula to its left to produce an A) are formulas.[1]

Figure 1 shows the natural deduction proof rules for the Lambek calculus (and the associated lambda term assignments).

The elimination rule for "/", labeled "$/E$" states that if we have a proof with conclusion A/B which is assigned term M (of type $U \to T$) and a proof with conclusion B which is assigned term N (of type U), then we can combine these two proofs to form a proof of A which is assigned lambda-term $(M\ N)$. The order of the premisses is important: B must occur adjacent to and to the right of A/B. The elimination rule for \ is left-right symmetric, with B occurring to the immediate left of $B\backslash A$.

[1] To keep the discussion simple, we do not present the natural deduction proof rules for the product $A \bullet B$, representing the concatenation of A and B.

The introduction rule, labeled "$/I$", states that if we have a proof of A with lambda-term M of some type T, which we have derived while using a hypothesis B, which is assigned a variable x of type U and which is the rightmost undischarged hypotheses of this proof, then we can discharge this B hypothesis to derive A/B of type $U \rightarrow T$ with term $\lambda x.M$. The discharged hypothesis is co-indexed with the rule, using an index i unique to the proof (for the Lambek calculus without product, this index is strictly speaking superfluous, since the leftmost and rightmost undischarged hypotheses are uniquely determined for each subproof). The introduction rule for $\backslash I$ is again left-right symmetric, requiring B to be the leftmost undischarged hypothesis.

We will write $A_1,\ldots,A_n \vdash C$ for a proof with undischarged hypotheses $A_1,\ldots,A_n$ (in the given order) and conclusion C.

As an example, the following Lambek calculus proof shows that "moons which Galileo discovered" is a noun *n*. To make the proof more readable, the lexical entries have been indicated as the conclusions of a rule *Lex* with the word occurring above it and the corresponding formula assigned by the lexicon below it (we will add the lambda terms later).

$$\dfrac{\dfrac{\text{moons}}{n}\,Lex \qquad \dfrac{\dfrac{\text{which}}{(n\backslash n)/(s/np)}\,Lex \qquad \dfrac{\dfrac{\dfrac{\text{Galileo}}{np}\,Lex \qquad \dfrac{\dfrac{\text{discovered}}{(np\backslash s)/np}\,Lex \qquad [np]_1}{np\backslash s}\,/E}{s}\,\backslash E}{s/np}\,/I_1}{n\backslash n}\,/E}{n}\,\backslash E$$

We can read off the lexical assignments from the undischarged leaves of the proof above. So "discovered" is a transitive verb, looking for a noun phrase (*np*, its object) to its right, then for a noun phrase (its subject) to its left to form a sentence *s*. The relativiser "which" looks for a complex formula *s*/*np* (that is a sentence missing a noun phrase in its rightmost position) to its right and for a noun to its left. The hypothetical *np* corresponds to a trace in mainstream syntactic theory. A weakness of the Lambek calculus is that this analysis does not extend to only slightly more complicated examples such as "moons which Galileo discovered in 1610", where the hypothetical noun phrase no longer occurs in a peripheral position. Many variants and extensions of the Lambek calculus have been developed with the goal of solving this and other problems (see, for example Moortgat 1997; Morrill 2011).

Given the lexicon, the phrase "moons which Galileo discovered" is a noun n iff the following holds.

$$n, (n\backslash n)/(s/np), np, (np\backslash s)/np \vdash n$$

The proof above shows this statement holds. Even though it is easy to verify this proof is correct by inspecting each rule application, it may not be immediately obvious how to find natural deduction proofs. In the next section, we will present a proof search procedure for the implicational fragment of Lambek calculus natural deduction.

2.2 *Proof search in natural deduction*

For our proof search procedure, the notion of *result* is useful.

Definition 2.2 *Given a formula F, its result is the atomic subformula of F defined as follows.*

$$\begin{aligned} result(A) &= A \qquad \textit{if } A \textit{ atomic} \\ result(A/B) &= result(A) \\ result(B\backslash A) &= result(A) \end{aligned}$$

Essentially, the result is the atomic formula we obtain once we have combined a formula with all its arguments. So the result of $(np\backslash s)/np$ is s and the result of $(n\backslash n)/(s/np)$ is n. Proof search for a sequent $A_1,\ldots,A_n \vdash C$ in natural deduction works as follows (a more precise description can be found in Moot and Retoré 2012).

1. If C is a complex formula, apply the appropriate introduction rules until we obtain an atomic formula p (this may add formulas to the left of A_1 and to the right of A_n).
2. Select an active hypothesis H of the proof such that $result(H) = p$ (that is, select a formula which eventually produces the current atomic goal formula).
3. Our current sequent is of form $A_1,\ldots,A_{i-1},H,A_{i+1},\ldots,A_n \vdash p$ and we need to subdivide the formulas to the left of H $(A_1,\ldots,A_{i-1})$ into m subsequences $\Gamma_1,\ldots,\Gamma_m$, where m is the number of arguments H takes to its left, and we need to subdivide the formulas to the right of H $(A_{i+1},\ldots,A_n)$ into k subsequences $\Delta_1,\ldots,\Delta_k$

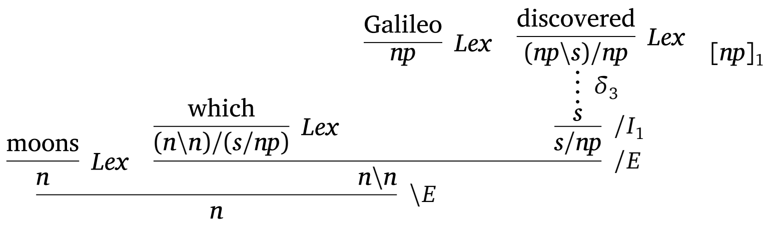
where k is the number of arguments H takes to its right (this fails if H selects no arguments to its left and $A_1,\ldots,A_{i-1}$ is not empty, and similarly if H has no arguments to its right and $A_{i+1},\ldots,A_n$ is not empty). We apply all elimination rules to H until we arrive at atomic formula p, then recursively find the proofs from step 1 for each of the arguments: proofs $\Gamma_q \vdash B_q$ for arguments to the left and proofs $\Delta_r \vdash D_r$ for arguments to the right. In the simplest case with a single argument on the right and a single argument on the left, $H = (B\backslash p)/D$; there is no need for splitting the A_i further and we simply try to find proofs for $A_1,\ldots A_{i-1} \vdash B$ and for $A_{i+1},\ldots,A_n \vdash D$. Succeed if all recursive steps succeed. If not, try other subdivisions of the hypotheses. Fail when there is no way to divide the hypotheses such that all subproofs succeed.

The algorithm above has non-determinism in two places. The first step is deterministic, but in the second step there may be several choices for the atomic goal formula and in the third step there may be several ways to split up the sequence of formulas (we need multiple arguments either to the right or to the left for this).

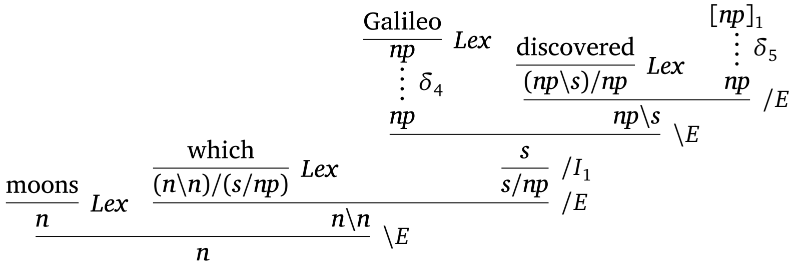
As an example, the sequent $n, (n\backslash n)/(s/np), np, (np\backslash s)/np \vdash n$ has an atomic conclusion, so nothing needs to be done for the first step. For the second step, there is the choice of two formulas: either n (corresponding to "moons") or $(n\backslash n)/(s/np)$ (corresponding to "which"). The first choice fails immediately since there are still formulas to the right which are not arguments of the formula producing the result (as it is atomic). The second choice provides a formula looking for an s/np to its right and an n to its left. Simply writing out the required elimination rules and separating the hypotheses produces the following.

$$\dfrac{\begin{array}{c}\dfrac{\text{moons}}{n}\,Lex\\ \vdots\,\delta_1\\ n\end{array} \qquad \dfrac{\dfrac{\text{which}}{(n\backslash n)/(s/np)}\,Lex \qquad \begin{array}{c}\dfrac{\text{Galileo}}{np}\,Lex \quad \dfrac{\text{discovered}}{(np\backslash s)/np}\,Lex\\ \vdots\,\delta_2\\ s/np\end{array}}{n\backslash n}\,/E}{n}\,\backslash E$$

We now need to complete the procedure recursively to find the proofs δ_1 and δ_2. The first subproof is trivial: we are looking for a noun and there is one, so δ_1 is empty and the n premiss and the n conclusion of δ_1 become the same formula occurrence. The second subproof has a complex goal, so according to step 1 we apply the introduction rule for "/" which produces the following.



Our subproof δ_3 requires us to prove $np, (np\backslash s)/np, np \vdash s$. Since s is atomic and only the transitive verb has s as its goal, this produces the following.



We can complete the proof by identifying the atomic noun phrases in δ_4 and in δ_5. The given proof procedure is top-down and enumerates eta-long beta-normal form proofs. In addition, different proofs correspond to different meanings, that is, different proofs will have different lambda terms assigned to them using the term assignment of Figure 1. This correspondence between natural deduction proofs and lambda-terms is the well-known Curry-Howard correspondence. It is not an isomorphism for the Lambek calculus, since not all intuitionistic proofs have a corresponding Lambek calculus proof. Even stronger, not all multiplicative intuitionistic linear logic proofs have a corresponding Lambek calculus proof: the Lambek calculus is a logic without contraction and weakening, like linear logic, but also without the exchange rule.

Adding the term assignment of Figure 1 produces the following proof.

$$\dfrac{\dfrac{\text{moons}}{n : m}\,Lex \quad \dfrac{\dfrac{\text{which}}{(n\backslash n)/(s/np) : w}\,Lex \quad \dfrac{\dfrac{\dfrac{\text{Galileo}}{np : g}\,Lex \quad \dfrac{\dfrac{\text{discovered}}{(np\backslash s)/np : d}\,Lex \quad [np : x]_1}{np\backslash s : (d\,x)}\,/E}{s : ((d\,x)\,g)}\,\backslash E}{s/np : \lambda x.((d\,x)\,g)}\,/I_1}{n\backslash n : (w\,(\lambda x.((d\,x)\,g)))}\,/E}{n : ((w\,(\lambda x.((d\,x)\,g)))\,m)}\,\backslash E$$

Each lexical assumption of the proof is assigned a unique variable (in the example above, this variable is the first letter of the corresponding word for the convenience of the reader) and these have exactly one free occurrence in the final term (these are linear lambda terms, since each abstraction binds exactly one variable as well). The lexicon assigns both a formula and a corresponding lambda term to each lexical entry. Computing the formal semantics corresponds to replacing each word by its lexical semantics. In the current case, ignoring complications such as tense and the plural, many words have a trivial meaning assignment, so we replace m by the constant $\mathit{moon}^{e\to t}$ (or, if we prefer, by its eta expansion $\lambda x^e.\mathit{moon}(x)$), d by the constant $\mathit{discover}^{e\to(e\to t)}$, g by the constant $\mathit{Galileo}^e$. The crucial case is the lexical assignment for "which", for which we replace w by $\lambda Q^{e\to t}\lambda P^{e\to t}\lambda y^e.(P\,y)\wedge(Q\,y)$. Making all lexical substitutions in our original term $((w\,(\lambda x.((d\,x)\,g)))\,m)$ produces the following term.

$$((\lambda Q\lambda P\lambda y.((P\,y)\wedge(Q\,y)))\,(\lambda x.((\mathit{discover}\,x)\,\mathit{Galileo})))\,\mathit{moon}$$

We apply beta-reduction by substituting $\lambda x.((\mathit{discover}\,x)\,\mathit{Galileo})$ for Q, which produces the following.

$$(\lambda P\lambda y.((P\,y)\wedge((\lambda x.((\mathit{discover}\,x)\,\mathit{Galileo}))\,y)))\,\mathit{moon}$$

A second beta-reduction replaces x by y as follows.

$$(\lambda P\lambda y.((P\,y)\wedge((\mathit{discover}\,y)\,\mathit{Galileo})))\,\mathit{moon}$$

The final beta-reduction replaces P by *moon* to produce the following normal form.

$$\lambda y.((\mathit{moon}\,y)\wedge((\mathit{discover}\,y)\,\mathit{Galileo}))$$

According to the standard notational conventions of Montague semantics (Gamut 1991), this corresponds to the following more natural term.

$$\lambda y.(moon(y) \wedge discover(Galileo, y))$$

That is, the y such that they are moons and were discovered by Galileo.

2.3 *Proof nets*

Type-logical grammars generally have multiple proof systems which are provably equivalent (in the sense that they derive the same theorems). Having multiple proof systems available is a great benefit, because meta-theoretical properties are often easier to prove in one system than in another.

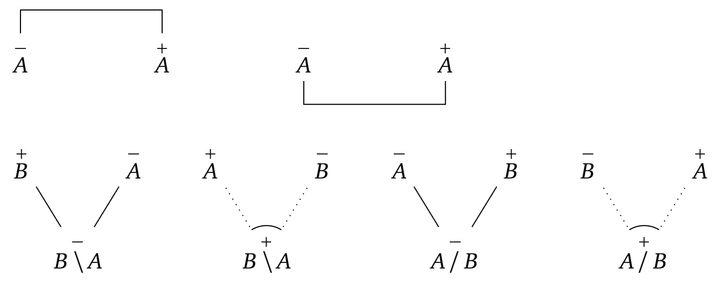
Even though natural deduction is a nice proof system producing proofs which are fairly easy to read and which have a direct connection to the semantics, we will introduce a second proof system, *proof nets*, which makes some aspects of proof combinatorics easier to see[2].

Proof nets are a proof system introduced for linear logic by Girard (1987). Proof nets represent proofs as (hyper)graphs, where the vertices are (polarized) formulas and the hyperlinks represent a connection between the main formula of a rule and its immediate subformulas. The links for the Lambek calculus are shown in Table 1. The formulas above a link are its premisses whereas the formulas below it are its conclusions. The axiom link, on the top left of Table 1 has no premisses and two conclusions (the order between them is irrelevant), whereas the cut link, on the top right, has two premisses (in any order) and no conclusions). The cut link is presented only for completeness, since the Lambek calculus satisfies cut elimination, we never need to use a cut link when using proof nets for proof search.

The links for the negative implications correspond to the natural deduction elimination rules, though the complex formula is the conclusion of the link and the main premiss of the elimination rule. The links for the positive implications correspond to the introduction rules, with the withdrawn hypothesis as the negative premiss of the rule.

Positive and negative formulas correspond essentially to unnegated and negated formulas (as in the classical equivalences be-

[2] Another advantage of proof nets is that, unlike natural deduction, adding the product rules to the proof net calculus presents no complications.



$\overset{-}{A}$ $\overset{+}{A}$
$\overset{-}{A}$ $\overset{+}{A}$
$\overset{+}{B}$ $\overset{-}{A}$ $\overset{-}{B\backslash A}$
$\overset{+}{A}$ $\overset{-}{B}$ $\overset{+}{B\backslash A}$
$\overset{-}{A}$ $\overset{+}{B}$ $\overset{-}{A/B}$
$\overset{-}{B}$ $\overset{+}{A}$ $\overset{+}{A/B}$

Table 1:
Links for Lambek calculus proof structures

tween $B \rightarrow A \equiv \neg B \vee A$ and $\neg(B \rightarrow A) \equiv B \wedge \neg A$). In the context of linear logic, polarities function as a restriction on classical formulas to ensure the intuitionistic restriction to a single conclusion. The distinction between solid and dotted links corresponds to the distinction between a logical conjunction (solid) and a logical disjunction (dotted). This distinction plays a key role in deciding the correctness of proof structures below.

Given a statement $A_1,\ldots,A_n \vdash C$, we obtain a proof frame by unfolding the formulas according to the logical links on the bottom row of Table 1, using the negative unfolding for the A_i and the positive unfolding for C, until we reach the atomic formulas. We then connect the atomic formulas by means of the axiom link shown on the top left of Table 1. We need to respect the linear order of the premisses for the logical links (and the linear order of the formulas in the sequent), but the axiom link can connect a positive and a negative atom in either order.

Figure 2 shows the formula unfolding corresponding to “moons which Galileo discovered”. The occurrences of the atomic formulas have been numbered to allow easy reference to them; these numbers are not a formal part of the proof structure. We saw the natural deduction proof for this noun in Section 2.1. When all axioms of a formula unfolding have been linked we call the resulting structure a *proof structure*. Not all proof structures correspond to proofs. The proof structures which do are *proof nets*. As we will see, we can distinguish proof nets from other proof structures just by looking at properties of the graph.

For the formula unfolding, it is immediately clear what the search space for potential proofs is: we need to find a 1-1 matching between positive and negative occurrences of the same atomic formula. So for Figure 2, we need to match the positive s_1 to the negative s_2 (there

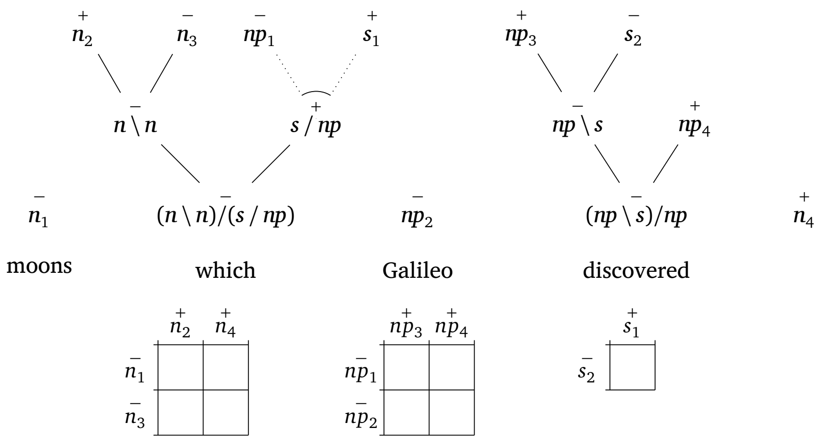


Figure 2: Formula unfolding for "moons which Galileo discovered"

is only one possible solution here), the two positive *n*'s to the two negative ones, and the two positive *np*'s to the two negative ones. The squares at the bottom of Figure 2 summarise the possibilities.

For Lambek calculus proof nets, the matching of atomic formulas must be planar. Planarity corresponds to non-commutativity of the logic and it therefore holds for the Lambek calculus but not for its extensions. Given that there is only one possibility for *s*, planarity constrains the possible axiom connections for the *np* formulas: when the two *s* formulas have been connected, the negative np_2 corresponding to "Galileo" can only be connected to the leftmost (subject) noun phrase np_3 of "discovered" since connecting it to the rightmost (object) noun phrase would force the link to cross the *s* axiom link. Similarly, the negative np_1 of "which" can only be linked to the object np_4 of "discovered" when we require a planar connection.

Finally, there are two planar matchings possible between the two positive and the two negative nouns: we either connect the negative n_1 of "moons" to the positive n_2 of "which" and the negative n_3 of "which" to the positive goal n_4, or vice versa (we had the same choice for natural deduction proof search in Section 2.2).

However, only one of these two possibilities produces a proof net. There are many graph-theoretical ways to characterise the proof nets among other proof structures. One simple way, due to Danos (1990), uses the graph contractions shown in Figure 3.

We first remove all formula information from a proof structure, replacing formula occurrence by unique vertex indices $v_0, v_1, \ldots,$ then

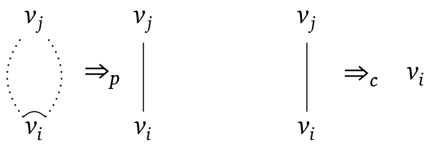


Figure 3:
Contractions for proof nets

applying the contractions. A proof structure is a proof net iff it contracts to a single vertex using these contractions. The condition on the rightmost contraction is that the two vertices are distinct. In other words, if the proof structure has a cycle containing only solid links, then we can use this contraction to reduce this cycle to a self-loop, but we can never eliminate it. Similarly, the contractions only shorten paths, but do not create new ones. Therefore, if a proof structure is disconnected, it will never contract to a single point. The leftmost reduction corresponds to the dotted links for the positive implications and it essentially requires us to "join" the two premisses of the link. As the previous discussion suggests, the contraction criterion is quite close to the more well-known acyclicity and connectedness condition (Danos and Regnier 1989). However, the contraction condition has the advantage of allowing a compact representation of intermediate structures in proof search, and is therefore more suitable for proof search (Moot 2017).

With this in mind, it is clear that of the two possible connections between the *n* formulas, connecting the two *n* formulas of "which" together produces a cycle of solid links, and therefore a structure which doesn't contract to a point. In addition, connecting the noun "moons" to the goal formula produces an axiom link disconnected from the rest of the structure. Therefore, the structure shown in Figure 4 is the only proof net given this sequence of formulas. It is easy to verify it contracts to a point given the contractions of Figure 3. One way of proceeding is eliminating all "dangling" links (that is, links connecting a single vertex to the rest of the structure). When such links have been recursively removed, we end up with the pair of dotted links and a solid path between the positive *s* and the negative *np* of this dotted link. We can contract this path to a single vertex, producing the correct configuration for contracting the pair of dotted links and apply the final contraction to the resulting solid link to produce a single vertex.

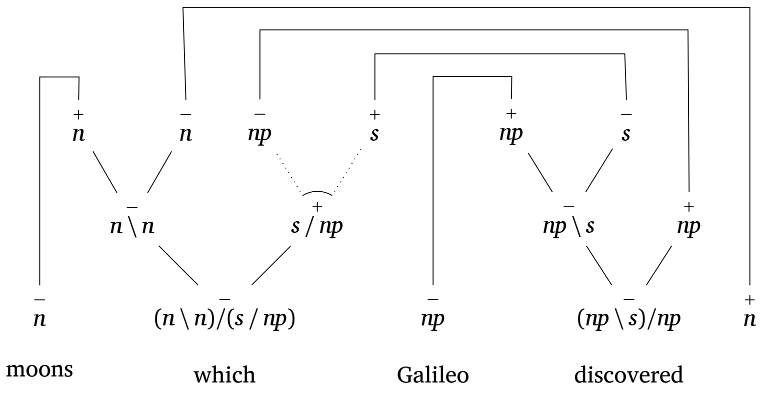


Figure 4: Proof net for "moons which Galileo discovered"

3 COMBINATORICS AND COMPLEXITY

Using type-logical grammars for computing the meaning of sentences and using the resulting meaning for different tasks (entailment, question answering, etc.) has the following bottlenecks.

1. Lexical lookup. For wide-coverage grammars, the number of formulas that the lexicon assigns to many common words is rather large.
2. Proof combinatorics. Finding a proof (or the best proof for some numerical definition of best) is NP-complete for most type-logical grammars.
3. Meaning computation. Computing the meaning of a sentence is done by substituting lexical lambda terms and then normalising the resulting simply typed lambda term. Normalising simply typed lambda terms is known to be of non-elementary complexity.
4. Meaning use. Questions of logical entailment between sentences are undecidable, even in the first-order case.

The focus of the rest of this article will be on Item 2, proof combinatorics, but I will offer some brief remarks on the other items.

A probabilistic lexicon The number of lexical formulas per word is a major problem for real-world applications. However, when we fix a set of possible formulas and have enough examples of sentences with the correct formula assignment, we can define a probability model over words, together with a limited amount of context (typically the two preceding and succeeding words). This general approach is called supertagging (Bangalore and Joshi 2011) and it has been applied successfully to many formalisms including type-logical grammars (Moot 2010, 2014b, presents supertaggers for multimodal type-logical grammars for Dutch and for French).

Since the topic of supertagging has been discussed at length elsewhere and provides good, practical solutions to the problem of lexical ambiguity we will not elaborate on it here.

Meaning computation Schwichtenberg (1982) shows that normalising simply typed lambda terms has non-elementary worst-case complexity. This complexity result essentially exploits recursive copying. However, there are many implementations of the simply typed lambda calculus for computational linguistics which perform rather efficiently, and this in spite of the fact that, in general, little effort is spent on optimising the implementation of the normalisation component. We claim that the meaning recipes necessary for the lexicon are all terms of soft linear logic, and hence can be reduced in polynomial time (Lafont 2004; Baillot and Mogbil 2004). This accounts for the observed fact that lambda term normalisation is not a real bottleneck in practice (Moot and Retoré 2016).

Logical entailment Given that logical entailment is undecidable in general, there are two basic strategies:

1. We can use an off-the-shelf theorem prover (generally using some time limit) and simply see whether it finds a proof. Bos and Markert (2005) use this approach (as well as some more approximative measures).
2. We can use an incomplete but decidable logical fragment for computing entailment. Abzianidze (2017) uses this approach.

In both cases, the result is a high-precision but low-recall system (that is, when the system produces an answer, it is usually right, but there are many correct answers for which no proofs are found). The main bottleneck to improving recall is adding a logical formalisation of a sufficient amount of world knowledge (without reducing prover performance), a classic problem in artificial intelligence.

4 WEIGHTED PROOF SYSTEMS

Given a sequent $\Gamma \vdash C$, the formula unfolding into a proof frame gives a compact representation of the proof combinatorics for the given sequent. We can combine positive and negative occurrences of the same atomic formulas until we obtain a proof structure. An atomic formula a with n positive and n negative occurrences (the number of positive and negative occurrences for each atomic formula must be equal if the sequent is derivable; this is called the *count check*) corresponds to an $n \times n$ matrix, and a potential reading for this sequent, that is a proof structure, is a perfect matching between positive and negative occurrences.

When we fill the matrix with weights (we will discuss some different ways of computing these weights below), it becomes possible to compute the best proof structure according to these weights. We can either use best-first search, connecting the "best" axiom links first for each local choice, or use a k-best proof structure computation, computing the k total links which are the best globally. Given that computing the k-best proof structures this way can be done in polynomial time (Kuhn 1955), there is no guarantee that the best proof structure is actually a proof net (unless $\mathrm{P} = \mathrm{NP}$). However, we can use a polynomial k-best system as an incomplete approximation of proof search.

Given two atomic formulas a_1 and a_2 of the same type (n, np, s) but of opposite polarity there are different ways of assigning a weight to the possible axiom link between them.

1. We can use the distance between words as weight, using distance 0 when the two atomic formulas are subformulas of the same formula occurrence, distance 1 between adjacent words, etc.
2. We can use a probability-like measure, estimated from proof nets in a large corpus.
3. We can use the word similarity measure between w_1 and w_2.

We will discuss each of these alternative measures in turn in the next sections.

4.1 *Word distance*

One simple metric to use is word distance, preferring axiom connections between closer words. This gives a strong preference to linking atomic subformulas of the same formula, followed by linking atomic subformulas of adjacent formulas.

Given a proof net with k axiom links, when processing this proof net left-to-right performing axiom links as soon as possible, this will mean each link of distance of n will cause the leftmost corresponding atom to be open/unlinked for n steps. Measuring the number of open axioms after each word has been proposed as a straightforward model of human sentence processing which, in spite of its simplicity, makes a number of correct predictions about processing (Johnson 1998; Morrill 1998, 2011).

We illustrate this by examining the contrast between Dutch and German verb clusters. Verb clusters in both Dutch and German can have a sequence of verb arguments followed by a sequence of verbs selecting these arguments. The difference between German and Dutch is that German verbs select these arguments right-to-left (that is, the leftmost verb selects the rightmost argument) and the Dutch verb select these arguments left-to-right (that is, the leftmost verb selects the leftmost argument). This is illustrated by the following contrast.

(1)

Wolfgang	hat	die Lehrerin	die Murmeln	aufräumen	helfen
Wolfgang	has	the teacher	the marbles	collect	help

Wolfgang helped the teacher collect the marbles

(2)

Jantje	heeft	de lerares	de knikkers	helpen	opruimen
Jantje	has	the teacher	the marbles	help	collect

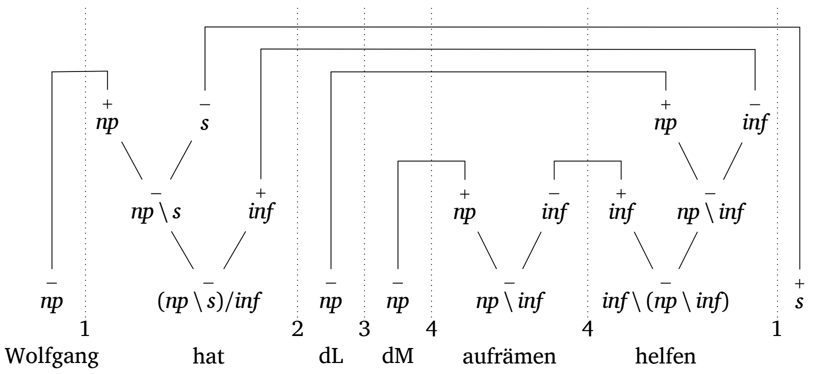
Jantje helped the teacher collect the marbles

Bach *et al.* (1986) find that, in an experimental setting, German sentences like the one above are harder for German native speakers than the Dutch sentences are for Dutch native speaker: although the difference is not very large, the German sentences are not only judged as harder by the German speakers, but the German test subjects also make more comprehension errors.

Figure 5 shows the proof net corresponding to sentence (1). The noun phrases "die Lehrerin" and "die Murmeln" have been not been treated as a combination of np/n and n but as a simple np to reduce the size of the proof net. This will not affect the comparison with the Dutch example.[3]

[3] Some other simplifications have been made: neither the German auxiliary "hat" nor the Dutch auxiliary "heeft" can select a simple infinitive argument (i.e.

Figure 5: Proof net for "Wolfgang hat die Lehrerin die Murmeln aufräumen helfen"



Computing the processing complexity using a proof net such as the one shown in Figure 5 requires us to make some choices. We can assume that the hearer knows to expect a sentence (and therefore put the goal formula initially). Morrill (2011) chooses this option. We can also keep the goal formula at the end, as done here.

There are some other potential complexity issues to take into account: some choices of lexical formulas and of axiom links lead to failure and it is possible that this affects processing complexity (at least it does so for a computer implementation). The *size* of the partial proof net constructed so far may also play a role (the size of the contracted partial proof net according to the contractions of Figure 3 seems a good candidate for such a size measure).

Morrill and Johnson use the simplest solution here, measuring complexity by the successful proof nets when processed left to right, counting the number of unlinked axioms at each step.

Figure 5 shows a dotted column after each word, together with a count of the number of axioms it crosses. In the example, after the first word, "Wolfgang", there is a single unlinked *np*, therefore the count is 1. After "hat", the *np* of "Wolfgang" becomes linked but an unlinked *s* and an unlinked *inf* are added, leaving a total of 2 unlinked atoms.

we have "Die Lehrerin hat die Murmeln aufgeräumt", with a past participle rather than an infinitive); they only take an infinitive argument when this infinitive itself selects for another infinitive. This can be solved either by adding features or by distinguishing the atomic types. Bach *et al.* (1986) note that for German (but not for Dutch) both grammar textbooks and speakers disagree over whether the final verb should be an infinitive or a past participle.

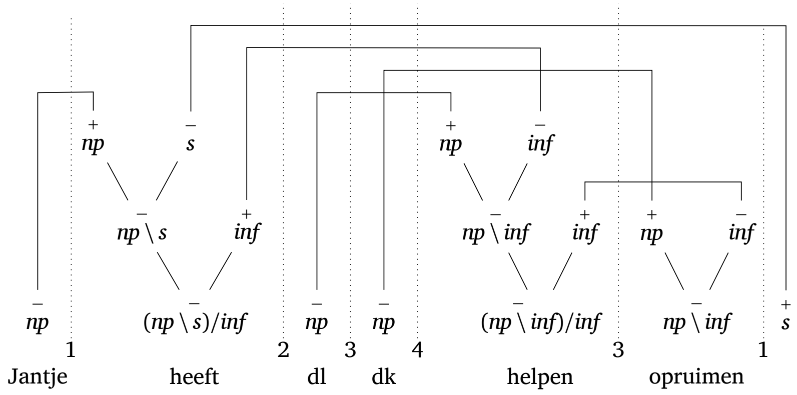


Figure 6: Proof net for "Jantje heeft de lerares de knikkers helpen opruimen"

In general, the complexity profile of verb clusters in German (and in Dutch) rises when the arguments of the verbs in the cluster are encountered (the noun phrases "die Lehrerin" and "die Murmeln"), then descends when the verbs start selecting their arguments. This provides an explanation for why these sentences quickly become unacceptable with multiple levels of embedding.

The complexity profile of Figure 5 has a maximum complexity of 4 and a total complexity of 15.

To provide a proof net for the Dutch example (2), we need somewhat more complex proof net machinery, since the crossed dependencies of Dutch cannot be handled by planar structures. Since these more complicated proof nets do not affect our chosen complexity measure, we will simply look at the complexity profile for the non-planar proof net in Figure 6: this is not a Lambek calculus proof net, and we assume a proof net calculus which allows only this structure for the example sentence.

The complexity profiles of the two sentences are the same until the first infinitive, but then the Dutch sentence has a slight advantage: its maximum complexity, like the German example, is 4; but its total complexity is 14, compared to 15 for the German example. This advantage becomes somewhat more pronounced when we add a third and a fourth verb.

These examples are interesting because many known psycholinguistic facts, even some fairly subtle ones like shown here, are a direct consequence of rather minimal assumptions about a model of processing. Morrill (2011) presents many other examples.

4.2 *Corpus estimation*

Given a sequent, it is not hard to define a probability distribution over its proof structures: to obtain such a probability distribution we simply need to fill the $n \times n$ matrix for each atomic formula in such a way that all rows and all columns sum to 1. In other words, each atomic formula is assigned a probability distribution over the atomic formulas of opposite polarity. For example, looking to Figure 6, for a full proof search the negative np corresponding to 'Jantje' can be linked to the positive np of 'heeft', the positive np of 'helpen' and the positive np of 'opruimen' and the sum of these probabilities must be 1. Similarly, the positive np of 'heeft' can be linked to the negative np of 'Jantje', the negative np of 'de lerares' and the negative np of 'de knikkers'.

Mathematically, it is much harder to define a probability distribution over proof *nets*. When we define a probability distribution over proof structures, we assign a non-zero probability to structures which do not correspond to proofs. Even though it makes sense a priori to want non-proofs to have zero probability, this entails that an underivable sequent should fail to be assigned a probability distribution at all, since no axiom link can contribute to a proof. However, this means assigning probabilities becomes an NP-hard problem, since we would be able to decide derivability of a sequent from the success or failure of computing a probability distribution.

Accepting the assignment of non-zero probability to axiom links which are not part of any proof is comparable to probabilistic context-free grammar parsers assigning non-zero probability to constituents which cannot be part of a derivation of the complete string.

The question for probability assignment is how likely is the nth atomic formula of word w_1 to combine with the kth atomic formula of w_2 (with some form of backoff, for example to the two part-of-speech tags). In general, we can use well-known statistical methods (Berger *et al.* 1996) to compute a probability function from any combination of properties from formulas, words and context. This possibility has so far been little explored in the context of type-logical grammars.

In terms of assigning weights to atomic formulas, this is not fundamentally different from the other weighted approaches discussed here. It has the advantage over the other methods that it can distinguish between different arguments of the same formula (neither the distance measure nor the vector similarity measure does this). However, it has the disadvantage that it requires a large amount of annotated data to estimate the probabilities.

4.3 *Vector similarity*

An advantage of using vector similarity rather than a large corpus of parsed text is that it is much easier to obtain the former than it is to obtain the latter: a high-quality parsed corpus of sufficiently large size requires an enormous effort in times of person-hours; on the other hand, computing word vectors can be done automatically and, using the enormous amount of text available on the internet, on a scale unrealistic for any manual method. Computing word vectors from the web still requires an important effort in crawling, cleaning, duplicate detection, etc., but nowhere near the person-hours needed to manually annotate a similar size corpus, something especially relevant for under-resourced languages: as discussed by Kilgarriff and Grefenstette (2003), even ‘smaller’ languages such as Icelandic, Basque, Latin and Esperanto have over 50 million words of text available according to conservative estimates. For many of the most-used languages on the internet, cleaned-up and (automatically) annotated versions of this content are freely available and can be used to extract word vectors (Baroni *et al.* 2009).

Weighting axiom links according to the similarity of the words given by distributional semantics means preferring connections between words with related meanings (this appears to be close to the notion of *discourse coherence* as used by Asher and Lascarides 2003, only in a more shallow, syntactic context).

Even though this basic idea is easily stated, implementing it requires making some choices. While distributional semantic similarity is easily defined for two words, defining it for two complex expressions is essentially the compositionality problem for vector space semantics.

A simple solution would be to choose the vector sum for composition, and this already performs surprisingly well on several similarity tasks. However, the vector sum approach is ill-adapted to preferences in type-logical proofs: given that we need to match the atomic subformulas of all words in a sentence in any case, and given that the vector sum operation is associative and commutative, this would not allow us to distinguish between different word groupings (or even between different word orders).

We therefore need a slightly more sophisticated method for combining vector similarity with proof rules. We adapt the basic idea of lexicalised parsing with context-free grammars and use a *head word* for each expression — the verb for a verb phrase, the noun for a noun phrase, etc. In the context of type-logical grammar, we therefore specify the following general principles, as sort of type-logical equivalences to the head percolation principles of Magerman (1994):

1. the head of a lexical hypothesis is the word itself,
2. the head of the combination of A/A with A and of A with $A\backslash A$ is the head of A
3. the head of np/n with n (resp. pp/np with np) is the head of the noun n (resp. the head of the noun phrase np)
4. the head of other formulas A/B and $B\backslash A$, with $A \neq B$, is the head of A.

Moreover, for the semantic assignments in a type-logical lexicon, we can distinguish between the lexical entries whose semantic content is purely logical (using only the logical constants like "$\neg$" "$\wedge$", "$\forall$", and a few other predicates whose meaning is invariant across models, like "$=$"and "$<$") and those whose semantic content is not (these typically contain predicates like "*love*" and "*book*" corresponding to the lexical entry itself).[4]

The basic elements in our formal setup are now triples containing a *formula*, a *head word*, and *vector distance weight*, with each lexical entry starting with the word lemma as its head and distance zero. For each elimination rule, the new head word is defined by the propagation rules above (it is the head of the argument when the functor is a modifier, a determiner or a preposition, and the head of the functor otherwise) and the weight is updated by adding the weights of the two premisses and additionally adding the distance of the two head words according to the vector model. The rule be-

[4] This contrast is unfortunately not as sharp as we would like it to be: while it is simple to see "all" and "some" as purely logical, it is much less easy to see how other words such as "few" and "many" can be interpreted in terms of purely logical operators.

low presents a general elimination rule operating on triples (with "$\multimap$" generalising over both "/" and "\") according to this description.

$$\frac{\langle w_1, h_1, A \multimap B\rangle \quad \langle w_2, h_2, A\rangle}{\langle w_1 + w_2 + d(h_1, h_2), h, B\rangle} \multimap E$$

From a purely logical point of view (that is, looking only at the third element of the triple), this rule operates just like a normal elimination rule. The head h of the conclusion will be either h_1 or h_2 depending on the head propagation rules described above. The weight computation uses a distance measure d computing the distance between the two head words and simple addition. Nothing in particular hinges on the use of "+" here; any function monotone in both its arguments can be used here. Many choices are also possible for the distance measure d, but in the examples below we use the simple cosine measure which is the most commonly used for distributional similarity. The cosine measure produces 1 when the vectors point in the exact same direction, 0 when the vectors are orthogonal.[5] This ensures the highest-weight proof combines the nearest vectors.

For the introduction rules, it is somewhat more difficult to define the proper elements for the discharged hypothesis of the rule. We can simply choose zero for its weight, but it is unclear what the proper head word for a hypothesised constituent is. One simple solution is to assign the empty word ϵ to such hypotheses and stipulate that the empty word has distance zero to all other words (i.e. for all w, $d(\epsilon, w) = d(w, \epsilon) = 0$). This would give the following introduction rule.

$$\frac{\begin{array}{c}\langle 0, \epsilon, B\rangle \\ \vdots \\ \langle w, h, A\rangle\end{array}}{\langle w, h, B \multimap A\rangle} \multimap I$$

We can also use a somewhat more sophisticated rule for subproofs of the following form.

[5] In principle, we can have -1 when the vectors point in the exact opposite direction, although many methods are guaranteed to obtain positive vectors only.

$$\dfrac{\begin{array}{c}\vdots\\ \langle w_1,h_1,(B\multimap A)\multimap C\rangle\end{array} \quad \dfrac{\begin{array}{c}[\langle 0,h_1,B\rangle]^k\\ \vdots\\ \langle w_2,h_2,A\rangle\end{array}}{\langle w_2,h_2,B\multimap A\rangle}\multimap I_k}{\langle w_1+w_2,h_1,C\rangle}\multimap E$$

Where for relative pronouns $A = s$, $B = np$ and $C = n\backslash n$, and for generalised quantifiers $A = s$, $B = np$ and $C = s$. Essentially, h_1 is propagated from the left premiss of the elimination rule to the hypothesis of the introduction rule. This works well since the head word of a determiner phrase (of type $(np \multimap s) \multimap s$) is its noun, and similarly, the noun argument of the relative pronoun is semantically identical to the extracted noun phrase $B = np$.

As a concrete example, a French fragment like “concert de piano gratuit”, like its English translation “free piano concert”, has two possible readings, one where there is a piano concert which is free and one where a free piano is used to give a concert. This type of ambiguity, although somewhat reduced by noun/adjective agreement, is quite common in the French Treebank (Abeillé *et al.* 2003) and apparently a difficult construction both for journalists and annotators. Treating this example according the the method described above provides the following (to reduce horizontal space, we have used w_1, w_2 and w_3 for the weight terms to be discussed later).[6]

$$\dfrac{\dfrac{\dfrac{\text{concert}}{\langle 0,\mathit{concert},n\rangle}\mathit{Lex} \quad \dfrac{\dfrac{\text{de}}{\langle 0,\mathit{de},(n\backslash n)/n\rangle}\mathit{Lex} \quad \dfrac{\text{piano}}{\langle 0,\mathit{piano},n\rangle}\mathit{Lex}}{\langle w_1,\mathit{piano},n\backslash n\rangle}/E}{\langle w_2,\mathit{concert},n\rangle}\backslash E \quad \dfrac{\text{gratuit}}{\langle 0,\mathit{gratuit},n\backslash n\rangle}\mathit{Lex}}{\langle w_3,\mathit{concert},n\rangle}\backslash E$$

The weight w_1 of the proof showing “de piano” is of type $n\backslash n$ is equal to the weight of its to premisses (both zero) plus the distance between the heads of the two premisses of the rule, “de” and “piano” in our case, which have a distance of 0.0740, so we conclude w_1 is 0.0740. We can now compute the weight of the proof showing “concert de piano” to be of type n by combining the weight 0 of “concert” with

[6] Here and elsewhere, all weights are computed used the models provided by Fauconnier (2016) at `http://fauconnier.github.io/#software`

the weight 0.0740 of “de piano” and adding the distance between the two head words “concert” and “piano’, which is 0.4398 (that is, these words are fairly close) to arrive at $w_2 = 0.5138$. Finally, we combine this *n* with the adjective “gratuit” to compute the final weight w_3 by adding the previously computed w_2 to 0 (the weight of “gratuit”) and adding the distance between “concert” and “gratuit”, which is 0.1921. This gives us a total weight w_3 of 0.7059 for this reading (note that this number is not a probability and meaningful only in comparison to similarly computed numbers).

The second proof looks as follows.

$$\dfrac{\dfrac{\text{concert}}{\langle 0, \mathit{concert}, n\rangle}\,\mathit{Lex} \quad \dfrac{\dfrac{\text{de}}{\langle 0,\mathit{de},(n\backslash n)/n\rangle}\,\mathit{Lex} \quad \dfrac{\dfrac{\text{piano}}{\langle 0,\mathit{piano},n\rangle}\,\mathit{Lex}\quad \dfrac{\text{gratuit}}{\langle 0,\mathit{gratuit},n\backslash n\rangle}\,\mathit{Lex}}{\langle w_1,\mathit{piano},n\rangle}\,\backslash E}{\langle w_2,\mathit{piano},n\backslash n\rangle}\,/E}{\langle w_3,\mathit{concert},n\rangle}\,\backslash E$$

Even though the second reading uses the exact same words, it combines them in a different way, and this affects the weight calculations. We now combine “piano’ and “gratuit” first, to obtain $w_1 = 0+0+d(\mathit{piano},\mathit{gratuit}) = 0.0695$. We then combine this result with “de” and calculate $w_2 = 0 + w_1 + d(\mathit{de},\mathit{piano}) = 0.0695 + 0.0740 = 0.1435$. Finally, we combine the previous result with “concert” and calculate $w_3 = 0 + w_2 + d(\mathit{concert},\mathit{piano}) = 0.1435 + 0.4398 = 0.5833$.

These calculations show a preference for the first reading, where the concert is free rather than the piano. The key difference between the two readings is that $d(\mathit{concert},\mathit{gratuit}) > d(\mathit{piano},\mathit{gratuit})$, whereas the other computed terms are equal.

We can use the same method to compute “voir la fille avec les lunettes” (to see the girl with the glasses), since $d(\mathit{voir},\mathit{lunettes}) > d(\mathit{fille},\mathit{lunettes})$, which gives a preference for “voir ... avec les lunettes” over “fille avec les lunettes”.

Using semantic relatedness like this is, of course, not without its defects. For example, the verb phrase “saw the star with the telescope” is structurally identical to the example above, but has only one plausible reading, where “with the telescope” is a verb phrase modifier. However, “star” and “telescope” are closer semantically than “girl” and “telescope” are. The problem here is essentially that semantic vector similarity as we are using it here doesn’t give us any information about argument structure.

This suggests the need for more sophisticated ways to combine vector semantics, such as used by Baroni and Lenci (2010). In the context of a real-world system, the lexicon is a probability distribution over a finite set of formulas and therefore the highest-weight proof for a sentence must be a combination of the probability over the formulas with the weight over the axioms. The right way of combining the weights of the supertagger (a probability distribution over formulas) with the vector weights needs to be determined empirically, of course. Two simple possibilities are:

1. taking the best supertagger sequence for which a proof is found, then finding the maximum weight proof for this sequence;
2. combine the supertagger probabilities with the weight of the proof into a single weighted sum; that is, we treat finding the relative importance of the two weights as a standard machine learning objective to be determined empirically.

In the case of “voir l’étoile avec le téléscope” (*see the star with the telescope*), the supertagger (Moot 2014b) strongly prefers adverbial use of “with” over adjectival use (26.4% against 0.9%), a very strong preference in favour of the preferred reading. Therefore, in a real-world system this problem disappears unless the weighted sum gives a very strong priority to the vector weight component.

4.4 *Vector similarity and proof nets*

We can adapt the above strategy with minor modifications to a proof net parser. This is done by replacing atomic formulas by binary predicates, where the first argument represents the head and the second argument the weight. For weights, we use the real numbers with the usual function “ $+$ ” and the $d(w_1, w_2)$ function computing the distance between two words w_1 and w_2. Instead of having extra-logical head percolation and weight computation principles, these now form a part of the lexical entries (although these extra-logical principles would explain many common patterns occurring in the lexical entries and would make it possible to create an automatic compilation step adding the head and weight arguments to a ‘standard’ lexicon). Using first-order arguments has many applications, including as a solution for the Dutch verb clusters we’ve seen in Section 4.1 (Moot 2014a). As we use them here, these arguments are extra-grammatical and serve

$$
\begin{aligned}
\mathit{lex}(\text{book}) &= \mathit{n}(\mathit{book}, 0)\\
\mathit{lex}(\text{the}) &= \mathit{np}(X, w)/\mathit{np}(X, w)\\
\mathit{lex}(\text{interesting}) &= \mathit{n}(X, w + d(\mathit{interesting}, X))/\mathit{n}(X, w)\\
\mathit{lex}(\text{read}) &= (\mathit{np}(X, w_1)\backslash \mathit{s}(\mathit{read}, w_1 + w_2 + d(w_1, \mathit{read}) + d(w_2, \mathit{read}))/\mathit{np}(Y, w_2)\\
\mathit{lex}(\text{which}) &= (\mathit{n}(X, w_1)\backslash \mathit{n}(X, w_1 + w_2))/(\mathit{s}(Y, w_2)/\mathit{np}(X, 0)\\
\mathit{lex}(\text{every}) &= (\mathit{s}(Y, w_1 + w_2)/(\mathit{np}(X, 0)\backslash \mathit{s}(Y, w_2)))/\mathit{n}(X, w_1)
\end{aligned}
$$

Table 2: Lexical assignments with head word and weight computation information

only as a way to compute preferences among different proofs using the same formulas.

As before, atomic content words, like the noun "book" are assigned the entry *n*(*book*, 0). That is, the head constituent of the noun is "book" itself and the weight assigned to this expression is zero. Table 2 lists some other lexical entries in the current context.

As shown in the table, the determiner "the" is assigned an entry simply copying both the head word and the weight, following the principles of a purely logical word in the previous section.

An adjective such as "interesting" on the other hand copies the head word, but adds the distance between the head word to the previous weight.[7]

Similarly, the transitive verb "read" adds both the distance between the verb and its subject and the distance between the verb and its object to the weight of its two arguments.

The weighted entry for "which" requires some explanation. First of all, the head word X of the resulting noun is the same as the head of the argument noun (this behaviour is consistent with a noun modifier, and it makes, for example, "book which ..." behave the same as "book" in this respect). Second, the extracted *np*, being a hypothetical element, starts at weight 0 and shares the head with the noun argument. This approach therefore expects the long-distance dependency

[7] Modifiers of modifiers (e.g. adverbs like "very") cannot be handled in the same way as in the natural deduction based account of the previous section. Since they modify the adjective or adverb they select, and since these no longer contain this adjective or adverb as their head word, we can no longer compute the distance between e.g. "very" and "interesting", and between "very" and "quickly" since both "interesting" and "quickly" are not arguments of any of their atomic subformulas. The natural deduction approach of the previous section assigns heads to formulas with no requirement that these be atomic, and this provides a potential benefit for these cases.

between a noun and a relative clause to occur most likely between the head noun and the verb which is semantically closest to it.

Compared to the standard proof net matching algorithm using minimisation (or maximisation) of weight, we have now added computation of weights to the matching process. This presents something of a complication. However, since we need to do only simple computations (addition, vector cosine) in each cell of the matrices representing the search space, this doesn't make a big difference computationally.

4.5 *Limitations*

The current method doesn't distinguish between object and subject arguments and this is an important weakness.[8] This limitation is essentially a consequence of the division of labour between the distributional and type-logical approaches: the type-logical component of the system is solely responsible for word order while the distributional component only tests for similarity between a verb and its argument, taking neither grammatical nor structural considerations into account. We therefore need either a more subtle similarity measure or another way of distinguishing the likely relations between a verb and its different arguments.

5 CONCLUSIONS AND FUTURE WORK

We have given an overview of several methods of adding weights to proof search in type-logical grammars. With the exception of Bonfante and de Groote (2001), this possibility has been seldom discussed in the type-logical grammar literature, to our surprise. We have given applications of weighted proof search to modelling human processing, to finding parses most similar to those found in a given corpus, and to finding parses which prefer grouping similar words together. These methods still need to be thoroughly evaluated beyond the manually calculated examples shown here. Fortunately, for many current type-logical grammars and their theorem provers, the groundwork for incorporating weighted proof search has already been laid down. Given

[8] As discussed in Section 4.2, we can incorporate this when estimating probabilities from a corpus, but not for the other methods discussed here, i.e. vector similarity and word distance.

the many potential applications of weighted proof search, we look forward to testing these methods against available data for parsing and human processing.

REFERENCES

Anne ABEILLÉ, Lionel CLÉMENT, and François TOUSSENEL (2003), Building a Treebank for French, in Anne ABEILLÉ, editor, *Treebanks*, volume 20 of *Text, Speech and Language Technology*, pp. 165–187, Springer.

Lasha ABZIANIDZE (2017), *A natural proof system for natural language*, Ph.D. thesis, Tilburg University.

Nicolas ASHER and Alex LASCARIDES (2003), *Logics of Conversation*, Cambridge University Press.

Emmon BACH, Colin BROWN, and William MARSLEN-WILSON (1986), Crossed and Nested Dependencies in German and Dutch: A Psycholinguistic Study, *Language and Cognitive Processes*, 1(4):249–262.

Patrick BAILLOT and Virgile MOGBIL (2004), Soft Lambda-calculus: A Language for Polynomial Time Computation, in *Foundations of software science and computation structures*, pp. 27–41, Springer.

Srinivas BANGALORE and Aravind JOSHI (2011), *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*, MIT Press, Cambridge, Massachusetts.

Marco BARONI, Silvia BERNARDINI, Adriano FERRARESI, and Eros ZANCHETTA (2009), The WaCky Wide Web: A Collection of Very Large Linguistically Processed Web-Crawled Corpora, *Language Resources and Evaluation*, 43(3):209–226.

Marco BARONI and Alessandro LENCI (2010), Distributional Memory: A General Framework for Corpus-based Semantics, *Computational Linguistics*, 36(4):673–721.

Adam BERGER, Stephen DELLA PIETRA, and Vincent DELLA PIETRA (1996), A Maximum Entropy Approach to Natural Language Processing, *Computational Linguistics*, 22(1):39–71.

Guillaume BONFANTE and Philippe DE GROOTE (2001), Stochastic Lambek Categorial Grammars, in Geert-Jan KRUIJFF, Larry MOSS, and Richard T. OEHRLE, editors, *Proceedings of FGMOL 2001*, volume 53 of *Electronic Notes in Theoretical Computer Science*, Elsevier.

Johan BOS and Katja MARKERT (2005), Recognising Textual Entailment with Logical Inference, in *Proceedings of the 2005 Conference on Empirical Methods in Natural Language Processing (EMNLP 2005)*, pp. 628–635.

Vincent DANOS (1990), *La logique linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul)* [Linear logic applied to the study of various normalisation processes (mainly of the lambda calculus)], Ph.D. thesis, University of Paris VII.

Vincent DANOS and Laurent REGNIER (1989), The Structure of Multiplicatives, *Archive for Mathematical Logic*, 28:181–203.

Jean-Philippe FAUCONNIER (2016), *Acquisition de liens sémantiques à partir d'éléments de mise en forme des textes : exploitation des structures énumératives* [Acquisition of semantic relations from text layout elements: exploitation of enumerative structures], Ph.D. thesis, Université de Toulouse.

L. T. F. GAMUT (1991), *Logic, Language and Meaning*, volume 2, The University of Chicago Press.

Jean-Yves GIRARD (1987), Linear Logic, *Theoretical Computer Science*, 50:1–102.

Mark JOHNSON (1998), Proof Nets and the Complexity of Processing Center-Embedded Constructions, *Journal of Logic, Language and Information*, 7(4):443–447.

Adam KILGARRIFF and Gregory GREFENSTETTE (2003), Introduction to the Special Issue on the Web as Corpus, *Computational Linguistics*, 29:333–347.

Harold W. KUHN (1955), The Hungarian Method for the Assignment Problem, *Naval Research Logistics Quarterly*, 2:83–97.

Yves LAFONT (2004), Soft Linear Logic and Polynomial Time, *Theoretical Computer Science*, 318(1):163–180.

Joachim LAMBEK (1958), The Mathematics of Sentence Structure, *American Mathematical Monthly*, 65:154–170.

David M. MAGERMAN (1994), *Natural language parsing as statistical pattern recognition*, Ph.D. thesis, University of Pennsylvania.

Jeff MITCHELL and Mirella LAPATA (2010), Composition in Distributional Models of Semantics, *Cognitive Science*, 34:1388–1429.

Michael MOORTGAT (1997), Categorial Type Logics, in Johan VAN BENTHEM and Alice TER MEULEN, editors, *Handbook of Logic and Language*, chapter 2, pp. 93–177, Elsevier/MIT Press.

Richard MOOT (2010), Automated Extraction of Type-logical Supertags from the Spoken Dutch Corpus, in Srinivas BANGALORE and Aravind JOSHI, editors, *Complexity of Lexical Descriptions and its Relevance to Natural Language Processing: A Supertagging Approach*, chapter 12, pp. 291–312, MIT Press, Cambridge, Massachusetts.

Richard MOOT (2014a), Extended Lambek Calculi and First-order Linear Logic, in Claudia CASADIO, Bob COECKE, Michael MOORTGAT, and Philip SCOTT, editors, *Categories and Types in Logic, Language, and Physics: Essays dedicated to*

Jim Lambek on the Occasion of this 90th Birthday, number 8222 in Lecture Notes in Artificial Intelligence, pp. 297–330, Springer, Heidelberg.

Richard MOOT (2014b), A Type-logical Treebank for French, *Journal of Language Modelling*, 2(2).

Richard MOOT (2017), The Grail Theorem Prover: Type Theory for Syntax and Semantics, in Zhaohui LUO and Stergios CHATZIKYRIAKIDIS, editors, *Modern Perspectives in Type Theoretical Semantics*, Springer.

Richard MOOT and Christian RETORÉ (2012), *The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics*, number 6850 in Lecture Notes in Artificial Intelligence, Springer, Heidelberg.

Richard MOOT and Christian RETORÉ (2016), Natural Language Semantics and Computability, Technical report, LIRMM.

Glyn MORRILL (1998), Incremental Processing and Acceptability, Technical Report LSI–98–46–R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya.

Glyn MORRILL (2011), *Categorial Grammar: Logical Syntax, Semantics, and Processing*, Oxford University Press, Oxford.

The Nghia PHAM (2016), *Sentential Representations in Distributional Semantics*, Ph.D. thesis, University of Trento.

Helmut SCHWICHTENBERG (1982), Complexity of Normalization in the Pure Typed Lambda-Calculus, in *The L. E. J. Brouwer Centenary Symposium*, pp. 453–457, North-Holland.