Order-Based Inference in Natural Logic

YAROSLAV FYODOROV, *Computer Science Faculty, Technion - IIT, Haifa, Israel. E-mail: yaroslav@cs.technion.ac.il*

YOAD WINTER, *Computer Science Faculty, Technion - IIT, Haifa, Israel. E-mail: winter@cs.technion.ac.il*

NISSIM FRANCEZ, *Computer Science Faculty, Technion - IIT, Haifa, Israel. E-mail: francez@cs.technion.ac.il*

Abstract

This paper develops a version of Natural Logic – an inference system that works directly on natural language syntactic representations, with no intermediate translation to logical formulae. Following work by Sánchez, we develop a small fragment that computes semantic order relations between derivation trees in Categorial Grammar. The proposed system has the following new characteristics: (i) It uses orderings between derivation trees as purely syntactic units, derivable by a formal calculus. (ii) The system is extended for conjunctive phenomena like coordination and relative clauses. This allows a simple account of non-monotonic expressions that are reducible to conjunctions of monotonic ones. (iii) A decision procedure for provability is developed for a fragment of Natural Logic.

Keywords: inference, monotonicity, natural logic, order, semantics

1 Introduction

Model-theoretic semantic theories of natural language assume that most linguistic expressions – or even all of them – represent objects in partially ordered domains so that meanings of expressions of the same category are naturally comparable. Formal semantics treats order relations between expressions of complex categories as compositionally derived from orders between expressions of simpler categories, according to the rules of a given grammar and certain semantic properties of words. For instance, the denotation of the nominal expression *tall student* is semantically 'smaller than' the denotation of the noun *student* in every model. This simple ordering, together with the 'order reversing' meaning of the determiner *no*, is responsible for the fact that the noun phrase *no tall student* is semantically 'greater than' the noun phrase *no student* in every model. At the top level, these order statements result in a semantic ordering of natural language sentences. In an adequate semantic theory, this ordering corresponds to an intuitively valid *entailment* relation between sentences. For instance, the aforementioned order statements, together with the other elements in the sentence, are responsible for the valid conclusion of the sentence *John saw no tall student* from the premise *John saw no student*.

Fruitful as this view on inference in natural language is, its formulation within model-theoretic semantics is not immediately helpful for the design of computationally

significant inference systems. This paper aims to develop an inference system for natural language using basic insights on order relations coming directly from model-theoretic semantics, but using only proof-theoretical symbolic manipulation of natural language syntactic representations with no appeal to models. On the other hand, the direct relationships between structures and meanings in model-theoretic semantics allows us to dispense with the translation of natural language syntactic representations into a level of logical representation. Rather, inferences are computed directly from semantically annotated syntactic derivations of expressions.

This methodology means that "higher order" properties of natural language expressions become an integral part of the system. We believe that this is descriptively necessary, and may furthermore have distinct advantages over common techniques of translation into First Order Logic. To consider one example for this point, an adjective like *tall* is a *restrictive* modifier of nouns: any *tall N* is an *N*, for every noun *N*. Similarly, the degree word *very* is a restrictive modifier of *adjectives*: anything that is *very A* is *A*, for any adjective *A*. In first order logic, it is hard (or even impossible) to represent such inferential properties of modifiers like *tall* and *very*.[1] In the proposed system of natural logic, one feature, the feature 'R' (restrictiveness of modifiers), with the proper inference rules, represents this property for all modificational syntactic categories. The main challenge of Natural Logic is to find an optimal set of semantic features that allows to capture computationally as much as possible of the inference properties of natural language.

In [10] and [7], a similar conception of *Natural Logic* is used for describing certain semantic properties of natural language expressions. Sánchez proposes a mechanism that decorates categorial grammar proofs of natural language expressions using signs that indicate the *monotonicity* properties of these expressions – whether they are 'order preserving' or 'order reversing'. Sánchez shows that this monotonicity marking can be used to account for non-trivial inferences in natural language. However, while Sánchez' annotation of proof trees is rigorously defined, the derivation of the order statements that use them is not fully formalised. A similar limitation exists in more recent extensions of Sánchez' work, as in [2] and [1]. Therefore, Sánchez' system and its current descendents do not fully derive inferences between natural language sentences. The first step we take in this paper is to use Sánchez' treatment of monotonicity for defining order statements as purely syntactic relations between derivation trees in categorial grammar, using a formal calculus that we call the *order calculus*. As a logic this 'natural logic' is different from traditional conceptions of logic as a closure of a set of atomic formulas under certain operators. Here each formula is a pair of derivation trees in Categorial Grammar and therefore the whole power of CG is needed to describe formulas.

Another non-obvious question about Sánchez' system is whether this treatment of monotonicity can extend to cover inferences with non-monotonic expressions. We show that at least for a (large) subset of non-monotonic items, such a treatment is possible using a novel treatment of *coordination* in natural logic that relies on the semantic fact that items like *and* and *or* are greatest lower bound/least upper bound operators respectively with respect to the order relations in the categories they apply

[1] Although there exists a standard representation of adjectives in FOL, in which *tall man* is represented as $\mathbf{tall}'(x) \wedge \mathbf{man}'(x)$, this representation treats *tall* as if it were *intersective*, which is not necessarily the case. Furthermore, an adjective *phrase* like *very tall* has no straightforward representation in FOL to begin with.

to. This immediately captures entailments with so called *continuous* non-monotonic expressions, which as proven in [8] are equivalent to conjunctions of monotonic expressions. After introducing this system, we present an algorithm that, under certain restrictions, finds a proof of any order relation that is provable in the order calculus.

Section 2 describes some notions and techniques from natural language semantics that are crucial for the following sections. Section 3 develops a categorial inference system that formalises Sánchez' work, and extends it to treat coordination. Section 4 gives a small lexicon for the system, which is used for illustrating its application for some inferences with natural language sentences. Section 5 develops the proposed preliminary decision procedure for provability and sketches briefly elements of its correctness proof. The more detailed proof can be found in [9]. Section 6 briefly discusses previous works on direct inference in natural language, and their relations with the present enterprise. Section 7 describes a working prototype for computing inferences in natural logic, which has been implemented upon Bob Carpenter's *tlg* categorial grammar parser ([3]).

2 Semantic order relations in natural language

Semantic order relations between expressions in natural language are very common, and, as we will show in this paper, are useful for the derivation of inferences in an effective way. There are three origins for linguistic order relations on which we would like to concentrate:

1. *Construction-based* order relations are generated from specific constructions or lexical items of the language.
2. *Lexical* order relations are relations between words that appear due to their lexical meaning.
3. *Imported* order relations are deduced by inference processes on input in natural language or are explicitly provided by some external source.

As an example for order relations of the first kind consider the following entailments:

(1) a. John is a tall student ⇒ John is a student
b. John is very tall ⇒ John is tall
c. John is a very tall student ⇒ John is a student.

These entailments reflect general properties of the words *tall* and *very* when they function as modifiers, which are informally written below as order relations, where **N** and **A** are a noun and an adjective respectively.

(2) a. $tall(\mathbf{N}) \leq \mathbf{N}$
b. $very(\mathbf{A}) \leq \mathbf{A}$
c. $(very(tal))(\mathbf{N}) \leq \mathbf{N}$.

It should be pointed out that the relation in (2c), which reflects a general property of the complex expression *very tall*, logically follows from the ordering properties of the words *tall* and *very* as described in (2a)and (2b).This kind of construction-based ordering is central to the inference processes in natural logic, which we aim to capture within what we call an *Order Calculus*.

Lexical order relations reflect knowledge about the meaning of words. For instance:

(3) a. *student* $\leq$ *person*
b. *run* $\leq$ *move*
c. *father* $\leq$ *parent*

Of course, lexical relations may be more complex than order relations. For instance, the following proposition reflects the familiar connection between the transitive verb *kill* and the intransitive verb *die*. The notations $\mathbf{kill}'$ and $\mathbf{die}'$ represent the binary predicate and the unary predicate that these words denote, respectively.

(4) $\forall x[\exists y[\mathbf{kill}'(y,x)] \rightarrow \mathbf{die}'(x)]$

The properties of the logic required to express lexical relations is a subject that we will not touch here. We simply assume that lexical *order relations* are given, as part of a lexical knowledge base that is not part of the order calculus itself. Additional 'imported' order relations may be inferred from other knowledge bases. For instance, a premise stating that *all swans are white* may be inferred from some description (linguistic or another) of animal life. Linguistic expressions like *every*, *all*, *always* etc. are useful for expressing such order statements. Lexical rules on the meanings of these words derive the appropriate order statements.

Let us now review some notions and techniques from natural language semantics, which are central for the development of the order calculus in the next section. We assume that expressions in natural language are associated with *types*. The primitive types are a finite set, in which we identify a subset of *primitive partial order (PO) types* and, in turn, its subset of *primitive boolean types*. In any given model, each primitive type is associated with a *domain*. The domains associated with primitive PO types are partially ordered by a given partial order relation, while the domains associated with primitive boolean types have a boolean algebra defined on them. Non-primitive types inductively describe the functional compounds of primitive types. Standardly, Currying allows us to describe n-ary functions by unary types. A type that describes n-ary functions with a PO or boolean range is called a (non-primitive) *PO type* or *boolean type* respectively. An ordering relation on non-primitive PO domains is defined *pointwise* using the orderings of primitive PO domains. These notions are formally defined below.

Definition 2.1 (types) Let $\mathbf{T}^0$ be some finite set of primitive types. The set of types is the smallest set $\mathbf{T}$ that satisfies:

1. $\mathbf{T}^0 \subseteq \mathbf{T}$,
2. If $\tau \in \mathbf{T}$ and $\sigma \in \mathbf{T}$ then also $(\tau\sigma) \in \mathbf{T}$.

Standardly, types e (for 'entities') and t (for 'truth values') are among the primitive types.

Definition 2.2 (domains) For each primitive type $\tau \in \mathbf{T}^0$, let D_τ be the corresponding non-empty domain. Let these primitive domains be mutually disjoint. For each non-primitive type $(\tau\sigma) \in \mathbf{T} \setminus \mathbf{T}^0$, let $D_{\tau\sigma}$ be the set of functions from D_τ to D_σ.

Definition 2.3 (PO and Boolean types) Let $\mathbf{T}^0_{po} \subseteq \mathbf{T}^0$ and $\mathbf{T}^0_{bool} \subseteq \mathbf{T}^0_{po}$ be two finite sets of primitive PO types and primitive boolean types respectively, s.t. $t \in$

$\mathbf{T}^0_{bool}$. The set of PO (Boolean) types is the smallest set $\mathbf{T}_{po} \subseteq \mathbf{T}$ ($\mathbf{T}_{bool} \subseteq \mathbf{T}$) that satisfies:

1. $\mathbf{T}^0_{po} \subseteq \mathbf{T}_{po}$ ($\mathbf{T}^0_{bool} \subseteq \mathbf{T}_{bool}$),
2. If $\tau \in \mathbf{T}$ and $\sigma \in \mathbf{T}_{po}$ ($\sigma \in \mathbf{T}_{bool}$) then also $(\tau\sigma) \in \mathbf{T}_{po}$ ($(\tau\sigma) \in \mathbf{T}_{bool}$).

Definition 2.4 (pointwise partial (boolean) order) For each primitive PO (boolean) type $\tau \in \mathbf{T}^0_{po}$ ($\tau \in \mathbf{T}^0_{bool}$), let '$\leq_\tau$' be a partial (boolean) order on D_τ. For each non-primitive PO (boolean) type $(\tau\sigma) \in \mathbf{T}_{po} \setminus \mathbf{T}^0_{po}$ ($(\tau\sigma) \in \mathbf{T}_{bool} \setminus \mathbf{T}^0_{bool}$), the *pointwise partial (boolean) order* '$\leq_{\tau\sigma}$' on $D_{\tau\sigma}$ is defined by:

$x \leq_{\tau\sigma} y$ holds iff for every $z \in D_\tau$, $x(z) \leq_\sigma y(z)$ holds.

We sometimes use the notation '$x \geq y$' instead of '$y \leq x$', when convenient. Standardly, the domain D_t of truth values is the doubleton $\{0, 1\}$, for falsity and truth respectively, with the numerical partial order 'less or equal'. This boolean order imposes similar boolean orders on the boolean domains whose type "ends with t". The domain D_e of entities is arbitrarily chosen (with no partial order assumed).

After defining the elementary semantic notions with respect to order relations, we may move on to special behaviour of functions with respect to these relations, and to some order-based notions that are important for the purposes of this paper.

Definition 2.5 (restrictive function) A function $f \in D_{\tau\tau}$, where τ is a PO type, is called *restrictive* iff for every $x \in D_\tau$: $f(x) \leq x$.

Using these definitions we get an intuitive account of entailments as in (1).The function that the adjective *tall* denotes is of type $(et)(et)$ – from a nominal of type et to a nominal of type et, and we assume that it is a restrictive function. Similarly, the function that the degree modifier *very* denotes is of type $((et)(et))((et)(et))$ – from an adjectival phrase of type $(et)(et)$ to an adjectival phrase of type $(et)(et)$, and we assume that it is a restrictive function as well. Consequently, the function that the adjectival phrase *very tall* denotes is of type $(et)(et)$. By restrictiveness of *very* we get the relation $very(tall) \leq_{(et)(et)} tall$. Hence by definition of '$\leq_{(et)(et)}$', we get the relation $(very(tall))(student) \leq_{et} tall(student)$. By restrictiveness of *tall* we get: $tall(student) \leq_{et} student$. Hence, by transitivity of '$\leq_{et}$': $(very(tall))(student) \leq_{et} student$.

Of course, the notion of restrictive functions can be naturally extended to n-ary functions, and then it is sometimes said that a function of type 'ending with τ' is restrictive on one of its τ type arguments. A special kind of restrictive functions is known as *greatest lower bound* (glb) functions. Consider the definition of the binary case, which is most useful for our purposes.

Definition 2.6 (greatest lower bound) A function $f \in D_{\tau(\tau\tau)}$, where τ is a boolean type, is called *glb* iff for all $x, y, z \in D_\tau$ the following two conditions hold:

1. $(f(x))(y) \leq x$ and $(f(x))(y) \leq y$;
2. if $z \leq x$ and $z \leq y$ then $z \leq (f(x))(y)$.

The first requirement states that f is restrictive, or returns a lower bound, on both its arguments; the second requirement states that f returns a *greatest* lower bound on its arguments. A glb function is sometimes denoted by '$\wedge$' (boolean *meet*), and then '$(f(x))(y)$' is abbreviated as '$x \wedge y$'.[2]

[2] From the definition of the *glb* follows its uniqueness whenever existing.

A symmetric notion is the notion of *least upper bound (lub)* functions.

Definition 2.7 (least upper bound) A function $f \in D_{\tau(\tau\tau)}$, where τ is a boolean type, is called *lub* iff for all $x, y, z \in D_\tau$ the following two conditions hold:

1. $(f(x))(y) \geq x$ and $(f(x))(y) \geq y$;
2. if $z \geq x$ and $z \geq y$ then $z \geq (f(x))(y)$.

The first requirement states that f returns an upper bound on both its arguments; the second requirement states that f returns a *least* upper bound on its arguments. A lub function is sometimes denoted by '$\vee$' (boolean *join*), and then '$(f(x))(y)$' is abbreviated as '$x \vee y$'.

In natural language there are at least three kinds of glb functions:

1. Conjunctions. The standardly assumed meaning of conjunctions such as *dance and smile*, *Mary danced and John smiled*, and *every teacher and some student* is the glb of the meanings of the conjuncts.
2. Relative clauses. A 'subject oriented' relative clause such as *child who sneezed* (e.g. in the nominal *the child who sneezed*) is treated as a glb of the noun (*child*) denotation and the verb phrase (*sneezed*) denotation.
3. Intersective adjectives. Adjectives like *blue* and *pregnant* are often assumed to denote 'intersective functions': functions of type $(et)(et)$ that intersect their argument with an implicit argument of type et. For instance, the nominal *blue car* is synonymous with the nominal *car that is blue*, which is formed using a glb relative.

A lub function in natural language is the disjunction *or*: the standardly assumed meaning of disjunctions such as *dance or smile*, *Mary danced or John smiled*, and *every teacher or some student* is the lub of the meanings of the disjuncts.

Another useful property of functions in natural language that is expressed in terms of order relations is *monotonicity*.

Definition 2.8 (monotonicity) Let σ_1 and σ_2 be PO types. A function $f \in D_{\sigma_1\sigma_2}$ is:

upward monotone iff for all $x, y \in D_{\sigma_1}$: $x \leq_{\sigma_1} y \Rightarrow f(x) \leq_{\sigma_2} f(y)$;
downward monotone iff for all $x, y \in D_{\sigma_1}$: $x \leq_{\sigma_1} y \Rightarrow f(x) \geq_{\sigma_2} f(y)$.

For instance, natural language *determiners* like *some* and *every* are treated as functions of type $(et)((et)t)$. The determiner *some* is analysed as a function that is upward monotone on both its arguments, due to entailments as in (5).The determiner *every* is analysed as a function that is downward monotone on its first argument and upward monotone on its second argument, due to entailments as in (6).

(5) a. Some tall student ran ⇒ Some student ran
b. Some student ran ⇒ Some student moved

(6) a. Every student ran ⇒ Every tall student ran
b. Every student ran ⇒ Every student moved

Note further that a determiner like *exactly one* is neither upward nor downward monotone on any of its arguments. This is verified by the following lack of entailments.

(7) a. Exactly one tall student ran $\begin{matrix}\not\Rightarrow \\ \not\Leftarrow\end{matrix}$ Exactly one student ran

b. Exactly one student ran $\begin{matrix}\not\Rightarrow\\ \not\Leftarrow\end{matrix}$ Exactly one student moved

3 The Categorial Inference Engine

In this section we introduce the system that will be used to derive semantic order relations between syntactic representations of expressions in natural language without resorting to their meanings. A categorial calculus, a variant of the simple AB-calculus, is responsible for assigning derivation trees to expressions of natural language. The categories in these trees are decorated by some *semantic features*, which give information on the order-based semantic properties of the expressions under the given syntactic analysis. An *order calculus* is responsible for deriving order statements between such decorated derivation trees. Especially, this calculus derives order relations between derivation trees of sentences, which correspond to semantic entailment relations between readings of sentences themselves.[3]

We first define the *decorated categories* of the grammar. These categories are based on a small set of *primitive categories* and its standard extension to the set of categories in applicative categorial grammar, with the additional feature that decorated categories can have signs marking certain semantic properties (like monotonicity, restrictiveness, or glb/lub behaviour of expressions that derive them). A standard mapping from (decorated) categories to functional semantic types is defined. This is needed not only for the standard definition of a type-theoretical semantics, but furthermore for the specification of which expressions (possibly of different syntactic categories) are of the same semantic type and hence should be comparable in the order calculus. The subset of categories that is of special interest in our inference system is the set of *partial order categories* (PO categories) and *boolean categories*, those categories that correspond to PO and boolean types. Order relations are defined for this set. We follow [4] and eliminate non-PO categories from the set of decorated categories. Here and henceforth, by *types* we refer to semantic types and by *categories* to syntactic categories.

The set of (undecorated) categories is standardly defined as in applicative categorial grammar (in 'result on left' style slash format), with a mapping **type** from categories to types. We assume that the set of primitive categories includes at least two designated categories, s (for sentences) and $\overline{\mathsf{s}}$ (a category whose use will become clear as we go along).

Definition 3.1 (categories) Let $\mathbf{CAT}^0$ be a finite set of primitive categories s.t. $\{\mathsf{s}, \overline{\mathsf{s}}\} \subseteq \mathbf{CAT}^0$. Let $\mathbf{type}^0 : \mathbf{CAT}^0 \rightarrow \mathbf{T}$ be a typing function for this set s.t. $\mathbf{type}^0(\mathsf{s}) = \mathbf{type}^0(\overline{\mathsf{s}}) = t$. The set of categories is defined, together with its typing function **type** extending $\mathbf{type}^0$, as the smallest set **CAT** that satisfies:

1. $\mathbf{CAT}^0 \subseteq \mathbf{CAT}$. For any $A \in \mathbf{CAT}^0 : \mathbf{type}(A) = \mathbf{type}^0(A)$;
2. If $A \in \mathbf{CAT}$ and $B \in \mathbf{CAT}$ then $A/B \in \mathbf{CAT}$ and $A\backslash B \in \mathbf{CAT}$, and $\mathbf{type}(A/B) = \mathbf{type}(A\backslash B) = (\mathbf{type}(B)\ \mathbf{type}(A))$.

Note that the type of a primitive category need not be a primitive type.

[3] Here and henceforth, when we talk about entailment relations, we also consider cases where there are two or more premises. In the next section it will become clear how the system captures such entailments.

Definition 3.2 (PO categories) The set $\mathbf{CAT}_{po}$ of PO categories is defined by $\{A \in \mathbf{CAT} : \mathbf{type}(A) \in \mathbf{T}_{po}\}$ - the set of categories whose type is PO.

We use annotations on elements from the set of PO categories to encode three kinds of semantic properties: *monotonicity*, *restrictive modification* and *conjunction/disjunction*. The annotation uses a semantic feature $F \in \{+, -, R, C, D\}$ on the main (back)slash constructor of the category, for denoting upward (+) monotonicity, downward (−) monotonicity, restrictive modification (R), conjunction (C) and disjunction (D), respectively.[4] The annotation F on a category $A/^{\mathsf{F}}B$ that is assigned to an expression α means that α is upward/downward monotone, restrictive etc.

Monotonicity: any category A/B or $A\backslash B$ where both A and B are PO categories can be decorated by a monotonicity $+/-$ sign, for upward/downward monotonicity. For instance, assigning the decorated category $(\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n}$ to the determiner *every* specifies the (correct) assumption that this determiner is downward monotone on its nominal argument and, furthermore, that the noun phrase it generates (of the category $\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np})$) is upward monotone on its verb phrase argument.

Restrictive Modification: Any non-primitive PO category that corresponds to a type $\tau\tau$ is a category of a *semantic modifier* and can be marked by 'R' to describe the fact that its semantic function is restrictive.[5] For instance, a restrictive adverb like *yesterday*, in its role as a modifier of intransitive verbs, will be assigned the decorated category $(\mathsf{s}\backslash\mathsf{np})\backslash^{\mathsf{R}}(\mathsf{s}\backslash\mathsf{np})$. A restrictive adjective like *tall*, in its role as a nominal modifier, will be assigned the decorated category $\mathsf{n}/^{\mathsf{R}}\mathsf{n}$.

Conjunction/Disjunction: A non-primitive PO category that corresponds to a type $\tau(\tau\tau)$, where τ is a boolean type, is in fact a 'coordinator', which can be marked by a 'C'/'D' sign, for conjunction/disjunction (=glb/lub). For instance, the relative pronoun *who* will get the marking $(\mathsf{n}\backslash\mathsf{n})/^{\mathsf{C}}(\mathsf{s}\backslash\mathsf{np})$ for describing its conjunctive role in relative clauses missing a subject (e.g. as in *a child who walks*).

The set $\mathbf{CAT}_d$ of *decorated categories* contains the PO categories, possibly marked for monotonicity, conjunction/disjunction or restrictiveness. A category that is marked for conjunction or disjunction does not have any other marking.[6]

Definition 3.3 (decorated categories) The set of decorated categories is defined, together with its typing function $\mathbf{type}^d$ extending $\mathbf{type}$, as the smallest set $\mathbf{CAT}_d$ that satisfies:

1. $\mathbf{CAT}_{po} \subseteq \mathbf{CAT}_d$. For any $A \in \mathbf{CAT}_{po}$: $\mathbf{type}^d(A) = \mathbf{type}(A)$.
2. If $A \in \mathbf{CAT}_d$ and $B \in \mathbf{CAT}_d$ then $A/^{\mathsf{F}}B \in \mathbf{CAT}_d$ and $A\backslash^{\mathsf{F}}B \in \mathbf{CAT}_d$, where F is in the set $\{+, -, R, C, D, \epsilon\}$ (ϵ is the empty marking) and the following conditions hold:
   (a) $\mathbf{type}^d(A/^{\mathsf{F}}B) = \mathbf{type}^d(A\backslash^{\mathsf{F}}B) = (\mathbf{type}^d(A)\ \mathbf{type}^d(B))$;
   (b) If F='R' then $\mathbf{type}^d(A) = \mathbf{type}^d(B)$;
   (c) If F='C' or F='D' then:

[4] In fact, these properties are not mutually exclusive. For instance, a restrictive modifier may or may not be upward monotone, but a conjunction is always upward monotone and restrictive on both its arguments (see fact 3.11 below). In [9], categories are decorated by *sets* of semantic features.

[5] Primitive categories of this type are always arguments, and therefore do not function as modifiers, despite their semantic type.

[6] The restrictiveness and upward monotonicity of conjunction and the upward monotonicity of disjunction will follow from the inference rules for these features.

(i) $A = C/^{\epsilon}D$ or $A = C\backslash^{\epsilon}D$;
(ii) $\mathbf{type}^d(B) = \tau$ where τ is a boolean type, and $\mathbf{type}^d(A) = (\tau\tau)$.

Notation: We use $A/^*B$ (or $A\backslash^*B$) as a pattern matching any of the categories $A/^FB$ (or $A\backslash^FB$), where $F \in \{+, -, R, C, D, \epsilon\}$.

We define two notions of equivalence between decorated categories. *Formal* equivalence between two decorated categories means that they are derived from the same standard syntactic category and may only be different in their semantic annotations. Two decorated categories are called *semantically equivalent* if their semantic type is the same. The formal equivalence classes of categories are the standard categories of applicative categorial grammar. We assume that the syntax is insensitive to semantic annotations, and therefore only these equivalence classes are relevant for the definition of the syntactic inference rules in the grammar.[7] Semantic equivalence between categories means that denotations of expressions of these categories are semantically comparable.

Definition 3.4 (formally equivalent categories) For any two decorated categories $A, B \in \mathbf{CAT}^d$ we say that A is formally equivalent to B, and denote $A \equiv_f B$, iff:
1. A and B are primitive and $A = B$ or $\{A, B\} = \{\mathsf{s}, \overline{\mathsf{s}}\}$; or
2. $A = A_1/^*A_2$, $B = B_1/^*B_2$, $A_1 \equiv_f B_1$ and $A_2 \equiv_f B_2$; or
3. $A = A_1\backslash^*A_2$, $B = B_1\backslash^*B_2$, $A_1 \equiv_f B_1$ and $A_2 \equiv_f B_2$.

Definition 3.5 (semantically equivalent categories) For any two decorated categories $A, B \in \mathbf{CAT}^d$ we say that A is semantically equivalent to B, and denote $A \equiv_s B$, iff $\mathbf{type}^d(A) = \mathbf{type}^d(B)$.

To give an example, it is standard to assign the type e to the primitive category np (for noun phrases) and to give the type et to the primitive category n (for nouns). Consequently, the categories n and $\mathsf{s}\backslash\mathsf{np}$ (for intransitive verbs) become semantically equivalent. Under the assumption above that the grammar assigns the category $(\mathsf{n}\backslash\mathsf{n})/^C(\mathsf{s}\backslash\mathsf{np})$ to the word *who*, the calculus is therefore able to compare the derivation tree of the nominal *child who walks slowly* with the derivation trees of the verb *walks*, and to prove that the former is 'smaller than' the latter, although they are of different syntactic categories.

We use a variant of the standard applicative AB *calculus* over the decorated categories, which is responsible for derivation of syntactic analyses for given expressions. The only difference from the standard AB calculus is that here the argument category B in a functor category A/B (or $A\backslash B$) may be decorated differently than the category B' that combines with the functor. Therefore we require formal equivalence between B and B', and not simply identity.

Definition 3.6 (AB calculus) For any two decorated categories $A/^*B$ (or $A\backslash^*B$) and $B' \equiv_f B$ we have the following (back)slash elimination rules.

$$\frac{A/^*B \quad B'}{A}\ {}_{/E} \qquad \frac{B' \quad A\backslash^*B}{A}\ {}_{\backslash E}$$

[7] We do not treat here the case of negative polarity items like *any* and *ever*, which are commonly assumed to be sensitive to monotonicity marking. For an account of NPI within a version of natural logic see [2].

Notation: We use the '|' sign as a pattern matching both '/' and '\', in cases where the direction of the slash is immaterial. In such cases the patterns

$$\frac{A|^*B \quad B'}{A}\ {}_{|E} \quad \text{and} \quad \frac{B' \quad A|^*B}{A}\ {}_{|E}$$

each match both of the above elimination rules.

The AB calculus together with a *lexicon* constitute a *grammar* that is used for assigning *derivation trees* to natural language expressions. The derivation trees are the syntactic objects for which the order calculus is defined below. The definition of the lexicon and derivation trees is straightforward. We add to any lexicon the word $w_\top$ (the 'top word') of category $\overline{\mathbf{s}}$, the use of which will become clear in the sequel.

Definition 3.7 (lexicon) A lexicon is a pair $\langle \Sigma, \mathbf{cat} \rangle$, where Σ is a finite set of words s.t. $w_\top \notin \Sigma$, and **cat** is a function from $\Sigma \cup \{w_\top\}$ to non-empty finite subsets of $\mathbf{CAT}_d$ s.t. $\mathbf{cat}(w_\top) = \{\overline{\mathbf{s}}\}$.

Definition 3.8 (derivation tree) A labelled tree T is a derivation tree for a string $\sigma \in \Sigma^+$ iff: (i) The frontier of T is labelled by the elements of σ, (ii) Every leaf x in T has no brother and $\mathbf{label}(\mathbf{mother}(x)) \in \mathbf{cat}(\mathbf{label}(x))$, and (iii) The internal nodes in T (those not in its frontier) form a proof tree in the AB calculus.

The main part of the system is the *order calculus* defined below, which is the engine that derives semantic order relations between derivation trees of natural language expressions. The *formulas* manipulated by this calculus are order statements between derivation trees of formally or semantically equivalent categories. Thus, the general form of formulas in this calculus is:

> $\alpha_A \leq \beta_B$, where α_A and β_B are derivation trees of decorated categories A and B respectively such that $A \equiv_s B$.

Notational conventions: '$T_2 \geq T_1$' instead of '$T_1 \leq T_2$'; '$T_2 \equiv T_1$' instead of '$T_1 \leq T_2$ and $T_2 \leq T_1$'. This (semantic) equivalence between derivation trees should be distinguished from formal or semantic equivalence between categories as defined above. Note that all the decorated categories in $\mathbf{CAT}_d$ are based on PO categories in $\mathbf{CAT}_{po}$. Therefore, it makes sense to compare any two derivation trees that have semantically equivalent categories at their roots, as the semantic domains of all decorated categories are guaranteed to be ordered. Note further that the notation '$\leq$' is now treated as a syntactic relation between trees, and not as a semantic relation between denotations as in Section 2.

Having specified the item form in the system, we can now move on to the definition of the inference rules in the order calculus. Inference rules with no premises are order *axioms*, and they impose order statements between natural language expressions of certain constructions. Other rules derive order statements from given order statements. Two simple rules in the system, reflecting basic properties of the '$\leq$' relation (independent of derivation trees that it relates) are *Reflexivity* and *Transitivity*. In addition, there are rules that treat specific structures generated by the AB calculus. We consider here three typical structures: function application, modification and coordination.

Function application is general application of (back)slash elimination rules with no further restrictions. In this case we derive an ordering between $f(x)$ and $g(y)$ using two rules: (i) *Monotonicity* – where $f \equiv g$ is a monotonic function and an ordering is

given between x and y; and (ii) *Function replacement* – where $x \equiv y$ and an ordering is given between f and g (which do not have to be monotonic).

Modification holds in the special case of function application where the domain and the range of f are the same. A *Restrictive Modification* axiom derives then an order between x and $f(x)$. *Coordination* is a situation where a 'Curried' function F is of boolean type $\tau(\tau\tau)$. That is, F has two arguments of the same type as the result.[8] Coordination rules derive an order between $F(x, y)$ and z using orders between z and x and/or between z and y. These rules reflect the fact that conjunction/disjunction are *greatest lower bound/least upper bound* operators with respect to the '$\leq$' relation. An additional rule, combining coordination and function application, reflects our assumption that all coordinators in natural language are *pointwise operators*.

The *order calculus* embodies these assumptions on top of a grammar that induced by the AB calculus. In the following definition of this calculus, the notation Φ_X, Ψ_Y stands for a derivation trees Φ, Ψ in the AB calculus where the derived category (at the root) is X, Y respectively. An order relation is therefore of the form $\Phi_X \leq \Psi_Y$, where the categories X and Y are semantically equivalent. Derivation trees that need to be more elaborate in order relations are often enclosed by boxes, to highlight the Φ_X and Ψ_Y sides of the order relation. We also use a $A|^{\pm}B$ notation to denote a category, which is marked for either upward or downward monotonicity. For instance, the notation $\Phi_{A|^{\pm}B}$ matches both derivation trees in the AB calculus of the forms $\Phi_{A/^{+}B}$, $\Phi_{A\backslash^{+}B}$, $\Phi_{A/^{-}B}$ and $\Phi_{A\backslash^{-}B}$.

Definition 3.9 (order calculus (OC))

Reflexivity: $\dfrac{\emptyset}{\alpha \leq \alpha}$ REFL **Transitivity**: $\dfrac{\alpha \leq \beta \quad \beta \leq \gamma}{\alpha \leq \gamma}$ TRANS

Monotonicity:

$$\frac{\alpha_{B'} \leq \beta_{B''}}{\boxed{\dfrac{\gamma_{A|^{\pm}B} \quad \alpha_{B'}}{A}\ |E} \quad \begin{matrix}\leq \\ \geq\end{matrix} \quad \boxed{\dfrac{\gamma_{A|^{\pm}B} \quad \beta_{B''}}{A}\ |E}}\ \text{MON}$$

'$\leq$' and '$\geq$' for '+' and '−' respectively; $B \equiv_f B' \equiv_f B''$

Function Replacement:

$$\frac{\alpha_{A|^{*}B} \leq \beta_{C|^{*}D} \quad \gamma_{B'} \equiv \delta_{D'}}{\boxed{\dfrac{\alpha_{A|^{*}B} \quad \gamma_{B'}}{A}\ |E} \quad \leq \quad \boxed{\dfrac{\beta_{C|^{*}D} \quad \delta_{D'}}{C}\ |E}}\ \text{FR}$$

where $A \equiv_s C$, $B \equiv_s D$, $B \equiv_f B'$, $D \equiv_f D'$.

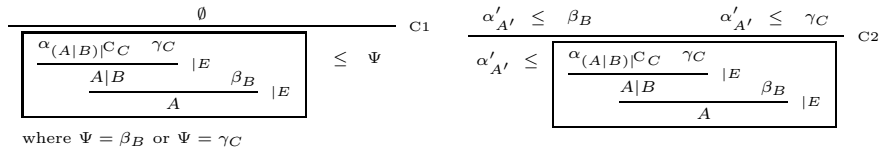
Restrictive Modification:

$$\frac{\emptyset}{\boxed{\dfrac{\alpha_{A'|^{\mathrm{R}}A} \quad \beta_{A''}}{A'}\ |E} \quad \leq \quad \beta_{A''}}\ \text{RMOD}$$

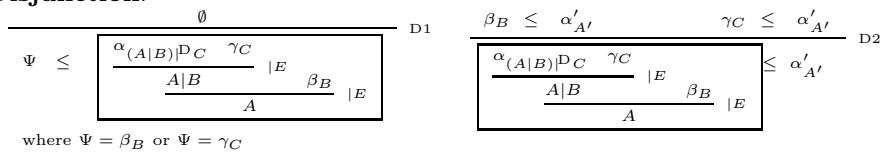
where $A \equiv_s A' \equiv_s A''$.

Conjunction:

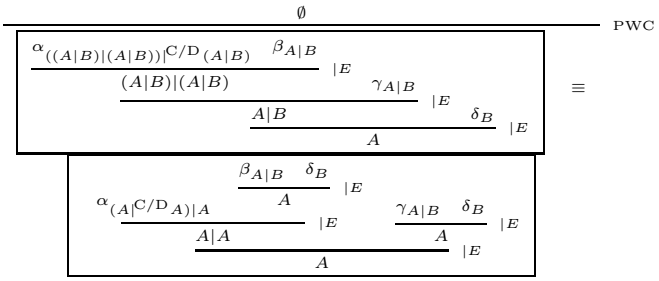
[8] In this description of coordination, items like relative pronouns are treated as coordinators, because their type can be assumed to be $(et)((et)(et))$. Here and henceforth, we refer to *coordination* in this extended, semantic, sense.



Disjunction:



Pointwise Coordination:



We define *provability* in the order calculus C as the relation that exists between a finite set $P = \{\alpha_i \leq \beta_i : i \in \{1...n\}\}$ of premise order statements and any order statement $\alpha \leq \beta$ derived from P in C using application of inference rules.

The Pointwise Coordination (PWC) rule presupposes that any conjunctive/disjunctive expression of category $((A|B)|(A|B))|^{C/D}(A|B)$ can also be analysed as category $(A|^{C/D}A)|A$ respectively, and conversely: any expression of category $(A|^{C/D}A)|A$ is also of category $((A|B)|(A|B))|^{C/D}(A|B)$, where B is a category derivable in the AB calculus from a given lexicon. The PWC rule partly follows from the Function Replacement, Conjunction and Disjunction rules. Let us abbreviate the conclusion of this axiom as stating that $(f \ \mathsf{coor}\ g)(x) \equiv f(x)\ \mathsf{coor}\ g(x)$, where coor is a cross-categorial coordinator (see the following section for our treatment of cross-categorial coordination). We observe the following fact.

Fact 3.10 The order statements $(f \wedge g)(x) \leq f(x) \wedge g(x)$ and $(f \vee g)(x) \geq f(x) \vee g(x)$ are derivable in the order calculus without the PWC rule.

A sketch of the proof of the first part of this fact is the following simplified derivation in the order calculus.

$$\dfrac{\dfrac{\dfrac{\emptyset}{f \wedge g \leq f}\ C1}{(f \wedge g)(x) \leq f(x)}\ FR \quad \dfrac{\dfrac{\emptyset}{f \wedge g \leq g}\ C1}{(f \wedge g)(x) \leq g(x)}\ FR}{(f \wedge g)(x) \ \leq \ f(x) \wedge g(x)}\ C2$$

The proof of the second part is similar.

Note further the following simple fact about this system.

Fact 3.11 (**i**) Any conjunctive category $(A|B)|^{C}C$ can be replaced by $(A|^{R}B)|^{C}C$ with no additional inferences derived by this notation. (**ii**) Any conjunctive/disjunctive category $(A|B)|^{C/D}C$ can be replaced by $(A|^{+}B)|^{+}C$ with no additional inferences derived by this notation.

In other words, this fact states that it is provable in the system that (i) any conjunction combines with its first argument to yield a restrictive modifier and that (ii) that conjunction and disjunction are both upward monotone on both arguments. This fact justifies our assumption in the definition of decorated categories that conjunctive and disjunctive categories are otherwise unmarked on their arguments.

Sketch of proof: (i) is directly by C1.

For (ii), the following inference (in abbreviated format) shows that conjunction is upward monotonic in its second argument:

$$\dfrac{\dfrac{\emptyset}{y \wedge z \leq y}\ \text{C1} \qquad \dfrac{\dfrac{\emptyset}{y \wedge z \leq z}\ \text{C1} \qquad z \leq z' \text{ (assumption)}}{y \wedge z \leq z'}\ \text{TRANS}}{y \wedge z \leq y \wedge z'}\ \text{C2}$$

The reasoning for the other cases with conjunction and disjunction is similar.

To illustrate the use of the order calculus, we give below some examples for order statements between natural language expressions that can be obtained by direct application of these rules to trees derived by a simple lexicon for the AB calculus like the one defined in the next section. For simplicity of notation we omit the derivation trees of the expressions and just add parentheses where necessary.

- *tall(student)* $\leq$ *student* (restrictive adjective modification)[9]
  yesterday(ran) $\leq$ *ran* (restrictive adverb modification)
- *some(tall student)* $\leq$ *some(student)* (upward monotonicity of *some* on first argument)
  every(student) $\leq$ *every(tall student)* (downward monotonicity of *every* on first argument)
- *(every student) (yesterday (ran))* $\leq$ *(every student) (ran)* (upward monotonicity of *every* on second argument)
  (no student) (ran) $\leq$ *(no student) (yesterday (ran))* (downward monotonicity of *no* on second argument)
- *(every student)(ran)* $\leq$ *(every tall student)(ran)* (function replacement)
- *student who ran* $\leq$ *student*,
  student who ran $\leq$ *ran* (conjunctive behaviour of relatives)
- *every student and some teacher ran* = *every student ran and some teacher ran* (pointwise coordination)

For proving the *soundness* of this system we use a standard extension of the AB calculus such that any derivable expression is assigned, besides a category A, also

[9] Non-restrictive adjectives like *fake* do not follow this pattern (e.g. a fake diamond is not a diamond), and we assume that this non-restrictiveness should be reflected in the category of the adjective, so that the modification rule does not apply to them.

a semantic value of type $\mathbf{type}^d(A)$. For lexical items such a value may come with restrictions (meaning postulates) according to the semantic feature in A.[10] Semantic values of *derivation trees for complex items* are defined inductively using the Curry-Howard isomorphism. The models for an order statement $T_1 \leq T_2$, where T_1 and T_2 are derivation trees for categories A_1, A_2 with semantic values ψ_1, ψ_2 respectively, are the models for which ψ_2 dominates ψ_1 in their PO-domain. Using these notions, it is easy to show that the above rules of the order calculus are sound relative to a standard semantics. We will not formally develop this point further here. However, in the appendix we illustrate a semantics for items in the lexicon that is provided there. While showing soundness of our system relative to standard semantics is pretty routine, a harder question is whether the system is complete. In other words: for a given lexicon of words with decorated categories, does the system derive all the valid order relations over this lexicon that semantically follow from the decorations under their standard semantics? We hypothesise that this is so, but we have not found a proof for this hypothesis yet.

4 The order calculus as an inference system for natural language

In this section we illustrate how the order calculus as defined above can be used for deriving inferences in natural language. To do that we have to add three ingredients to the system that has been described so far: a presentation of natural language sentences as order relations that can be manipulated by the order calculus, a concrete lexicon, and lexical/extra-logical order relations, which are used as additional *assumptions* of the order calculus. We describe these three parts and then give some examples for derivations in the resulting system.

4.1 Assertions as order relations

The premises and the goal in the system are assertions of natural language sentences. We regard such assertions as order statements of the form $\top \leq T$, where T is a derivation tree of a sentence and $\top$ ('top') is notation for the following special derivation tree.

$$\top = \begin{array}{c} w_\top \\ | \\ \mathsf{s} \end{array}$$

Thus, we treat assertions as a special case of order statements, between a derivation tree of a sentence and a 'top' element that corresponds to the truth value 'true'. This allows us to have a single uniform item form in the system, encoding both assertions of sentences and order relations between any linguistic expressions. Thus, the formulas of the order calculus that represent assertions of natural language sentences are of the following form.

$\top \leq \alpha_{\mathsf{s}}$, α_{s} is a derivation tree that derives s.

[10] For instance, for a lexical item ADJ of category $\mathsf{n}/^{\mathrm{R}}\mathsf{n}$ (a restrictive modifier of nouns), we assign in any model a value $\mathbf{ADJ}$ (of type $(et)(et)$) that satisfies the meaning postulate $\forall X_{et} \forall y_e[(\mathbf{ADJ}(X))(y) \rightarrow X(y)]$. See more on this strategy in the appendix.

Word(s)	Category
every	$(\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n}$
no	$(\mathsf{s}/^{-}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n}$
some	$(\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np}))/^{+}\mathsf{n}$
at least	$((\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np}))/^{+}\mathsf{n})/^{-}\mathsf{num}$
at most	$((\mathsf{s}/^{-}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n})/^{+}\mathsf{num}$
exactly	$((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))/\mathsf{n})/\mathsf{num}$
two,three,four	num
student, teacher, person	n
boys, girls, people	n
ran, walked, smiled, moved	$\mathsf{s}\backslash\mathsf{np}$
hugged, kissed, touched, admired	$(\mathsf{s}\backslash\mathsf{np})/\mathsf{np}$
tall, short, young, old	$\mathsf{n}/^{\mathrm{R}}\mathsf{n}$
very, extremely	$(\mathsf{n}/\mathsf{n})/^{\mathrm{R}}(\mathsf{n}/\mathsf{n})$
deliberately, yesterday	$(\mathsf{s}\backslash\mathsf{np})\backslash^{\mathrm{R}}(\mathsf{s}\backslash\mathsf{np})$
who	$(\mathsf{n}\backslash\mathsf{n})/^{\mathrm{C}}(\mathsf{s}\backslash\mathsf{np})$
and	$(\mathsf{s}\backslash\mathsf{s})/^{\mathrm{C}}\mathsf{s}$

TABLE 1. lexicon

4.2 Lexicon and ad hoc *syntactic rules*

We use a lexicon that will allow us to demonstrate the main properties of the system in treating monotonic and non-monotonic expressions. The set $\mathbf{T}^0$ of primitive types that we use includes the types t (for truth values), e (for entities) and v (for natural numbers). The primitive PO types in $\mathbf{T}^0_{po}$ are t and v and the only primitive boolean type is t. The set of primitive categories with their corresponding types (assigned by the $\mathbf{type}^0$ function) is:

$$\mathsf{s}{:}t \quad \overline{\mathsf{s}}{:}t \quad \mathsf{np}{:}e \quad \mathsf{n}{:}(et) \quad \mathsf{num}{:}v$$

The categories s and $\overline{\mathsf{s}}$ are for sentences, np is for noun phrases, n is for nouns and num is for numerals. A lexicon that uses these categories is given in table 1.

In order to give a proper treatment of coordination and of quantified NPs in object position, the AB calculus itself is insufficient. In order to avoid complications in the syntactic calculus that are unnecessary for our present purposes, we use *ad hoc* schemes that add elements to the lexicon given in figure 1. These schemes add more categories to lexical items of two kinds:

1. Lexical items of categories that contain the category $\mathsf{s}/^{\mathrm{F}}(\mathsf{s}\backslash\mathsf{np})$, of quantifiers in subject position. The scheme replaces this category by the corresponding category for quantifiers in object position: $(\mathsf{s}\backslash\mathsf{np})\backslash^{\mathrm{F}}((\mathsf{s}\backslash\mathsf{np})/\mathsf{np})$. For instance: we add to the determiner *every* the category $((\mathsf{s}\backslash\mathsf{np})\backslash^{+}((\mathsf{s}\backslash\mathsf{np})/\mathsf{np}))/^{-}\mathsf{n}$.
2. Coordinators of category $(\mathsf{s}\backslash\mathsf{s})/^{\mathrm{F}}\mathsf{s}$ are also given a category $(\mathsf{A}\backslash\mathsf{A})/^{\mathrm{F}}\mathsf{A}$, for each category A of a boolean type in the (finite) category closure of categories by the AB calculus in the lexicon that is given in Table 1.

The usage of these two schemes is illustrated in the appendix by the extended lexicon that they derive from the "core lexicon" in table 1. In the proofs below, we use this extended lexicon rather than the one above.

In addition, we add the following restriction on the usages of the coordinator categories:

- A derivation tree of category $(X|Y)|^{\mathrm{C/D}}Z$ appears only in derivation trees where is combines first with derivation trees Z and Y.

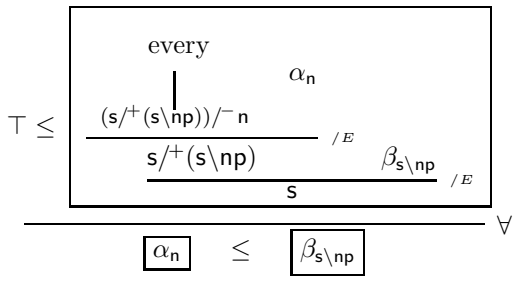


FIG. 1. postulate on *every*

This assumption rules out derivation trees where the category $(X|Y)|^{\mathrm{C}/\mathrm{D}}Z$ combines with $T|((X|Y)|Z)$ or where $(X|Y)|^{\mathrm{C}/\mathrm{D}}Z$ combines first with Z and the result $X|Y$ combines with $T|(X|Y)$. The linguistic motivations behind this assumption is that (i) the category $(X|Y)|^{\mathrm{C}/\mathrm{D}}Z$ corresponds actually to a binary function; (ii) a coordinator canot be an argument of another function.

We do not give full formalizations of this assumption and the *ad hoc* schemes above. They are used only in order to allow a simplest version of categorial grammar so to concentrate on the problems of computing order statements. A more comprehensive syntactic calculus (e.g. as in [3]) would eliminate these schemes altogether.

4.3 Lexical/extra-logical order statements

We postulate the following order statements between derivation trees of simple expressions (for simplicity, only the expressions are given, without the corresponding derivation trees in the grammar):

- *exactly* = *at least and at most*
- *two* ≤ *three* ≤ *four*
- *student* ≤ *person*, *teacher* ≤ *person*, *boys* ≤ *people*, *girls* ≤ *people*
- *ran* ≤ *moved*, *walked* ≤ *moved*
- *hugged* ≤ *touched*, *kissed* ≤ *touched*

Another *ad hoc* order relation is derived by the presence of the determiner *every* and is given in Figure 1. This rule is used to enrich the system with more order relations that are derived from assumptions in natural language (in the form '*every x y*'). Of course, a more complete treatment of quantifiers would eliminate this rule.

4.4 Examples

Example 1 [7] observes that in a premise sentence the number of upward/downward monotonic functions that take scope over a given expression determines whether a replacement of this expression by a 'bigger'/'smaller' expression is truth-preserving. For instance, consider the two sound inferences below.

(8) $\dfrac{\text{every student kissed every teacher}}{\text{every student kissed every tall teacher}}$

(9) $\dfrac{\text{every student who kissed every tall teacher smiled}}{\text{every student who kissed every teacher smiled}}$

In inference (51),the object noun phrase *every teacher* can be replaced by *every tall teacher*, preserving the truth of the premise. This is by virtue of the fact that the derivation trees for the nominals *teacher*/*tall teacher*, which are ordered according to the Modification axiom, are arguments of *every* in a position that is downward monotonic, and the whole object NP is in the scope of the noun phrase *every student* in a position that is upward monotonic. By contrast, in inference (52)the same noun phrase appears in a downward monotonic position: the nominal *student who kissed every tall teacher* is a first argument of *every*, which is marked as downward monotone on this argument. Consequently, (52) shows a reverse inference relation to (51): in (52)the sentence with the modified nominal entails the sentence with the non-modified nominal, and is not entailed by it. The opposite directions for these inferences are not provable thanks to the soundness of the system.

Let us see how the above inference system derives this fact. First, the system derives a '$\leq$' order relation between the two verb phrases *kissed every teacher* and *kissed every tall teacher*. This is formally described in figure 2. In inference (51),the verb phrase *kissed every (tall) teacher* is an argument of the (lifted) noun phrase *every student*, whose category is marked with '+' on this argument. Therefore, the system directly proves the $\leq$ ordering between the premise S_1 and the conclusion S_2 using the MON rule. Once such an order statement $S_1 \leq S_2$ is proven between sentences, the fact that the assertion of S_1 is given in the form $\top \leq S_1$ allows deriving by Transitivity $\top \leq S_2$, which is the assertion we need. By contrast, the inference in (52)is derived because the category of the relative *who* is marked by 'C', hence by fact 3.11 it behaves like a category marked by '+'. This derives the order statement *who kissed every teacher* $\leq$ *who kissed every tall teacher*. Further, Function Replacement derives *student who kissed every teacher* $\leq$ *student who kissed every tall teacher*. But using MON and the '$-$' sign on the category of *every* we now derive *every student who kissed every teacher* $\geq$ *every student who kissed every tall teacher*, and another Function Replacement step derives a $\leq$ order statement between the premise and the conclusion in (52).

Example 2 Another property of the system is that when a given premise is manipulated using the Monotonicity rule, different arguments of one function may require substitution by other arguments, and intermediate derivation trees may need to be generated in the proof process. For instance, consider the sound inferences in (54).

(10) $\dfrac{\text{no teacher ran}}{\text{no tall teacher ran yesterday}}$ $\qquad$ $\dfrac{\text{every teacher ran yesterday}}{\text{every tall teacher ran}}$

Using only one application of the Monotonicity rule we cannot derive these inferences, and we need to use intermediate sentences like *no tall teacher ran* (or *no teacher ran yesterday*) and *every teacher ran* (or *every tall teacher ran yesterday*). The conclusion is then proven using the Transitivity rule. For instance, the order statement *every teacher ran yesterday* $\leq$ *every tall teacher ran yesterday* is directly derived using RMOD and downward monotonicity of the determiner *every* on its first argument. The order statement *every tall teacher ran yesterday* $\leq$ *every tall teacher ran* is directly derived using RMOD and upward monotonicity of *every* on its second argument.

Example 3 The use of more than one premise and the conjunctive meaning of the relative is exemplified in the proof of the following inference.

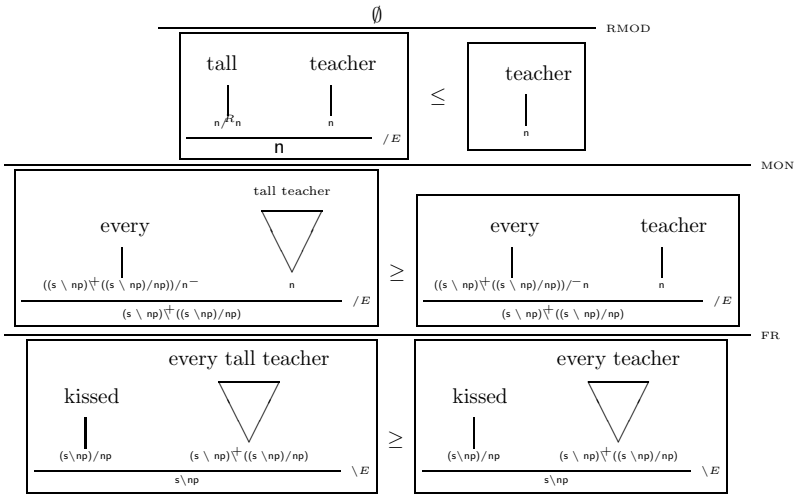


FIG. 2. kissed every teacher ≤ kissed every tall teacher

(11) $$\frac{\text{every student smiled} \quad \text{no student who smiled walked}}{\text{no student walked}}$$

The proof is based on the following steps. First we obtain *student* ≤ *smiled* using the *ad hoc* rule on simple *every* sentences. Then by Reflexivity *student* ≤ *student* and by conjunction rule C2 we get *student* ≤ *student who smiled*. Downward monotonicity of *no* and Function Replacement derive the statement *no student who smiled walked* ≤ *no student walked*. The formal process is given in figure 3.

Example 4 One of the surprising features of this system is that using the monotonicity and conjunction rules we can prove some non-trivial inferences with *non-monotonic* expressions. [8] notes that many non-monotone expressions in natural language can be expressed as conjunctions of monotonic expressions. For instance, the non-monotone item *exactly* is equivalent to a conjunction of the monotonic *at least* and *at most*. This fact makes it possible to prove inferences like the following.

(12) $$\frac{\text{exactly four tall boys walked} \quad \text{at most four boys walked}}{\text{exactly four boys walked}}$$

The proof starts by using our lexical assumption *exactly* = *at least and at most*. Consecutive steps of Pointwise Coordination and Function Replacement lead to *at least and at most four tall boys walked* = *at least four tall boys walked and at most four tall boys walked*. By C1 the derivation tree of this sentence is smaller than *at least four tall boys walked*, which in turn, by upward monotonicity of *at least four* and RMOD, is smaller than *at least four boys walked*. The conclusion ⊤ ≤ *at least four boys walked* and the second premise in (57)lead by C2 to ⊤ ≤ *at least four boys walked and at most four boys walked*, which using some steps of FR and Pointwise Coordination lead to *at least and at most four boys walked*, which as before is equal to the conclusion in (57).

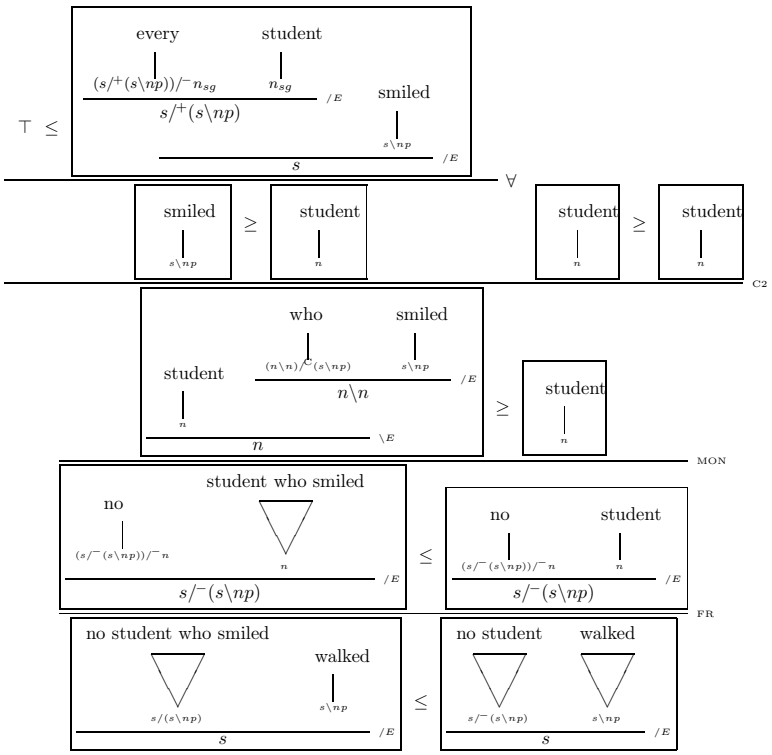


FIG. 3. conjunctive behaviour of relatives

5 Decision procedure for provability

This section introduces a decision procedure for provability in the order calculus and sketches its correctness proof, under certain limitations. Subsection 5.1 informally describes the proposed algorithm, which is further discussed in Subsection 5.2. Subsection 5.3 gives a sketch of the correctness proof, which is more formally given in [9].

5.1 Description of the algorithm

The algorithm we introduce below is a recursive function **derive** that decides whether a proof of an order statement $T_0 \leq T$ exists in the order calculus, given a (possibly empty) finite set A of n order statement premises $T_1 \leq T_1'$, $T_2 \leq T_2'$ etc. The trees $T_0, T, T_1, T_1', T_2, T_2'$ etc. are all derivation trees in the AB calculus and a given lexicon (not necessarily the one given in Section 4). This algorithm can be immediately used to search for a proof of an assertion $\top \leq S$ given a set of premises $\top \leq S_1$, $\top \leq S_2$ etc., where S, S_1, S_2 etc. are derivation trees of natural language sentences.

The *Goals* parameter keeps track of all the pairs of trees that appeared in recursive calls to the algorithm. Thus, the initial call is with $Goals = \emptyset$. A proof of an order relation $T_0 \leq T$ can be viewed as a series of order relations $T_0 = V_1 \leq ... \leq V_m = T$, where the proof of each order relation $V_j \leq V_{j+1}$ does not involve any Transitivity step $V_j \leq U \leq V_{j+1}$. In attempting to prove an order relation between T_0 and T, we

$$\dfrac{\dfrac{\dfrac{\emptyset}{\text{very(tall)} \leq \text{tall}}\ \text{RMOD}}{(\text{very(tall)})(\text{student}) \leq \text{tall(student)}}\ \text{FR} \quad \dfrac{\dfrac{\emptyset}{\text{tall(student)} \leq \text{student}}\ \text{RMOD}}{}}{(\text{very(tall)})(\text{student}) \leq \text{student}}\ \text{TRANS}$$

FIG. 4. very tall student $\leq$ student

distinguish between two cases, with complementary assumptions on the order relation that is being searched for:

1. Assume that in the order relation $T_0 = V_1 \leq ... \leq V_m = T$ that is being searched for, there is no full replacement of a tree using a premise. In other words: no $i \in \{1..n\}$ and no $j \in \{1..m-1\}$ satisfy $V_j = T_i$ and $V_{j+1} = T'_i$. In this case, the algorithm attempts the following possibilities:
   (a) The possibilities to recursively prove the order relation $T'_0 \leq T$, where T'_0 is a subtree of T_0 that is guaranteed to satisfy $T_0 \leq T'_0$ by one of the rules of the order calculus. Symmetrically: the possibilities to recursively prove the order relation $T_0 \leq T'$, where T' is a subtree of T that is guaranteed to satisfy $T' \leq T$ by one of the rules of the order calculus.
   These possibilities are searched, according to the form of the rules, in a way that will be described below.
   (b) The possibilities to prove directly the order relation $T_0 \leq T$ using the FR (function replacement) and MON (monotonicity) rules.
   This process is performed using a function that is called **subderive**.
2. Otherwise, assume that in the order relation $T_0 = V_1 \leq ... \leq V_m = T$ that is being searched for, there is a full replacement of a tree using a premise. In other words: for some $i \in \{1..n\}$ and some $j \in \{1..m-1\}$, $V_j = T_i$ and $V_{j+1} = T'_i$. Assuming that the leftmost occurrence of such a replacement is a replacement of a tree T_i by T'_i using the premise $T_i \leq T'_i$, the algorithm attempts to derive T_i from T_0 using the function **subderive**. This use of **subderive** is justified because obviously there is no full replacement of trees using a premise left to T_i. Then the algorithm recursively continues the attempt to **derive** T from T'_i.

In each recursive call to **derive** or **subderive**, we add the present goal $\langle T_0, T \rangle$ to the set of goals in the *Goals* parameter. A proof for this goal is not searched for recursively. This prevents repetition of attempts to prove the same goal, which may cause the algorithm not to terminate.

Before introducing the algorithm itself, we make a small change in the proof system of the order calculus, which does not affect provability, but which is more convenient for the description of the algorithm. Consider the valid inference in the order calculus that is given in Figure 4.

Let us denote the derivation tree for *very tall student* in this figure by T_0, and the derivation tree for *student* by T. The function category of the derivation of *very tall* is $\mathsf{n/n}$, which is not marked as a restrictive modifier. Consequently, it is impossible to generate T directly from T_0 using the RMOD rule. In addition, T is a simple word, without a function-argument structure, and therefore no direct use of the Function Replacement or Monotonicity rules is possible. The needed order relation between the

trees for *very tall* and *tall* are therefore not derived using the algorithm as sketched above. This problem is more general: it would reappear even if *very* were not defined as restrictive but is provable to be restrictive. However, there is a simple way to change the RMOD rule in a way that allows us to derive this order relation. Consider the following simple fact.

Fact 5.1 A function f of type $\tau\tau$, where τ is a poset type, is restrictive if and only if $f \leq id$, where id is the identity function of type $\tau\tau$.

We replace the RMOD axiom by the following rules:

$$\frac{\emptyset}{\alpha_{A'|^{\mathrm{R}}A} \leq id_{A'|A}}\ \text{RMOD}_1 \qquad \frac{\alpha_{A'|^{*}A} \leq id_{A'|A}}{\boxed{\dfrac{\alpha_{A'|^{*}A} \quad \beta_{A''}}{A'}\ |_E} \quad \leq \quad \beta_{A''}}\ \text{RMOD}_2$$

where $A \equiv_s A' \equiv_s A''$.

We assume that for each pair of PO categories A and A', s.t. $\mathbf{T}(A) = \mathbf{T}(A')$, that are in the (finite) category closure of the lexicon by the AB calculus, there exists a word $id_{A'|A}$. Note that this restatement of RMOD is only for the sake of the statement of the algorithm. To show that this does not change the order calculus in any significant way, it is sufficient to observe that the replacement of RMOD by RMOD_1 and RMOD_2 and the addition of the needed id elements to the lexicon do not change the provability of order relations that do not involve id. The id elements do have of course a natural semantic correlate: the identity functions in the respective domains. The proof of the order relation between T_0 and T is now as given in figure 5.

And this proof is found by the algorithm that is described below.

To describe the algorithm itself, let us first observe the following fact about the axioms C1, PWC and RMOD_1. Given a left hand tree T_l in the conclusion $T_l \leq T_r$ of these rules, the tree T_r can be defined as a function of T_l. Let us call the corresponding functions for these rules $f_{C1,r}$, $f_{PWC,r}$ and $f_{\text{RMOD}_1,r}$.[11] The letter r designates the fact that the right hand side of the conclusion is generated from the left hand side. In a similar way we can define the functions $f_{D1,l}$ and $f_{PWC,l}$ that generate the left hand sides of the conclusions of D1 and PWC from their right hand side. Observe now a similar point about the derivation rules C2 and D2. In C2, the premises can be expressed as a function of the conclusion $T_l \leq T_r$. Let us therefore denote the premises by $T_l \leq f_{C2,1}(T_r)$ and $T_l \leq f_{C2,2}(T_r)$. Similarly, in D2, we denote the premises by $f_{D2,1}(T_l) \leq T_r$ and $f_{D2,2}(T_l) \leq T_r$.

The function **subderive** uses these functions in trying to extract provable "smaller" order relations from the two derivation trees in the goal $T_0 \leq T$. The functions for the coordination rules C1, C2, D1, D2 and PWC are used in a straightforward way to search for such "smaller" order relations. Similarly for the restrictive modification rules RMOD_1 and RMOD_2. The Monotonicity and Function Replacement rules are used in a combined manner: when two function-argument structures are given, the functions are compared to each other, and the arguments are compared to each other according to monotonicity marking of the functions, to search for a proof of an order relation between the two structures. We denote $\alpha \in \mathbf{MON} \uparrow$ or $\alpha \in \mathbf{MON} \downarrow$ when

[11] The time complexity for computing these functions is constant, independently of their arguments. Note further that the function $f_{\text{RMOD}_1,r}$ needs only the categories A and A' in the derivation tree $\alpha_{A'|^{\mathrm{R}}A}$, so that $f_{\text{RMOD}_1,r}(\alpha)$ is $id_{A'|A}$.

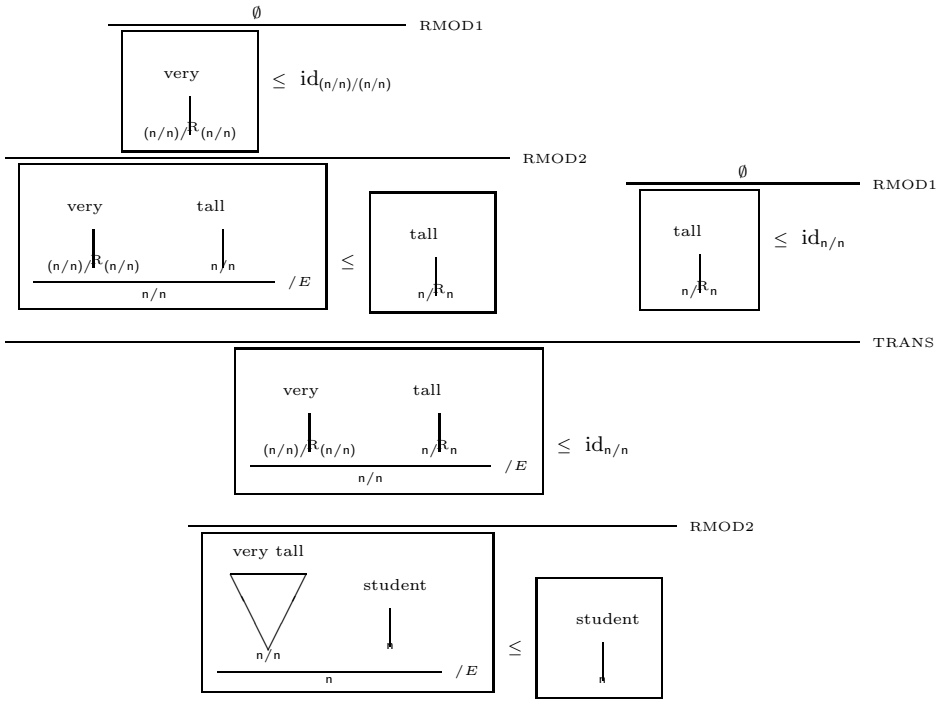


FIG. 5. very tall student ≤ student

the monotonicity of the category that is derived by a derivation tree α is marked by $+$ or $-$ respectively. The **derive** and **subderive** functions are given below. Recall that A is the set of given premises and that $n = |A|$.

derive$(T_0, T, Goals) =$

1. if $\langle T_0, T\rangle \in Goals$ then return **false**
2. $Goals' \leftarrow Goals \cup \{\langle T_0, T\rangle\}$
3. if **subderive**$(T_0, T, Goals')$ then return **true**
4. for each $i \in \{1..n\}$:
   if **subderive**$(T_0, T_i, Goals')$ and **derive**$(T_i', T, Goals')$
   then return **true**
5. return **false**

subderive$(T_0, T, Goals) =$

1. if $T_0 = T$ then return **true**

2. for each $ax \in Axioms$:

2.1 if $f_{ax,r}(T_0)$ is defined and **derive**$(f_{ax,r}(T_0), T, Goals)$ then return **true**.

2.2 if $f_{ax,l}(T)$ is defined and **derive**$(T_0, f_{ax,l}(T), Goals)$ then return **true**.

3. 3.1 if $f_{D2,1}(T_0)$ and $f_{D2,2}(T_0)$ are defined and **derive**$(f_{D2,1}(T_0), T, Goals)$ and

derive$(f_{D2,2}(T_0), T, Goals)$ then return **true**.

3.2 if $f_{C2,1}(T)$ and $f_{C2,2}(T)$ are defined and **derive**$(T_0, f_{C2,1}(T), Goals)$ and **derive**$(T_0, f_{C2,2}(T), Goals)$ then return **true**.

4. if $T_0 = \dfrac{\alpha \quad \beta}{c}$ and $T = \dfrac{\gamma \quad \delta}{c}$ and **derive**$(\alpha, \gamma, Goals)$

4.1 if $\alpha \in$ **MON** $\uparrow$ or $\gamma \in$ **MON** $\uparrow$ and **derive**$(\beta, \delta, Goals)$ return **true**.

4.2 if $\alpha \in$ **MON** $\downarrow$ or $\gamma \in$ **MON** $\downarrow$ and **derive**$(\delta, \beta, Goals)$ return **true**.

4.3 if ($\alpha \in$ **MON** $\downarrow$ and $\gamma \in$ **MON** $\uparrow$) or ($\alpha \in$ **MON** $\uparrow$ and $\gamma \in$ **MON** $\downarrow$) return **true**.

4.4 if **derive**$(\beta, \delta, Goals)$ and **derive**$(\delta, \beta, Goals)$ return **true**.

5. If $T_0 = \dfrac{\alpha_{A|A'} \quad \beta_{A''}}{A}$, where $A \equiv_s A' \equiv_s A''$ and **derive**$(\alpha, id_{A|A'}, Goals)$ and **derive**$(\beta, T', Goals)$ return **true**.

6. return **false**

Step 2 in the algorithm covers the C1, D1 and PWC axioms: for each of these axioms, the algorithm attempts to derive a an order relation between a modification of T_0 (in the case of D1 and PWC) and T, or between T_0 and a modification of T (in the case of C1 and PWC). Step 3 covers the derivation rules C2 and D2: it attempts to derive order relations between subtrees of T_0 (in the case of D2) and T, between T_0 and subtrees of T (in the case of C2). Step 4 covers the MON and FR rules, and step 5 covers the RMOD_1 and RMOD_2 rules.

5.2 Discussion

The *Goals* parameter in **derive** and **subderive** is needed in order to prevent a situation where the same goal is used in recursive steps of the algorithm, which may lead to non-termination. To see this, consider the following (artificial) example. Let the premises be the following, in a grammar where α and γ are derivation trees of category c.

Premise 1: $T_1 \leq T_1'$, where $T_1 = \alpha$ and $T_1' = \dfrac{\alpha \quad \beta}{c}\ _{|E}$

Premise 2: $T_2 \leq T_2'$, where $T_2 = \dfrac{\gamma \quad \beta}{c}\ _{|E}$ and $T_2' = \gamma$.

Now consider a call to **derive**, with parameters $T_0 = \alpha$, $T = \gamma$, and $Goals = \emptyset$. In Step 3 of **derive**, with $i = 1$, the algorithm calls **subderive**$(T_0, T_1, Goals')$ which returns **true**, since $T_0 = T_1$ $(=\alpha)$. The function **derive** will then be recursively called with $T_0 = \dfrac{\alpha \quad \beta}{c}\ _{|E}$ and $T = \gamma$. When Step 3 is performed inside this call with $i = 2$, the algorithm calls **subderive** with $T_0 = T_1'$ and $T = T_2$. Step 4.4 of **subderive** will now call **derive** with $T_0 = \alpha$ and $T_1 = \gamma$ (the left subtrees of T_1' and T_2 respectively). This pair is already in *Goals*, as it is the initial order statement we tried to prove, and consequently **derive** will return **false**. In the absence of the *Goals* parameters, the algorithm would not have terminated, because it would attempt to prove the order relation $\alpha \leq \gamma$ again and again.

It is now possible to see the problem for the method of the algorithm with the original RMOD rule. A proof as in (4) cannot be found by our method, since there is no simple direct way to derive from the goal $(very(tall))(student) \leq student$ the

needed subgoal $very(tall) \leq tall$. With the revised rules RMOD_1 and RMOD_2, step 5 in **subderive** generates the subgoal $very(tall) \leq id_{\mathsf{n}|\mathsf{n}}$, and for this subgoal, the subsequent **subderive** call generates the subgoal $very \leq id_{(\mathsf{n}|\mathsf{n})|(\mathsf{n}|\mathsf{n})}$. The latter subgoal guarantees $very(tall) \leq tall$, although it is not needed to generate this specific order relation as a subgoal in the proof search process.

5.3 Correctness of the algorithm

In this subsection we sketch the main restrictions that guarantee correctness of the above algorithm. The full correctness proof appears in [9]. The *soundness* of the algorithm, i.e. the fact that any order statement for which it returns **true** has a proof in the Order Calculus, follows quite directly from the way the algorithm was defined. *Termination* is guaranteed by two facts: first, given a goal $T_0 \leq T$ and a set of premises there is only a finite set of derivation trees that can appear as arguments of the **derive** function in the recursive search process for a proof of $T_0 \leq T$. Second, no pair of trees in this finite set appears more than once as an inner node of the search tree induced by the recursive search process in the algorithm. This situation is guaranteed by the *Goal* parameter of **derive**, in the way exemplified above.

In order for the algorithm to be *complete*, i.e. to return **true** for each order statement that has a proof in the order calculus, we have to adopt certain assumptions about the lexicon and the premises in the system. Recall Fact 3.10 about the PWC axiom, and let us call usages of PWC that are provable in the OC without PWC *trivial usages* of PWC.

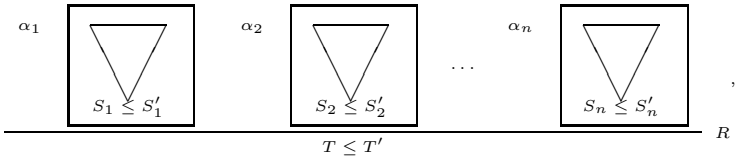
Let us define now a *normal form* of a proof in the order calculus. We will claim that under certain assumptions, a normal form exists for every proof in the Order Calculus that has no usage of PWC (or only trivial usages). Then we classify a subset of normal forms that do not include any sequences of order relations of *mixed-monotonicity*. We claim that normal forms that satisfy this restriction are arrived at by the algorithm, and speculate that the elimination of mixed monotonicity sequences (unlike the elimination of PWC!) from normal forms, is linguistically innocuous.

Let us first define three (non-disjoint) sets of inference rules and axioms in the Order Calculus:

1. *Right Productive* (R-PROD) rules are rules where the right hand tree T_r in the derived order relation $T_l \leq T_r$ is a function of the left hand tree T_l. These rules are REFL, PWC, RMOD_1, RMOD_2, C1, and D2.
2. *Left Productive* (L-PROD) rules are rules where the left hand tree T_l in the derived order relation $T_l \leq T_r$ is a function of the right hand tree T_r. These rules are REFL, PWC, C2 and D1.
3. *Subtree Replacement* (STR) rules are rules in which T_r is a simple replacement of subtrees in T_l, given order relations between those subtrees. These rules are MON and FR.

Definition 5.2 (normal form) Let $T \leq T'$ be an order statement that is derivable in the order calculus from the premises $T_1 \leq T'_1, T_2 \leq T'_2, ..., T_n \leq T'_n$. Then the *normal forms* of a proof of $T \leq T'$ is one of the following structures:

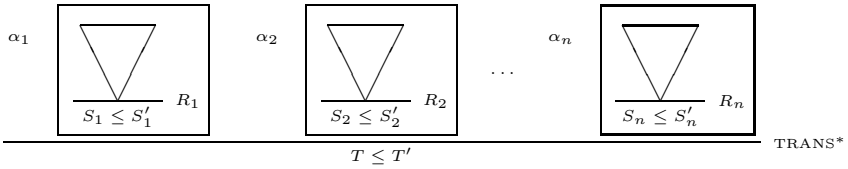
1. A *type 1* normal form is



   where $R \neq TRANS$, $n \geq 0$ and $\alpha_1, \ldots, \alpha_n$ are normal forms.

2. $\dfrac{\emptyset}{T_i \leq T'_i}\ PREM$
   where $T_i \leq T'_i$ is a premise.

3. A *type 2* normal form is



   where $n \geq 2$, $\alpha_1, ..., \alpha_n$ are type 1 normal forms, $T = S_1, S'_1 = S_2, S'_2 = S_3, ..., S'_{n-1} = S_n, S'_n = T$ and the string formed by the rules $R_1 R_2 ... R_n$ belongs to the regular language[12]:
   (PREM*, R-PROD*, STR*, L-PROD*) (PREM$^+$, R-PROD*, STR*, L-PROD*)* PREM*.

In words, a *normal form* of a proof is a structure that satisfies the following requirements:

- All occurrences of the transitivity rule are "flattened".
- The last steps of subproofs in a "flattened" transitivity step are repetitions of the pattern *premises, right productive rules, subtree replacement rules, left productive rules, premises.*
- The subproofs in each normal form are normal forms.

Next, we define the notion of a normal form that is free from sequences of mixed monotonicity.

Definition 5.3 (mixed monotonicity) Let $f_1(x_1), \ldots, f_n(x_n)$ be derivation trees, where $f(x)$ is a derivation $\dfrac{f_{A|B} \quad x_B}{A}$. We say that a part of normal form:

$$f_1(x_1) \leq f_2(x_2) \leq \ldots \leq f_n(x_n)$$

has mixed monotonicity iff one the following holds:

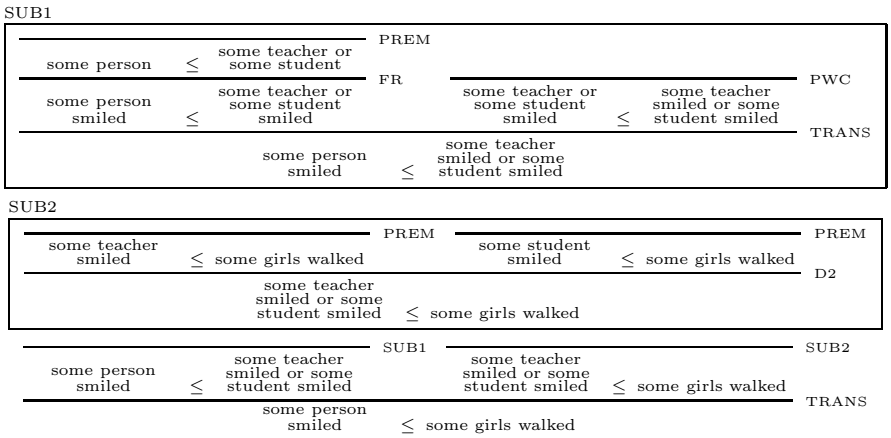
- $f_1, f_n \in \mathbf{MON}\uparrow \cup \mathbf{MON}\downarrow$ and there exist i, j, k: $1 \leq i \leq j \leq k \leq n$, s.t. $f_i, f_k \in \mathbf{MON}\uparrow, f_j \in \mathbf{MON}\downarrow$ or $f_i, f_k \in \mathbf{MON}\downarrow, f_j \in \mathbf{MON}\uparrow$.
- $f_1 \notin \mathbf{MON}\uparrow \cup \mathbf{MON}\downarrow$ or $f_n \notin \mathbf{MON}\uparrow \cup \mathbf{MON}\downarrow$ and there exist i, j: $1 \leq i \leq j \leq n$ s.t. $f_i \in \mathbf{MON}\uparrow, f_j \in \mathbf{MON}\downarrow$ or $f_i \in \mathbf{MON}\downarrow, f_j \in \mathbf{MON}\uparrow$.

[12] We use standard notation for regular expressions where E^* denotes zero or more repetitions of the regular expression E, E^+ denotes one or more repetitions of E, $E?$ — zero or one occurrences of E and $(E1|E2)$ is $E1$ or $E2$.

- $f_1, f_n \notin$ **MON** $\uparrow \cup$**MON** $\downarrow$ and there exists i: $1 < i < n$, s.t. $f_i \in$ **MON** $\uparrow \cup$**MON** $\downarrow$.

In [9] it is shown that under some syntactically plausible assumptions (see below) on the usage of coordination categories and restrictive modification, the algorithm discovers every proof that is PWC-free, provided its normal form is free from sequences of mixed monotonicity. At present we are not aware of any linguistic implication of this last assumption, but this is an open question.

Proofs that are not PWC-free Unlike the restriction on mixed monotonicity, the requirement that the proof does not involve a (non-trivial) usage of the PWC rule is of course a limitation of the algorithm. One example for a proof that the algorithm cannot find is the proof of the inference (57),described in Section 4.4. A simpler example, which illustrates better the problem of the proposed algorithm to cope with non-trivial usages of PWC, is the following proof (in simplified format).



The problem for the algorithm to find the proof is because in the attempt to derive sentences that are 'less or equal' than the sentence *some girls walked*, no attempt is made to use the disjunction *some teacher smiled or some student smiled*. Further, the algorithm does not attempt to derive *some teacher or some student smiled* from *some person smiled*. Whether or not there is a correct decision procedure for provability in the Order Calculus including (non-trivial usages of) PWC is unknown to us.

In the rest of this section, we discuss syntactic assumptions that are needed on coordination and restrictive modification.

Assumption on coordination We have already assumed above that a derivation tree α of category $A|A$ for a "half-coordinated" expression *coor X* combines only with derivation trees of category A (and not of category $B|(A|A)$). This is a reflection of the fact that coordination is an n-ary syntactic function (with $n \geq 2$), and hence Currying as in the categorial fragment above is not a satisfying syntactic treatment of this phenomenon. In addition, we now assume that no premise is an order relation $\alpha \leq \beta$ or $\beta \leq \alpha$, for such a derivation tree α of a "half coordination". Given this, it is provable that every proof using FR of an order relation that involves a coordination α *coor*^{0} β on its left side is of the following form (*coor*^{0} and *coor*^{1} are coordinators):

$$\dfrac{\dfrac{\alpha_c \quad \dfrac{coor^0_{c\backslash c/c} \quad \beta_c}{c\backslash c}\,{\backslash E}}{c}\,{/E} \leq \dfrac{\dfrac{\alpha'_c \quad \dfrac{coor^1_{c\backslash c/c} \quad \beta'_c}{c\backslash c}\,{\backslash E}}{c}\,{/E}}{\cdots}\,\text{FR}$$

where $\alpha \equiv \alpha'$, $\beta \equiv \beta'$ and $coor^0 \leq coor^1$. A similar fact holds when we assume that a coordination appears on the right side of the proven order relation. Assume now that in the process of mapping a proof to a normal form we get a structure of the following form:

$$\dfrac{\dfrac{\dfrac{...}{\alpha \leq \alpha \vee \beta}\ D1 \quad \dfrac{...}{\alpha \vee \beta \leq \alpha' \vee \beta'}\ FR}{\gamma \leq \delta} \quad ...}{} \tag{13}$$

As we said above, in the FR step it must be the case that $\alpha \equiv \alpha'$ and $\beta \equiv \beta'$. Taking into account that $\beta \equiv \beta'$ is equivalent to $\beta \leq \beta'$ and $\beta \geq \beta'$, we can replace (77)by the following, smaller, form:

$$\dfrac{\dfrac{...}{\alpha \leq \alpha'} \quad \dfrac{\emptyset}{\alpha' \leq \alpha' \vee \beta'}\ D1}{\gamma \leq \delta}$$

This reduction of the size of the proof allows us to rely on an induction assumption that each proof of a smaller size than the original proof that ends in a derivation step as in (77)has a normal form.

A simpler assumption on coordinator categories could be that they are only lexical. Thus, we could assume that the C/D marking for conjunction and disjunction appears only in categories of the form $(X|Y)|^{\mathrm{C/D}}Z$. This assumption will also be linguistically plausible: we do not know any need for a category $X|Y$, where X is a coordinator category.

Assumption on restrictive modification In order for a normal form to exist, we assume that no restrictive modifier is created from application of a lexical category to other categories. That is, no lexical category is of the form $((...|^{\mathrm{R}}...)|A)$. Suppose for instance that a derivation tree $f(x)$ has a category that is marked by R. Then the following is a proof of $g(x) \leq id$, but it is easy to see that there is no normal proof for this order relation.

$$\dfrac{\dfrac{g \leq f}{g(x) \leq f(x)}\ \mathrm{FR} \quad \dfrac{\emptyset}{f(x) \leq id}\ \mathrm{RMOD_1}}{g(x) \leq id}\ \mathrm{TRANS} \tag{14}$$

There is no linguistic motivation to rule out lexical items with categories of the form $((...|^{\mathrm{R}}...)|A)$. However, the only example we know of a lexical item that violates this rule is prepositions, and we expect prepositions to be comparable to other prepositions only. Since all the prepositions are restrictive on their second argument, we expect that they are marked for restrictiveness in a consistent manner, in which case there exists an alternative proof for (78)directly using the $\mathrm{RMOD_1}$ rule:

$$\dfrac{\emptyset}{[g(x)]^R \leq id_{A|A}}\ \mathrm{RMOD_1}$$

Therefore, we will require that no restrictive modifier is created by function application, taking into account that: (a) prepositions violate this requirement, (b) this violation does not affect the possibility to normalise the proof.

6 Notes on previous work

There are many works that deal with formal inference in natural language, and we will not attempt to give here a comprehensive overview. Here we only mention two works that concentrate on direct parallelism between natural language structures and formal proofs.

Purdy proposes in [6] a sound and complete inference system that is based on operations with n-ary relations. The expressive power of this system lies between the predicate calculus without identity and the predicate calculus with identity. Purdy then shows how sentences in a simple fragment of English can be easily translated into this formalism. It is not clear to us, however, to what extent the structure of Purdy's fragment resembles the structure of English. Most notably, the English noun phrase is missing from this fragment. This is because the analysis of structures of the form *Determiner - Noun - N-ary Predicate* requires a representation for each of the three elements in Purdy's system. If the determiner is to combine with the noun first, as in most English grammars, then Purdy's system would require translation procedures that are not simpler than those needed for the standard predicate calculus. Consequently, it is hard to see how Purdy's system can deal with common phenomena like NP coordination in an elegant way.

In [5] McAllester and Givan propose an inference system that is sound and complete, and moreover decidable in polynomial time. The system is based on so-called *class expressions* – expressions that denote sets. A determiner like *every* can combine with two class expressions to form an atomic formula. Alternatively, a determiner, a class expression and a binary relation can form together another class expression. These operations are quite parallel to natural language structures and furthermore, the time complexity of the system is rather low. However, the proposed logical system is pretty weak. For instance, since negation operates only on atomic formulae, a determiner like *no* can be handled only when it appears in subject position (e.g. *no student smiled*) but not in object position (e.g. *John likes no student*). In addition, the inference rules in the system are specialised for deduction in the proposed fragment of the predicate calculus, and it seems hard to extend them for richer constructions.

These limitations of inference systems for natural languagemotivate our study of "natural logic", logical formalism with better correspondence to natural language syntax, and with inference rules that are more directly related to model-theoretic semantics of natural language. Such systems may be applicable for a larger variety of fragments. The price to be paid for the larger syntactic flexibility is that completeness results are harder to obtain. Indeed, as mentioned above, whether the system that was defined in this paper is complete or not is unknown to us at the moment.

7 An implementation

The decision procedure for provability that was described in Section 5 was used to implement a working demo of an inference system for natural language. The system consists of four major parts: a parser for Categorial Grammar, a proof engine for the Order Calculus, a lexicon and a user interface.

As defined in Section 4, items of the Order Calculus are order statements between derivation trees in Categorial Grammar. Therefore, the input to the inference system

is a set of premises that are specified as a list of pairs of natural language expressions, each pair describing an order statement between the corresponding derivation trees, and an additional pair of natural language expressions, providing as a goal order statement that should be proved by the system. Consequently, all the natural language expressions that are fed into the system are first parsed by the CG parser. If the parser fails to parse one of the NL expressions, the inference process stops and a negative response is returned.[13] If all the expressions are accepted by the CG parser, the OC prover either shows a proof for the goal order statement or a negative reaction is given. When an order statement is provable in the OC, there usually exist a number of proofs, with different possible orders for applying the rules, with the different number of the uses of REFL rule or even with altogether different structures (e.g. one proof that uses PWC rule and another that does not). The system returns some of the proofs sorted by the order of their detection. Usually, these are the shortest proofs.

The CG parser is responsible for the creation of derivation trees in CG for natural language expressions. The parser that is used in the system is an adaptation of the *tlg* prover written by Bob Carpenter ([3]). The *tlg* prover is a bottom-up chart parser for categorial grammar using hypothetical reasoning as a deduction system for CG. In addition to the derivation trees it produces lambda-terms representing the semantics of NL expressions. Since we use a much simpler categorial grammar and we do not make use of semantic representations, we have changed the parser in order to adapt it for our purposes.

The lexicon that is used in the system is a superset of the lexicon defined in Table 1. However, there are some deviations from the way the lexicon is defined in Section 4 in the treatment of noun phrases and coordination. Instead of assigning each noun phrase a number of categories, which are used in different positions inside the sentence, noun phrases are described by one category, corresponding to the subject position, from which the CG parser can deduce new categories, corresponding to object position or to the position of the noun phrase in prepositional phrases. The *Order Calculus* was extended to treat such transformations correctly. For more details on this extension see [9]. Also coordination is treated not by lexical rules, but by a unification scheme in PROLOG. Also this procedure is described in more detail in [9].

The order calculus prover is a pretty straightforward implementation in Prolog of the decision procedure that was defined in Section 5, except for a number of technical details. While in the decision procedure the premises of the proof are treated as global parameters, in the Prolog implementation the premises are passed to the **derive** and the **subderive** predicates explicitly. A more important change is that the prover, instead of merely deciding about the provability or non-provability of an order statement, returns the proof of the order statement whenever existing. Since every step of the algorithm corresponds to a usage of a specific rule of the OC, a proof in the OC can be retrieved from the call tree of the algorithm.

The system is available either as a stand-alone Prolog application with a minimalistic interface to the prover or as a CGI application installed on the web-server through a more complex web-based interface. The web-based user interface uses HTML/JavaScript as a front-end and CGI/Prolog as a back-end. Figures 6 and 7

[13] For inference with ambiguity we adopt a liberal approach: **true** is returned if there exists an assignment of readings to all the ambiguous expressions, s.t. the desired inference can be proven.

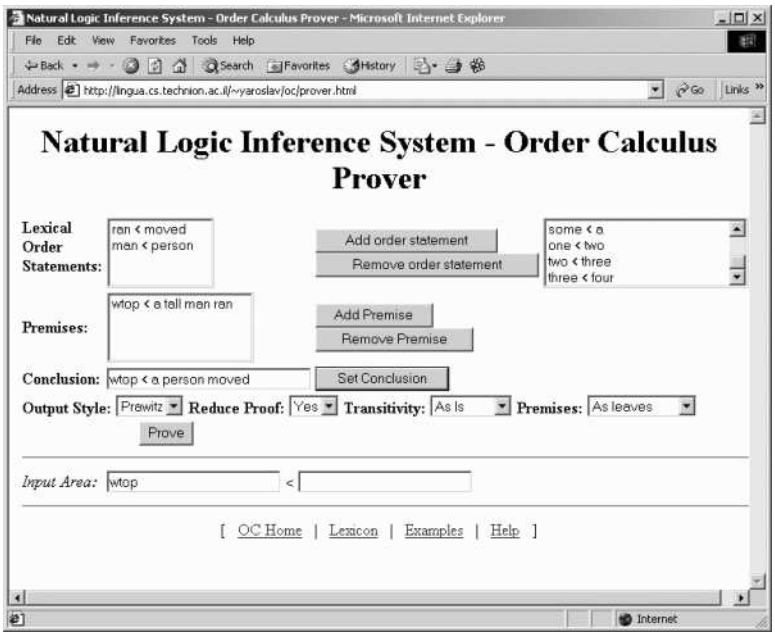


FIG. 6. A screenshot of the web-based GUI

show an example of the web-based GUI and an example of the system's output.

The implementation was done using SICStus Prolog. However, the use of features specific to SICStus was kept to the minimum and they are mainly restricted to the CGI-Prolog interface, so it should be possible to adapt the stand-alone application of the prover to any other ISO compliant Prolog implementation.

The Prolog implementation was successfully ran with SICStus Prolog 3.8.5 under RedHat Linux 2.2.16-3smp running on i686 processor and SICStus Prolog 3.8.3 under SunOs 5.8 running on sun4u processor. The web-server we have used was in both cases Apache web-server. The system is available online at `http://lingua.cs.technion.ac.il/~yaroslav/oc/`.

8 Conclusions

The design of a proof system for natural language is of course a huge task, most of whose limits are presently still unknown. We believe that in order to explore these limits it is helpful to attempt directions that use as many insights as possible from logic, computer science and linguistics. In this paper we tried to show that an attempt to use natural language syntax together with principles from higher order model-theoretic semantics may be worthwhile. Much work is left to be done on extending the system, exploring possible completeness results and improving the decision procedure for provability, which is presently of (at least) exponential complexity, and has some non-trivial restrictions on the input grammar and derivable order relations. On the other hand, the semantic and syntactic flexibility of the proposed inference system, together with its conceptual simplicity, suggest that such an enterprise could be highly

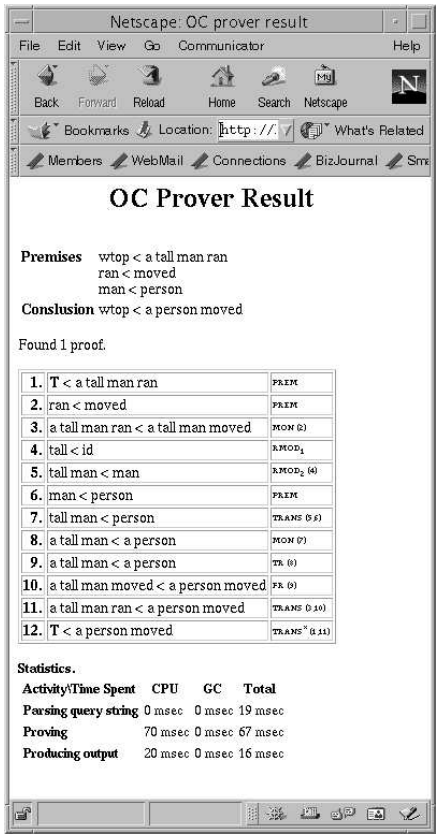


FIG. 7. A proof shown in Fitch presentation style

rewarding.

A Extended lexicon

Table 2 includes part of the extended lexicon that is derived for the lexicon in table 1.[14] Some semantic values are given in the table using their description in Montague's IL. Semantic values that are not given in the table are specified below.

1. *Non-logical Constants* (‡): Nouns like *student, boys* are assumed to denote arbitrary semantic values of type et.
2. *Restrictive modifiers* denote values of type $\tau\tau$ with the appropriate meaning postulate: for M is a value of type $\tau\tau$, the meaning postulate is:
   - For $\tau = et$ (§): $\forall X_{et} \forall y_e[(M(X))(y) \rightarrow X(y)]$.
   - For $\tau = (et)(et)$ (¶): $\forall R_{(et)(et)} \forall X_{et} \forall y_e[((M(R))(X))(y) \rightarrow (R(X))(y)]$.
3. For the *numerals two, three* etc. (†) we assume semantic values that are the corresponding natural numbers, with standard numerical order. The items *at least, at most* and *exactly* (∗) are accordingly defined as functions from natural numbers to determiners (of type $(et)((et)t)$. For example:

[14] We do not include here lexical items that are generated but are unnecessary for proofs throughout this paper.

the semantic value for *at least* maps any natural number n to the determiner D of type $(et)((et)t)$ that maps any set object A_{et} to the generalised quantifier $\{B_{et} : |A \cap B| \leq n\}$.

Acknowledgments
The parts of the second and third authors were partly supported by the fund for the promotion of research at the Technion, research no. 120-042, and by a BSF grant "Extensions and Implementations of Natural Logic". Part of the research by the second author was curried out during a stay at the Utrecht University, which was supported by an NWO grant no. B30-541. We are grateful to Johan van Benthem, Raffaella Bernardi, Ed Keenan, Michael Moortgat, Rani Nelken, Ian Pratt-Hartmann and the participants of ICOS-2 for their remarks on this work.

References

[1] Carlos Areces and Raffaella Bernardi. Polarity in the base logic. Unpublished ms., Utrecht University, 2000.

[2] R. Bernardi. Monotonic reasoning from a proof-theoretical perspective. In *Proceedings of Formal Grammar*, 1999.

[3] B. Carpenter. *Type-Logical Semantics*. MIT Press, Cambridge, Massachusetts, 1997.

[4] E. Keenan and L. Faltz. *Boolean Semantics for Natural Language*. D. Reidel, Dordrecht, 1985.

[5] D. A. McAllester and R. Givan. Natural language syntax and first-order inference. *Artificial Intelligence*, 56:1–20, 1992.

[6] W. C. Purdy. A logic for natural language. *Notre Dame Journal of Formal Logic*, 32:409–425, 1991.

[7] V. Sánchez. *Studies on Natural Logic and Categorial Grammar*. PhD thesis, University of Amsterdam, 1991.

[8] E. Thijsse. On some proposed universals of natural language. In A. ter Meulen, editor, *Studies in Modeltheoretic Semantics*. Foris, Dordrecht, 1983.

[9] Yaroslav Fyodorov. Implementing and Extending Natural Logic. MSc thesis, Technion - IIT, 2002. `http://www.cs.technion.ac.il/~yaroslav/thesis`

[10] Johan van Benthem. Meaning: interpretation and inference. *Synthese*, 73:451–470, 1987.

[11] van Benthem, J. *Language in Action: categories, lambdas and dynamic logic*. North-Holland, Amsterdam. 1991.

Received 15 January 2001.

Word(s)	Category	Semantics
every *(subject position)*	$(\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n}$	$\lambda P_{et}.\lambda Q_{et}.\forall x[P(x) \rightarrow Q(x)]$
every *(object position)*	$((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))\backslash^{+}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n}$	$\lambda P_{et}.\lambda Q_{e(et)}.\lambda y_e.\forall x[P(x) \rightarrow Q(x)(y)]$
no *(subject position)*	$(\mathsf{s}/^{-}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n}$	$\lambda P_{et}.\lambda Q_{et}.\forall x[P(x) \rightarrow \neg Q(x)]$
no *(object position)*	$((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))\backslash^{-}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n}$	$\lambda P_{et}.\lambda Q_{e(et)}.\lambda y_e.\forall x[P(x) \rightarrow \neg Q(x)(y)]$
some *(subject position)*	$(\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np}))/^{+}\mathsf{n}$	$\lambda P_{et}.\lambda Q_{et}.\exists x[P(x) \wedge Q(x)]$
some *(object position)*	$((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))\backslash^{+}(\mathsf{s}\backslash\mathsf{np}))/^{+}\mathsf{n}$	$\lambda P_{et}.\lambda Q_{e(et)}.\lambda y_e.\exists x[P(x) \wedge Q(x)(y)]$
at least *(subject position)*	$((\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np}))/^{+}\mathsf{n})/^{-}\mathsf{num}$	*
at least *(object position)*	$(((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))\backslash^{+}(\mathsf{s}\backslash\mathsf{np}))/^{+}\mathsf{n})/^{-}\mathsf{num}$	*
at most *(subject position)*	$((\mathsf{s}/^{-}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n})/^{+}\mathsf{num}$	*
at most *(object position)*	$(((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))\backslash^{-}(\mathsf{s}\backslash\mathsf{np}))/^{-}\mathsf{n})/^{+}\mathsf{num}$	*
exactly *(subject position)*	$((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))/\mathsf{n})/\mathsf{num}$	*
exactly *(object position)*	$(((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))\backslash(\mathsf{s}\backslash\mathsf{np}))/\mathsf{n})/\mathsf{num}$	*
two,three,four	num	†
student, teacher, person	n	‡
boys, girls, people	n	‡
ran, walked, smiled, moved	$\mathsf{s}\backslash\mathsf{np}$	‡
hugged, kissed, touched, admired	$(\mathsf{s}\backslash\mathsf{np})/\mathsf{np}$	‡
tall, short, young, old	$\mathsf{n}/^{\mathrm{R}}\mathsf{n}$	§
very, extremely	$(\mathsf{n}/\mathsf{n})/^{\mathrm{R}}(\mathsf{n}/\mathsf{n})$	¶
deliberately, yesterday	$(\mathsf{s}\backslash\mathsf{np})\backslash^{\mathrm{R}}(\mathsf{s}\backslash\mathsf{np})$	§
who	$(\mathsf{n}\backslash\mathsf{n})/^{\mathrm{C}}(\mathsf{s}\backslash\mathsf{np})$	$\lambda P_{et}.\lambda Q_{et}.\lambda x_e.P(x) \wedge Q(x)$
and *(sentences)*	$(\mathsf{s}\backslash\mathsf{s})/^{\mathrm{C}}\mathsf{s}$	$\lambda P_t.\lambda Q_t.P \wedge Q$
and *(NP in subject position)*	$((\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np}))\backslash(\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np})))/^{\mathrm{C}}(\mathsf{s}/^{+}(\mathsf{s}\backslash\mathsf{np}))$	$\lambda P_{(et)t}.\lambda Q_{(et)t}.\lambda R_{et}.P(R) \wedge Q(R)$
and *(NP in object position)*	$(((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))\backslash(\mathsf{s}\backslash\mathsf{np}))\backslash((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))\backslash(\mathsf{s}\backslash\mathsf{np})))/^{\mathrm{C}}$ $((\mathsf{s}/(\mathsf{s}\backslash\mathsf{np}))\backslash(\mathsf{s}\backslash\mathsf{np}))$	$\lambda P_{(e(et))(et)}.\lambda Q_{(e(et))(et)}.\lambda R_{e(et)}.$ $\lambda x_e.P(R)(x) \wedge Q(R)(x)$
and *(nouns)*	$(\mathsf{n}\backslash\mathsf{n})/^{\mathrm{C}}\mathsf{n}$	$\lambda P_{et}.\lambda Q_{et}.\lambda x_e.P(x) \wedge Q(x)$
and *(verb phrases)*	$((\mathsf{s}\backslash\mathsf{np})\backslash(\mathsf{s}\backslash\mathsf{np}))/^{\mathrm{C}}(\mathsf{s}\backslash\mathsf{np})$	$\lambda P_{et}.\lambda Q_{et}.\lambda x_e.P(x) \wedge Q(x)$
and *(adjectives)*	$((\mathsf{n}/\mathsf{n})\backslash(\mathsf{n}/\mathsf{n}))/^{\mathrm{C}}(\mathsf{n}/\mathsf{n})$	$\lambda P_{(et)(et)}.\lambda Q_{(et)(et)}.\lambda R_{et}.\lambda x_e.$ $P(R)(x) \wedge Q(R)(x)$
$w_{\top}$	$\overline{\mathsf{s}}$	1_t

TABLE 2. Extended lexicon with semantics