Thomas F. Icard, III

Inclusion and Exclusion in Natural Language

Abstract. We present a formal system for reasoning about inclusion and exclusion in natural language, following work by MacCartney and Manning. In particular, we show that an extension of the Monotonicity Calculus, augmented by six new type markings, is sufficient to derive novel inferences beyond monotonicity reasoning, and moreover gives rise to an interesting logic of its own. We prove soundness of the resulting calculus and discuss further logical and linguistic issues, including a new connection to the classes of weak, strong, and superstrong negative polarity items.

Keywords: Surface reasoning, Logic and grammar, Exclusion relations, Polarity.

Introduction and overview

One way of determining whether some sentence of natural language logically follows from another is to translate the sentences into some formalism, either in logical symbols or directly in terms of models, and decide whether the one representation follows from the other in the given logic. At least since Frege, semanticists have been interested in giving logical interpretations of natural language expressions so that the meanings, and in particular the inferential patterns, of sentences could be predicted from the meanings of the words that compose them. To the extent that semanticists are interested in closely approximating a full model theoretic interpretation of a given sentence, we might say that formal semantics has been concerned with *deep* aspects of meaning and inference. In contrast, yet quite complementary to this endeavor, a number of researchers have investigated formalisms for studying natural language entailment that stay closer to the *surface*, and in one way or another derive inferences without full interpretation into a logical formalism. This tradition, which has gone under the heading of *natural logic*, has ranged from proof systems operating directly over restricted, bare fragments of natural language ([11, 14]), to reasoning systems that operate over syntactic parse trees ([2, 4, 10, 15]). One of the guiding ideas of this work, particularly in the latter group, is that logic and grammar are closely

Special issue: Logic and Natural Language
Edited by **Nissim Francez and Ian Pratt-Hartmann**

related, and that many entailment patterns of interest can be extracted from purely syntactic information.

Perhaps the most influential and well known instantiation of the natural logic program is the so called *Monotonicity Calculus*, appearing originally in [2] and [13]. Starting with a very simple underlying syntax based on categorial grammar, the idea is to mark different expressions (or better, expression types) according to their *monotonicity* features, an idea that can be traced directly back to C.S. Pierce, and indirectly all the way to the Middle Ages (see [13], Ch. 2). For example, the quantifier *every* is *antitone* in its first argument and *monotone* in its second argument. This means that, holding the second argument constant, we can replace expressions appearing as first argument with expressions of a smaller extension *salva veritate*, e.g. *Every country is responsible* entails *Every large country is responsible*; and likewise, holding the first argument constant, we can replace expressions appearing as second argument with expressions of a larger extension, e.g. *Every vote is counted* entails *Every vote is acknowledged.* Other quantifiers have different monotonicity properties, e.g. *no* is antitone in both arguments, while *some* is monotone in both arguments. Indeed, it is possible to study the monotonicity properties of quantifiers whose symbolization would require languages stronger than that of first-order logic. For instance, while *few* and *most* are not definable by any formula in first-order logic [1], it is easy to see that *most* is monotone in its second argument, while *few* is antitone.

The main objective of the Monotonicity Calculus is to define algorithms assigning monotonicity markings to expressions of arbitrary type, at arbitrary places in a given expression, so as to facilitate inferences that may depend on several embeddings of quantifiers or other expressions that interact in complex ways with polarity. For instance, in order to show the following inference is valid:

$$\textit{Few cyclists see every car} \quad \Longrightarrow \quad \textit{Few cyclists see every vehicle}$$

we have to know how antitone contexts (the first argument of *every*) behave under the scope of another antitone context (the second argument of *few*).

The Monotonicity Calculus is therefore concerned with the relation of *inclusion* between expressions of the same given type. Recent work by MacCartney and Manning has proposed an extension of the Monotonicity Calculus to deal with relations of *exclusion* in addition [10, 8]. One of their basic insights is that quantifiers and other expressions project exclusion relations in a systematic way, very much analogous to projection of inclusion relations. For example, from the fact that the set of six-foot-tall things is excluded from the set of four-foot-tall things, we can conclude that *Everyone* *is six-feet-tall* is also excluded by *Everyone is four-feet-tall*, while *Someone is six-feet-tall* and *Someone is four-feet-tall* are perfectly compatible. In this more general setting, inclusion is only a special case of the possible relations that can obtain between expressions of the same type. Notably, MacCartney and Manning have observed that new instances of inclusion can result, most interestingly at the sentence level, by considering this wider class of relations, so that the extra relations could be seen merely as a subsidiary means of broadening the range of entailments between sentences. A number of examples will be given in what follows.

The aim of this work has been to improve and supplement natural language processing techniques for recognizing textual entailment, and their NatLog System has been successful in this endeavor. When hybridized with the Stanford RTE System, the result was a gain of 4% accuracy (see [9] for discussion). Given these practical interests, some of the formal and logical foundations are left unspecified. In particular it is unclear exactly how a calculus based on inclusion and exclusion together would relate to, or build on, the Monotonicity Calculus, not to mention whether the resulting calculus is sound, confluent, and so on.

In this paper we propose a formalization of the main logical ideas from [8, 10] in the style of the Monotonicity Calculus. The latter is based on two classes of functions, monotone and antitone, and posits two monotonicity signatures, for contexts of positive and negative polarity. What we shall call the *Projectivity Calculus*, designed to handle the additional relations of exclusion, requires only four more classes of functions and six additional *projectivity signatures*. The logic of these signatures turns out to be of interest in itself, and allows us to prove results analogous to those for the Monotonicity Calculus and other natural logics (Sections 1 and 2). In particular we can prove soundness of our marking algorithm with respect to standard denotational semantics. As a consequence, we can define a calculus $\mathcal{C}$ of relations between terms in our formal language (Section 3), which we incidentally observe does not possess a Church-Rosser property. However, $\mathcal{C}$ is sound and is strong enough to capture a number of interesting inferences, which in principle go beyond those afforded by monotonicity reasoning alone. Our main example in Section 3.2 will be to show that the following inference is a consequence of $\mathcal{C}$, along with assumptions about *every* and *not every*:[1]

[1]One might object that *every* should not carry existential import, and that therefore this inference should be invalid. Importantly, this is not a feature of the logical framework to be presented, but simply a choice about the meaning of *every*. In the NatLog system universal quantifiers (*each*, *every*, *all*) are assumed to carry existential import, and we follow suit. But this is a free parameter, not forced by the underlying calculus.

Every job that involves a giant squid is dangerous $\Longrightarrow$
Not every job that involves a cephalopod is safe

Inferences like this one depend on projecting exclusion relations between terms, as well as joining different exclusion relations together.

The Monotonicity Calculus is generally thought to work at a level close to syntax, in part because polarity information must already be represented somehow in order to predict the distribution of so called *negative polarity items* (see [5] for a recent overview). As we shall see, an analogous argument may be made for our Projectivity Calculus. It turns out that the extra properties of functions that we need to project exclusion relations, namely *anti-additivity* and *anti-multiplicativity*, correspond exactly to those proposed by Zwarts and others to capture the syntactic distribution of increasingly strong classes of negative polarity items. This connection will be explained briefly in Section 4.

1. Ordered Domains

In model theoretic semantics, it is customary to think of implication as a special case of *inclusion*, most typically of one set of situations or possibilities in another. To say that P logically implies Q is, roughly, to say that the P situations are included in the Q situations. This is a special case because it amounts to inclusion in a particular type domain, namely that of the type for truth values. But we also have inclusion relations in other types. For instance, it is natural to think of domains of predicate types as ordered sets, again by inclusion of one set of individuals in another.

For the purposes of the Monotonicity Calculus, it is enough to assume our domains are preordered sets (posets). Suppose $\mathbb{A} = (A, \leq)$ and $\mathbb{B} = (B, \leq)$ are posets. Let $mon(\mathbb{A}, \mathbb{B})$ denote the set of monotone functions from A to B: functions such that $a \leq a'$ in $\mathbb{A}$ implies $f(a) \leq f(a')$ in $\mathbb{B}$. The set of *antitone* functions from $\mathbb{A}$ to $\mathbb{B}$, for which $a \leq a'$ implies $f(a') \leq f(a)$, coincides with the set of monotone functions from $\mathbb{A}$ to $\mathbb{B}^{op}$, where $\mathbb{B}^{op} = (B, \geq)$, the result of turning $\mathbb{B}$ upside-down. Let $ant(\mathbb{A}, \mathbb{B})$ denote this set of monotone functions. Both $mon(\mathbb{A}, \mathbb{B})$ and $ant(\mathbb{A}, \mathbb{B})$ admit a canonical preordering, where $f \leq g$ just in case for all $a \in A$, $f(a) \leq g(a)$. Clearly $(mon(\mathbb{A}, \mathbb{B}), \leq)$ and $(ant(\mathbb{A}, \mathbb{B}), \leq)$ are both preorders, provided $\mathbb{A}$ and $\mathbb{B}$ are. In this way, functional domains can inherit the appropriate structure from more basic domains. With this much foundation, the Monotoncity Calculus can already be defined and studied (see any of [2, 11, 13]).

However, type domains usually have much more underlying structure than mere posets. For example, the domain for truth value types can be taken as the two element structure $\mathbb{2} = (\{0,1\}, +, \cdot, 0, 1)$, which also happens to be the simplest non-trivial Boolean lattice. The domain for predicate types is usually the set of functions from some unordered domain D to $\mathbb{2}$, or equivalently the powerset lattice $\mathbb{D} = (\mathcal{P}(D), \cup, \cap, \varnothing, D)$. This extends to generalized quantifiers, which take elements of $\mathbb{D}$ to functions from $\mathbb{D}$ to $\mathbb{2}$, and indeed to any set of functions whose range is a Boolean lattice.

Proposition 1.1. *If $\mathbb{B} = (B, \vee, \wedge, 0, 1)$ is a Boolean lattice and A is any set, the set of functions $f : A \longrightarrow B$ forms a Boolean lattice, in which $f \vee g(a) = f(a) \vee g(a)$, $f \wedge g(a) = f(a) \wedge g(a)$, and $\mathbf{0}$ and $\mathbf{1}$ are the constant functions sending all $a \in A$ to 0 and 1, respectively.*

Whereas Monotonicity Calculus deals only with inclusion relations defined in terms of $\leq$, one can view MacCartney and Manning's work as capitalizing on this extra structure to capture inferences involving a larger collection of relations which now become defined in terms of $\vee$, $\wedge$, 0, and 1, in particular *exclusion* relations:

Definition 1.2 (The Set $\mathcal{R}$ of Relations). The following are notations for relations between elements in a Boolean lattice, slightly modified from [10].

$$\begin{array}{lll} x \sqsubseteq y & x \wedge y = x & (x \leq y) \\ x \sqsupseteq y & x \vee y = x & (x \geq y) \\ x \mid y & x \wedge y = 0 & \\ x \smile y & x \vee y = 1 & \end{array}$$

We write $x \equiv y$ if both $x \sqsubseteq y$ and $x \sqsupseteq y$; write $x \curlywedge y$ if both $x \mid y$ and $x \smile y$; and write $x\#y$ for the universal (uninformative) relation. Thus we define the set $\mathcal{R}$ of relations to be: $\{\equiv, \sqsubseteq, \sqsupseteq, \curlywedge, \mid, \smile, \#\}$.

These relations can also be defined in wider classes of domains, which shall become important in what follows. Let us call a structure $(A, \vee, 1)$ a $\vee$*-1-semilattice* if $(A, \vee)$ is a join-semilattice with top element 1; and likewise call $(A, \wedge, 0)$ a $\wedge$*-0-semilattice*. Any structure $(A, \vee, \wedge, 0, 1)$ that is both a $\vee$-1-semilattice and a $\wedge$-0-semilattice is a *bounded lattice*, provided that $\wedge$ and $\vee$ satisfy the absorption laws. It is a Boolean lattice if moreover $\wedge$ and $\vee$ satisfy the distributive laws and each element has a complement. Notice that the relations $\sqsubseteq$, $\sqsupseteq$, and $\equiv$ can be defined on any poset. The relation $\mid$ is only defined in $\wedge$-0-semilattices, while $\smile$ is defined in $\vee$-1-semilattices. The complementary relation $\curlywedge$ is only defined in bounded lattices.

In Boolean lattices, the set $\mathcal{R}$ of relations is rather well behaved. A first observation is that there is a natural ordering $\ll$ among the relations in $\mathcal{R}$. We say that $R \ll R'$, if xRy implies $xR'y$.[2] Then $(\mathcal{R}, \ll)$ is given as follows:

$$\begin{tikzcd}
 & \equiv & & \curlywedge & \\
\sqsubseteq \arrow[ur] & \sqsupseteq \arrow[u] & & \smile \arrow[u] & \mid \arrow[ul] \\
 & & \# \arrow[ull] \arrow[ul] \arrow[uul] \arrow[uur] \arrow[ur] \arrow[urr] & &
\end{tikzcd}$$

Lemma 1.3. *In any Boolean lattice, if x and y are distinct from 0 and 1, there is some $R \in \mathcal{R}$ such that, if $xR'y$ then $R' \ll R$.*

This fails when x or y is 1 or 0. For example, for any x we have both $x \smile 1$ and $x \sqsubseteq 1$, but these are incomparable in terms of $\ll$. Nonetheless, for elements other than 0 and 1, Lemma 1.3 guarantees the existence of some maximally informative relation between them.

Building on top of $(\mathcal{R}, \ll)$, we can introduce another important operation on $\mathcal{R}$, which turns out to play a central role in the NatLog system.

Definition 1.4. The join of R and R', denoted $R \bowtie R'$, is the $\ll$-maximal relation $R^* \in \mathcal{R}$ such that, if xRy and $yR'z$ then xR^*z.

It is a rather involved task to ensure that $\bowtie$ is well defined on $\mathcal{R}$. By exhaustive check, which we do not repeat here, one can verify that the join of any two relations corresponds to that given in [8, 10].

Lemma 1.5. *The join $R \bowtie R'$ of relations $R, R' \in \mathcal{R}$ is given by this table.*

$\bowtie$	$\sqsubseteq$	$\sqsupseteq$	$\curlywedge$	$\mid$	$\smile$
$\sqsubseteq$	$\sqsubseteq$	$\#$	$\mid$	$\mid$	$\#$
$\sqsupseteq$	$\#$	$\sqsupseteq$	$\smile$	$\#$	$\smile$
$\curlywedge$	$\smile$	$\mid$	$\equiv$	$\sqsupseteq$	$\sqsubseteq$
$\mid$	$\#$	$\mid$	$\sqsubseteq$	$\#$	$\sqsubseteq$
$\smile$	$\smile$	$\#$	$\sqsupseteq$	$\sqsupseteq$	$\#$

Furthermore, $\equiv \bowtie R = R = R \bowtie \equiv$, and $\# \bowtie R = \# = R \bowtie \#$, for all $R \in \mathcal{R}$.

Lemma 1.6 is a direct consequence of Definition 1.4, which will justify one of our two main rules of inference:

Lemma 1.6. *In any Boolean lattice, if xRy and $yR'z$ then $x(R \bowtie R')z$.*

[2] That is to say, in any Boolean lattice, if xRy holds then $xR'y$ also holds; or equivalently, $xR'y$ can be derived from xRy using the laws of Boolean algebra.

1.1. Functions on Domains

Monotonicity calculi focus on the polarity properties of functions on domains with respect to the basic two relations $\sqsubseteq$ and $\sqsupseteq$. Monotone functions f preserve these relations:

$$a \sqsubseteq a' \text{ implies } f(a) \sqsubseteq f(a'),$$

whereas antitone functions g reverse them:

$$a \sqsubseteq a' \text{ implies } g(a') \sqsubseteq g(a).$$

With three new relations (two new primitive relations, and one new defined relation), many new sorts of functions become relevant. The number of mappings from $\mathcal{R}$ to $\mathcal{R}$ is, after all, 7^7. Of course, not all of these are consistent with the meanings of these relations, and even fewer are likely to occur in natural language. In what follows, we shall only consider six additional classes of functions, which encompass all the expressions of English we have encountered so far, including all examples considered in [10]. Each of these classes refines either the monotone functions or the antitone functions.

Definition 1.7. Let $f : \mathbb{L} \longrightarrow \mathbb{L}'$ be a function on Boolean lattices.[3]

1. f is *additive* if $f(x \vee y) = f(x) \vee f(y)$, and *completely additive* if in addition $f(1) = 1$.
2. f is *multiplicative* if $f(x \wedge y) = f(x) \wedge f(y)$, and *completely multiplicative* if in addition $f(0) = 0$.
3. f is *anti-additive* if $f(x \vee y) = f(x) \wedge f(y)$, and *completely anti-additive* if in addition $f(1) = 0$.
4. f is *anti-multiplicative* if $f(x \wedge y) = f(x) \vee f(y)$, and *completely anti-multiplicative* if in addition $f(0) = 1$.

Note that each does in fact refine either monotonicity or antitonicity.

Lemma 1.8. *Let $f : \mathbb{L} \longrightarrow \mathbb{L}'$ be a function. The following are equivalent:*

(i) *f is monotone;*

(ii) $f(x) \vee f(y) \leq f(x \vee y)$*;*

(iii) $f(x \wedge y) \leq f(x) \wedge f(y)$.

[3]The notions of additive and anti-additive functions seem to have been first used in the study of language by Hoeksema [6]. See Zwarts [16] for discussion of anti-multiplicative functions. We shall revisit this connection in Section 4.

Proof. $(i) \Rightarrow (ii)$: Suppose f is monotone. Then since $x \leq x \vee y$ and $y \leq x \wedge y$, we have $f(x) \leq f(x \vee y)$ and $f(y) \leq f(x \vee y)$. But seeing as $f(x) \vee f(y)$ is the join of $f(x)$ and $f(y)$, we conclude $f(x) \vee f(y) \leq f(x \vee y)$.

$(ii) \Rightarrow (i)$: Assume $x \leq y$, and so $y = x \vee y$. Then $f(y) = f(x \vee y) \geq f(x) \vee f(y)$, which means $f(x) \leq f(y)$.

The equivalence of (ii) and (iii) follows by duality. ■

Lemma 1.9. *Let $f : \mathbb{L} \longrightarrow \mathbb{L}'$ be a function. The following are equivalent:*

(i) *f is antitone*;

(ii) $f(x \vee y) \leq f(x) \wedge f(y)$;

(iii) $f(x) \vee f(y) \leq f(x \wedge y)$.

Proof. Analogous to the proof of Lemma 1.8. ■

Furthermore, the duality between monotone and antitone functions extends naturally to these classes of functions. In the case of bounded lattices $\mathbb{L} = (L, \vee, \wedge, 0, 1)$, the dual $(L, \wedge, \vee, 1, 0)$ is denoted by $\mathbb{L}^{op}$. So, just as in the case of preordered sets, $x \leq y$ in $\mathbb{L}$ if and only if $x \geq y$ in $\mathbb{L}^{op}$. This extends to semilattices in the obvious way, so that $\mathbb{L} = (L, \vee, 1)$ is a $\vee$-1-semilattice if and only if $\mathbb{L}^{op} = (L, \wedge, 0)$ is a $\wedge$-0-semilattice.

Lemma 1.10. *1. The set of anti-additive functions from $\mathbb{L}'$ to $\mathbb{L}$ is equal to the set of additive functions from $\mathbb{L}'$ to $\mathbb{L}^{op}$.*

2. The set of anti-multiplicative functions from $\mathbb{L}'$ to $\mathbb{L}$ is equal to the set of multiplicative functions from $\mathbb{L}'$ to $\mathbb{L}^{op}$.

There are numerous examples of all consistent combinations of these properties from Definition 1.7. In the cases of typical quantifiers in English, the step from property P to property *completely* P amounts to assuming non-trivial domains, an assumption we shall adopt in what follows. For example, *some* is additive in its second argument, yet *some astronaut who is not an astronaut* is the constant function sending everything to 0, and so it is not, strictly speaking, completely additive. Henceforth, by P we mean *completely* P. The following are some examples (c.f. [10, 16]).

* *Few* fails all of these properties in its first argument.
* *At least two* is (only) monotone in both arguments.
* *If* is (only) antitone in its first argument.
* *Some* is additive in both arguments.

* *No* is anti-additive in both arguments.

* *Most* is multiplicative in its second argument.

* *Not every* is anti-multiplicative in its second argument.

* *Is* is additive and multiplicative.

* *Not* is anti-additive and anti-multiplicative.

These considerations suggest the need for nine separate *projectivity signatures*, which we shall eventually use to label functional types.

Definition 1.11 (Projectivity Signatures). The set Σ of projectivity signatures is defined to be $\{\bullet, +, -, \diamondplus, \diamondminus, \boxplus, \boxminus, \oplus, \ominus\}$.

Intuitively, $\bullet$ is associated with the class of all functions, $+$ with the monotone functions, $-$ with antitone, $\diamondplus$ with additive, $\diamondminus$ with anti-additive, $\boxplus$ with multiplicative, $\boxminus$ with anti-multiplicative, $\oplus$ with additive and multiplicative, and $\ominus$ with anti-additive and anti-multiplicative. This will be made more precise in the next section.

To end this section, we present some basic facts that will become useful in what follows. Below, $\mathbb{L}$ and $\mathbb{L}'$ are assumed to be $\vee$-1-semilattices or $\wedge$-0-semilattices, as appropriate.

Lemma 1.12. *Suppose we are given two semilattices $\mathbb{L}$ and $\mathbb{L}'$.*

1. *The additive functions $f : \mathbb{L} \longrightarrow \mathbb{L}'$ form a $\vee$-1-semilattice.*
2. *The multiplicative functions form a $\wedge$-0-semilattice.*
3. *The anti-additive functions form a $\vee$-1-semilattice.*
4. *The anti-multiplicative functions form a $\wedge$-0-semilattice.*

Proof. We show only parts 1 and 3, as 2 and 4 are analogous.

Suppose f and g are additive functions. We must show $f \vee g$, defined so that $f \vee g(a) = f(a) \vee g(a)$ for all a, is also additive. But this is clear, since $f \vee g(a \vee b) = f(a \vee b) \vee g(a \vee b) = (f(a) \vee f(b)) \vee (g(a) \vee g(b)) = (f(a) \vee g(a)) \vee (f(b) \vee g(b)) = f \vee g(a) \vee f \vee g(b)$. Moreover $f \vee g(1) = f(1) \vee g(1) = 1 \vee 1 = 1$. The top element $\mathbf{1}$ in the $\vee$-semilattice of additive functions is the constant function sending all a to 1.

Suppose next that f and g are anti-additive. It follows that $f \vee g(a \vee b) = f(a \vee b) \wedge g(a \vee b) = (f(a) \wedge f(b)) \wedge (g(a) \wedge g(b)) = (f(a) \wedge g(a)) \wedge (f(b) \wedge g(b)) = f \vee g(a) \wedge f \vee g(b)$. Likewise, $f \vee g(1) = f(1) \wedge g(1) = 0 \wedge 0 = 0$. And the top element is the constant function sending all a to 0. Since the anti-additive functions are identified with the additive functions from $\mathbb{L}$ to $\mathbb{L}'^{op}$, this is indeed the top element in the ordered set. ■

2. Projectivity Marking

2.1. Types

Our set of types is generated from three basic types: p, v, and t, which correspond to predicates, intransitive verbs, and truth values, respectively.[4]

DEFINITION 2.1. The set $\mathcal{T}$ of *types* consists of all symbols generated by the following grammar:[5]

$$\begin{aligned}
\tau_b &::= p \mid v \mid t \mid \tau_b \longrightarrow \tau_b \mid \tau_b \xrightarrow{+} \tau_b \mid \tau_b \xrightarrow{-} \tau_b \mid \tau_b \xrightarrow{\oplus} \tau_b \mid \tau_b \xrightarrow{\ominus} \tau_b\\
\tau_u &::= \tau_b \mid \tau_u \xrightarrow{\diamond\!\!+} \tau_u \mid \tau_u \xrightarrow{\diamond\!\!-} \tau_w\\
\tau_w &::= \tau_b \mid \tau_w \xrightarrow{\boxplus} \tau_w \mid \tau_w \xrightarrow{\boxminus} \tau_u
\end{aligned}$$

As will become evident in the next definition, the domains of types generated as τ_b are bounded lattices, those of τ_u are $\wedge$-0-semilattices, and those of τ_w are $\vee$-1-semilattices (see Lemma 1.12).

DEFINITION 2.2. We define (ordered) domains $\mathbb{A}_\tau = (A_\tau, \leq)$ by induction.

1. $\mathbb{A}_p = \mathbb{A}_v = (\mathcal{P}(D), \subseteq)$, for some given set D.
2. $\mathbb{A}_t$ is the ordered two-element set $\mathbb{2} = (\{0, 1\}, \leq)$.
3. In the following, we assume the canonical ordering on sets of functions.
   $\mathbb{A}_{\tau \longrightarrow \sigma}$ is the set of all functions $f : \mathbb{A}_\tau \longrightarrow \mathbb{A}_\sigma$.
   $\mathbb{A}_{\tau \xrightarrow{+} \sigma}$ is the set of monotone functions $f : \mathbb{A}_\tau \longrightarrow \mathbb{A}_\sigma$.
   $\mathbb{A}_{\tau \xrightarrow{-} \sigma}$ is the set of antitone functions $f : \mathbb{A}_\tau \longrightarrow \mathbb{A}_\sigma$.
   $\mathbb{A}_{\tau \xrightarrow{\diamond\!\!+} \sigma}$ is the set of additive functions $f : \mathbb{A}_\tau \longrightarrow \mathbb{A}_\sigma$.
   $\mathbb{A}_{\tau \xrightarrow{\diamond\!\!-} \sigma}$ is the set of anti-additive functions $f : \mathbb{A}_\tau \longrightarrow \mathbb{A}_\sigma$.
   $\mathbb{A}_{\tau \xrightarrow{\boxplus} \sigma}$ is the set of multiplicative functions $f : \mathbb{A}_\tau \longrightarrow \mathbb{A}_\sigma$.
   $\mathbb{A}_{\tau \xrightarrow{\boxminus} \sigma}$ is the set of anti-multiplicative functions $f : \mathbb{A}_\tau \longrightarrow \mathbb{A}_\sigma$.
   $\mathbb{A}_{\tau \xrightarrow{\oplus} \sigma} = \mathbb{A}_{\tau \xrightarrow{\diamond\!\!+} \sigma} \cap \mathbb{A}_{\tau \xrightarrow{\boxplus} \sigma}$. $\qquad \mathbb{A}_{\tau \xrightarrow{\ominus} \sigma} = \mathbb{A}_{\tau \xrightarrow{\diamond\!\!-} \sigma} \cap \mathbb{A}_{\tau \xrightarrow{\boxminus} \sigma}$.

Lemma 1.12 and Definition 2.1 together guarantee that all of these ordered domains are well defined. This correspondence henceforth justifies our calling a function that has the property corresponding to a given projectivity marking φ a *φ-function.* For example, a $\diamond\!\!+$-function is an additive function.

As discussed above, monotone functions project $\sqsubseteq$ as $\sqsubseteq$, and antitone functions project $\sqsubseteq$ as $\sqsupseteq$. We are now in a position to study the projectivity behavior of all of these classes of functions for all the relations in $\mathcal{R}$.

[4]We diverge from using the standard primitive types e and t. (C.f. [11], [13]).

[5]The plain functional type symbol $\tau \longrightarrow \sigma$ is used in place of $\tau \xrightarrow{\bullet} \sigma$, for convenience.

DEFINITION 2.3. [Projection] If $R \in \mathcal{R}$ and $\varphi \in \Sigma$, the *projection of R under φ* is the $\ll$-maximal $R^* \in \mathcal{R}$ for which the following holds:

Whenever xRy and f is a φ-function, it follows that $f(x)R^*f(y)$.

We write $[R]^\varphi$ for the projection of R under φ.

LEMMA 2.4. *The operation $[R]^\varphi$ is well defined, and is given by this table:*

$[\,]$	$\sqsubseteq$	$\sqsupseteq$	$\curlywedge$	$\mid$	$\smile$
$+$	$\sqsubseteq$	$\sqsupseteq$	$\#$	$\#$	$\#$
$\diamondplus$	$\sqsubseteq$	$\sqsupseteq$	$\smile$	$\#$	$\smile$
$\boxplus$	$\sqsubseteq$	$\sqsupseteq$	$\mid$	$\mid$	$\#$
$\oplus$	$\sqsubseteq$	$\sqsupseteq$	$\curlywedge$	$\mid$	$\smile$

$[\,]$	$\sqsubseteq$	$\sqsupseteq$	$\curlywedge$	$\mid$	$\smile$
$-$	$\sqsupseteq$	$\sqsubseteq$	$\#$	$\#$	$\#$
$\diamondminus$	$\sqsupseteq$	$\sqsubseteq$	$\mid$	$\#$	$\mid$
$\boxminus$	$\sqsupseteq$	$\sqsubseteq$	$\smile$	$\smile$	$\#$
$\ominus$	$\sqsupseteq$	$\sqsubseteq$	$\curlywedge$	$\smile$	$\mid$

Moreover, note that $[R]^\bullet = \#$ for all $R \in \mathcal{R}$.

Lemma 2.5 follows directly from Definition 2.3 and Lemma 2.4.

LEMMA 2.5. *For any φ-function $f : \mathbb{A} \longrightarrow \mathbb{B}$, if xRy then $f(x)[R]^\varphi f(y)$.*

2.2. Function Composition

Like the set $\mathcal{R}$ of relations, the set Σ of signatures also comes with its own structure. First of all, it has a natural ordering $\leqslant$, defined so that $\varphi \leqslant \psi$ just in case any φ-function is also a ψ-function. Then $(\Sigma, \leqslant)$ is given by the following graph (with transitive arrows left implicit):

$$\begin{tikzcd}
 & \oplus & & \ominus & \\
\boxplus \arrow[ur] & \diamondplus \arrow[u] & & \diamondminus \arrow[u] & \boxminus \arrow[ul] \\
 & + \arrow[ul] \arrow[u] & & - \arrow[u] \arrow[ur] & \\
 & & \bullet \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Using this ordering we can also define a sort of composition operator on Σ:

DEFINITION 2.6. [Composition] If $\varphi, \psi \in \Sigma$, then $\varphi \circ \psi$ is defined to be the $\leqslant$-maximal signature $\chi \in \Sigma$, such that the following holds:

If f is a φ-function and g is a ψ-function, then $f \circ g$ is a χ-function.

Abusing terminology, $\varphi \circ \psi \in \Sigma$ is called the *composition of* φ *and* ψ.

The composition can be shown well-defined, and in fact we have:

LEMMA 2.7. $(\Sigma, \circ, \oplus)$ *is a monoid.*

The full table is reproduced here, noting also that $\varphi \circ \bullet = \bullet = \bullet \circ \varphi$, and $\oplus \circ \varphi = \varphi = \varphi \circ \oplus$, for all $\varphi \in \Sigma$.

$\circ$	$+$	$-$	$\diamondplus$	$\diamondminus$	$\boxplus$	$\boxminus$	$\ominus$
$+$	$+$	$-$	$+$	$-$	$+$	$-$	$-$
$-$	$-$	$+$	$-$	$+$	$-$	$+$	$+$
$\diamondplus$	$+$	$-$	$\diamondplus$	$-$	$+$	$\boxminus$	$\boxminus$
$\diamondminus$	$-$	$+$	$\diamondminus$	$-$	$-$	$\boxplus$	$\boxplus$
$\boxplus$	$+$	$-$	$+$	$\diamondminus$	$\boxplus$	$-$	$\diamondminus$
$\boxminus$	$-$	$+$	$-$	$\diamondplus$	$-$	$+$	$\diamondplus$
$\ominus$	$-$	$+$	$\diamondminus$	$\diamondplus$	$\boxminus$	$\diamondplus$	$\oplus$

Notably, $\circ$ is not commutative, e.g. $\boxminus \circ \diamondminus = \diamondplus \neq \boxplus = \diamondminus \circ \boxminus$. Nor does $(\Sigma, \circ, \oplus)$ form a group, as only $\oplus$ and $\ominus$ have inverses, namely themselves.

Lemma 2.7, together with Lemma 2.5, will be crucial for the projectivity marking algorithm in the next section. It is the analogue in this more general setting to the fact that $+$ and $-$ alone are well-behaved with respect to composition (the upper left quadrant of the figure), which allows propagating polarity information up (or down) a parse tree in the Monotonicity Calculus.

2.3. Typed Language

Every term t of our language $\mathcal{L}$, defined below in Definition 2.8, has a type τ, for which we write $t : \tau$. In addition, we can associate with every term (or type) an *unmarked type*, which is simply the result of erasing all of the markings on τ, notated τ° (following [15]). Note that by Proposition 1.1 the domain for every unmarked type is a Boolean lattice.

The set $\mathcal{T}$ of types also has a natural ordering, according to the restrictions they put on their domain. The ordering $\leqslant \in \mathcal{T} \times \mathcal{T}$ is defined to be the least ordering such that, for all $\sigma, \tau, \sigma', \tau' \in \mathcal{T}$:

1. $\tau \leqslant \tau$.
2. If $\sigma' \leqslant \sigma$, $\tau \leqslant \tau'$, and $\varphi \leqslant \psi$, then $\sigma \xrightarrow{\varphi} \tau \ \leqslant \ \sigma' \xrightarrow{\psi} \tau'$.

Intuitively, $\tau \leqslant \sigma$ means, anything of type τ could also be considered of type σ, with possible loss of information. Notice, whenever $\tau \leqslant \sigma$ we have $\tau^\circ = \sigma^\circ$.

Definition 2.8. In the language $\mathcal{L}$ we allow countably many variables (and assume at least one) of each type, and we allow countably many constants of any given type. In addition, we have the following complex terms:

If $t : \sigma \xrightarrow{\varphi} \tau$ and $s : \rho \leqslant \sigma$, then $t(s) : \tau$.

A model $\mathcal{M} = (D, [\![\ ]\!], \nu)$ of $\mathcal{L}$ consists of a domain D of entities, an interpretation function $[\![\ ]\!]$, and a typed map ν. The interpretation function $[\![\ ]\!]$ is defined, so that if $t : \tau$ is a constant then $[\![t]\!] \in A_\tau$; and if t is a functional term $s(u)$, $[\![s(u)]\!] = [\![s]\!]([\![u]\!])$. The variable assignment ν sends variables to elements of the appropriate type domain.

We can say $\mathcal{M}$ is a model of a relational statement tRt' in the obvious way: this holds if t and t' are of the same unmarked type τ° and $[\![t]\!]$ bears the relation R to $[\![t']\!]$ in $\mathbb{A}_{\tau^\circ}$. These relations are always well defined, since $\mathbb{A}_{\tau^\circ}$ is a Boolean lattice (Proposition 1.1). This allows us to say, e.g. that *every* should stand in the relation $|$ to *no* in the unmarked type domain of generalized quantifiers $\mathbb{A}_{p \longrightarrow (v \longrightarrow t)}$. We write $\Gamma \vDash tRt'$ if any model of all the assumptions in Γ is also a model of tRt'.

2.4. Projectivity of Contexts

Terms with a single variable can be understood as functions. This allows marking the projectivity of a given occurrence of a subterm in a given closed term.[6] If t contains no variables, it is called a *ground term.* We define the *topmost projectivity* of a term t, denoted by $top(t)$, to be $\oplus$ if t is of basic type, and φ if $t : \sigma \xrightarrow{\varphi} \tau$ for $\varphi \in \Sigma$. A *context* t is a term with exactly one variable, x, and the projectivity of context t is defined as follows:

Definition 2.9. The projectivity of a context t with variable x, written $pro(t)$, is defined by induction on t:

1. If $t = x$, then $pro(t) = \oplus$;
2. If $t = s(u)$ and x is a subterm of s, then $pro(s(u)) = pro(s)$;
3. If $t = s(u)$ and x is a subterm of u, then $pro(s(u)) = top(s) \circ pro(u)$.

If $t : \tau$ has a single variable $x : \sigma$, we can associate with the context t a function $f_t : \mathbb{A}_\sigma \longrightarrow \mathbb{A}_\tau$, which is defined so that:

1. If $t = x$, then $f_t : A_\tau \longrightarrow A_\tau$ is the identity function;
2. If $t = s(u)$ and x is a subterm of s, then for $a \in A_\sigma$, $f_{s(u)}(a) = f_s(a)([\![u]\!])$;

[6] What follows derives in large part from the presentation in [12]. See also [11].

3. If $t = s(u)$ and x is a subterm of u, then $f_{s(u)} = [\![s]\!] \circ f_u$.

Proposition 2.10 (Soundness of Projectivity Marking). *Given a context t, if $pro(t) = \varphi$ then f_t is a φ-function.*

Proof. We proceed by induction on t. If t just is the variable x, then since f_t is simply the identity on A_τ, certainly f_t is additive and multiplicative.

Next, suppose $t = s(u)$ and x is a subterm of s. In this case $pro(s(u)) = pro(s)$, say it is φ. We give the result only for the case where $\varphi = \ominus$, as the other seven cases are either easier or analogous. We must show $f_{s(u)}$ is anti-additive. By the induction hypothesis f_s is anti-additive, so $f_{s(u)}(g \vee h) = f_s(g \vee h)([\![u]\!]) = (f_s(g) \wedge f_s(h))([\![u]\!]) = f_s(g)([\![u]\!]) \wedge f_s(h)([\![u]\!]) = f_{s(u)}(g) \wedge f_{s(u)}(h)$. Likewise, $f_{s(u)}(1) = f_s(1)([\![u]\!]) = 0([\![u]\!]) = 0$.

In the last case, if $f = s(u)$ and x is a subterm of u, then $f_{s(u)} = [\![s]\!] \circ f_u$. By Definition 2.9, $pro(s(u)) = top(s) \circ pro(u)$. Suppose that $u : \rho$, and so $s : \rho \xrightarrow{\varphi} \tau$ for some $\varphi \in \Sigma$, which means $[\![s]\!]$ is a φ-function. If $pro(u) = \psi$, then by induction hypothesis f_u is a ψ-function. We can conclude $f_{s(u)} = [\![s]\!] \circ f_u$ is a $\varphi \circ \psi$-function, which by Lemma 2.7 is well defined and in Σ. ■

Finally, we can define the projectivity of an occurrence of a subterm in a ground term in a straightforward way. If t is a term and s is a subterm occurence of t, let $t^{s' \leftarrow s}$ denote the term that results from replacing the single occurrence of s by s' in t.

Definition 2.11. If s is any subterm occurrence of ground term t, there is a unique context $t' = t^{x \leftarrow s}$ with the single variable x. We say the *projectivity of occurrence s in t* is $pro(t') \in \Sigma$. We abbreviate this signature by $o(s,t)$.

Corollary 2.12. *If $o(s,t) = \varphi$ and $t' = t^{s' \leftarrow s}$ with sRs', then $t[R]^\varphi t'$.*

This corollary will form the basis of the calculus defined in the next section, justifying the main inference rule for substitution.

3. A Projectivity Calculus

Based on the projectivity marking algorithm we have defined, a number of reasoning calculi are now conceivable. In this section we present one concrete example, which captures rather closely the logic underlying the NatLog system [10]. NatLog uses the notion of an *edit distance* between two expressions, familiar from computational linguistics, which incorporates substitutions of terms, in addition to insertions and deletions. We do not explicitly include insertion and deletion rules; however, our Substitution

Rule, working at the level of types rather than terms, is sufficiently general to capture much of what is accomplished with those operations. For instance, instead of a *not*-insertion rule, we could allow x to be substituted by *not* x.

3.1. Calculus $\mathcal{C}$ of Relations

Suppose $\mathcal{L}$ is our typed language, and the relations in $\mathcal{R}$ are defined for some given primitive constants. The Projectivity Calculus $\mathcal{C}$ provides a method for extending relations in $\mathcal{R}$ to arbitrary terms. We have a few basic rules, which are clearly sound:

$$\text{Reflexivity: } \frac{}{t \sqsubseteq t}$$

$$\text{Symmetry: } \frac{t \sqsubseteq t'}{t' \sqsupseteq t} \qquad \frac{t \sqsupseteq t'}{t' \sqsubseteq t} \qquad \frac{t \mid t'}{t' \mid t} \qquad \frac{t \smile t'}{t' \smile t}$$

$$\text{Absurdity: } \frac{t \mid t}{sRs'}$$

The main interest in $\mathcal{C}$ stems from the next two rules. The first allows us to compose relations, according to the join operation (see Lemma 1.5).

$$\text{Composition: } \frac{tRu \qquad uR'v}{t(R \bowtie R')v}$$

The second is the Substitution Rule, which justifies calling $\mathcal{C}$ a *projectivity* calculus. The soundness of this rule follows from Corollary 2.12. We allow any substitutions of terms with the same unmarked type, provided that the resulting expression is still well formed according to Definition 2.8.

$$\text{Substitution: } \frac{sRs'}{t[R]^{\varphi}t^{s' \leftarrow s}}\ o(s,t) = \varphi$$

If we have a set of relational assumptions Γ, we write $\Gamma \vdash_{\mathcal{C}} tRt'$ just in case tRt' follows from Γ by applying Reflexivity, Symmetry, Absurdity, Composition, or Substitution some finite number of times.

THEOREM 3.1 (Soundness of $\mathcal{C}$). *If* $\Gamma \vdash_{\mathcal{C}} tRt'$, *then* $\Gamma \vDash tRt'$.

PROOF. This follows by induction on length of proofs in $\mathcal{C}$, citing Corollary 2.12 for the Substitution Rule and Lemma 1.6 for the Composition Rule. ■

$\mathcal{C}$ will not in general be complete. For example, for any two terms t and t' of types $\sigma \xrightarrow{\oplus} \tau$ and $\sigma \xrightarrow{\ominus} \tau$, respectively, it will always hold that $[\![t]\!] \mid [\![t']\!]$ since no function is both additive and anti-additive. But it is easy to see that $t \mid t'$ is not derivable from the empty set of assumptions. The question, how to turn $\mathcal{C}$ into a complete system, we leave for future work.

3.2. A Worked Example

In this section we define a simple grammar for a small fragment of English, using marked types of different projectivity signatures.

Constant	Type
every	$p \xrightarrow{\diamondminus} (v \xrightarrow{\boxplus} t)$
some, a	$p \xrightarrow{\diamondplus} (v \xrightarrow{\diamondplus} t)$
no	$p \xrightarrow{\diamondminus} (v \xrightarrow{\diamondminus} t)$
not every	$p \xrightarrow{\diamondplus} (v \xrightarrow{\boxminus} t)$
job, giant squid, cephalopod	p
safe, dangerous	$p \xrightarrow{\oplus} p$
is	$(p \xrightarrow{\oplus} p) \xrightarrow{\oplus} v$
involves	$(v \longrightarrow t) \xrightarrow{\oplus} v$
that	$v \xrightarrow{\oplus} (p \xrightarrow{\oplus} p)$

Define the set Γ to consist of the following relational assumptions (obviously not complete, but sufficient for the purpose of our examples).

$\textit{every} \curlywedge \textit{not every}$	$\textit{some} \curlywedge \textit{no}$	$\textit{squid} \sqsubseteq \textit{cephalopod}$
$\textit{no} \mid \textit{every}$	$\textit{safe} \mid \textit{dangerous}$	

From this we can derive typical monotonicity inferences:

$$\frac{\textit{giant squid} \sqsubseteq \textit{cephalopod}}{\textit{every cephalopod} \sqsubseteq \textit{every giant squid}}$$

But we can also capture inferences and relations that apparently require going beyond monotonicity. For instance, we do not need an extra postulate to tell us that $\textit{no} \sqsubseteq \textit{not every}$, since it is already derivable from Γ.

$$\dfrac{\dfrac{\textit{no} \mid \textit{every} \qquad \textit{every} \curlywedge \textit{not every}}{\textit{no}\ (\mid \bowtie \curlywedge)\ \textit{not every}}}{\textit{no} \sqsubseteq \textit{not every}}$$

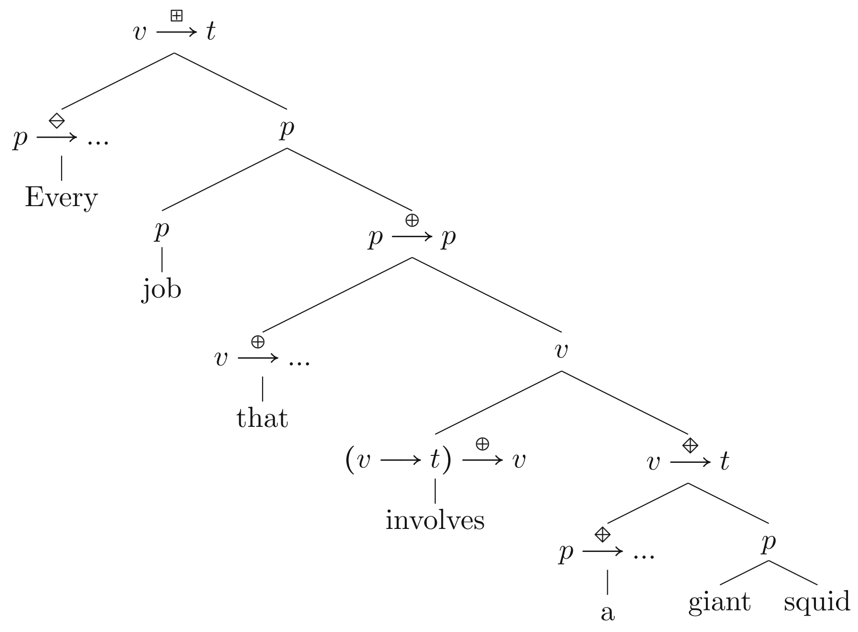
The main example in this section is intended to demonstrate that genuinely new entailment relations among sentences can be captured in $\mathcal{C}$. We show that the following holds in $\mathcal{C}$ using Γ as premises:

t = *Every job that involves a giant squid is dangerous* $\sqsubseteq$
t' = *Not every job that involves a giant squid is safe*

Instead of writing out the entire derivation at once, we go through it in steps. First of all, notice that there are three substitutions leading from t to t', which use the following axioms from Γ:

$$\textit{every} \curlywedge \textit{not every} \qquad \textit{giant squid} \sqsubseteq \textit{cephalopod} \qquad \textit{dangerous} \mid \textit{safe}$$

Below is a partial tree depicting the crucial function types that will determine the projectivity of the relevant contexts (the verb argument is left off):


$v \xrightarrow{\boxplus} t$
$p \xrightarrow{\diamondminus} \dots$
Every
p
p
job
$p \xrightarrow{\oplus} p$
$v \xrightarrow{\oplus} \dots$
that
v
$(v \longrightarrow t) \xrightarrow{\oplus} v$
involves
$v \xrightarrow{\diamondplus} t$
$p \xrightarrow{\diamondplus} \dots$
a
p
giant
squid

Six different substitution orders are possible, but the reader may check that each leads to the same conclusion. Arbitrarily, we begin by substituting *safe* for *dangerous*. By Definition 2.9, the projectivity of this occurrence of *dangerous* is given by $top(every(job...)) \circ top(is) = \boxplus \circ \oplus = \boxplus$. Since $[\mid]^{\boxplus} = \mid$, we first have an inference of the following form.

$$\frac{\textit{safe} \mid \textit{dangerous}}{t \mid t^{\textit{safe} \leftarrow \textit{dangerous}}}\; o(\textit{dangerous}, t) = \boxplus$$

Next we can substitute *cephalopod* for *giant squid* in the resulting term $u = t^{\textit{safe} \leftarrow \textit{dangerous}}$. The projectivity of its occurrence is,

$$top(every) \circ top(that) \circ top(involves) \circ top(a) = \diamondminus \circ \oplus \circ \oplus \circ \diamondplus = \diamondminus$$

which results in the following inference:

$$\frac{\textit{giant squid} \sqsubseteq \textit{cephalopod}}{u \sqsupseteq u^{\textit{cephalopod} \leftarrow \textit{giant squid}}}\ o(\textit{giant squid}, u) = \diamondminus$$

Finally, we can substitute *not every* for *every* in $v = u^{\textit{cephalopod} \leftarrow \textit{giant squid}}$, in which case this occurrence is not under the scope of any function, so its projectivity marking is simply $\oplus$.

$$\frac{\textit{every} \curlywedge \textit{not every}}{v \curlywedge v^{\textit{not every} \leftarrow \textit{every}}}\ o(\textit{every}, v) = \oplus$$

Where $t' = v^{\textit{not every} \leftarrow \textit{every}}$, the Composition Rule gives us the following.

$$\frac{\dfrac{t \mid u \qquad u \sqsupseteq v}{t \mid v} \qquad v \curlywedge t'}{t \sqsubseteq t'}$$

It is clear that such an inference is not derivable in a system that uses only polarity markings, as it crucially relies on exclusion relations $\mid$ and $\curlywedge$.

3.3. The Church-Rosser Property

We know that $\mathcal{C}$ is sound, in the sense that it never derives from sound assumptions a relational statement that does not hold. But is it also confluent? In the example just given we noted the order of substitution did not matter. But is this true in general? The answer is negative, for a very simple reason. The problem is that it is very easy to derive the uninformative relation #, and once this happens it is impossible to derive any other relation since the composition of any relation with # is again #. So in one sense, we have a kind of confluence since we can always reach the # relation given a rich enough language. However, the system lacks the kind of confluence one might hope for. That is, one must be clever with one's choice of substitutions. We illustrate with an example.

In one step we can derive that *some squid* $\sqsubseteq$ *some cephalopod*, simply by substituting in *cephalopod* for *squid*. However, once we make a wrong substitution, this conclusion is unreachable. For instance, given the assumption that *squid* $\mid$ *octopus*, by one substitution we derive *some squid* # *some octopus*. Then, even though we also have *octopus* $\sqsubseteq$ *cephalopod* and thus *some octopus* $\sqsubseteq$ *some cephalopod*, our result is $\# \circ \sqsubseteq = \#$. It is not actually false that *some squid* # *some cephalopod*, but by a smarter choice of rule applications we can derive something stronger.

4. Negative Polarity Items

In the introduction, we said that the goal of a natural logic is to facilitate inferences using only shallow, syntactic features. On the face of it, the projectivity signatures we use to mark types are paradigmatically semantic, even though they stop short of full semantic interpretation. However, a surprising discovery is that this much information in fact seems to be needed to predict distribution patterns of certain classes of expressions across a number of languages. These are the so called negativity polarity items.

It is well known since the work of Ladusaw [7] that negative polarity items like *any* or *ever* generally appear only in antitone contexts. In other words, there seem to be independent reasons to keep track of monotonicity information in a formal grammar, simply to account for the well formed expressions. Subsequent work by Zwarts on English, Dutch, and German argued that there are at least three distinct subclasses of negative polarity items that exhibit different distributional behavior: weak, strong, and superstrong [16]. The weak negative polarity items like *yet* can appear in any antitone context, as illustrated by the fact that 1 and 2 are grammatical:

1. Not everyone is here yet.
2. Few students are here yet.

The strong negative polarity items, Zwarts noticed, seem to require anti-additive contexts. An example of the strong type is the expression *in years.* Thus, while 3 is ungrammatical, 4 is grammatical since *no* is anti-additive in its second argument.

3. # Few scholars have written about it in years.
4. No scholar has written about it in years.

Finally, the superstrong negative polarity items, like *a tad bit*, seem to require contexts that are both anti-additive and anti-multiplicative. Thus, 5 seems ungrammatical, while 6 is perfectly grammatical since *not* is anti-additive and anti-multiplicative.

5. # No customer was a tad bit happy.
6. The manager was not a tad bit happy.

Zwarts provides a rather comprehensive classification of quantifiers and other contexts in terms of these function types. As our results show, such a classification is already sufficient to account for the projectivity behavior of these expressions. In some sense, then, the notions we use to capture significant reasoning patterns based on inclusion and exclusion are independently motivated by grammatical concerns that at first may have seemed unrelated.

5. Conclusion

We have shown how a projectivity calculus can be defined on top of the Monotonicity Calculus with only minimal additions. In effect, as soon as one is able to assign marked types to expressions in a given fragment of language and specify some axioms about what relations hold among basic lexical items, it is possible to apply the Projectivity Calculus to derive new relations among complex terms. We close by discussing where the Projectivity Calculus might lie in the space of familiar logical systems.

One question is whether the Projectivity Calculus is a genuine *extension* of the Monotonicity Calculus. In fact, as observed in [4], an exclusion relation between x and y can equivalently be seen as a containment relation between x and $-y$, the complement of y. Note that $x \mid y$ is equivalent to $x \sqsubseteq -y$, $x \smile y$ is equivalent to $x \sqsupseteq -y$, and hence $x \curlywedge y$ is equivalent to $x \equiv -y$. Each of these is a special case of containment. Indeed, we could have set things up this way. But this should not blur the fact that we still would need to define the same additional projectivity signatures to deal with these effectively different kinds of containment relations. After all, we have seen that different quantifiers with the same monotonicity markings must be assigned different projectivity markings (e.g. *some* and *at least two*). In some sense, it is the new signatures based on enriched domains that constitute an extension of the Monotonicity Calculus, not just the exclusion relations by themselves. We could just as well have considered the single relation $\sqsubseteq$ and unary negation operation $-$.

From the other side, an outstanding question is how close $\mathcal{C}$ is to the expressive and inferential power of first-order, or higher-order logic. As we remarked at the beginning, one of the interests in natural logic and surface reasoning is that we are not restricted to fragments for which we already have approximate first-order translations. Still, the question does arise: If we consider a fragment of English including only *every*, *some*, and *not*, with their appropriately marked types, plus a number of individual constants, predicates and relations (i.e. proper nouns, common nouns, adjectives, verbs, etc.), what fragment of first-order logic do we obtain? We know, for example, the De Morgan laws are not in general derivable. For example, entailments like *Every job is safe* $\Longrightarrow$ *No job is dangerous* require more information than is represented in our projectivity markings. That is, the inference from *Q job is safe* to *Q' job is dangerous* is not sound for just any quantifiers $Q : p \xrightarrow{\diamondminus} (v \xrightarrow{\boxplus} t)$ and $Q' : p \xrightarrow{\diamondminus} (v \xrightarrow{\diamondminus} t)$. What sort of fragment of first-order, or higher-order, logic this is, and just how far this type marking could take us, we leave as questions for future work.

Acknowledgements. The author would like to thank Lawrence S. Moss for helpful comments on an early version of this paper, and for many formative discussions. Thanks also to Johan van Benthem, two anonymous reviewers, and the editors of this issue of *Studia Logica*.

References

[1] Barwise, J., and R. Cooper, Generalized quantifiers and natural language, *Linguistics and Philosophy* 4:159–219, 1981.

[2] van Benthem, J., *Essays in Logical Semantics*, Reidel, Dordrecht, 1986.

[3] van Benthem, J., *Language in Action*, North Holland, Amsterdam, 1991.

[4] van Benthem, J., Natural logic: a view from the 1980's, in M. K. Chakraborty et al. (eds.), *Logic, Navya-Nyāya and Applications*, College Publications, London, 2008.

[5] Giannakidou, A., Negative and positive polarity items, in C. Maienborn, K. von Heusinger, and P. Portner (eds.), *Semantics: An International Handbook of Natural Language Meaning*, Mouton de Gruyter, 2011.

[6] Hoeksema, J., Negative polarity and the comparative, *Natural Language and Linguistic Theory* 1:403–434, 1983.

[7] Ladusaw, W., *Polarity Sensitivity as Inherent Scope Relations*, Ph.D. Dissertation, University of Texas Austin, 1979.

[8] MacCartney, B., *Natural Language Inference*, Ph.D. Dissertation, Stanford University, 2009.

[9] MacCartney, B., and C. D. Manning, Modeling semantic containment and exclusion in natural language inference, *The 22nd International Conference on Computational Linguistics (Coling-08)*, Manchester, 2008.

[10] MacCartney, B., and C. D. Manning, An extended model of natural logic, *Proceedings of the Eighth International Conference on Computational Semantics* , 2009.

[11] Moss, L. S., *Logics for Natural Language Inference*, ESSLLI 2010 Course Notes.

[12] Moss, L. S., The Soundness of Internalized Polarity Marking, *Studia Logica* 100(4):683–704, 2012. (this issue)

[13] Sánchez, V., *Studies on Natural Logic and Categorial Grammar*, Ph.D. Dissertation, Universiteit van Amsterdam, 1991.

[14] Suppes, P., Logical inference in English, *Studia Logica* 38(4):375–391, 1979.

[15] Zamansky, A., N. Francez, and Y. Winter, A 'natural logic' inference system using the Lambek Calculus, *Journal of Logic, Language, and Information* 15(3):273–295, 2006.

[16] Zwarts, F., Three types of polarity, in F. Hamm and E. Hinrichs (eds.), *Plurality and Quantification*, Kluwer, 1998.

Thomas F. Icard, III
Department of Philosophy
Stanford University
Stanford, California, USA
icard@stanford.edu