Monotonicity Reasoning in Formal Semantics Based on Modern Type Theories

G. E. Lungu and Z. Luo*

Royal Holloway, University of London
Egham, Surrey TW20 0EX, U.K.
Georgiana.Lungu.2013@live.rhul.ac.uk
zhaohui.luo@hotmail.co.uk

Abstract. Modern type theories (MTTs) have been developed as a powerful foundation for formal semantics. In particular, it provides a useful platform for natural language inference (NLI) where proof assistants can be used for inference on computers. In this paper, we consider how monotonicity reasoning can be dealt in MTTs, so that it can contribute to NLI based on the MTT-semantics. We show that subtyping is crucial in monotonicity reasoning in MTTs because CNs are interpreted as types and therefore the monotonicity relations between CNs should be represented by the subtyping relations. In the past, monotonicity reasoning has only been considered for the arrow-types in the Montagovian setting of simple type theory. In MTT-semantics, richer type constructors involving dependent types and inductive types are employed in semantic representations. We show how to consider monotonicity reasoning that involve such type constructors and how this offers new useful mechanisms in monotonicity reasoning in MTTs.

1 Introduction

Natural Logic arose from the observation that monotonicity plays an important role in natural language reasoning. Its early studies were conducted in different fields including, for example, Sommers [23] in philosophy, van Benthem [24,25] in logic, Purdy [20] in computer science and Dowty [8] in linguistics. In general, the studies have been based on the Montague semantics (and hence Church's simple type theory) and influenced by the research on generalised quantifiers.

Formal semantics in Modern Type Theories (MTTs), or MTT-semantics for short, was first studied by Ranta in Martin-Löf's type theory [22] and, more recently, developed by the second author and his colleagues in a series of papers such as [14,6]. It provides us with not only a viable alternative to the Montague

* This work is partially supported by the research grant F/07-537/AJ of the Leverhulme Trust in U.K.

Semantics, but potentially an attractive full-blown semantic tool with advantages in many respects. In particular, it provides a useful platform for natural language inference (NLI) where the proof technology provided by proof assistants can be used for inference on computers; for example, there has been work on NLI in the Coq proof assistant based on the MTT-semantics [5].

In this paper, we study monotonicity reasoning in MTTs and this will provide the basis for monotonicity reasoning in NLI based on MTT-semantics. Since CNs are interpreted as types (not as predicates) in MTT-semantics, the notion of monotonicity between CNs needs to be expressed by a relationship between types – here, the subtyping relation plays a crucial role. More precisely, the logical inclusion relationship between CNs N_1 and N_2 is represented by means of the subtyping judgement $N_1 \leq N_2$.[1] We spell out some of the rules with NLI examples for explanation.

In the past, monotonicity reasoning has only been considered for the arrow-types (non-dependent function types) in the Montagovian setting of simple type theory. In MTT-semantics, richer type constructors are employed in semantic representations, examples of which include various dependent types and inductive types such as dependent sum types, dependent product types, disjoint union types, among others. Some other type constructors not in the traditional type theories are also used – an example of this is the dot-types used to represent copredication [14,28]. We show how to consider monotonicity reasoning that involves such type constructors and how this offers new useful mechanisms in monotonicity reasoning in MTTs.

The main contribution of the current paper is two-fold: It shows how to deal with monotonicity reasoning in MTTs, on the one hand, and studies monotonicity reasoning for various type constructors in MTTs and shows that this brings new power of monotonicity reasoning, on the other. We shall start with the introduction of monotonicity reasoning in MTTs in the following section, showing how coercive subtyping is used essentially in monotonicity reasoning in MTTs. Then, in §3, based on a typing of generalised quantifiers in MTTs, we analyze the left (right) increasing (decreasing) monotonicity rules, as it is defined in Montague's grammar but from the perspective of MTTs. Monotonicity involving various type constructors are studied in §4, where we consider as examples Σ-types, Π-types, disjoint union types and vector-types, showing the corresponding rules and NLI reasoning examples involving them. In the concluding remarks, future and some related work is discussed.

[1] Note that, in MTTs, the subtyping relationship $N_1 \leq N_2$ is *not* a logical proposition (of type *Prop*). Rather $N_1 \leq N_2$ is a *judgement*, at the same level as typing judgements like $a : A$. Coercive subtyping [11,17], implemented in the proof assistants Coq [7], Lego [16], Matita [19] and Plastic [2], is an adequate subtyping mechanism for MTTs and is employed here for monotonicity reasoning as well.

2 Monotonicity Reasoning in MTTs: an Introduction

In order to conduct monotonicity reasoning in MTTs and the associated proof assistants, we need to spell out how monotonicity should be formulated in MTTs. The original idea of monotonicity reasoning (see, for example, [25]) is to generalise the notion of logical consequence to arbitrary types in the simple type theory to arrive at a partial order $\Longrightarrow$ for each type. Notationally, we write

$$X \Longrightarrow_A Y$$

to mean that $X \Longrightarrow Y$ for X and Y of type A. For instance, for arrow types, one can define that, for $e, e' : A \to B$, $e \Longrightarrow_{A\to B} e'$ if, and only if, $e(x) \Longrightarrow_B e'(x)$ for all $x : A$. We shall extend this to other types in MTTs and, in this paper, we shall work in the type theory UTT [10].

The basic laws for monotonicity reasoning are:

$$\frac{X \Longrightarrow_A Y \quad F : A \to B \quad F\uparrow}{F(X) \Longrightarrow_B F(Y)} \qquad \frac{X \Longrightarrow_A Y \quad F : A \to B \quad F\downarrow}{F(Y) \Longrightarrow_B F(X)}$$

where $F \uparrow$ and $F \downarrow$ mean that F is $\Longrightarrow$-preserving and $\Longrightarrow$-reversing, respectively.

For some types, the monotonicity order is specialised. For instance, monotonicity on propositions is identified with logical consequence/implication; in symbols,[2]

$$p \Longrightarrow_{Prop} q \quad \text{is defined to be} \quad p \supset q.$$

From this and based on the above rules, it is rather straightforward to derive the following:

- For predicates P and P' of the same domain type A (formally, $P,\ P' : A \to Prop$), $P \Longrightarrow_{A\to Prop} P'$ if, and only if, $P(a) \supset P'(a)$ for every $a : A$.

In an MTT with subtyping, monotonicity on types is identified with the subtyping relation: for types A and B,

$$A \Longrightarrow B \quad \text{if and only if} \quad A \leq B,$$

where $A \leq B$ means that $A \leq_c B$ for some functional operation from A to B. For instance, let CN be the type universe of (the interpretations of) common nouns.[3] Then, for N_1 and N_2 in CN, $N_1 \Longrightarrow_{\text{CN}} N_2$ if, and only if, $N_1 \leq N_2$.

[2] In UTT, *Prop* is the impredicative universe of logical propositions. For those unfamiliar with MTTs, it may also be good to know that every proposition in *Prop* is a type and this fact is used, for example, to form the Σ-type $\Sigma([\![man]\!], [\![handsome]\!])$ where $[\![handsome]\!]$ is a predicate, i.e., a proposition-valued function and hence a type-valued function.

[3] In the MTT-semantics, CNs are interpreted as types. This is different from the Montague semantics, where CNs are interpreted as predicates. See [13] for more information about this.

The above has set up the basic framework for monotonicity reasoning in MTTs. Here are some examples, which are mostly taken from [26] after slight modifications. They are examples that can already be done in simple type theory, though we consider them in the MTT setting. Further examples about binary quantifiers and other MTT type constructors can be found in §3. (We ignore the syntax/semantics difference in the examples.)

- Adverbs as sentence modifiers. For example, when `gracefully` is considered as a sentence modifier, we have: (34) and (35) imply (3). (Such explanations are omitted below.)

  (1) `Mary dances` $\Longrightarrow_{Prop}$ `Mary moves`

  (2) `gracefully` : $Prop \to Prop$ is $\Longrightarrow$-preserving.

  (3) `Mary dances gracefully` $\Longrightarrow_{Prop}$ `Mary moves gracefully`

- Alternatively, one may say that `gracefully` actually modifies verbs such as `dance` and `move`. If analysed like this, we have:

  (4) `dance` $\Longrightarrow_{Human \to Prop}$ `move`

  (5) `gracefully` : $(A : \text{CN})(A \to Prop) \to (A \to Prop)$ and `gracefully`(A) is $\Longrightarrow$-preserving for each A.

  (6) `dance gracefully` $\Longrightarrow_{Human \to Prop}$ `move gracefully`

  Note that, from (6), we obtain (3) as a consequence as well.

- Sentence modifiers such as negation.

  (7) `Mary dances` $\Longrightarrow_{Prop}$ `Mary moves`

  (8) `no` : $Prop \to Prop$ is $\Longrightarrow$-reversing.

  (9) `Mary does not move` $\Longrightarrow_{Prop}$ `Mary does not dance`

- Order-preserving quantifiers.

  (10) `dance` $\Longrightarrow_{Human \to Prop}$ `move`

  (11) `everyone` : $(Human \to Prop) \to Prop$ and is $\Longrightarrow$-preserving.

  (12) `Everyone dances` $\Longrightarrow_{Prop}$ `Everyone moves`

- Order-reversing quantifiers.

  (13) `dance` $\Longrightarrow_{Human \to Prop}$ `move`

  (14) `nobody` : $(Human \to Prop) \to Prop$ and is $\Longrightarrow$-reversing.

  (15) `Nobody moves` $\Longrightarrow_{Prop}$ `Nobody dances`

3 Binary Quantifiers in Monotonicity Reasoning

Generalised quantifiers have been studied extensively in the Montagovian setting (see, for example, [1,27]). In the MTT-semantics, similar considerations can be taken where, for example, a binary quantifier is of the following type:

$$\Pi A : \text{CN}.\ (A \to Prop) \to Prop.$$

In other words, a binary quantifier is an operator Q that takes a common noun N and a predicate P over N to form a proposition $Q(N, P)$. For instance, `every` is a binary quantifier whose semantics $[\![every]\!]$ is of the above type. Therefore, the following sentence (16) has its interpretation (17):

(16) Every student talks.

(17) $[\![every]\!]([\![student]\!], [\![talk]\!])$

where the formula in (17) is well-typed because $[\![student]\!]$ is of type CN and $[\![talk]\!]$ is of type $[\![human]\!] \to Prop$ which is a subtype of $[\![student]\!] \to Prop$.

In the Montagovian setting, a generalised quantifier Q of type $\langle 1, 1\rangle$ is said to be *right (or left) monotone increasing* if $\forall A \subseteq A', B \subseteq B' \subseteq M$, $Q_M(A, B) \supset Q_M(A, B')$ (or $Q_M(A, B) \supset Q_M(A', B)$), where M is the universe we relate to. It is said to be *right (left) monotone decreasing* if $Q_M(A, B') \supset Q_M(A, B)$ (respectively $Q_M(A', B) \supset Q_M(A, B)$).

The binary quantifiers in MTTs can be studied similarly (see also the discussion in 5). A binary quantifier Q of type $\Pi A : \text{CN}.\ (A \to Prop) \to Prop$ takes a 'left' (first) argument A and a 'right' (second) argument B to form a proposition $Q(A, B)$. We can then consider:

- monotonicity (or the reverse monotonicity) in its first argument as expressed by the relation $\Longrightarrow_{CN}$, which is the subtyping relation $\leq$; and
- monotonicity (or the reverse monotonicity) in its second argument as expressed by the relation $\Longrightarrow_{A \to Prop}$, which is the predicate inclusion relation $\subseteq$ (i.e., $B \subseteq B' =_{\text{df}} \forall x : A.\ B(x) \supset B'(x)$).

Treated like this, we have the derived rules in Figure 1, which can be viewed as defining properties of the following notions, for any binary quantifier Q:

- Q is said to be *right increasing* if the rule (MON ↑) holds.
- Q is said to be *right decreasing* if the rule (MON ↓) holds.
- Q is said to be *left increasing* if the rule (↑ MON) holds.
- Q is said to be *left decreasing* if the rule (↓ MON) holds.

Here are some examples for the monotonicity rules (we only spell out the semantic interpretations in the first example):

- An example of left decreasing monotonicity i.e. $\Longrightarrow$-reversing in the first argument is $\forall$, which satisfies (↓ MON); because of (18), we have (19) implies (20) (i.e., (19) $\Longrightarrow_{Prop}$ (20)).

  (18) Scientists are human. (semantics: $[\![scientist]\!] \Longrightarrow_{\text{CN}} [\![human]\!]$)

$$\text{(MON}\uparrow\text{)}\quad \frac{A:\text{CN}\quad B,\ B':A\to Prop\quad B\Longrightarrow_{A\to Prop} B'}{Q(A,B)\Longrightarrow_{Prop} Q(A,B')}\quad (Q\ \texttt{is}\ MON\uparrow)$$

$$\text{(MON}\downarrow\text{)}\quad \frac{A:\text{CN}\quad B,\ B':A\to Prop\quad B'\Longrightarrow_{A\to Prop} B}{Q(A,B)\Longrightarrow_{Prop} Q(A,B')}\quad (Q\ \texttt{is}\ MON\downarrow)$$

$$(\uparrow\text{MON})\quad \frac{A,\ A':\text{CN}\quad B':A'\to Prop\quad A\Longrightarrow_{\text{CN}} A'}{Q(A,B')\Longrightarrow_{Prop} Q(A',B')}\quad (Q\ \texttt{is}\ \uparrow MON)$$

$$(\downarrow\text{MON})\quad \frac{A,\ A':\text{CN}\quad B:A\to Prop\quad A'\Longrightarrow_{\text{CN}} A}{Q(A,B)\Longrightarrow_{Prop} Q(A',B)}\quad (Q\ \texttt{is}\ \downarrow MON)$$

Fig. 1. Derived reasoning rules for binary quantifiers.

(19) Every human is mortal. (semantics: $[\![every]\!]([\![human]\!],[\![mortal]\!])$)

(20) Every scientist is mortal. (semantics: $[\![every]\!]([\![scientist]\!],[\![mortal]\!])$)

– Examples of right increasing binary quantifiers include $\forall$, $\exists$ and $Most$, which are $\Longrightarrow$-preserving w.r.t. its second argument and satisfy (MON $\uparrow$).

(21) Humans are mortal.

(22) Most scientists are human.

(23) Most scientists are mortal.

– Examples of left increasing binary quantifiers include $\exists$, which is $\Longrightarrow$-preserving w.r.t. its first argument and satisfy ($\uparrow$ MON).

(24) Scientists are human.

(25) Some scientists are mortal.

(26) Some humans are mortal.

– Examples of right decreasing binary quantifiers include `none`, which is $\Longrightarrow$-reversing w.r.t. its second argument and satisfy (MON $\downarrow$).

(27) Humans are mortal.

(28) No philosophers are mortal.

(29) No philosophers are human.

4 Monotonicity Reasoning with MTT Type Constructors

MTTs have a rich type structure and, in an MTT-semantics, these types are proved to be very useful in interpretations of various linguistic features. Here

we study monotonicity in this rich type structure and consider some of the type constructors with respect to monotonicity reasoning.[4]

Dependent sums (Σ-types). An example of dependent types is *dependent sums* or *Σ-types* which have been employed in the MTT-semantics to interpret modified CNs. For example, "handsome men" can be interpreted as $\Sigma(Man, handsome)$ (or $\Sigma m{:}Man.\ handsome(m)$ in a more traditional notation), where $handsome : Human \to Prop$. There are two natural subtyping relationships that can be considered.

– In the first place, $\Sigma(Man, handsome) \leq_{\pi_1} Man$ because a handsome man is a man. In general, we have

$$\Sigma(A, B) \leq_{\pi_1} A.$$

In other words, when A interprets a CN and $\Sigma(A, B)$ a modified CN, we have

$$\Sigma(A, B) \Longrightarrow_{\text{CN}} A.$$

For example, we have that (30) implies (31).

(30) John is a handsome man.

(31) John is a man.

– Another subtyping relationship for Σ-types describes its structural relationships of its parameters. Informally, if A is a subtype of A' and B is a 'subtype' of B', then $\Sigma(A, B)$ is a subtype of $\Sigma(A', B')$. Formally, we have (for simplicity, we omit some premises), where we assume that $A,\ A' : \text{CN}$, $B : A \to Prop$ and $B' : A' \to Prop$:

$$(*) \qquad \frac{A \Longrightarrow_{\text{CN}} A' \quad B \Longrightarrow_{A \to Prop} B'}{\Sigma(A, B) \Longrightarrow_{\text{CN}} \Sigma(A', B')}$$

For example, we have that (32) implies (33):

(32) John is a smart and diligent student.

(33) John is a smart human.

When $[\![smart\ and\ diligent\ students]\!]$ and $[\![smart\ human]\!]$ are interpreted as $\Sigma(Student, S\&D)$ and $\Sigma(Human, S)$, respectively, the reasoning from (32) to (33) follows from the above rule. Note that, in this example, both premises involve strict monotonicity relations and are used essentially.

[4] Since the monotonicity relationship $\Longrightarrow$ over types is expressed by the subtyping relation $\leq$, the reader may be interested in the general study of the latter and, if so, please consult the papers [11,15].

Disjoint union types. When A and B are types, we can form the disjoint union type $A + B$. This is employed in [4] to interpret privative adjectives like fake. Let us consider the type Car and the classification "fake cars" and "real cars". A car exclusively belongs to one of these two categories. In this case we say $Car = Car_{fake} + Car_{real}$. By disjoint union types are used to interpret privative adjectives, as the above example illustrates, both of the injections $inl : A \to A + B$ and $inr : B \to A + B$ are declared as coercions. For instance, we regard either a real car or a fake car as a car.

For disjoint union types, we can consider the following rule:

$$\frac{A \leq A' \quad B \leq B'}{A + A' \leq B + B'}$$

One may think that a real (fake) car is a real (fake) vehicle and hence, if we define $Vehicle = Vehicle_{fake} + Vehicle_{real}$, we have from the above rule that

$$Car = Car_{fake} + Car_{real} \leq Vehicle_{fake} + Vehicle_{real} = Vehicle.$$

Dependent products (Π-types). Dependent Π-types are a basic form of dependent types. In MTT-semantics, it is employed for many purposes and has proved very useful. For instance, it is used in [14] which models adverbs acting as verb modifiers. In general, in MTT-semantics, a verb-modifying adverb is of the following type:

$$\Pi A : \text{CN}.\ (A \to Prop) \to (A \to Prop).$$

For example, the adverb $gracefully$ in the examples in §2 is interpreted as of the above type.

For Π-types, we can introduce the following monotonicity rule, where $A,\ A' : \text{CN}$, $B : A \to Type$ and $B' : A' \to Type$:[5]

$$\frac{A \Longrightarrow_{\text{CN}} A' \quad B \Longrightarrow_{A \to Type} B'}{\Pi(A, B) \Longrightarrow_{Type} \Pi(A', B')}$$

Vector types. For $A : CN$ and $n : Nat$, $Vec(A, n)$ is the inductive type of vectors whose lengths are n. Vector types may be used to deal with some linguistic features such as those involve collective verbs [3]. For instance, we may interpret meet as of the following type:

$$[\![meet]\!] : \Pi n : Nat.\ Vec([\![human]\!], n + 2) \to Prop.$$

The vector-types are monotone increasing for every n with the following rules:

$$\frac{A \Longrightarrow_{\text{CN}} A' \quad n : Nat}{Vec(A, n) \Longrightarrow_{Type} Vec(A', n)}$$

For instance, we have (34) implies (35).

(34) Three men met.

(35) Three humans met

[5] Here, $Type$ is a type universe (and hence the formation of $A \to Type$ as a type is legal). This is a technicality that we may ignore here.

Dot-types. These types are not inductive types found in traditional MTTs. They were introduced to model the linguistic feature of copredication [12]. If A and B are types do not share common components, then we can form a dot-type $A \bullet B$ for which we have two coercions from $A \bullet B$ to A and from $A \bullet B$ to B. For instance, in a sentence like

John picked up the book and mastered it,

book can be interpreted as of type $Phy \cdot Info$, where Phy and $Info$ are type of physical objects and informational objects, respectively. Indeed, this sentence implies both that 'John picked up the book (as a physical object) and 'John mastered the book (as an informational object)'. The monotonicity rule for dot-types is:

$$\frac{A \leq A' \quad B \leq B'}{A \bullet B \leq A' \bullet B'}$$

At the time of writing this paper, we have some rather artificial NL examples for this and it would be interesting to find some real NLI examples concerning dot-types.

5 Concluding Remarks

We have considered monotonicity reasoning in the MTT setting and developed the corresponding rules for the relevant type constructors. An obvious future work is to implement this, extending the NLI machinery in the Coq proof assistant developed by Chatzikyriakidis [5]. It is expected that this would improve the reasoning engine as a whole.

Reasoning in Natural Logic has been studied by several researchers, including MacCartney's work in implementing an NLI system based on monotonicity reasoning [18]. Embedding monotonicity reasoning in MTTs allows simpler ways of reasoning in many cases and its automation in proof assistants is expected. We can then compare this to MacCartney's work. In this respect, we should also mention the recent work of implementing natural logic in the Coq system [9], which is based on the Montagovian setting of simple type theory (without using Coq's MTT mechanisms in formal semantics).

In this paper, while analyzing monotonicity and the impact the coercive subtyping relation has, we touched a couple of points that could be the basis for further research. The work reported here is part of the development of a theory of generalised quantifiers in the MTT framework, hopefully to offer an richer alternative to than that in the Montagovian setting. For instance, when considering a binary quantifier $Q(A, B)$ in an MTT, the domain of the second argument B needs to be a supertype of the first argument A (since $D_B \rightarrow Type \leq A \rightarrow Type$ if $A \leq D_B$). Furthermore, the quantifier properties such as

conservativity and extensibility are also closed related to the situations in which these quantifiers are evaluated to see whether they are satisfied. This leads to the current work of the second author on representing (possibly infinite) situations or incomplete possible worlds by means of (finite) contexts in type theory. Some promising results have been obtained in this direction (and are to be reported elsewhere).

Another interesting direction is to study how to interpret mass nouns in MTTs. In [13], the second author has taken a set/type-theoretic view (instead of a mereological view as advocated by Quine [21] and others) and used the intuition of languages with classifiers like Chinese to motivate some uses of mass nouns: eg, in `a bucket of water`, the measure word (or classifier) is `bucket`, that provides the criterion of equality associated with CNs in general. It would be interesting to study this in connection with plurals and collective verbs, although further work is called for about their interpretations in MTTs.

For some topics such as the study of proof rules for monotonicity, a categorical framework may help to arrive at better understandings. For instance, it would be interesting to look at the relationships between $\Sigma(Human, handsome)$ and Man, for $Man \leq_c Human$, from a categorial perspective. Consider the following two derivations (due to transitivity of the coercive subtyping relations and the $(*)$ rule in §4):

$$\frac{\Sigma(Man, handsome) \leq_{\pi_1} Man \quad Man \leq_c Human}{\Sigma(Man, handsome) \leq_{c \circ \pi_1} Human}$$

$$\frac{\Sigma(Man, handsome) \leq_{c_\Sigma} \Sigma(Human, handsome) \quad \Sigma(Human, handsome) \leq_{\pi_1} Human}{\Sigma(Man, handsome) \leq_{\pi_1 \circ c_\Sigma} Human}$$

where $\circ$ is the composition operator and c_Σ is the coercion for the conclusion of the rule $(*)$ in §4, where this coercion is omitted. Coherence in coercive subtyping requires uniqueness of coercion and, in this example, we should have $\pi_1 \circ c_\Sigma = c \circ \pi_1$. This means that the following diagram should commute:[6]

$$\begin{array}{ccc} \Sigma(Man, handsome) & \xrightarrow{\pi_1} & Man \\ \downarrow{\scriptstyle c_\Sigma} & & \downarrow{\scriptstyle c} \\ \Sigma(Human, handsome) & \xrightarrow[\pi_1]{} & Human \end{array}$$

Furthermore, we may refine the type Man to be defined as, where $male : Human \to Prop$:

$$Man = \Sigma(Human, male),$$

[6] Note that there is also the interesting difference between the intensional equality in MTTs and the extensional equality in category theory, although the latter gives us interesting clues on some of the research challenges.

and then the situation is linked to the following diagram:

$$\begin{array}{ccc} \Sigma(Human, male \wedge handsome) & \xrightarrow{\pi_1} & Human \\ {\scriptstyle handsome^* male}\downarrow & & \downarrow{\scriptstyle handsome} \\ Human & \xrightarrow[male]{} & Prop \end{array}$$

which connects logical conjunctions to pullbacks in the category theory.

Acknowledgement We'd like to thank Nicholas Asher, Stergios Chatzikyriakidis and the anonymous LACL reviewers for helpful comments to an earlier version, which have helped a lot in improving the paper.

References

1. J. Barwise and R. Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4(2):159–219, 1981.
2. P. Callaghan and Z. Luo. An implementation of LF with coercive subtyping and universes. *Journal of Automated Reasoning*, 27(1):3–27, 2001.
3. S. Chatzikyriakidis and Z. Luo. An account of natural language coordination in type theory with coercive subtyping. *Lecture Notes in Computer Science*, 8114:31–48, 2012.
4. S. Chatzikyriakidis and Z. Luo. Adjectives in a modern type-theoretical setting. *Formal Grammar 2013*, Lecture Notes in Computer Science 8036:159–174, 2013.
5. S. Chatzikyriakidis and Z. Luo. Natural language inference in Coq. *EACL Workshop on Type Theory and Natural Language Semantics. Goteborg*, 2013. Submitted manuscript.
6. S. Chatzikyriakidis and Z. Luo. Natural language reasoning using proof assistant technology: Rich typing and beyond. *EACL Workshop on Type Theory and Natural Language Semantics. Goteborg*, 2014.
7. The Coq Development Team. *The Coq Proof Assistant Reference Manual (Version 8.1), INRIA*, 2007.
8. D. Dowty. The role of negative polarity and concord marking in natural language reasoning. *Proc of the 4th SALT Conference*, 1994.
9. Rutger Huijben. A Coq module for natural logic. Master's thesis, Technische Universiteit Eindhoven, 2013.
10. Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*. Oxford University Press, 1994.
11. Z. Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(1):105–130, 1999.
12. Z. Luo. Type-theoretical semantics with coercive subtyping. *Semantics and Linguistic Theory 20 (SALT20), Vancouver*, 2010.
13. Z. Luo. Common nouns as types. In D. Bechet and A. Dikovsky, editors, *Logical Aspects of Computational Linguistics (LACL'2012). LNCS 7351*, 2012.

14. Z. Luo. Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophy*, 35(6):491–513, 2012.
15. Z. Luo and R. Adams. Structural subtyping for inductive types with functorial equality rules. *Mathematical Structures in Computer Science*, 18(5), 2008.
16. Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Dept of Computer Science, Univ of Edinburgh, 1992.
17. Z. Luo, S. Soloviev, and T. Xue. Coercive subtyping: theory and implementation. *Information and Computation*, 223, 2012.
18. B. MacCartney. *Natural Language Inference*. PhD thesis, Stanford Universisty, 2009.
19. The Matita proof assistant. Available from: `http://matita.cs.unibo.it/`, 2008.
20. W. C. Purdy. A logic for natural language. *Notre Dame Journal of Formal Logic*, 32, 1991.
21. W. Quine. *Word & Object*. MIT Press, 1960.
22. A. Ranta. *Type-Theoretical Grammar*. Oxford University Press, 1994.
23. F. Sommers. *The Logic of Natural Language*. Cambridge Univ Press, 1982.
24. J. van Benthem. *Essays in Logical Semantics*. Reidel, 1986.
25. J. van Benthem. *Language in Action: Categories, Lambdas and Dynamic Logic*. Elsevier, 1991.
26. J. van Eijck. Natural logic for natural language. *LNCS 4363*, 2007.
27. D. Westerstahl and S. Peters. *Quantifiers in Language and Logic*. Oxford University Press, 2006.
28. T. Xue and Z. Luo. Dot-types and their implementation. *Logical Aspects of Computational Linguistics (LACL 2012). LNCS 7351*, 2012.