Noname manuscript No.
(will be inserted by the editor)

Natural Language Inference in Coq

Stergios Chatzikyriakidis · Zhaohui Luo

Received: date / Accepted: date

Abstract In this paper we propose a way to deal with Natural Language Inference (NLI) by implementing Modern Type Theoretical semantics in the proof-assistant Coq. The paper is a first attempt to deal with NLI and Natural Language reasoning in general by using proof-assistant technology. Valid NLIs are treated as theorems and as such the adequacy of our account is tested by trying to prove them. We use Luo's Modern Type Theory with coercive subtyping as the formal language in which we translate Natural Language semantics to, and we further implement these semantics in the Coq proof-assistant. It is shown that the use of a Modern Type Theory with an adequate subtyping mechanism can give us a number of promising results as regards NLI. Specifically, it is shown that a number of inference cases, i.e. quantifiers, adjectives, conjoined Noun Phrases and temporal reference among others can be successfully dealt with. It is then shown, that even though Coq is an interactive and not an automated theorem prover, automation of all of the test examples is possible by introducing user-defined automated tactics. Lastly, the paper offers a number of innovative approaches to NL phenomena like adjectives, collective predication, comparatives and factive verbs among others, contributing in this respect to the theoretical study of formal semantics using Modern Type Theories.

Stergios Chatzikyriakidis
Dept of Computer Science, Royal Holloway, Univ of London
Egham, Surrey TW20 0EX, U.K; Open University of Cyprus
E-mail: stergios.chatzikyriakidis@cs.rhul.ac.uk

Zhaohui Luo
Dept of Computer Science, Royal Holloway, Univ of London
Egham, Surrey TW20 0EX, U.K
E-mail: zhaohui.luo@hotmail.co.uk

1 Introduction

Natural Language Inference (NLI), i.e. the task of determining whether an NL hypothesis can be inferred from an NL premise, has been an active research theme in computational semantics in which various approaches have been proposed (see, for example [26] and some of the references therein). In this paper, we study NLI based on formal semantics in MTTs with coercive subtyping [24] and its implementation in the proof assistant Coq [11].

A *Modern Type Theory* (MTT) is a dependent type theory consisting of an internal logic, which follows the propositions-as-types principle. This latter feature along with the availability of powerful type structures make MTTs very useful for formal semantics. Research on MTTs has been extremely fruitful in analyzing NL semantics and a number of problematic phenomena in NL semantics have been managed to be tackled. Earlier work by Sunbdholm [36] and Ranta [34], among others, have managed to deal in a rather adequate way with a number of semantic phenomena like e.g quantifiers, anaphora and donkey sentences among others. The second author of the current paper has further developed MTT-based semantics via employing the impredicative type theory UTT [19] enriched by an adequate subtyping mechanism, i.e. coercive subtyping [20, 25]. MTT semantics has now gradually becoming a serious alternative to Montague semantics [30] as regards formal semantics.

A proof assistant is a computer system that assists the users to develop proofs of mathematical theorems. A number of proof assistants implement MTTs. For instance, the proof assistant Coq [11] implements pCIC, the predicative Calculus of Inductive Constructions[1] and supports some very useful tactics that can be used to help the users to automate (parts of) their proofs. Proof assistants have been used in various applications in computer science (e.g., program verification) and formalised mathematics (e.g., formalisation of the proof of the 4-colour theorem in Coq).

The above two developments, the use of MTT semantics on the one hand and the implementation of MTTs in proof assistants on the other, has opened a new research avenue: the use of existing proof assistants in dealing with NLI. In this paper, we present our work as regards NLI by implementing MTT semantics for NL in Coq. The purpose is to show how an interactive proof assistant such as Coq can help deal with NLI. In particular, we implement MTT semantics in Coq and then use Coq to reason about these semantics by dealing with various examples from the FraCas test suite [10]. What we would like to show is that a large class of NLI cases can be straightforwardly dealt with under this approach, which basically treats NLIs as valid theorems in Coq. Also, we believe that in many cases we have given satisfactory and innovative semantic treatments of various semantic phenomena like e.g. the generalization of Σ types not only to adjectives but to VP adverbs, comparatives and factive

[1] pCIC is a type theory that is rather similar to UTT, especially after its universe *Set* became predicative since Coq 8.0. A main difference is that UTT does not have co-inductive types. The interested reader is directed to Goguen's PhD thesis [16] as regards the meta-theory of UTT.

verbs, which we believe are useful in themselves. Furthermore, it is shown that many NLI cases, in fact all of the NLI cases we have dealt with, can be automatedly performed by an automated combination of Coq's in-built proof tactics.

The paper is structured as follows: in §2 we present an introduction to formal semantics based on MTTs. Specifically, we concentrate on how to use type theory, in particular the Unified Theory of dependent Types (UTT) with the addition of coercive subtyping, to represent NL formal semantics. In §3, we present a very short introduction to the way the Coq proof-assistant works. In §4, we discuss the implementation of MTT semantics in Coq, in order to deal with various examples from the FraCas test suite. Lastly, the issue of doing automated theorem proving in interactive theorem provers is discussed in the final section, together with some informal comparison with other relevant work as well as some directions from future work.

2 Formal Semantics in Modern Type Theories

In this section, we give a brief introduction to the formal semantics based on Modern Type Theories (MTTs) [34,21,24]. A Modern Type Theory (MTT) is a variant of a class of type theories in the tradition initiated by the work of Martin-Löf [27,28], which have dependent types and inductive types, among others. We choose to call them Modern Type Theories in order to distinguish them from Church's simple type theory [9] that is commonly employed within the Montagovian tradition in formal semantics.

Among the variants of MTTs, we are going to employ the Unified Theory of dependent Types (UTT) [19] with the addition of the coercive subtyping mechanism (see, for example, [20,25] and below). UTT is an impredicative type theory in which a type *Prop* of all logical propositions exists.[2] This stands as part of the study of linguistic semantics using MTTs rather than simply typed ones, including the early studies such as [36,34] inter alios.

2.1 Formal Semantics Based on MTTs: the Basics

In semantics based on MTTs, the basic ways to interpret various linguistic categories are as follows, with basic examples shown in Figure 1, where we also compare them to those in Montague semantics:

- A sentence (S) is interpreted as a proposition of type *Prop*.
- A common noun (CN) can be interpreted as a type.
- A verb (IV) can be interpreted as a predicate over the type D that interprets the domain of the verb (ie, a function of type $D \to Prop$).
- An adjective (ADJ) can be interpreted as a predicate over the type that interprets the domain of the adjective (ie, a function of type $D \to Prop$).

[2] This is similar to simple type theory where a type t of truth values exists.

	Example	Montague semantics	MTT-based semantics
CN	man, human	$[\![man]\!]$, $[\![human]\!] : e \to t$	$[\![man]\!]$, $[\![human]\!] : Type$
IV	talk	$[\![talk]\!] : e \to t$	$[\![talk]\!] : [\![human]\!] \to Prop$
ADJ	handsome	$[\![handsome]\!] : (e \to t) \to (e \to t)$	$[\![handsome]\!] : [\![man]\!] \to Prop$
MCN	handsome man	$[\![handsome]\!]([\![man]\!])$	$\Sigma m : [\![man]\!].\ [\![handsome]\!](m) : Type$
S	A man talks	$\exists m : e.\ [\![man]\!](m) \& [\![talk]\!](m)$	$\exists m : [\![man]\!].\ [\![talk]\!](m) : Prop$

Fig. 1 Examples in formal semantics.

– Modified common nouns (MCNs) can be interpreted by means of Σ-types (see below).

In what follows, we shall give further explanations of various aspects of MTT-based semantics, explicating along the way the basic features of MTTs and coercive subtyping. We try to bring out the linguistic relevance of the system used rather than being meticulous as regards the formal details in each case.

2.2 Common Nouns as Types and Many-sortedness of MTTs

A key difference between formal semantics based on MTTs and Montague semantics lies in the interpretation of common nouns (CNs) which is in turn based on the fact that MTTs are essentially 'many-sorted' logical systems.

In Montague semantics [30], the underlying logic (Church's simple type theory [9]) can be seen as 'single-sorted' in the sense that there is only one type e of all entities. The other types such as t of truth values and the function types generated from e and t do not stand for types of entities. In this respect, there are no fine-grained distinctions between the elements of type e and as such all individuals are interpreted using the same type. For example, *John* and *Mary* have the same type in simple type theories, the type e of individuals. An MTT, on the other hand, can be regarded as a 'many-sorted' logical system in that it contains many types and as such one can make fine-grained distinctions between individuals and further use those different types to interpret subclasses of individuals. For example, one can have $John : [\![man]\!]$ and $Mary : [\![woman]\!]$, where $[\![man]\!]$ and $[\![woman]\!]$ are different types.

An important trait of MTT-based semantics is the interpretation of common nouns (CNs) as *types* [34] rather than sets or predicates (i.e., objects of type $e \to t$) as it is the case within the Montagovian tradition. The CNs *man*, *human*, *table* and *book* are interpreted as types $[\![man]\!]$, $[\![human]\!]$, $[\![table]\!]$ and $[\![book]\!]$, respectively. Then, individuals are interpreted as being of one of the types used to interpret CNs. Modified common nouns (MCNs in Figure 1) can be interpreted by means of Σ-types, types of dependent pairs. For instance, 'handsome man' can be interpreted as the type $\Sigma m : [\![man]\!].\ [\![handsome]\!](m)$, the type of pairs of a man and a proof that the man is handsome.

This many-sortedness (i.e., the fact that there are many types in an MTT) has the welcoming result that a number of semantically infelicitous sentences

like *the ham sandwich walks*, which are however syntactically well-formed, can be explained easily given that a verb like *walk* will be specified as being of type $Animal \to Prop$ while the type for the ham sandwich will be $[\![food]\!]$ or $[\![sandwich]\!]$, which is not compatible with the typing for *walk*:[3]

(1) $the\ ham\ sandwich : [\![food]\!]$

(2) $walk : [\![human]\!] \to Prop$

The idea of common nouns being interpreted as types rather than predicates has been argued in [23] on philosophical grounds as well. There, the second author argues that Geach's observation that common nouns in contrast to other categories have criteria of identity that enable common nouns to be compared, counted or quantified, has an interesting link with the constructive notion of set/type: in constructive mathematics, sets (types) are not constructed only by specifying their objects but they additionally involve an equality relation. The argument is then that the interpretation of CNs as types in MTTs is explained and justified to a certain extent.[4]

Interpreting CNs as types rather than predicates has also a significant methodological implication: the various subtyping relations one may consider in formal semantics become compatible. For instance, in representing NL semantics, one may introduce various subtyping relations by postulating a collection of subtypes (physical objects, informational objects, eventualities, etc.) of the type of entities [1]. It is clear that, if CNs are interpreted as predicates as in the traditional Montagovian setting, introducing such subtyping relations would cause major problems: even some basic semantic interpretations would go wrong and it is very difficult to deal with some linguistic phenomena like e.g. copredication satisfactorily. Instead, if CNs are interpreted as types, as in Type Theoretical semantics based on MTTs, copredication can be given a straightforward and satisfactory treatment [21].

2.3 Subtyping in Formal Semantics

As briefly explained above, because of many-sortedness of MTTs, CNs can be interpreted as types. For instance, in a Montagovian setting, all of the verbs below are given the same type $e \to t$, but in an MTT, we can have

(3) $drive : [\![human]\!] \to Prop$

(4) $eat : [\![animal]\!] \to Prop$

(5) $disappear : [\![object]\!] \to Prop$

which have different domain types. This has the advantage of disallowing interpretations of some infelicitous examples like *the ham sandwich walks*.

[3] This is based of course on the assumption that the definite NP is of a lower type and not a Generalized Quantifier. More on Generalized Quantifiers and the determiner *the* later on in this chapter.

[4] See [23] for more details on this.

However, interpreting CNs by means of different types could lead to serious undergeneralizations without a subtyping mechanism. For instance, consider the interpretation of the sentence 'A man talks' in Figure 1: for m of type $[\![man]\!]$ and $[\![talk]\!]$ of type $[\![human]\!] \to Prop$, the function application $[\![talk]\!](m)$ is only well-typed because we have that $[\![man]\!]$ is a subtype of $[\![human]\!]$.

Coercive subtyping [20,25] provides an adequate framework to be employed for MTT-based formal semantics [21,24].[5] It can be seen as an abbreviation mechanism: A is a (proper) subtype of B ($A < B$) if there is a unique implicit coercion c from type A to type B and, if so, an object a of type A can be used in any context $\mathfrak{C}_B[\_]$ that expects an object of type B: $\mathfrak{C}_B[a]$ is legal (well-typed) and equal to $\mathfrak{C}_B[c(a)]$.

As an example, in the case that both $[\![man]\!]$ and $[\![human]\!]$ are base types, one may introduce the following as a basic subtyping relation:

(6) $[\![man]\!] < [\![human]\!]$

In case that $[\![man]\!]$ is defined as a composite Σ-type (see §2.4 below for details), where $male\colon [\![human]\!] \to Prop$:

(7) $[\![man]\!] = \Sigma h\colon [\![human]\!].\ male(h)$

we have that (6) is the case because the above Σ-type is a subtype of $[\![human]\!]$ via the first projection π_1:

(8) $(\Sigma h\colon [\![human]\!].\ male(h)) <_{\pi_1} [\![human]\!]$

Equipped with this coercive subtyping mechanism, the undergeneration problems can be straightforwardly solved while still retaining the ability to rule out semantically infelicitous cases like *the ham sandwich walks*. In effect, many-sortedness in MTTs turns out to be superior than single sortedness in the simple type theories, at least in this respect. Furthermore, many inferences involving monotonicity on the first argument in generalized quantifiers can be directly captured using the subtyping mechanism. An inference of the sort exemplified in example (12) below, can be captured given that $[\![man]\!] < [\![human]\!]$:

(9) Some man runs ⇒ Some human runs

Thus, an $x\colon [\![man]\!]$ can be used as an $x\colon [\![human]\!]$, and as such the inference goes through for 'free' in a way.[6]

[5] It is worth mentioning that subsumptive subtyping, i.e. the traditional notion of subtyping that adopts the subsumption rule (if $A \leq B$, then every object of type A is also of type B), is inadequate for MTTs in the sense that it would destroy some important metatheoretical properties of MTTs (see, for example, §4 of [25] for details).

[6] These kinds of inferences can be straightforwardly proven in Coq by using a standard analysis for quantifier *some* plus the subtyping relation $[\![man]\!] < [\![human]\!]$.

2.4 Some Type Constructions in MTTs

In this subsection, we shall discuss several type constructors as well as some more advanced features of MTTs (like for example universes) focusing on the way these can be used in formal semantics.

Dependent Σ-types. One of the basic features of MTTs is the use of Dependent Types. A dependent type is a family of types depending on some values. Here we explain two basic constructors for dependent types, Σ and Π, both highly relevant for the study of linguistic semantics.

The constructor/operator Σ is a generalization of the Cartesian product of two sets that allows the second set to depend on values of the first. For instance, if $[\![human]\!]$ is a type and $male\colon [\![human]\!] \to Prop$, then the Σ-type $\Sigma h\colon [\![human]\!].\ male(h)$ is intuitively the type of humans who are male.

More formally, if A is a type and B is an A-indexed family of types, then $\Sigma(A, B)$, or sometimes written as $\Sigma x : A.B(x)$, is a type, consisting of pairs (a, b) such that a is of type A and b is of type $B(a)$. When $B(x)$ is a constant type (i.e., always the same type no matter what x is), the Σ-type degenerates into product type $A \times B$ of non-dependent pairs. Σ-types (and product types) are associated projection operations π_1 and π_2 so that $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$, for every (a, b) of type $\Sigma(A, B)$ or $A \times B$.

The linguistic relevance of Σ-types can be directly appreciated once we understand that in its dependent case, Σ-types can be used to interpret linguistic phenomena of central importance, like for example adjectival modification [34].[7] For example, *handsome man* is interpreted as a Σ-type (80), the type of handsome men (or more precisely, of those men together with proofs that they are handsome):

(10) $\Sigma m\colon [\![man]\!].\ [\![handsome]\!](m)$

where $[\![handsome]\!](m)$ is a family of propositions/types that depends on the man m.

Adjectival modification is however notoriously difficult to deal with, given that besides examples of adjectives like *black* or *handsome*, there exists a number of other more difficult categories, i.e. privative adjectives like e.g. *fake* or non-committal adjectives like *alleged*. Within the Montagovian tradition, these different adjectival categories have been dealt with by using a number of meaning postulates in each case. [8] have proposed a way of dealing with all adjectival categories using the framework presented in this paper. In particular, it was shown that meaning postulates were not needed for most of the cases, the exception being cases of non-committal adjectives. The idea in this paper is to use typing alone to capture what in the Montagovian tradition is done via means of meaning postulates. The interested reader is directed there for more information on the issue.

[7] Σ-types can also provide the tools for the proper semantic interpretation of the so-called 'Donkey-sentences' [36].

Dependent Π-types The other basic constructor for dependent types is Π. Π-types can be seen as a generalization of the normal function space where the second type is a family of types that might be dependent on the values of the first. A Π-type degenerates to the function type $A \to B$ in the non-dependent case. In more detail, when A is a type and P is a predicate over A, $\Pi x : A.P(x)$ is the dependent function type that, in the embedded logic, stands for the universally quantified proposition $\forall x : A.P(x)$. For example, the following sentence (11) is interpreted as (12):

(11) Every man walks.

(12) $\Pi x\colon [\![man]\!].[\![walk]\!](x)$

Π-types are very useful in formulating the typings for a number of linguistic categories like VP adverbs or quantifiers. The idea is that adverbs and quantifiers range over the universe of (the interpretations of) CNs and as such we need a way to represent this fact. For this reason, Π-types can be used, universally quantifying over the universe CN. (13) the type for VP adverbs[8] while (14) is the type for quantifiers:

(13) $\Pi A\colon \textsc{cn}.\ (A \to Prop) \to (A \to Prop)$

(14) $\Pi A\colon \textsc{cn}.\ (A \to Prop) \to Prop$

Further explanations of the above types will be given after we have introduced the concept of type universe below.

Type Universes. An advanced feature of MTTs, which will be shown to be very relevant in interpreting NL semantics, is that of universes. Informally, a universe is a collection of (the names of) types put into a type [28].[9] For example, one may want to collect all the names of the types that interpret common nouns into a universe $\textsc{cn}\colon Type$. The idea is that for each type A that interprets a common noun, there is a name $\overline{A}$ in CN. For example,

$$\overline{[\![man]\!]}\colon \textsc{cn} \quad \text{and} \quad T_{\textsc{cn}}(\overline{[\![man]\!]}) = [\![man]\!].$$

In practice, we do not distinguish a type in CN and its name by omitting the overlines and the operator $T_{\textsc{cn}}$ by simply writing, for instance, $[\![man]\!]\colon \textsc{cn}$. Thus, the universe includes the collection of the names that interpret common nouns. For example, in CN, we shall find the following types:

(15) $[\![man]\!]$, $[\![woman]\!]$, $[\![book]\!]$, ...

[8] This was proposed for the first time in [22].

[9] There is quite a long discussion on how these universes should be like. In particular, the debate is largely concentrated on whether a universe should be predicative or impredicative. A strongly impredicative universe U of all types (with $U\colon U$ and Π-types) is shown to be paradoxical [15] and as such logically inconsistent. The theory UTT we use here has only one impredicative universe *Prop* (representing the world of logical formulas) together with infinitely many predicative universes which as such avoids Girard's paradox (see [19] for more details).

(16) $\Sigma m\colon [\![man]\!].[\![handsome]\!](m)$
(17) $G_R + G_F$

where the Σ-type in (19 is the proposed interpretation of 'handsome man' and the disjoint sum type in (17) is that of 'gun' (the sum of real guns and fake guns – see above).

Having introduced the universe CN, it is now possible to explain (13) and (14). The type in (14) says that for all elements A of type CN, one gets the function type $(A \to Prop) \to Prop$. The idea is that the element A becomes the type used. To illustrate how this works let us imagine the case of quantifier *some* which has the typing in (14). The first argument needed, has to be of type CN. Thus *some human* is of type $([\![human]\!] \to Prop) \to Prop$ given that the A here is $[\![human]\!]\colon$ CN (A becomes the type $[\![human]\!]$ in $([\![human]\!] \to Prop) \to Prop$). Then given a predicate like $walk\colon [\![human]\!] \to Prop$, we can apply *some human* to get $[\![some\ human]\!]([\![walk]\!])\colon Prop$.

The idea of universes has been proved useful in giving an account of NL coordination in an MTT. Specifically, in [**?**], we have introduced a universe of Linguistic Types, *LType* to capture the flexibility associated with NL coordination.

3 NL Semantics and Inference in Coq: an Introduction

Coq is a dependently typed interactive theorem prover, implementing the calculus of Inductive Constructions (pCiC, see [11]). Coq, and in general proof-assistants, provide assistance in the development of formal proofs. Specifically, the use of Coq has been extremely fruitful and a number of exciting results have been produced via its use, notably the proof of the four-colour theorem (see [17]) and CompCert, a formally verifiable compiler for C (see[3]) among others. The idea is simple: you use Coq in order to see whether statements as regards anything that has been either pre-defined or user-defined (definitions, parameters, variables) can be proven or not. In order to see how this works, imagine the following three variables, P Q and R of type $Prop$. One may want to check whether the following statement involving these variables is a theorem:

(18) $(P \to Q) \to (Q \to R) \to (P \to R)$

In order to do this, you just tell Coq that this is a theorem to be proven:

(19) $Theorem\ Propositional : (P \to Q) \to (Q \to R) \to (P \to R)$

This will put Coq into proof-mode, where the user is asked to interactively guide the assistant to the proof. In order to do this, the user has a number of proof-tactics that s/he can use. More complicated tactics can be further defined and a number of libraries with complementary tactics exist. But how

can such an assistant be used in order to reason about NL semantics? This is to what we turn in the next section. For more introductory information about proving Theorems in Coq, please see Appendix A.1 explicating the proof of (19) as well as further more advanced proof techniques (A.2).

Coq implements an MTT (pCiC). For this reason it is highly suitable to implement MTT semantics, given that this is quite close to the language "it understands" so to say. Indeed, this has been already noted by Luo and colleagues and some first attempts at implementing MTT semantics in Coq have been made [21,22,7]. However, this is not all that Coq has to offer. Given that Coq is a powerful theorem prover, it can further reason about the implemented semantics. In fact, one may very well use Coq's proving ability to prove valid NL inferences, in the same sense it is used for proving valid mathematical or logical theorems. Given that semantic entailment corresponds to an implication relation between two different semantic structures, entailment relations can be translated into constructed theorems that need to be proven. In such a context, a valid semantic entailment will very simply mean the implication relation between the two semantic structures is a valid theorem. A very simple case of semantic entailment, that of example (20), will therefore be formulated as the following theorem in Coq (21):

(20) John walked $\Rightarrow$ some man walked

(21) Theorem ex: John walked $\rightarrow$ some man walked

Then depending on the semantics of the individual lexical items one may or may not prove the theorem that needs to be proven in each case. Inferences like the one shown in (21) are easy cases in Coq. Assuming the semantics of *some* which specify that given any A of type CN and a predicate of type $A \rightarrow Prop$, there exists an $x\colon A$ such that $P(x)\colon Prop$, such cases are straightforwardly proven.[10]

A few notes about the lexical entries. We use Coq's $Prop$ type, corresponding roughly to the type of truth-values (t) in Montague Semantics. We define CN to be of type *set* and interpret CNs like *man, human* as being of type CN (thus we have for example $man, human\colon CN$).[11] Verbs are defined as predicates requiring types $A\colon CN$ as their arguments. The exact type of this A argument depends on the verb itself. Thus, *walks* is defined as being of type $human \rightarrow Prop$ whereas eat as $animal \rightarrow Prop$. Subtyping relations are supported by Coq's coercion mechanism and thus all the relevant subtyping relations can be declared.[12] Adjectives are defined as predicates, and adjectival modification as Σ types (see the discussion in section 2). Quantifiers and

[10] See A.1, A.2 and A.3 in the appendix for examples of how this can be done.

[11] In Luo's MTT, CN is the universe containing the names that interpret CNs. Since the possiblity of introducing new universes is not an option we approximate this idea by having CN being of type set.

[12] Note that ambiguous paths are not allowed and as such given two types A and B (with $A, B\colon CN$), there is no possiblity of defining both $A < B$ and $B < A$.

VP adverbs are defined as types ranging over CN (see (13) and (14)). The definition for *some*, already mentioned, is shown below:

(22) Definition some:= fun A:CN,fun P: A → Prop⇒ exists x:A, P(x).

Ignoring tense and aspect for the moment and defining *walked* as being of type $[\![human]\!] \rightarrow Prop$ and *John* as being of type $[\![man]\!]$ with $[\![man]\!]$ ¡ $[\![human]\!]$, we can prove the theorem in (21) quite easily. We first use the proof tactic *intro* to move the implicans to the hypotheses. Then, we apply $unfold$ to *some* (*unfold some*). *Unfold* does what it says: it unfolds the definition associated with a lexical entry (if there is a definition). We then substitute *John* for x, and via the tactic assumption the theorem is proven. This example as well as all of the examples discussed in this paper can also be proven automatically by using Coq's predefined automation tactics or via using user-defined automation tactics.[13]

4 NL Inference with FraCas Examples in Coq

The FraCas Test Suite [10] arose out of the FraCas Consortium, a huge collaboration with the aim to develop a range of resources related to computational semantics. The FraCas test suite is specifically designed to reflect what an adequate theory of NL inference should be able to capture. It comprises NLI examples formulated in the form of a premise (or premises) followed by a question and an answer. For instance,

(23) Either Smith, Jones and Anderson signed the contract.
Did Jones sign the contract? [Yes]

(24) No delegate finished the report.
Did any delegate finished the report on time? [No]

The examples are quite simple in format but are designed to cover a very wide spectrum of semantic phenomena, e.g. generalized quantifiers, conjoined plurals, tense and aspect related phenomena, adjectives and ellipsis, among others.

In this section, we use a number of these examples in FraCas (except the last subsection §4.8 on collective predicates) to see how and if they can be dealt with using the MTT-semantics implemented in Coq. The formulation we are going to follow will transform the question in each example of the FraCas test suite into a declarative hypothesis that needs to be proven.[14] All of these examples are formalised in Coq[15]; the Coq code and proof for the first example, as described in §4.1 below, can be found in Appendix A.3.

[13] See the discussion on automation in the end.

[14] The same modification can be also found in [26]). In general, if one uses a theorem prover to deal with NL inference (e.g. analyses in the style of [2, 4, 5]) such modifications are necessary.

[15] The source codes can be obtained by sending an email request to the first author: stergios.chatzikyriakidis@cs.rhul.ac.uk.

4.1 Quantifiers and Monotonicity

A great deal of the FraCas examples are cases of inference that result from the monotone properties of quantifiers. Examples concerning monotonicity on the first argument are very easily treated in a system encoding an MTT with coercive subtyping, by employing the subtyping relations between CNs (c.f., §2.3).

To put this claim in context, let us look at the following example (3.55) from the FraCas test suite:

(25) Some Irish delegates finished the survey on time.
Did any delegate finish the report on time [Yes]

In an MTT-based semantics, the sentences in the above example become:[16]

(26) $\exists s : [\![Irishdelegate]\!], [\![ONTIME]\!][\![finish]\!](s, (the\, report))$
Let Q $= \exists s : [\![delegate]\!], [\![ONTIME]\!][\![finish]\!](s, (the\, report))$. Is Q true?

where $[\![finish]\!]$: $[\![object]\!] \to [\![human]\!] \to Prop$, $[\![ONTIME]\!]$: $forall A : CN, (A \to Prop) \to (A \to Prop)$[17], $[\![Irishdelegate]\!] < [\![delegate]\!] < [\![human]\!]$ and $[\![report]\!] < [\![object]\!]$. Some notes of explanation on the subtyping relation $[\![Irishdelegate]\!] < [\![delegate]\!]$: We follow the second author's implementation of Σ-types as the Coq record types [22], and we interpret the modified CN *Irish delegate* as the following record type:

(27) Record Irishdelegate : CN := mkIrishdelegate [d :> delegate; _ : Irish d]

Given the subtyping relation between *Irish delegate* and *delegate* (reflected by means of the syntactical notation :> in Coq), (25) is correctly captured.

It is straightforward to prove the above formula Q, by the Coq commands for applying the introduction rule and $\exists$-introduction. In fact, theorems like Q can be proved automatedly by means of the tactics in Coq (see Appendix A.3).

Similar considerations apply to the examples like the one shown below (FraCas example 3.49):

(28) A Swede won a Nobel Prize.
Every Swede is a Scandinavian.
Did a Scandinavian win a Nobel prize? [Yes]

Given the subtyping relation $Swede < Scandinavian < human$ the above inference is correctly predicted. Note that the second premise is expressed via means of the subtyping relation $Swede < Scandinavian$. Specifically, we have $[\![win]\!]$: $[\![object \to [\![human]\!] \to Prop.$ and $prize < Object$. The subtyping

[16] For this first example, we shall detail its formal semantics in a type theory. You can also find the Coq implementation of this in Appendix A.3. For the examples later on, we shall omit such details.

[17] This is the type for VP adverbs used in [22]. We will see later on that a slightly modified type will be used for VP adverbs. For the moment, we keep this type given that it does not play any role whatsoever in proving the inference.

relation $[\![Nobel\_Price]\!] < [\![price]\!]$ is true because that $[\![Nobel\_Price]\!]$ may be defined as $\Sigma p : [\![price]\!].\ Nobel(p)$ which is a subtype of $[\![price]\!]$ via the first projection (c.f., §2.3).

All the other examples in the same section of the FraCas test suite (3.1.4) can be dealt with accordingly. Some of them are more complicated since they do involve genitival constructions (e.g. expressions like 'residents of the North American continent' in example 3.50) but in principle can be treated in the same way with slight modifications.

Adverbial Modifications: a Digression. We now move to cases involving monotonicity on the second rather than the first argument. Such an example with upwards monotonicity is shown below:

(29) Some delegates finished the survey on time.
Did any delegate finish the survey? [Yes]

Monotonicity on the second argument can be treated in a similar way as above. However, the above example (29) is a little bit trickier since an adjunct, i.e. the PP *on time*, is involved. As mentioned in §2.4, VP adverbs such as *on time* are given the following type (to repeat (13) here):

(30) $\Pi A\colon \textsc{cn}.\ (A \to Prop) \to (A \to Prop)$

In order to consider such adverbial phrases in inference, we make use of an auxiliary object *ONTIME*:

(31) $ONTIME\colon \Pi A\colon \textsc{cn}\Pi v\colon A \to Prop.\ \Sigma p\colon A \to Prop.\forall x : A.p(x) \supset v(x)$

For any common noun A and any predicate v over A, $ONTIME(A, v)$ is a pair (p, m) such that for any $x\colon A$, $p(x)$ implies $v(x)$. Taking the sentence (29) as an example, for the CN *delegate* and predicate $[\![finish]\!]$[18], we define

(32) $on\ time = \lambda A\colon \textsc{cn}.\lambda v\colon A \to Prop.\ \pi_1(ONTIME(A, v))$

which is of type (30). As a consequence, for instance, any delegate who finished the survey on time did finish the survey.

Note that the Σ-type in (31) might be considered as a general form of conjunction and, thinking in this way, it is not difficult to see that the above analysis is intuitively compatible with a Davidsonian analysis of VP adverbs where the adverb modifies an event argument [13].

A similar example involving downward monotonicity on the second argument is shown below:

(33) No delegate finished the report.
Did any delegate finish the report on time [No]

In the above case, the only thing we need to do is turn the hypothesis into its negation and then try to prove it, as we did in Coq.

[18] Note that $[\![finish]\!]\colon [\![human]\!] \to Prop < [\![delegate]\!] \to Prop$.

4.2 Conjoined Noun Phrases

In the section of the FraCas test suite involving inferences with conjoined NPs, one can find the following example:

(34) Smith, Jones and Anderson signed the contract.
Did Jones sign the contract? [Yes]

In [7], we have proposed a polymorphic type for binary coordinators that extends over the constructed universe $LType$, the universe of linguistic types. This can be extended to n-ary coordinators. For example, the coordinator *and* may take three arguments, as in the premise of (34). In such cases, the type of the coordinator, denoted as and_3 in semantics, is:

(35) $and_3 : \Pi A : LType.\ A \to A \to A \to A.$

Intuitively, we may write this type as $\Pi A : LType.\ A^3 \to A$. For instance, the semantics of (34) is (36), where c is 'the contract':

(36) $[\![sign]\!](and_3(s, j, a), c)$

In order to consider such coordinators in reasoning, we consider the following auxiliary object (similarly as in the last subsection when we consider adverbial phrases) and define and_3 as follows:

(37) $AND_3 : \Pi A : LType.\ \Pi x, y, z : A.\ \Sigma a : A.\ \forall p : A \to Prop.\ p(a) \supset p(x) \wedge p(y) \wedge p(z).$

(38) $and_3 = \lambda A : LType. \lambda x, y, z : A.\ \pi_1(AND_3(A, x, y, z))$

Having defined the coordinators such as *and* in such a way, we shall have the desired inference as expected. For example, from the semantics (36), we can infer that 'Jones signed the contract', the hypothesis in (34).[19]

Coordinators such as *or* can be defined in a similar way. More complex examples like the one shown below can be also proven:

(39) Either Smith, Jones or Anderson signed the contract.
If Smith and Anderson did not sign the contract, did Jones sign the contract? [Yes]

4.3 Adjectives

Inferences involving adjectives pose a number of difficulties given the semantic asymmetries associated with different classes of adjectives. The semantics of adjectives is a notoriously difficult issue in theoretical semantics and a number of approaches have been put forth, particularly in a classical Montagovian

[19] A note about Coq is in order here: building new universes is not an option in Coq (or, put in another way, Coq does not support us to build a new universe). Instead, we shall use an existing universe in Coq in conducting our examples for coordination.

setting (see, for example, [29,18,31,32]). In Modern Type Theories, Σ-types have been proposed for intersective adjectives [34] and, recently, the current authors have studied adjectives more systematically using the framework used in this paper [8]].

The FraCas test suite uses different terminology than those usually found in the literature on the formal semantics of adjectives. The basic classification is between affirmative and non-affirmative adjectives:

(40) Affirmative: Adj(N) $\Rightarrow$ (N)

(41) Non-affirmative: Adj(N) $\not\Rightarrow$ (N)

We shall follow this latter terminology in this paper.

4.3.1 Affirmative and Non-affirmative Adjectives

Cases of affirmative adjectives are handled well with the existing record mechanism already used for adjectives. The following inference as well as similar inferences are correctly captured, given that a CN modified by an intersective adjective is interpreted as a Σ-type which is a subtype of the CN via means of the first projection:

(42) John has a genuine diamond $\Rightarrow$ John has a diamond.

Non-affirmative adjectives involve cases like *former*. The problem with these types of adjectives is that they do not give rise to categorical intuitions as regards inference. Thus, the following inference is valid for some but non-valid for some others:

(43) John is a former president $\Rightarrow$ John is not a president

The same goes for adjectives like *fake*.[20] We leave the discussion concerning adjectives like *former* until temporal inference is going to be discussed. We will then propose an account, assuming that *former* is indeed non-affirmative.

4.3.2 No Comparison Class adjectives

Adjectives like *four-legged* do not need reference to a comparison class (FraCas 3.202):

(44) Every mammal is an animal.
Is every four-legged mammal a four-legged animal? [Yes]

Assuming that four_legged is of type $Animal \rightarrow Prop$ and given that $Mammal < Animal$, the above inference is correctly predicted.

[20] In the case of *fake*, [32] tried to provide an account where *fake* is treated as a subsective adjective, i.e. affirmative in the classification given in the FraCas test suite, via using the disjoint union type.

4.3.3 Opposites

This section deals with adjectives of the same comparison class which are opposites of one another like e.g. *large* and *small*. For these adjectives, the following inferences hold:

(45) Small(N) $\Rightarrow \neg$ Large(N).

(46) Large(N) $\Rightarrow \neg$ Small(N)

(47) $\neg$ Small(N) $\not\Rightarrow$ Large(N).

(48) $\neg$ Large(N) $\not\Rightarrow$ Small(N)

What is the most difficult part here is the avoidance of the inferences (47) and (48). Something which is not small might not be large, given that sizes do not come in the form of a binary opposition. Thus, a way to treat this is to introduce another size, let us say *normalsized*, and have *small* hold in case the negation of both *large* and *normalsized* hold:[21]. We introduce the following:

(49) Definition Small:= fun A:CN,fun x: A=> $\sim$Large x $\wedge$ $\sim$Normalsized x.

This approach is successful in getting the inferences as regards opposites right.[22]

4.3.4 Extensional Comparison Classes

Adjectives like *large* and *small* are only relevant for the comparison class they refer to. Thus, inferences like the following are found:

(50) All mice are small animals.
Mickey is a large mouse.
Is Mickey a large animal? [No]

In order to deal with these cases, we introduce a polymorphic type for adjectives like these ranging over the universe CN.[23] The type for *large* is shown in (51), which is (52) in Coq's notation:

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

(52) Parameter large : forall A:CN, A$\to$Prop

Using the above type, we have many instances of *large* depending on the choice of A. If $A = Man$ then we get $large(Man)\colon Man \to Prop$; if $A = Animal$, we get $large(Animal) : Animal \to Prop$, and so on. In this respect,

[21] Of course, depending on context more fine grained distinctions might be needed but the idea is applicable to these cases as well.

[22] *Small* is defined after *Large* has been declared. The opposite is also possible, i.e. defining *Large* after *Small* has been declared first. This might seem strange from a theoretical point of view, but for implementation purposes it is not.

[23] This is based on the authors' analysis of subsective adjectives [8].

we get different 'larges' as such for different As. With this, one can capture the meaning of subsective adjectives, i.e. that if something is A (where A an adjective), it is only large for its class denoted by the CN (a large mouse is thus only large as a mouse, but not as an elephant). This way of treating subsective adjectives will correctly account for the inferences like that in (50).[24]

4.3.5 A note on intersective adjectives

Intersective adjectives comprise a class of adjectives in the theoretical literature on adjectives which can be characterized by the following inferential schema:

(53) Affirmative: $\text{Adj(N)(x)} \Rightarrow \text{Adj(x)} \wedge \text{N(x)}$

Thus, *black man* means something that is *black* and a *man*. With intersective adjectives, one should be able to get inferences like the following:

(54) Adj_{inter} man $\Rightarrow$ Adj_{inter} human

A concrete example would be *black man* implying *black human*. Given that coercions according to Luo's MTT propagate via the various type constructors, we have: $\Sigma([\![man]\!], black) < \Sigma([\![human]\!], black)$.[25]

4.4 Comparatives

Comparatives such as *shorter than* can be given semantics either directly or by means of an explicit measure. We shall consider both alternatives.

Comparatives without Measures. We shall consider *shorter than* as a typical example. Intuitively, *shorter than* should be of type $Human \rightarrow Human \rightarrow Prop$ as in the following example:

(55) Mary is shorter than John.

We assume that there be a predicate $short\colon Human \rightarrow Prop$, expressing that a human is short. Intuitively, if Mary is shorter than John and John is short, then so is Mary. Furthermore, one should be able to take care of the transitive properties of comparatives. Thus, if A is $COMP$ than B and B is $COMP$ than C, then A is also $COMP$ than C. All these can be captured by considering

[24] The interested reader is directed to [8] for more information on the treatment of subsective as well as the other types of adjectives in MTT with coercive subtyping.

[25] In Coq, we cannot have the first projection as a general coercion. Instead, we have to declare it for the instances we need. This is a weakness of Coq that does not allow us to implement the more general treatment. Such a general coercion is possible to get declared in Plastic, an interactive theorem prover that implements Luo's UTT and coercive subtyping [6].

$SHORTER\_THAN$ of the following Σ-type and define *shorter than* to be its first projection:

(56) $SHORTER\_THAN$: Σp: $Human \rightarrow Human \rightarrow Prop.\ \forall h_1, h_2, h_3 : Human.\ p(h_1, h_2) \wedge p(h_2, h_3) \supset p(h_1, h_3) \wedge \forall h_1, h_2 : Human.p(h_1, h_2) \supset short(h_2) \supset short(h_1)$.

(57) $[\![shorter\ than]\!] = \pi_1(SHORTER\_THAN)$

With the above, we can easily show that the inferences like (58) can be obtained as expected.

(58) John is shorter than George.
George is shorter than Stergios.
Is John shorter than Stergios? [Yes]

Comparatives with Measures. In giving an analysis of compratives, one may consider measures, taking into consideration different degrees of the measure used in each case, e.g height in the case of comparatives like *short*, weight in the case of adjectives like *heavy* or speed in the case of adjectives like *fast*. For example, we can analyze *shorter than* as a relation between nouns that do come with implicit measures, in which the first noun has less height than the second.

Such measures can be taken care of explicitly by extending the above treatment by dependent typing over measures. Let's consider *shorter than* as an example, taking heights to be measured by the type $Height$ of numbers such as 1.70.[26] We are then led to consider the family of types $Human$: $Height \rightarrow Type$ indexed by heights: $Human(n)$ is the type of humans of height n. Then, *shorter than* is defined as follows:[27]

(59) $SHORTER\_THAN$: $\Pi i, j$: $Height.\ \Sigma p$: $Human(i) \rightarrow Human(j) \rightarrow Prop.\ \forall h_1 : Human(i) \forall h_2 : Human(j).\ p(h_1, h_2) \leftrightarrow i < j$.

[28]

(60) $[\![shorter\ \ than]\!](i, j) = \pi_1(SHORTER\_THEN(i, j))$: $Human(i) \rightarrow Human(j) \rightarrow Prop$

We can now take care of the inferences like (61) as expected:

(61) John is shorter than George.
George is 1.70.
Is John less than 1.70 tall? [Yes]

[26] Here we do not spell out the type $Height$. One might take $Height$ to be the type of natural numbers and use 170 to stand for 1.70, etc.

[27] The transitive properties of comparatives are not encoded in this example for reasons of simplicity. One may very well do so having as a guide the previous entry without measures.

[28] This is a bi-implcation, given that if the height of human x is less than the height of another human y, then it is also the case that x is shorter than y. The definition also works as an implication.

To see the details, the semantics of the above sentences are given in (64), where J and G are the semantics of *John* and *George*, involving height parameters:

(62) $J, G : \Sigma x : height.HUMAN(x)$

We can further define j and g as the second projection of J and G respectively:

(63) $j, g : \pi_2(J, G)$

With these at hand, (61) can be formulated as follows:

(64) $[\![shorter\ than]\!](J, G)$.
$g = 1.70$.
Is Q true, where $Q = j < 1.70$?

It is easy to show that the above inference (64) can be proven in Coq.

It may worth remarking that superlatives can be defined once comparatives are: for example, for any x: $Human$, $shortest(x)$ if and only if x is shorter than or equal to any y: $Human$. Similar treatment can account for the rest of the examples involving comparatives in the FraCas test suite.

4.5 Temporal Reference

A way to deal with temporal reference without employing a temporal logic of some sort, is to introduce a type $Time$ of times to deal with the extra parameter of $Time$ (see e.g. [34] for such a view).

With such a type $Time$, one provides a very simple model of tense and, over $Time$, we have a precedence relation $\leq$ and a specific object now: $Time$, standing for 'the current time' or 'the default time'. Simple tenses like the simple present or the simple past can then be easily captured.[29] Also, in this model, verbs are assumed to involve a $Time$ argument as well.[30] Thus a verb like *walk* is not simply of type $[\![human]\!] \rightarrow Prop$ anymore, but rather of $[\![human]\!] \rightarrow Time \rightarrow Prop$.

The type $Time$ can be specified as an inductive type in an MTT, where one may consider the following as one of its constructors:[31]

(65) $date$: $DATES \rightarrow Time$

[29] One may even employ this model to capture composite tenses like the past perfect, but we do not discuss this here. See [34] for an idea of how this can be done within such a framework.

[30] The assumption that verbs involve an event/situation argument goes back at least to Davidson [13]. See [14] and reference therein, for a history of events/situations in linguistic theory.

[31] An inductive type is specified by a number of constructors whose types must be strictly positive (see, for example, Chapter 9 of [19] for formal details.) $Time$ as an inductive type may have other constructors but we only detail *date* here.

where $DATES$ consists of the triples (y, m, d) where y ranges over integers to represent years, m over Jan to Dec to represent months, and d over the days $1, 2, ...$ to represent days.[32] For example, $date(1970, Oct, 5)$ stands for the time 'Oct 5, 1970'.

Now, consider the inferences like the following example:

(66) Last year John signed the contact.
Today is June 18, 2013.
Did John sign the contract in 2012? [Yes]

With the above, the above sentences in (66) are interpreted as those in (67), where $c = [\![the\ contract]\!]$:

(67) $\exists t\colon Time, \exists m : month, \exists d : day.date(year(now) - 1, m, d) \wedge m \leq 12 \wedge d \leq 30 \wedge sign(j, c, t)$.
$now = date(2013, June, 18)$.
Is Q true, where $\exists t\colon Time, \exists m : month, \exists d : day.date(2012, m, d) \wedge m \leq 12 \wedge d \leq 30 \wedge sign(j, c, t)$?

This can now be shown to be valid inference in Coq. Cases involving temporal adverbs like *yesterday, today* or PPs like *next month, next year* can be treated accordingly.

Similarly examples like the following can be accounted for, assuming that currently identifies the time of the proposition to be equal with the *default time*. The *Time* argument of the proposition has already been identified as being the *default time* via means of the present tense verb and as such, examples like the one below are very easily proven to be valid inferences:

(68) ITEL has a factory in Birmingham.
Does ITEL currently have a factory in Birmingham? [Yes]

The sections in the Fracas test suite that deal with *in* and *for* adverbials need a solid account of lexical aspect as well as a fuller account of tense which at the present we do not have to offer. We leave these sections unresolved until such an account is provided. However, some of the inferences can be effectively dealt with in pretty much the same way as the monotone on the second argument examples. One such example is shown below:

(69) Smith lived in Birmingham for two years.
Did Smith live in Birmingham? [Yes]

The idea is to introduce lists of elements of type $Time$ in order to deal with intervals. We further define a parameter $Interval\colon list\ Time \to Time$ which

[32] Note that,in detail, the range of days depends on the year and month. This can be represented by means of dependent types: the type $Day(y, m)$ depends on $y\colon Year$ and $m\colon Month$: for example, if there are only 28 days in Feb of 1970, $Day(1970, Feb) = \{1, 2, ..., 28\}$, the enumeration type consisting of 1, 2, ..., 28 only. Formally, $DATES$ can be defined as $\Sigma y\colon Year.\Sigma m\colon Month.\ Day(y, m)$.

applies to a list of Times and returns a $Time$ argument. In effect, time intervals can function as time arguments. Now, we further define that for all elements $x\colon Time$ included in the given interval, let us say $Interval(T)\colon Time$, as well as for all subintervals $T_1 : listTime$ and for $P\colon Time \to Prop$, $P(Interval(T))$ as well as $P(t)$ and $P(Interval(T_1))$ hold. Similar cases can be treated accordingly.

4.5.1 The Case of former

Adjectives such as *former* or *past* may be treated in the temporal model we have considered.[33] We assume that some CNs are indexed by the time parameter. For example, instead of being interpreted just as a type, a CN like *president* is interpreted as a family of types indexed by $t\colon Time$:

(70) $president(t)\colon \textsc{cn}$.

For example, as $now\colon Time$ stands for the 'current time', $president(now)$ is the president at the current time.

With the above mechanisms available, we can now interpret CNs modified by *former* as follows: for example,[34]

(71) $[\![former\ president]\!] = \neg president(now) \wedge \exists t\colon Time.\ t < now \wedge president(t)$.

In general, we have $[\![former]\!]\colon (Time \to \textsc{cn}) \to \textsc{cn}$,[35] obtained by abstracting *president* in the above definition: for any $p\colon Time \to \textsc{cn}$,

(72) $[\![former]\!](p) = \neg p(now) \wedge \exists t\colon Time.\ t < now \wedge p(t)$.

With $president\colon Time \to \textsc{cn}$, we have $[\![former\ president]\!] = [\![former]\!]([\![president]\!])$.

This kind of analysis will predict that former president entails a past president but not a current president.

4.6 Epistemic, Intentional and Reportive Attitudes

This section involves verbs taking a sentential argument. The difference is between verbs that presuppose the truth of the propositions expressed by their sentential arguments and verbs that do not:

[33] Another approach to dealing with such adjectives is to follow Partee [31] and assume that *former* behaves similarly to privative adjectives like *fake* or *imaginary*. If so, one may follow the proposed MTT-interpretation by the authors to use the disjoint union type to interpret *former*. See [8] for details.

[34] For understandability of the readers who are unfamiliar with MTTs, we abuse the notation here, using $\neg A$ to stand for $A \to \emptyset$, $\wedge$ for $\times$ and $\exists$ for Σ. One may ignore these formal details.

[35] In Coq this is translated as $(Time \to \textsc{cn}) \to Prop$ given that definitions always end in Prop.

(73)Smith knew that Itel had won the contract 1991.
Did Itel win the contract in 1991? [Yes]

(74)Smith believed that Itel had won the contract 1991.
Did Itel win the contract in 1991? [Don't know]

Again, we will not dwell on a discussion on how suitable semantics for attitude verbs should be given. There are so many issues to take into consideration in this respect, starting with questions so general like 'what is belief', that such a discussion cannot be carried out here. However, we can provide an account of these types of inferences without necessarily solving the issues associated with the semantics of Attitude verbs.

What we need is to encode that some epistemic verbs presuppose their argument's truth while others do not. For instance, *know* belongs to the former class and its semantics is given as follows:

(75) $KNOW = \Sigma p : Human \rightarrow Prop \rightarrow Prop.\ \forall h : Human \forall P : Prop.\ p(h, P) \supset P$

(76)$[\![know]\!] = \pi_1(KNOW)$

With this, the inference (73) can be obtained as expected. Intensional verbs like *believe* on the other hand do not imply their arguments and inferences like (74) cannot be shown to be valid inferences.

In the FraCas test suite there are also examples concerning 'veridicality'; this is basically the property that verbs like *know* show – 'know P' $\Rightarrow$ P, so we do not need to discuss these cases again.

4.7 Substitution and Existential Instantiation

Substitution refers to the ability of substituting two equivalent terms and retaining the meaning after substitution, as (77) shows:

(77)Smith saw Jones sign the contract.
Jones is the chairman of ITEL
Did Smith see the chairman of ITEL sign the contract?[Yes]

Substitutions like those in the second premise above can be easily done in Coq via the *replace* tactic. Thus, cases like these are easy to capture.

There are also examples where existential quantifiers and their instantiations are involved. For example,

(78)Smith knows that Jones signed the contract.
Jones is a person.
There is a person such that Smith knows he signed the contract

Existential quantification is introduced to the semantics because of the second sentence; this becomes clear if we spell out the semantic interpretations of the sentences in (78) as those in (79) below, where $c = [\![the\ contract]\!]$:

(79) $[\![know]\!](s, sign(j, c))$.
$\exists x : Person.\ j = x$.
$\exists x : Person.\ [\![know]\!](s, sign(x, c))$.

It is easy to see that the first two imply the third.

4.8 Collective predication

We want to be able to get the following inferences (note that these cases are not part of the FraCas test suite):

(80) Stergios and Zhaohui met ⇒ Stergios met Zhaohui and Zhaohui met Stergios

(81) Stergios and Zhaohui hit each other ⇒ Stergios hit Zhaohui and Zhaohui hit Stergios

(82) Stergios and Zhaohui are Greek and Chinese respectively ⇒ Stergios is Greek

For such collective predicates, we use the Vector-analysis proposed by the authors in [7]. Verbs like *meet* in their collective guise take a vector argument with at least two elements (ie, an object of type $Vec(Human, n+2)$), as given in (83).[36] This account can also give us a natural treatment of reflexives like *each other*. The idea is that *each other* in English turns a transitive predicate into an intransitive one whose sole argument is a vector whose length is at least 2, as in (84). Lastly, *respectively* can be seen as a big functor which takes two vector arguments and returns a proposition, as in (85).

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

(84) $[\![each\ other]\!]$: ΠA: CN, Πn: Nat. $(A \to A \to Prop) \to Vec(A, n+2) \to Prop$

(85) $[\![respectively]\!]$: ΠA: CN.Πn: Nat. $Vec(A, n+2) \to Vec((A \to Prop), n+2) \to Prop$

With the above typings, in order to get expected inferences, we need to assume more information concerning these words. For example, for *each other*, we assume that the following be true: for any A: CN, n: nat, P: $A \to A \to Prop$ and v: $Vec(A, n+2)$,

(86) $[\![each\ other]\!](A, n, P, v) \supset \forall i, j : nat.\ i \leq n+1 \wedge j \leq n+2 \wedge i \neq j \supset P(v_i, v_j) \wedge P(v_j, v_i)$,

[36] $Vec(A, n)$ can be seen as a collection of elements of type A with an explicit nat argument counting the elements.

where if $v = (a_1, ..., a_{n+2})$ then $v_i = a_i$ and $v_j = v_j$. It is now straightforward to get the expected inferences such as those in (80,81,82).[37]

Remark 1 A promising aspect of using vector types is that they can potentially be used for proper semantic treatments of some of the non-classical quantifiers including, for example, *exactly three* or *most*.[38] We do not know how far can one go with vector types as regards a general way of dealing with plurals. We have not yet explored the possibilities as well as the consequences of this proposal with respect to a general theory of plurals. This is a topic which we will pursue in future work. However, one can already see a way to treat cases of negated plurals like the ones shown below:

(87) Just one accountant attended the meeting
Did no accountant(s) attend the meeting? [No]

(88) Just one accountant attended the meeting
Did any accountant(s) attend the meeting? [Yes]

We can assume that plural CNs are in the plural part of CN, CN_{pl}, with $\text{CN}_{pl} < \text{CN}$. Now, we can consider typings of quantifiers with vectors for plural CNs, something along the line of the following type:

(89) $\Pi A\colon \text{CN}_{pl}\Pi n\colon Nat.\Pi k\colon Nat.\ Vec([\![human]\!], k) \rightarrow Prop$

5 NLI: Discusion on different appraches and automation

5.1 Informal comparison with other relevant approaches

The most obvious difference between the system presented here and deep approaches to NLI that use first-order logic as their translation language like e.g. [4,5], is the use of a many-sorted typed system rather than an untyped one . This, in conjunction with the coercive subtyping mechanism, takes care of a number of inferences via typing only (e.g. monotone on the first argument or adjectival inferences). In systems translating to first-order logic, this information must be added separately as axioms .[39] Furthermore, dependent typing offers a number of welcomed results. One such result was developed in this paper and concerns the employment of Σ types not only in dealing with existential[40] or adjectival modification but to interpret adverbial modification as well as the semantics of factive verbs. Again, the advantage in this case is the

[37] On the assumption that *meet* and *respectively* are also assumed to involve extra information in the same vein with *each other*.

[38] Also, this does not mean that the Vector-treatments are superior as compared to some existing semantics of these quantifiers (see, for example, [37]).

[39] For example, the Montagovian meaning postulates for the different kinds of adjectives have to be defined as axioms (see e.g. [33]). In our case, and at least for intersective and subsective adjectives, their inferential properties are derived via typing only (see [8] for more information).

[40] Even though we do not use Σ types to represent existential quantification.

ability of using dependent types in order to take care of the desired inferences without resorting to meaning postulates. Abstracting away from the detalis of each line of approach, like for example the phenomena that are treated in one of the approaches but not in the other,[41] the basic difference seems to boil down to the use of two rather different logical languages in interpreting NL semantics, first-order logic on the one hand and an MTT on the other.

However, and as already mentioned, the system presented in this paper is not yet a full-blown system, given that only the part of the inferential process is shown and not any of the other components of a successful NLI system, namely a wide-coverage parser recognizing grammatical strings of text as well as various components that perform some kind of pre-processing of the goal before the latter is handed to the prover. For example, [5] a wide coverage CCG parser is used, while Background Knowledge is encoded via translating any relations found in WordNet (e.g. hyponymy relations) to first-order logic. The same is done for generic knowledge (e.g. passives, spatial information). Furthermore, deep approaches usually involve a shallow approach component as it is the case for example in [4] where some form of relation between the premise and the hypothesis are derived. This is done via searching for word overlaps between the premise and hypothesis by taking into consideration Wordnet relations. This process results in the assignment of a similarity measure between the premise and hypothesis.[42] Such hybrid approach will be interesting to use once a more complete version of the system presented here is ready. Another idea we would like to use is that that of entailment approximation discussed in [4]. The intuition behind it is simple and is based on the informal observation that when the prover has almost found a proof, the relation is usually an entailment. Of course, this is very difficult to formalize in practice for obvious reasons. In [4], a model builder is used for this reason. Again the idea is simple: in case we are dealing with an entailment, the entities of the model are the same in both the premise and hypothesis. In case we are not dealing with an entailment, the domain size is different. Domain size is then used to approximate entailment: larger distances in the domain size point to non-entailment, smaller distances to entailment. Thus, a possible future direction is to try and see how can the concept of entailment approximation be translated within our system. Obviously, we will not be using any kind of model builder[43] but however, one can measure the domain size via the number of entities or relations between the entities that are needed in the local context of the proof in order for the premise or the hypothesis to be true. Then the same idea used in [4], based on the distance in the domain size of the entities plus relations between the premise and hypothesis, can be used.

[41] E.g. treatment of anaphora that is lacking in our account or the treatment of collective predication temporal reference lacking in deep approaches like [4, 5, 33].

[42] The account proposed in [26], as already mentioned, is a kind of hybrid approach with both a shallow and a deep component. It is out of the scope of this paper to look at the state-of-the-art shallow approaches to NLI. However, the interested reader is directed to [26] and references therein for more information on these type of approaches.

[43] It is in itself contradictory to use model-theoretic semantics in a constructive framework!

It is clear from the above that the next step for us will have to be the development of a full-blown NLI system. This will ideally involve the development of a parser using Ranta's Grammatical Framework (GF, [35]), its purpose being twofold: a) parse grammatical Engish sentences and b) linearize these parsed input into the syntax of Coq. In effect, Coq syntax is treated like an ordinary language. The system will involve an abstract syntax but two concrete syntaxes, one for English and one for the Coq language.[44] Additional information like BK or generic knowledge can be expressed via means of axioms or even typing (e.g. hyponymy relations) drawn from WordNet or similar sources (VerbNet, ConceptNet) . The same holds for generic knowledge.This all remain part of our future work and our in our opinion feasible.

Furthermore, and even though we have covered a number of issues in this paper, there are sections in the FraCas test suite that we have not tried yet. For example the section (or subsections) discussing issues relevant to the aspectual system have not yet been properly tried out. It is our intention to attempt a proper formalization as well as implementation of aspect in Coq and extend the preliminary implementation of tense as shown in this paper as well. Similar considerations apply to other sections of the FraCas test suite like e.g. the section dealing with inference in elliptical environments.[45] Lastly, if such a system is to have broader practical applications one needs to test against real text and not examples constructed ad hoc for the sake of testing various categories of inference as the FraCas test suite is basically doing. The next step will thus be to test the proposed account against the RTE challenge suites [12]. This, along with what we have mentioned already, consists the basis of what our future work is directed towards to.

5.2 Interaction and automation

Coq is an interactive theorem prover. As such, theorems, in our case NLIs, are proven interactively and not automatically. One may argue, that such a system is not really helpful for NLI, since what we want is a way to prove these inferences automatically. This is a valid point and of course we do agree. However, the idea of using an interactive theorem has a number of advantages. One of them is that by using an interactive theorem prover one is able to see the reason a given theorem cannot be proven. This last fact alone can be quite helpful in designing automated tactics for NLI. Furthermore, Coq itself has a number of build-in tactics that are designed to automate trivial parts of proofs. For example, some of our examples can be solved with *intuition* or *jauto* once the *cbv delta* tactic has been executed. *Cbv delta* replaces the

[44] This is one of the core ideas of GF parsers, i.e defining one abstract syntax that corresponds to multiple concrete ones.

[45] Dealing with ellipsis successfully is of course largely dependent on the adequacy of the parser, given that if the parser succeeds in parsing elliptical constructions it will then linearize these structures into the Coq language where the elided information will be present. From this stage on, inferences are easy to be proven. However, this issue is left for future work.

occurences of a defined notion by the definition itself in the current goal (or in any of the hypotheses) while *intuition* just looks for first-order intuitionistic logic tautologies. We can thus define a new tactic which first calls *cbv delta*, followed by *intuition* and *jauto* in order to automate a number of example cases.[46] We have introduced such a tactic (AUTO in the source code) and indeed a number of example cases can be automatically be proven by using this tactic (e.g. 25, 28, 42). For more advanced cases, one can use more elaborate proof-techniques in order to achieve automation. For example in cases where a Σ type analysis is used, like e.g. in the case of VP adverbs, one needs to use destruct specifically for the auxiliary objects (e.g. ONTIME for ontime). One can thus devise an automatic tactic which is however context dependent, depending on the example. In the same case one might need to instruct Coq to apply a specific premise. One tactic that does both the aforementioned is shown below:

(90)Ltac AUTO1 a b:= cbv; destruct a; eapply b; AUTO.

The above tactic can take care of the Σ type cases (note the use of AUTO within AUTO1). A similar more advanced automated tactic has been defined for cases of collective predication. These three tactics are then all we need to automate all our proof-examples. In order to achieve full automation, we can further use one composite tactic which tries one of the three tactics and succeeds in case one of them does. Assuming we have three tactics a, b and c, one can define the following tactic, say d:

(91)Ltac d:= solve[a|b|c].

Using this technique, one can actually automate all the examples discussed in this paper. It will be very interesting to see how far can one go with automation, in particular whether automation is still possible when the examples are comprised of bigger texts, like e.g. some examples from the RTE challenges. In fact, proof-automation in Coq is an on-going research topic within the community and a number of researchers have provided interesting results like for example work by [38] on inductive proof-automation. Further work is needed on the feasibility of automation as regards NLI but the first results seem promising. We hope that this paper will be the start of a new research direction, in which MTT semantics (or in general formal semantics) and proof-assistant technology work on a par in order to deal with NL reasoning.

6 Conclusion

In this paper we presented the first attempt to use proof-assistant technology in order to deal with NLI. Furthermore, this paper proposed the use of MTTs as

[46] *Jauto* is part of the *LibTactics* library, containing extra tactics than the standard ones.

the logical language for dealing with NLI. We provided an account of a number of cases from the FraCas test suite using Luo's TT with coercive subtyping. It was shown that using a considerably richer language than first-order logic, can give us a number of welcomed results as regards NLI. In particular, the coercive subtyping mechanism as well as the use of dependent typing have been shown to be very helpful in dealing with various NLI cases. The account was then tested in Coq where the FraCas test suite examples were encoded as Coq axioms. Lastly, it was shown that one cannot only use Coq in order to reason about NL semantics, but to further automate the proof process by developing used-defined tactics.

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

References

1. Asher, N.: Lexical Meaning in Context: a Web of Words. Cambridge University Press (2012)
2. Blackburn, P., Bos, J.: Representation and Inference for Natural Language. CSLI Publications (2005)
3. Blazy, S., Dargaye, Z., Leroy, X.: Formal verification of a C compiler front-end. In: FM 2006: Int. Symp. on Formal Methods. Lecture Notes in Computer Science, vol. 4085, pp. 460–475. Springer (2006), `http://gallium.inria.fr/ xleroy/publi/cfront.pdf`
4. Bos, J., Markert, K.: Recognising textual entailment with logical inference. In: Proc. of the 2005 Conference on Empirical Methods in Natural Language Processing (EMNLP),. pp. 98–103 (2005)
5. Bos, J., Markert, K.: When logical inference helps determining textual entailment (and when it doesnt). In: Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment (2006)
6. Callaghan, P., Luo, Z.: An implementation of LF with coercive subtyping and universes. Journal of Automated Reasoning 27(1), 3–27 (2001)
7. Chatzikyriakidis, S., Luo, Z.: An account of natural language coordination in type theory with coercive subtyping. In: Parmentier, Y., Duchier, D. (eds.) Proc. of Constraint Solving and Language Processing (CSLP12). LNCS 8114. pp. 31–51. Orleans (2012)
8. Chatzikyriakidis, S., Luo, Z.: Adjectives in a modern type-theoretical setting. In: Morrill, G., Nederhof, J. (eds.) Proceedings of Formal Grammar 2013. LNCS 8036. pp. 159–174 (2013)
9. Church, A.: A formulation of the simple theory of types. J. Symbolic Logic 5(1) (1940)
10. Cooper, R., Crouch, D., van Eijck, J., Fox, C., van Genabith, J., Jaspars, J., Kamp, H., Milward, D., Pinkal, M., Poesio, M., Pulman, S.: Using the framework. Technical Report LRE 62-051r (1996), http://www.cogsci.ed.ac.uk/ fracas/.
11. The Coq Development Team: The Coq Proof Assistant Reference Manual (Version 8.1), INRIA (2007)
12. Dagan, I., Glickman, D., Magnini, B.: The pascal recognising textual entailment challenge. In: Quionero-Candela, J., Dagan, I., Magnini, B.and d'Alch-Buc, F. (eds.) Machine Learning Challenges. LNCS 3944. pp. 177–190 (2006)
13. Davidson, D.: Compositionality and coercion in semantics: The semantics of adjective meaning. In: Rescher, N. (ed.) The Logical Form of Action Sentences, pp. 81–95. University of Pittsburgh Press (1967)
14. Davidson, D.: The logical form of action sentences. In: Rescher, N. (ed.) The Logic of Decision and Action. University of Pittsburgh Press (1967)

15. Girard, J.Y.: Une extension de l'interpretation fonctionelle de gödel à l'analyse et son application à l'élimination des coupures dans et la thèorie des types'. Proc. 2nd Scandinavian Logic Symposium. North-Holland (1971)
16. Goguen, H.: A Typed Operational Semantics for Type Theory. Ph.D. thesis, University of Edinburgh (1994)
17. Gonthier, G.: A computer-checked proof of the Four Colour Theorem (2005), `http://research.microsoft.com/~gonthier/4colproof.pdf`
18. Kamp, H.: Formal semantics of natural language. In: Keenan, E. (ed.) Two theories about adjectives, pp. 123–155. Cambridge University Press (1975)
19. Luo, Z.: Computation and Reasoning: A Type Theory for Computer Science. Oxford Univ Press (1994)
20. Luo, Z.: Coercive subtyping. Journal of Logic and Computation 9(1), 105–130 (1999)
21. Luo, Z.: Type-theoretical semantics with coercive subtyping. Semantics and Linguistic Theory 20 (SALT20), Vancouver 84(2), 28–56 (2010)
22. Luo, Z.: Contextual analysis of word meanings in type-theoretical semantics. In: Logical Aspects of Computational Linguistics (LACL'2011). LNAI 6736. pp. 159–174 (2011)
23. Luo, Z.: Common nouns as types. In: Bechet, D., Dikovsky, A. (eds.) Logical Aspects of Computational Linguistics (LACL'2012). LNCS 7351. pp. 173–185 (2012)
24. Luo, Z.: Formal semantics in modern type theories with coercive subtyping. Linguistics and Philosophy 35(6), 491–513 (2012)
25. Luo, Z., Soloviev, S., Xue, T.: Coercive subtyping: theory and implementation. Information and Computation 223, 18–42 (2012)
26. MacCartney, B.: Natural Language Inference. Ph.D. thesis, Stanford Universisty (2009)
27. Martin-Löf, P.: An intuitionistic theory of types: predicative part. In: H.Rose, J.C.Shepherdson (eds.) Logic Colloquium'73 (1975)
28. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)
29. Montague, R.: The proper treatment of quantification in ordinary English. In: Hintikka, J., Moravcsik, J., Suppes, P. (eds.) Approaches to Natural Languages. pp. 221–242 (1973)
30. Montague, R.: Formal Philosophy. Yale University Press (1974)
31. Partee, B.: Compositionality and coercion in semantics: The semantics of adjective meaning. In: Bouma, G., Krämer, I., Zwarts, J. (eds.) Cognitive foundations of interpretation, pp. 145–161. Royal Netherlands Academy of Arts and Sciences, (2007)
32. Partee, B.: Privative adjectives: Subsective plus coercion. In: Bauerle, R., Reyle, U. (eds.) Presuppositions and Discourse: Essays Offered to Hans Kamp, pp. 123–155. Emerald group publishing (2010)
33. Pulman, S.: Second order inference in NL semantics. Talk given at the KCL Language and Cognition seminar, London (2013)
34. Ranta, A.: Type-Theoretical Grammar. Oxford University Press (1994)
35. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammar. CSLI Publications (2011)
36. Sundholm, G.: Proof theory and meaning. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic III: Alternatives to Classical Logic, pp. 471–506. Reidel (1986)
37. Sundholm, G.: Constructive generalized quantifiers. Synthese 79(1), 1–12 (1989)
38. Wilson, S., Fleuriot, A., Smaill, A.: Inductive proof automation for coq,. In: Proceedings of the 2nd Coq Workshop, EPTCS (2010)

A Coq Code of Examples

All of the examples in this paper have been tried in the Coq proof assistant. The source codes can be obtained by sending an email request to stergios.chatzikyriakidis@cs.rhul.ac.uk. Here, we shall give a simple example of the Coq script (Appendix A.1), an example with Coq tactics (Appendix A.2) and some examples in linguistic semantics (Appendix A.3).

A.1 A Simple Example of a Coq proof

```
Variables P Q R: Prop.
Theorem P Q R: (P->Q)->(Q->R)->(P->R).
intro. intro. intro.apply H0. apply H. assumption.
```

The variables P, Q and R are introduced as being of type $Prop$. We want to prove $(P \to Q) \to (Q \to R) \to (P \to R)$. We apply introduce three times. Now want to prove R, our assumptions being $(P \to Q), (Q \to R)$ and P. We apply $(Q \to R)$, we now have to prove Q. We apply $(P \to Q)$. We now have to prove P, which is easy, given that it is one of our assumptions.

A.2 A slightly more advanced example

We want to prove that if the law of the excluded middle holds then so is Peirce's law.

```
Definition lem: A \/ ~ A.
Definition Peirce:= ((A->B)->A)->A.
Theorem lemP: lem -> Peirce.
unfold Peirce. unfold LEM.unfold Peirce. intros. elim H.intros.
assumption.intros.apply H0.intros.absurd A.assumption. assumption.
```

We unfold the definitions, apply *intros* and *elim H*:

```
lemP < elim H.
2 subgoals
H : A \/ ~ A
H0 : (A -> B) -> A
============================
A -> A
subgoal 2 is:
~ A -> A
```

Then, *intro*, *assumption* and *intro* again:

```
lemP < intros.
1 subgoal
H : A \/ ~ A
H0 : (A -> B) -> A
H1 : ~ A
============================
A
```

We use *apply H0* and now we have to prove $A \to B$. We apply *intro*:

```
lemP < intro.
1 subgoal
H : A \/ ~ A
H0 : (A -> B) -> A
H1 : ~ A
H2 : A
============================
B
```

We use *absurd A* and now we need to prove A and $\neg A$, which can be done via two applications of *assumption*.

The above can be proved automatically as well, using automated user-defined tactics. For this case, we can define a tactic which unfolds all the definitions and then applies *tauto*, which tries intuitionistic propositional tautologies:

```
Ltac AUTO:= cbv delta;tauto
```

This suffices to prove our example automatically.

A.3 An Example from the FraCas Test Suite

FraCas example 3.55

(92) Some Irish delegates finished the report on time.
Did any delegate finish the report on time [Yes]

```
Parameter delegate survey: CN
Record Irishdelegate : CN := mkIrishdelegate { c :> delegate; _ : Irish c }.
Parameter on_time: forall A:CN, (A -> Prop) -> (A->Prop).
Parameter finish: Object -> Human -> Prop.
Axiom so:survey->Object. Coercion so: survey>->Object. *subtyping*
Axiom dh:delegate->Human. Coercion dh: delegate>->Human. *subtyping*
Theorem IRISH: (some Irishdelegate)(On_time(finish(the survey)))->(some delegate)
(On_time (finish(the survey))).
```

We unfold the definitions for *a* and move the premise to the assumptions via intro and we apply the elimination tactic *elim*:

```
IRISH < elim H.
1 subgoal

H : exists x : Irishdelegate, On_time (finish (the survey)) x
============================
forall x : Irishdelegate,
On_time (finish (the survey)) x ->
exists x0 : delegate, On_time (finish (the survey)) x0
```

We apply *intros*:

```
IRISH < intro.
1 subgoal
H : exists x : Irishdelegate, On_time (finish (the survey)) x
x : Irishdelegate
H0 : On_time (finish (the survey)) x
============================
exists x0 : delegate, On_time (finish (the survey)) x0
```

With x: $Irishdelegate$ as an assumption, we can now substitute x_0 in the conclusion with x thanks to the subtyping mechanism:

```
IRISH < exists x.
1 subgoal
H : exists x : Irishdelegate, On_time (finish (the survey)) x
x : Irishdelegate
H0 : On_time (finish (the survey)) x
============================
On_time (finish (the survey)) x
```

We apply assumption and the *proof* is over. The above can be proved using automated tactics as well. For the puproses of this paper the following tactic has been defined:

```
Ltac AUTO:= cbv delta;intuition;try repeat congruence;  jauto;intuition.
```

The above unfolds all definitions, then tries all intuitionistic first-order tautologies (*intuition*). Then, *congruence* deals with any equalities (for the example interested there are

no equalities). Then *jauto* is applied, which is basically Coq's predefined *auto* tactic along with some pre-processing of the goal. Then again *intuition* is applied. This automated tactic can prove what we want (and many more).