On the Interpretation of Common Nouns: Types Versus Predicates

Stergios Chatzikyriakidis and Zhaohui Luo

Abstract When type theories are used for formal semantics, different approaches to the interpretation of common nouns (CNs) become available w.r.t whether a CN is interpreted as a predicate or a type. In this paper, we shall first summarise and analyse several approaches as found in the literature and then study a particularly interesting and potentially challenging issue in a semantics where some CNs are interpreted as types – how to deal with some of the negated sentences and conditionals. When some CNs are interpreted as types (e.g., *Man*: *Type*), a sentence like `John is a man` can be given a judgemental interpretation j: *Man*, rather than the traditional Montagovian interpretation $man(j)$. In such a setting, the question is then how to interpret negated sentences like `John is not a man` (or more complicated sentences like conditionals). A theory for predicational forms of judgemental interpretations is introduced and is shown to be able to deal with negated sentences and conditionals appropriately. A number of examples are considered to show that the theory provides an adequate treatment in various situations. Furthermore, experiments in the proof assistant Coq are performed in order to provide more supporting evidence for this adequacy. Besides the above, we also briefly study the use of indexed types in order to deal with CNs exhibiting temporal sensitivity and gradability.

S. Chatzikyriakidis—Supported by the Centre of Linguistic Theory and Studies in Probability in Gothenburg.
Z. Luo—Partially supported by the research grants from Royal Academy of Engineering, EU COST Action CA15123, and the CAS/SAFEA International Partnership Program for Creative Research Teams.

S. Chatzikyriakidis (✉)
University of Gothenburg, Dicksonsgatan 4, 41256 Göteborg, Sweden
e-mail: stergios.chatzikyriakidis@gu.se

S. Chatzikyriakidis
Open University of Cyprus, B1 33, Latsia, Nicosia, Cyprus

Z. Luo
Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK
e-mail: zhaohui.luo@hotmail.co.uk

S. Chatzikyriakidis and Z. Luo (eds.), *Modern Perspectives in Type-Theoretical Semantics*, Studies in Linguistics and Philosophy 98, DOI 10.1007/978-3-319-50422-3_3

1 Introduction

The interpretation of common nouns is a central topic for any theory of formal semantics. When type theories are used for formal semantics, there are several approaches as regards interpreting CNs:

- One can interpret CNs as predicates in Church's simple type theory (Church 1940). This is the traditional and dominating approach taken by Montague and his followers (Montague 1973, 1974);
- One may interpret CNs as types. This is the view taken in the formal semantics using Modern Type Theories (MTT-semantics for short) (Ranta 1994; Luo 2012; Luo 2012a);
- One may employ a dependent type theory for formal semantics but still take the traditional view of CNs as predicates (as suggested in, for example, Krahmer and Piwek 1999; Bekki 2014; Tanaka et al. 2015); and
- One may take a view that some CNs are interpreted both as types and predicates at the same time (Retoré 2013).

Interpreting CNs in the above different ways has important implications. To lay down the background of our work in this paper, we shall first, in Sect. 2, briefly discuss these approaches and, in particular, explicate their advantages and potential problems.

Interpreting CNs as types, as in MTT-semantics (the second approach of the above), has several advantages as compared with a CNs-as-predicates approach. These include, for example, the treatment of selectional restrictions as typing and its compatibility with subtyping, whose use has been found useful in recent semantic studies of advanced linguistic features such as copredication (a CNs-as-predicates approach has met difficulties in these respects – see Sect. 2 for more details). However, it is not the case that a CNs-as-types approach does not have its own problems. An important issue to study is the way to interpret some of the negated sentences and conditionals when one takes the CNs-as-types approach.[1] To understand the issue, we need first to appreciate the difference between judgements and propositions in MTTs. In MTT-semantics, besides interpreting most of the sentences as propositions (for example, propositions of type *Prop* in UTT Luo 1994), one may interpret (1) as the judgement in (2), where the CN *man* is interpreted as a type *Man*:

(1) John is a man.
(2) $j : Man$.

Can such a judgemental interpretation of a sentence be turned into a predicational form? Furthermore, in such a setting, how should negated sentences like (3) and conditionals like (4) be interpreted?

[1] Please note that this is not just an issue for MTT-semantics; it is a general problem whenever one has many types to interpret (even some) CNs in formal semantics. Several people have raised this issue of how to interpret negated sentences in MTT-semantics to the second author including, to name a few, G. Morrill (during ESSLLI 2011), N. Asher (in email communications about a paper in LACL 2014) and K. Mineshima (in ESSLLI 2014). We are grateful for their interesting discussions and comments.

(3) Tables do not talk.
(4) If John is a student, he is happy.

We shall study a proposal for predicational forms of judgemental interpretations, including those for negated sentences and conditionals. In particular, in Sect. 3, we shall propose the predicational form for the simple judgements like (2) and further propose and study a special negation operator NOT to be used for interpreting negated sentences like (3) and conditionals like (4). In Sect. 4, one can find many examples of semantic interpretations of negated sentences and conditionals. Experiments of these semantic examples and some related inference examples are done in the proof assistant Coq (The Coq Team 2007) and are reported in Sect. 4. In Sect. 5, we shall illustrate, from a particular angle, that dependent types as found in MTTs are useful in giving semantics to CNs when more advanced issues are considered. In particular, we shall show how to use indexed types, types that are dependent on indexes in a type, to deal with temporal sensitivity and gradability. A summary is given in the Conclusion.

2 Common Nouns as Types or Predicates: Different Approaches

As summarised above, when considering type-theoretical semantics, there are several approaches to the interpretation of CNs. The key difference is whether to interpret a CN as a predicate or as a type. We shall start by discussing the differences between predicates and types and then consider the different approaches in the literature to discuss some of the advantages and potential problems of each of them.

2.1 Types Versus Predicates

A *predicate* in a higher-order logic can be seen as a representation of a set, standing for a collection of elements.[2] For example, in Montague semantics, a predicate of type $e \to t$ represents a subset of the type e of entities. A *type* is also a representation of a collection of objects. For example, in Martin-Löf's type theory, the type N represents the inductively defined collection of natural numbers. Therefore, since both of them are representations of collections, one may think that predicates and types are rather similar. However, their similarities stop here. In fact, predicates (or sets) and types are very different, as to be discussed below. CNs are standing informally for collections: in doing formal semantics, one can either interpret a CN as a predicate or as a type.

[2]A predicate in a higher-order logic may be seen as different from a predicate in first-order logic. Here, for those who are interested, it is instructive to mention Quine's view on this difference (Quine 1986) (although the authors should not be regarded as agreeing with it).

But because predicates and types have important differences, the choice of using one of them to interpret CNs has profound implications.

We shall briefly discuss several differences between predicates (or sets) and types.

Types as Manageable Sets. Intuitively, both sets and types represent collections. However, types are much more 'manageable' than sets (as in set theory). For instance, the membership relation $s \in S$ is a logical formula and whether such a formula is true is undecidable. In contrast, whether the judgement $a : A$, expressing that the object a is of type A, is correct is decidable (in a technical jargon, we say that type checking is decidable). This decidability is essential for type theories in many respects including, for example, the employment of the embedded logic according to the propositions-as-types principle (see below) and the computer implementations of MTTs in proof assistants. This also shows that types are manageable (and simpler than sets or predicates).

In simple type theory (Church 1940) where Montague semantics is based, e and t are (base) types and so are function types such as $e \to t$. In an MTT, such as Martin-Löf's type theory (Martin-Löf 1984; Nordström et al. 1990) or UTT (Luo 1994), besides the function types, there are many more types such as dependent function types (Π-types), dependent sum types (Σ-types), type universes, etc. However, in MTTs there are still much fewer types than sets. For example, one can form unions or intersections of sets, but one cannot do so for types: if one did this, type checking would become undecidable.

Embedded Logic. MTTs have consistent embedded logics according to the propositions-as-types principle (Curry and Feys 1958; Howard 1980). For instance, in Martin-Löf's type theory, the embedded logic is a (non-standard) first-order logic as explained in Martin-Löf (1984) and, in UTT, the embedded logic is the higher-order logic as explained in Chap. 5 of Luo (1994). Based on the normalisation theorem (e.g., in the case of UTT), one can show that the embedded propositions-as-types logic is consistent.

This is in contrast to some other systems such as set theory. It is important to mention that, in order to employ a consistent logic (e.g., higher-order logic in the Montagovian setting or propositions-as-types logics in MTTs), type checking must be decidable. This is because, as a logical system, given a formula A and a potential proof a, it should be decidable whether a is a proof of A. If one uses the propositions-as-types principle, this means that type checking (i.e., the derivability of judgements $a : A$) must be decidable.[3] Therefore, in systems like set theory, or some systems with :-judgements that do not have the decidability property, one does not have an embedded logic for applications (e.g., in giving a formal semantics).

Equalities Associated with Types. MTTs were originally developed as foundational languages for constructive mathematics, where types represent constructive

[3]Decidability of type checking means that of the judgement $a : A$. In Church's simple type theory (as used in Montague semantics), it means that, for example, it is decidable to check whether $j : e$, $p : t, f : e \to t$, etc. This obviously has to be decidable for, otherwise, the embedded logic (HOL in Montague's semantics) could not be used in a feasible way (for example, we would not be able to apply its rules).

collections. Constructive collections are associated with their own equalities.[4] This is in contrast to set theory where there is a universal equality between sets (and their elements which formally are sets as well). When types in MTTs are used to interpret CNs, it is natural to think that the CNs are associated with their own equalities as well, although many of them share the same ones (but some of them do not). Note that this is different from set-theoretical semantics such as Montague where a universal equality exists. From a philosophical point of view, it is argued in Luo (2012) that the idea of interpreting CNs as types ties well with Geach's criterion of identity for CNs (Geach 1962). We shall not talk about this in the following pages and the interested readers may consult (Luo 2012) where this has been studied.

2.2 *Different Approaches to Interpretation of CNs*

Common nouns are usually interpreted as standing for collections in formal semantics. In type theories, as we have already mentioned, they can be interpreted either as predicates or as types. We describe several approaches in the literature and briefly analyse their advantages and, in particular, potential problems.

CNs as Predicates. In the formal semantics literature, the traditional approach is to interpret common nouns as predicates, as initiated in Montague Grammar (Montague 1974) where CNs are predicates of type $e \to t$. This has been the dominating approach in the last several decades and the most obvious advantage is its robustness: it has been extensively studied in various scenarios and is well-understood.

However, this does not mean that the CNs-as-predicates approach does not have its problems. For example, some critics have pointed out that Montague semantics is model-theoretic and lacks support in proof-theoretical studies and therefore, for example, it is not suitable for implementation of reasoning tools. Some people think that this is a regrettable weakness for model-theoretic semantics because reasoning is one of the major motivations of conducting formal semantics in the first place.

Another problem with the CNs-as-predicates approach is its incompatibility with subtyping, as shown and discussed in Luo (2010, 2012a). When modelling some advanced linguistic features, one may introduce various subtypes of entities. For instance, consider the linguistic feature of copredication as exemplified in the following sentence:

(5) John picked up and mastered three books in the library.

where *book* exhibits both a physical and an informational aspect. In dealing with this, Asher (2012) considers the subtypes $\textsc{Phy} < e$ and $\textsc{Info} < e$ of physical objects and informational objects respectively, and interprets `pick up` as of type $\textsc{Phy} \to t$ and `master` as of type $\textsc{Info} \to t$. Furthermore, in the Montagovian setting where CNs are predicates, one would interpret CNs such as `book` as predicates of dot-type $\textsc{Phy} \bullet \textsc{Info}$:

[4] We cannot explain this in details. See, for example, Beeson (1985) among others.

$$book : \text{PHY} \bullet \text{INFO} \rightarrow t.$$

Unfortunately, the introduction of subtypes PHY and INFO is incompatible with the above interpretation of *book* as a predicate. To see the problem, consider the adjective `heavy` which can only be applied to physical objects:

$$heavy : (\text{PHY} \rightarrow t) \rightarrow (\text{PHY} \rightarrow t)$$

Now, how can one interpret the phrase `heavy book`? In order to type

$$heavy(book),$$

the type of `book` should be a subtype of the domain type of `heavy`, i.e.:

$$\text{PHY} \bullet \text{INFO} \rightarrow t \ \leq \ \text{PHY} \rightarrow t,$$

and, by contravariance, we must have

$$\text{PHY} \leq \text{PHY} \bullet \text{INFO}.$$

But this is in the wrong direction (just the other way around!) – intuitively, not every physical object has an informational aspect. We shall show below that, when CNs are interpreted as types, the above problem disappears and things work as intended. Therefore, we may conclude that the CNs-as-predicates approach is incompatible with the introduction of subtypes such as PHY and INFO in dealing with advanced linguistic features.

CNs as Types. An alternative to the CNs-as-predicates view comes from Modern Type Theories: instead of interpreting CNs as predicates, one may interpret them as types. This CNs-as-types approach was first studied in Ranta's seminal work on using Martin-Löf's type theory in formal semantics (Ranta 1994). For instance, the sentence (6) is interpreted as (7):

(6) Every student talks.
(7) $\forall x : Student.\ talk(x)$

where *Student* is a type that interprets the CN `student` and $talk : Human \rightarrow Prop$ interprets the verb `talk`.

The CNs-as-types approach, as we have already briefly mentioned, has several advantages as compared with the CNs-as-predicates approach. For example, it has been used successfully to deal with selectional restrictions and copredication (Luo 2010) and has been applied to practical reasoning by implementing these ideas in the Coq proof assistant (Chatzikyriakidis and Luo 2014, 2015). For instance, selectional restrictions like the ones shown below are handled by virtue of the many sorted system, with sentences like these producing a semantic type mismatch and therefore function application is impossible:

(8) (#) The television ate the cake.

where the predicate $[\![eat]\!]$: *Animal* $\rightarrow$ *Prop* needs an argument of type *Animal* while *the television* is a physical object not of type *Animal*.

Adopting the CNs-as-types approach, it is important to note that there must be a compatible subtyping mechanism in the type-theoretical framework for, otherwise, the approach would not be viable. For instance, in the semantics (7) of the above example, *Student* is a subtype of *Human*; it is this subtyping relationship that makes the application $talk(x)$ in (7) well-typed. Fortunately, there is a subtyping mechanism called *coercive subtyping* (Luo 1999; Luo et al. 2012) that is suitable for modern type theories and that, as shown by the second author (Luo 2010, 2012a), makes the CNs-as-types approach viable. The authors of the current paper have studied this approach systematically to show how various classes of CNs with adjectival and adverbial modifications can be interpreted as types in MTTs (Chatzikyriakidis and Luo 2013, 2016). We shall consider this further in Sect. 5, i.e. how dependent types can be used to interpret CNs in more advanced situations.

Interpreting CNs as types, one can have a straightforward treatment of copredication. Considering the above example (5) (cf., the discussion following that), `book` is interpreted as a type, a subtype of PHY $\bullet$ INFO (a book has both physical and informational aspects):

$$Book \leq \text{PHY} \bullet \text{INFO}.$$

So in MTT-semantics, especially with the dot-types as defined in Luo (2010), the coordination in (5) can be treated as intended since `pick up` and `master` are both of type *Book* $\rightarrow$ *Prop*:

$$\texttt{pick up} : \text{PHY} \rightarrow Prop \leq \text{PHY} \bullet \text{INFO} \rightarrow Prop \leq Book \rightarrow Prop$$
$$\texttt{master} : \text{INFO} \rightarrow Prop \leq \text{PHY} \bullet \text{INFO} \rightarrow Prop \leq Book \rightarrow Prop$$

Put in another way, the problem of incompatibility with subtyping is not a problem in the CNs-as-types approach.

Of course, the CNs-as-types approach, as a relatively new way to interpret CNs, may have its own problems or issues worth being investigated further. One of them is how to deal with negated sentences as mentioned in the introduction; we shall devote Sects. 3 and 4 in the current paper to propose a solution to this.

Another issue worth paying a special attention to is the need of *proof irrelevance* in formal semantics based on MTTs. This issue has been studied by the second author in Luo (2012), where it is shown that, in order to give an adequate formal semantics, the MTT that is used must have the ability to impose proof irrelevance. Intuitively, proof irrelevance means that, for any logical proposition P and any two proofs p and q of P, p and q are equal. The importance of this can be explained by means of an example using Σ-types to interpret some of the CNs modified by adjectives. For instance, the phrase `handsome man` can be interpreted as $\Sigma(Man, handsome)$, where *Man* is a type, *handsome*: *Man* $\rightarrow$ *Prop* and the Σ-type consists of pairs (m, p) such that p is a proof that m is handsome. In such examples, proof irrelevance

is required in order to obtain correct identification criteria for handsome men: we need that $(m, p) = (m', p')$ if and only of $m = m'$; put in another way, as soon as m and m' the same, so are p and q. (See Luo 2012 for a more detailed explication.)

It is worth pointing out that, if the above argument (or that given in Luo 2012) is correct, then it would not be adequate to use Martin-Löf's type theory for formal semantics because one cannot impose that every two proofs should be identified since, in Martin-Löf's type theory, propositions and types are identified. If one did identify all proofs/objects of the same proposition/type in Martin-Löf's type theory, one would not have types with more than one object anymore! That would of course be absurd. Therefore, it seems that it is better to use an MTT where there is a clear distinction between propositions and types. In our work on MTT-semantics, we have used UTT (Luo 1994) where there is a distinctive (and impredicative) type *Prop* of logical propositions. UTT is used in the examples in the current paper as well.

CNs as Predicates in a Dependent Type Theory. A number of researchers working on the application of constructive type theories into linguistic semantics have taken the traditional route, interpreting CNs to be predicates rather than types. Such proposals can be found in Krahmer and Piwek (1999); Bekki (2014); Tanaka et al. (2015) among others. For instance, Bekki and his colleagues have considered DTS (Bekki 2014), a system with Π-types, Σ-types, the type of natural numbers and the equality types (as in Martin-Löf's type theory), together with a newly introduced @-operator to deal with anaphora and presupposition. The propositions-as-types principle is adopted so that the embedded logic as in Martin-Löf's type theory can be used for formal semantics.[5]

In DTS, CNs are interpreted as predicates. For example, instead of interpreting the CN `man` as a type, the DTS-interpretation of `man` is a predicate **Man** with domain **e**, the type of entities.[6] This employs the traditional Montagovian approach to CNs and has the advantage of being well-understood (but of course looses the advantageous points of the CNs-as-types approach discussed above).

In DTS, Π and Σ are used as logical operators and play a useful role in defining its notational devices called dynamic conjunction/disjunction for dealing with anaphoric expressions/discourses. Unlike the CNs-as-types approach, using DTS for formal semantics does not use other types available in a modern type theory to interpret CNs or to deal with other linguistic issues (for example, it does not employ disjoint union types or type universes as in MTT-semantics to interpret more

[5]DTS as described in Bekki (2014) does not seem to contain the other logical type constructors such as disjoint union or the empty type. It is unclear how to define some of the logical operators such as negation without the empty type. We might assume that DTS be extended with such types and type constructors, since a logic in DTS is needed for doing formal semantics. (Bekki and Mineshima, in a private communication with the authors after seeing this footnote, have clarified that DTS is intended to contain these type constructors; this is further clarified by adding a new paragraph in Sect. 1.3 of their paper in the current collection Bekki and Mineshima 2016).

[6]Formally, the type of a predicate is the function type $\mathbf{e} \to U$, where U is either a type universe or a totality of propositions. We assume that DTS be extended with such a type.

sophisticated noun phrases such as privative adjectival modification). Like the traditional CNs-as-predicates approach in Montague semantics, it suffers from the incompatibility with subtyping extensions. For example, contrary to the claim made in Tanaka et al. (2015), if we extended DTS with types PHY of physical objects and INFO of informational objects, the problem as discussed above would still exist when one tries to interpret the phrases such as `heavy book`.

CNs as Both Predicates and Types. Another interesting alternative suggestion proposed by researchers like Retoré (2013) (and, to some extent, Asher et al. 2016), argues that we could employ *both* predicates and types in interpreting (some) CNs. For instance, the CN `man` can be interpreted as both a type *Man* and a predicate $man : e \to t$ (for example, both *Man* and *man* being constants). Then, we could use a suitable interpretation: to use *Man* when a type is required and *man* when a predicate is. For instance, one can use a CNs-as-types interpretation when dealing with the copredication feature and a CNs-as-predicates interpretation in some other situations. This may sound very desirable indeed, but we think it has a potentially serious problem: it is unclear how the two interpretations of the same CN relate to each other. For this to work, we would need, for example:

(*) For any $x : e$, $x : Man$ if and only if $man(x)$ is true.

In other words, $(*)$ is the basis for having two interpretations of `man` (type *Man* and predicate *man*) at the same time! Unfortunately, we cannot simply stipulate that the requirements like $(*)$ be the case, for if we did this in general, there would be problems with the setting (in Church's simple type theory) including that type checking would become undecidable. This is because the truth of a logical formula is undecidable and, as a consequence, one cannot employ the embedded logics in such systems. We do not see how we can sidestep this problem, if at all. Research seems called for to clarify this further.

3 Predicational Forms of Judgements in Formal Semantics

As mentioned in Sect. 1, the MTT-semantics, which takes the CNs-as-types approach, faces the problem of interpreting negated sentences and conditionals. We make a proposal for predicational forms of type-theoretical judgements so that the corresponding predicational forms of judgemental interpretations in MTT-semantics can be used in interpreting composite sentences with those sentences as components.[7]

[7]This proposal was started by considering a predicational form for a non-hypothetical judgement, proposed in an email communication by the second author with Koji Mineshima during ESSLLI 2014. It was further developed to consider the logical operator NOT, first proposed in Chatzikyriakidis and Luo (2014) and further developed here, for negated judgements $\nvdash a : A$ to interpret negated VPs and hypothetical judgements (e.g. conditionals).

3.1 Predicational Forms of Non-hypothetical Judgements

As explained above, in MTT-semantics, most of the sentences are interpreted as propositions. However, because CNs are interpreted as types, some sentences are better interpreted as judgements, rather than propositions. For instance, the sentence (1) can be interpreted as (2), repeated here as (9) and (10), respectively:

(9) John is a man.
(10) $j : Man$

where *Man* is a type. A judgement of the form (10) is called non-hypothetical because it does not depend on other contextual assumptions. Formally, a non-hypothetical judgement is a judgement $\Gamma \vdash a : A$ such that the context Γ is empty.

In general, the predicational form of the non-hypothetical judgement $a : A$ is the proposition $p_A(a)$, defined in the following definition.

Definition 1 (*predicate* p_A) Let A be a type. Then, predicate $p_A : A \to Prop$ is defined as: for any $x : A$, $p_A(x) = true$, where $true : Prop$ is any (fixed) tautological proposition.

It is straightforward to see that $a : A$ is derivable if, and only if, $p_A(a)$ is true (in the sense that the synthetic judgement '$p_A(a)$ *true*' is derivable).

Example. Consider (9) and its judgemental interpretation (10). The corresponding propositional interpretation of (9) is (11), which is a proposition:

(11) $p_{Man}(j)$.

Therefore, for example, the composite sentence (12) can be given semantics in (13):

(12) John is a man and he is happy.
(13) $p_{Man}(j)$ & $happy(j)$.

Remark It is worth noting that the semantic meanings of two logically equivalent propositions may be (intensionally) different. For instance, when $j : Man$, the proposition $p_{Man}(j)$ is logically equivalent to the true proposition *true*. However, the well-typedness of $p_{Man}(j)$, i.e., that $p_{Man}(j)$ is a proposition of type *Prop*, *presupposes* the derivability (or, informally, correctness) of the judgement $j : Man$, while the well-typedness of *true* does not.

3.2 Predicational Forms of Negated Judgements

Consider the following negated sentence:

(14) John is not a man.

Intuitively, the interpretation of (14) would correspond to the negated judgement $\not\vdash j : \mathit{Man}$, where j is the interpretation of John. What would be a predicational form of such a negated judgement?

Before answering this question, we should first note that it would not be always correct to interpret (14) as $\neg p_{\mathit{Man}}(j)$ because its well-typedness presupposes that $j : \mathit{Man}$ (cf., the above remark). Put in another way, we should realise that, in (14), the interpretation of John may not always be of type *Man*; instead, it could be some object which is not a man.

In order to interpret such cases of negation in NL sentences like (14), we have proposed the following operator NOT[8]:

$$\text{NOT} : \Pi A : \text{CN}.\ (A \to \mathit{Prop}) \to (\mathit{Obj} \to \mathit{Prop}),$$

where *Obj* is the top type in the universe CN of (the interpretations of) common nouns. That is, for every $A : \text{CN}$, $A \leq_{c_A} \mathit{Obj}$ for some c_A.

Notation We shall use the following notations.

- For $a : A$, we shall use $\bar{a}$ to denote $c_A(a)$.
- We often omit the first argument of NOT and the coercion operator of its third argument by writing $\text{NOT}(p, b)$ for $\text{NOT}(A, p, \bar{b})$.

The negation operator NOT satisfies several logical laws including the following concerning its relationship with the predicate p_A as defined in Definition 1:

(L1) For any $A : \text{CN}$ and $a : A$, $\text{NOT}(p_A, a) \iff \neg p_A(a)$.
(L2) If $A \leq B$, then $\text{NOT}(p_B, c) \implies \text{NOT}(p_A, c)$, for any $c : C : \text{CN}$.

Using NOT, the predicational interpretation of (14) is the proposition in (15):

(15) $\text{NOT}(p_{\mathit{Man}}, j)$.

There are two possibilities according to the interpretation j of John: (1) if $j : \mathit{Man}$ then, by the above law (L1), (15) is logically equivalent to $\neg p_{\mathit{Man}}(j)$, which is logically false; (2) if $j : A$ for some A which is not *Man*, then the truth value of (15) could be either true or false, or even unknown.

Here are some further examples with negated VPs. The propositional interpretations of (16) and (17) are (18) and (19), respectively:

(16) Tables do not talk.
(17) Red tables do not talk.
(18) $\forall x : \mathit{Table}.\ \text{NOT}(\mathit{talk}, x)$.
(19) $\forall y : \Sigma(\mathit{Table}, \mathit{red}).\ \text{NOT}(\mathit{talk}, y)$.

where $\mathit{talk} : \mathit{Human} \to \mathit{Prop}$, $\mathit{red} : \mathit{Phy} \to \mathit{Prop}$ and $\mathit{Table} \leq \mathit{Phy}$. We have used Σ-types to interpret modified CNs: $[\![\mathit{red\ table}]\!] = \Sigma(\mathit{Table}, \mathit{red})$. Because $\Sigma(\mathit{Table},$

[8] NOT was first proposed in Chatzikyriakidis and Luo (2014) to deal with negated VPs. We develop the idea further in this paper.

red) $\leq$ *Table* (assuming that the first projection be a coercion), it is straightforward to see that (18) implies (19) (see Sect. 4 for these and more examples and their Coq verifications, including uses of the laws (L1) and (L2) for NOT).

3.3 Conditionals: Predicational Forms of Hypothetical Judgements

Conditionals occur in sentences like (20):

(20) If John is a student, he is happy.

Since its premise has a judgemental interpretation, (20) may be interpreted as corresponding to the following judgement:

(21) $j : \mathit{Student} \vdash \mathit{happy}(j)\ \mathit{true}$

What is the predication form of (21)? In general, what are the predicational forms of hypothetical judgements of the form $\Gamma \vdash J$, where $\Gamma \equiv x_1 : A_1,\ \ldots,\ x_n : A_n$ is non-empty (i.e., $n \geq 1$)? This requires us to consider the following predicate P_A (with P in capital), defined by means of the NOT operator.

Definition 2 (*predicate* P_A) Assume that $A,\ B : \mathrm{CN}$. Then, predicate $P_A : B \to \mathit{Prop}$ is defined as: for any $x : B$, $P_A(x) = \neg\mathrm{NOT}(p_A, \bar{x})$, where p_A is the predicate defined in Definition 1.

Intuitively, one may regard $P_A(b)$ as a propositional representation of the judgement that b is of type A, even in a position as a premise. For instance, the predicational form of the hypothetical judgement (21) that interprets (20) is (22), which is the abbreviation of (23):

(22) $P_{\mathit{Student}}(j) \implies \mathit{happy}(j)$
(23) $\neg\mathrm{NOT}(p_{\mathit{Student}}, j) \implies \mathit{happy}(j)$

where $j : B$ for some $B : \mathrm{CN}$.

Remark Note that it would not be correct to interpret (20) as the following proposition with the predicate p_{Student} (with a small p):

(24) $p_{\mathit{Student}}(j) \implies \mathit{happy}(j)$

since the well-formedness of (24) would have presumed that $j : \mathit{Student}$ which may not be the case when one utters (20). Put in a more precise way, there are two possibilities here in understanding (20):

1. $j : \mathit{Student}$. In this case, (24) would be correct and logically equivalent to (22), by the law (L1) for NOT.
2. $j : A$ (and A is not *Student*; e.g., $A = \mathit{Human}$).[9] In this case, (24) would be incorrect ($p_{\mathit{Student}}(j)$ is not well-typed) and the correct interpretation of (20) is (22).

[9] This is the case, for example, when we do not know the type of John in the NL context.

4 Examples with Coq Experiments

We have put the above proposal for predicational forms into the proof assistant Coq (The Coq Team 2007), where the MTT-semantics has been implemented (Chatzikyriakidis and Luo 2014). In particular, we have implemented the setting of predicational forms, considered several relevant examples and their Coq implementations, and conducted examples of reasoning for the corresponding predicational semantics. Space considerations do not allow us to give a detailed exposition of how Coq works – see Appendix A where a brief introduction to how Coq works is given, and Appendix B where one can find all of the data and assumptions for the examples in this paper to go through.

4.1 General Set-Up for Predicational Forms

We shall implement the basic set-up of predicational forms in Coq. First of all, the predicate p_A in Definition 1 is defined as `pr` in Coq as follows:

```
Definition pr:= fun (A:CN)(a:A)=>True.
```

With `pr`, the predicational form of j: *Man* that interprets 'John is a man', is the following proposition (cf. (9–10))

```
pr Man John: Prop
```

If we unfold the definition, this will just return `True`.

Now, the next step is to introduce the negation operator NOT.

```
Parameter NOT: forall A:CN, (A->Prop)->(Object->Prop).
```

When declaring `NOT`, we have made the first CN-argument to be *implicit*, using Coq's command `Set Implicit Arguments`; this means that, when we write `NOT`-expressions, its first argument is omitted, just like our informal notations. For example, instead of `NOT Human talk John`, we omit `Human` by writing `NOT talk John`, which Coq now accepts.

We need then to implement the logical laws (L1-L2) for NOT. We implement these laws using the *Variable* declarations. With this, and given that we have introduced a local section, these laws will appear as part of the context of the relevant proofs. However, the logical laws (L1-L2) are generic: (L1) is for arbitrary $A \leq Obj$ and (L2) is for any $A \leq B$. Both of these conditions are not representable generically in Coq,[10] although every individual case can be done. For instance, for the first law (L1): for

[10]To get such genericity would require the so-called *bounded quantification* which is usually not available in a type theory of any sophistication.

any $a: A \leq Obj$, $\neg\text{NOT}(p_A, a) \iff p_A(a)$, Coq will complain that $\text{NOT}(p_A, a)$ is ill-typed because there is no way to assume an arbitrary subtype A of Obj. Although to do this for each individual case is tedious, it still can be done. We have to declare the laws for each type in the case of the first law and each subtyping relation for the second one. We present the rules for the type *Man* and the subtyping relation $Man < Human$.

```
Variable NOT1_m: forall a:Man, (NOT (pr Man) a) <->
not (pr Man a).
Variable NOT2_hm: forall (a : Human),  NOT  (pr Human)
a -> NOT (pr Man) a.
```

Finally, the predicate P_A in Definition 2 is defined as PR in Coq. Again the same problems of generality arise so we show the definition with the B in the original definition replaced by types *Man* and *Object* respectively (we note this with (m, o) following *PR* in the code):

```
Definition PR_m := fun A:CN, fun x:Man => not (NOT
(pr A) x).
Definition PR_o := fun A:CN, fun x:Object => not (NOT
 (pr A) x).
```

4.2 Examples with Explanations

We now give some examples to demonstrate how predicational forms are used to give propositional semantics of negated sentences and conditionals when judgemental interpretations are involved. For each example, we shall give its MTT-semantics as well as its Coq representation and, if needed, we shall explain the related reasoning about the semantics.

Example 1

(25) John is a man and John is not a man.
(26) $p_{Man}(j)$ & $\text{NOT}(p_{Man}, j)$, where $j: Man$
(27) `pr Man John` $\wedge$ `NOT (pr Man) John`

Note that, by the first law (L1) of NOT, the above sentence is equivalent to false.

Example 2

(28) Tables do not talk.
(29) $\forall x: Table.\ \text{NOT}(talk, x)$.
(30) `forall x:Table, NOT talk x`

where $talk: Human \rightarrow Prop$.

Example 3

(31) Red tables do not talk.
(32) $\forall y\colon \Sigma(\mathit{Table}, \mathit{red}).\ \mathrm{NOT}(\mathit{talk}, y)$.
(33) `(all redtable) (NOT talk).`

where `redtable` is defined as a Σ-type, implemented by means of Coq's equivalent dependent record mechanism (see Chatzikyriakidis and Luo 2014):

```
Record redtable : CN := mkredtable{rt :> Table; _ : Red rt}.
```

where `Red : Object -> Prop` being a predicate over the top type `Object` in CN, and `all` is the name of the standard universal quantifier:

```
Definition all := fun A:CN, fun P:A->Prop => forall x:A,
 P x.
```

We can now prove the following theorem since `redtable` is a subtype of `Table` with the first projection as coercion:

```
Theorem TABLE : (all Table) (NOT talk) -> (all redtable)
 (NOT talk).
```

To prove this, we `unfold` all the definitions and apply `intros`, moving the consequent as a hypothesis:

```
TABLE < cbv. intro.
1 subgoal
H : forall x : Table, NOT talk x
x : redtable
============================
 NOT talk (let (r3, _) := x in r3)
```

Then, applying H will complete the proof.

Example 4

(34) It is not the case that John is not a man.
(35) $\neg\mathrm{NOT}(p_{\mathit{Man}}, j)$, where $j\colon \mathit{Man}$.
(36) `not (NOT (pr Man) John)`

We can show that this sentence is true. We turn this into a theorem and see if we can prove it. If it is true, this should be possible:

```
Theorem NOTNOTMAN: not (NOT (pr Man) John).
```

Here, for simplicity, we have introduced the classical law (as an axiom). Coq goes into proof-mode:

```
Classical_Negation : forall P : Prop, ~ ~ P <-> P
NOT_1m : forall x : Man, NOT (pr Man) x -> ~ pr Man x
============================
 ~ NOT (pr Man) John
```

where the hypothesis `NOT1_m` is an instance of the (L1) law for the type *Man* (as discussed earlier). We unfold the definitions with *cbv* and also unfold *pr* in the type of `NOT1_m`. Then, we call a variant of Coq's built-in *auto* tactic, *eauto*. This suffices to get a proof.

Example 5

(37) If John is a student, he is happy.
(38) $P_{Student}(j) \implies happy(j)$
(39) `PR_m Student John -> happy John.`

Note that the semantics of the above sentence is not $p_{Student}(j) \implies happy(j)$; instead, $P_{Student}$ as defined in Definition 2 should be used. As remarked at the end of Sect. 3.3, if $j : Student$, we can show that $P_{Student}(j)$ is equivalent to true and the above sentence is logically equivalent to $happy(j)$.

Example 6

(40) Some linguists are logicians.
(41) $\exists l : Linguist.\ \text{NOT}(p_{Logician}, l)$
(42) `(some Linguist) (NOT (pr Logician)).`

Unfolding the definition of `some` in (42), we get:

(43) `exists x:Linguist, NOT (pr Logician) x`

Please note that the semantics of this example cannot be $\exists l : Linguist.\ \neg p_{Logician}(l)$, which would be ill-typed because l is not of type *Logician*.

Example 7

(44) It is not the case that John is a student.
(45) $\neg P_{Student}(j)$, where $j : Human$.
(46) `Theorem JOHNNOTSTUDENT: not (NOT (pr Student) John).`

Note that, by Definition 2, (45) is the abbreviation of $\neg\neg\text{NOT}(p_{Student}, j)$, which is (classically[11]) equivalent to $\text{NOT}(p_{Student}, j)$, the semantics of 'John is not a student'.

[11] In order to do this in Coq, we need to introduce a classical axiom (say, the double negation axiom).

Example 8

(47) It is not the case that every linguist is a logician.
(48) $\neg\forall l\colon \mathit{Linguist}.\ P_{\mathit{Logician}}(l)$
(49) `not (forall l:Linguist, PR_l Logician l)`

where `PR_l` is the predicate P_{Linguist}, similarly defined as `PR_m` at the end of Sect. 4.1. Classically, the semantics is equivalent to $\exists l\colon \mathit{Linguist}.\ \textsc{not}(p_{\mathit{Logician}}, l)$. Indeed, we can prove the following theorem in Coq (the only if part of this equivalence):

```
Theorem NOTLL : not (forall l:Linguist, PR_l Logician l)
                -> exists l:Linguist, NOT (pr Logician) l.
```

Example 9

(50) It is not the case that every human is a logician.
(51) $\neg\forall h\colon \mathit{Human}.\ P_{\mathit{Logician}}(h)$
(52) `not (forall h:Human, PR_l Logician h)`

Similarly to the previous example, the semantics is equivalent to $\exists h\colon \mathit{Human}.\ \textsc{not}(p_{\mathit{Logician}}, h)$. Most importantly we can prove (51) from (48):

```
Theorem NOTHL: not (forall l:Linguist, PR_l Logician l)
            -> not (forall h:Human, PR_h Logician h).
```

Example 10

(53) If John is not a human, then John is not a man.
(54) $\textsc{not}(p_{\mathit{Human}}, j) \implies \textsc{not}(p_{\mathit{Man}}, j)$
(55) `NOT (pr Human) John -> NOT (pr Man) John`

By the second law (L2) for NOT, the above is true because $\mathit{Man} \leq \mathit{Human}$. In Coq, we can prove the following:

```
Theorem NOTHUMANMAN: NOT (pr Human) John -> NOT (pr Man) John.
```

5 Indexed Types for Interpretation of CNs

The semantics of common nouns involve a number of fine-grained distinctions that require careful treatments that go beyond a straightforward standard interpretation. For example, there are clear cases like gradable nouns or nouns exhibiting temporal sensitivity whose semantic treatments are difficult, if not impossible, to be handled using the standard view:

(56) Dead rapper fired shot first.
(57) What an enormous idiot!

In the first example taken from Tonhauser (2002), the rapper fired the shot while still alive and not after he died. In order to capture this, one has to allow a time argument for the nominal that can be potentially different to the time at which the rest of the sentence is true. In the second example, the most natural reading we get, as Morzycki (2009) argues, is not one about large physical size but rather of the nominal holding to a high degree. Although the problem of handling this type of cases is orthogonal to the interpretation of CNs as predicates or as types, to deal with them in the CNs-as-types approach may be more challenging because there are fewer types than predicates and in general, it is more delicate when dealing with types rather than with predicates. Also, the issue of providing accounts for gradable nouns or the temporal sensitivity of nouns is far too complex to engage with in full in the current paper. In this section, we will try to show how one can approach this type of phenomena in MTTs, taking the CNs-as-types view. In particular, we will show how more fine-grained issues such as those above can be handled using indexed types in MTTs.

Indexed types are special kinds of dependent types – types dependent on indexes. Put in another way, indexed types are just families of types that are indexed by a type parameter whose type is usually a simple one. Example types of indexes include, for example, the type N of natural numbers, the type *Human* of human beings, and the type *Height* of heights (see below). For instance, one may consider, for h: *Human*, a family of types $Evt(h)$ of events that are conducted by h; such a type family was used in Asher and Luo (2012) to study linguistic coercions in formal semantics.[12] In this section, we show that indexed types also provide useful tools to deal with the extra fine-grainedness required in advanced treatment of CNs.

The general idea is that CNs like *Human*, *Man* and *Table* might come in different guises depending on the type they are indexed each time. Their general form is a CN-valued function from indexes. For example, one might wish to have the family of types of humans that are indexed with a *Height* parameter:[13]

(58) $Human : Height \rightarrow \text{CN}$

Then, $Human(n)$ is the family of types of humans with height n. This idea has been used by the authors in Chatzikyriakidis and Luo (2014) to deal with gradable adjectives as well as adjectives involving temporal relations like *former*. According

[12] In general, dependent types are useful in describing dependent event types, giving a fine-grained treatment of Davidsonian event semantics (Davidson 1967; Parsons 1990). In this respect, an initial study of how event semantics can be considered in dependent type theories has been carried out (Luo and Soloviev 2016) and further work is at the moment in progress.

[13] A similar idea has been used by de Groote (2007) in discussing type-theoretic extensions of abstract categorial grammars. However, indices there are used on the (linguistic) syntactic level, trying to capture morphological considerations (e.g. gender, number etc.); in effect those indices correspond to elements that are morphologically realized. In our case, the approach deals with things that are not morphologically realized (e.g. grades) and is purely on the (linguistic) semantic level.

to Chatzikyriakidis and Luo (2014), gradable adjectives are interpreted as predicates that take an indexed CN argument. For example in the case of *tall*, the argument is of type $Human(n)$, which is indexed by n, a height parameter. This is essential if one wants to reason about heights, e.g. in the inference example below:

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

See Chatzikyriakidis and Luo (2014, 2016) for more details on the analysis of comparatives.

The above idea of types indexed by heights can be generalised in that types can be indexed by any degree of measurement. One can introduce a type universe *Degree* of all degrees which contain all measurement types such as $Height$, $Weight$, $Width$, etc. Also, some nouns are indexed by more than one parameter. For instance, we may have that humans are indexed by one's height as well as one's weight. In such a case, we would have: $Human : Height \rightarrow Weight \rightarrow$ CN and, therefore, a type $Human(n, w)$ would be indexed by both height n and weight w.

Now, let us go back to example (57) repeated below:

(60) What an enormous idiot!

The problem here is that *enormous idiot* does not designate an idiot who is enormous in terms of size but rather enormous in terms of idiocy. Morzycki (2009) argues that these nouns have similarities with gradable adjectives and attempts a similar analysis to adjectives like *tall* for the noun *idiot*. The idea is that *idiot* involves a measure function from individuals to their degree of idiocy. On the other hand, Constantinescu (2013) assumes that the interpretation in *huge idiot* is one of size as in the case of *huge man*, albeit with the following difference: nouns involving abstract parameters as well as abstract nouns denote kind of tropes. The idea of tropes has been developed by Moltmann (2003, 2007) and in simple terms it defines kinds of tropes as a 'universal whose instances are concrete property manifestations' (Moltmann 2003, 2007). Constantinescu (2013) exploits this idea and proposes two types of nouns involving tropes: type A, i.e. nouns involving abstract parameters like *idiot* and type B, i.e. abstract nouns like *wisdom*, *generosity* etc. The proposal is that type A nouns involve types of individuals in which a property (for example idiocy) is instantiated while type B nouns are analyzed as two place predicates, basically a relation between tropes and individuals that instantiate this trope (for example generosity(p,x)).

Such an idea is very naturally interpreted using Sigma-types. Let us explain. According to Luo (2012), a possible characterization of CNs is by means of Σ-types. For example, the CN *passenger* can be represented by the following Σ-type:

(61) $Passenger[T] = \Sigma p : Person.Journey[T](p)$

In the above example, *Person* is the type of persons and *Journey[T]*(p) the finite type of journeys in T that the person p has made. There is a way to deal with nouns with

abstract parameters and abstract nouns in the same sense. For example, *idiot* can be represented as the following Σ-type:

(62) $Idiot = \Sigma i : Idiocy.Human(i) \times i > STND$

Intuitively, an object of the above type is a triple (i, h, p) consisting of a degree i of idiocy, a human h whose idiocy degree is i and a proof that i is bigger than the idiocy threshold *STND*. Note that this account has not only similarities with the ideas proposed in Constantinescu (2013) but also brings out a connection with gradable adjectives in the sense that they both involve a degree parameter. However, these two constructions are clearly different in terms of their formal status, adjectives being predicates while nouns types.

With these assumptions in line, we now have to show to get the interpretation we need in the cases like *enormous idiot*. As already said, we take gradable adjectives to take arguments of type $D : Degree \rightarrow \text{CN}$ indexed by grade parameters $D : Degree$. Thus, we can assume the following type for enormous:

(63) $Enormous : \Pi D : Degree.\Pi A : D \rightarrow \text{CN}.\ A(d) \rightarrow Prop$

However, this type does not say anything about the semantics of *enormous*. intuitively, what we want to get in the case of *enormous idiot* is someone who is an idiot to a high degree. This means that this degree must be higher than the degree of idiocy needed for someone to be considered an idiot (*i* in the case of *idiot)*. Assume that there are different standards depending on the case: first there is a standard of idiocy. One needs to exceed this in order to be counted as an idiot. Then, there is a polymorphic standard for enormous. Above this standard a given degree is considered as being enormous:

(64) $STND_{id} : Idiocy$
(65) $STND_{en} : \Pi D : Degree.D$

With this in line, one can assume the following type for *enormous*:

(66) *enormous*:
$\Pi D : Degree \quad \Pi A : D \rightarrow \text{CN} \quad \Pi p : (\Sigma d : D.\ \ d > STND_{en}(D)).\ \ A(\pi_1(p)) \rightarrow Prop$

So, what about the interpretation of *enormous idiot*? This is given below as a Σ type in line with various proposals within the MTT literature to treat adjectives as Σ types (Ranta 1994; Chatzikyriakidis and Luo 2013; Luo 2011) among others:

(67) $Enormous\ Idiot = \Sigma h : Idiot.\ enormous(Idiocy, Human, (\pi_1(h), q), \pi_1(\pi_2(\pi_1(h))))$
where $q\ :\ \pi_1(h) > STND_{id}$.

If you take the first two arguments to be implicit (a form of syntactic sugar), this is rewritten as:

(68) $Enormous\ Idiot = \Sigma h : Idiot.\ enormous((\pi_1(h), q), \pi_1(\pi_2(\pi_1(h))))$
where $q\ :\ \pi_1(h) > STND_{id}$.

Now, even though we concur with Constantinescu as regards the idea of (kinds of) tropes, we do not agree that no degrees are involved in these cases. On the other hand, as Solt (2009) notes in a rather interesting discussion, assuming that nouns and adjectives pattern together as regards degree structure, then the fact that there is no coherent class of adnominal modifiers (in contrast to adjectival modifiers, e.g. too/very etc.) is a puzzling fact. According to what we have proposed in this chapter, this fact can at least be partially explained. This is because, even though nouns like *idiot* involve a degree parameter, they are however not predicates but rather Σ types. This is a major difference as compared to gradable adjectives and can be used as the basis for distinguishing their very different distributional properties. The exact details of this idea are however left for future research.

Another issue discussed in the literature on gradable nouns and their modification by adjectives, is the so-called bigness generalization (Morzycki 2009). This generalization specifies that low degree interpretations by size adjectives are not possible, i.e. cases like *small idiot*. This is, we believe, a non-existing generalization. Cross-linguistically one can find violations of this generalization. In Greek one can have *megalos anthropos* 'a big human' and *mikros anthropos* 'small human' to mean someone who shows a great degree of humanness (or achievement as a human) and low degree of humanness (or achievement) respectively. So, if this generalization is not there, why cases like *small idiot* are deviant? We believe that Constantinescu is right in pointing out that being an idiot involves idiocy to a relatively big extent. *Small* on the other hand, specifies that the degree should hold to a small degree, thus leading to a contradiction (basically one would end up with contradictory statements like $i > STND_{id} \wedge i < STND_{id}$). We believe that this account is on the correct track. It is surely simpler as it treats adjectives like *small* and *big* in the same sense by positing that the infelicitousness is actually due to the lexical semantics of nouns like *idiot.*

The idea that CNs may involve index parameters has been pursued with time parameters as well in Chatzikyriakidis and Luo (2014). There, it was proposed that a CN like *president* is interpreted as a family of types indexed by $t : Time$:

(69) $president(t) : \text{CN}$.

The question that quickly comes to mind is what is the relation between a regular CN type like *Human* and its indexed counterpart, e.g. $Human(n)$. One idea we can pursue is to consider that the type $Human(n)$ be actually a subtype of *Human* of all human beings, in effect:

(70) $Human(n) <_c Human$,

where is a natural number argument representing human heights. In general, one can consider a coercion parametrized by the elements of the universes CN and *Degree*: for every type $A : \text{CN}$ and any $D : Degree$, one can consider a parametrized coercion from the indexed CNs $A_D(d)$:

(71) $A_D(d) <_{c(A,D)} A$.

In the particular case (70), A is *Human*, D is *Height* and $A_D(d)$ is $Human(n)$.

Something like the above has to be further restrained given that it will otherwise predict that every CN induces a coercion between its parametrized versions and itself. Thus, it will predict nouns with abstract parameters and abstract nouns to be compatible with height indexed versions and so on. In order to deal with this, one might assume that every type A specifies its range of possible refinements. For example *Human* will specify that refinements $Human_D$ with D: *Degree* will be possible for the types $Height, Weight, Size, Width$: *Degree* (the list of types is not exhaustive here). One can have specific sub-universes of *Degree* containing the types needed for each case. Then, parameterisations are considered over such sub-universes.

Also, potentially, one may need to further allow more than one parameter to be present in nouns. For example, this will be needed in cases like the following:

(72) The dead rapper was an enormous idiot

In the above example, we need to capture the fact that the dead rapper was an idiot to a great extent, but also the fact that this was before he was dead! In this sense, we need both the *i* grade as well as a time parameter for *idiot*. Such a flexible treatment is available but we will not get into details here.

6 Concluding Remarks

In this paper, after summarising and discussing several different approaches in the interpretation of common nouns in type-theoretical semantics, we have proposed a solution to the issue of interpreting negated sentences and conditionals in MTT-semantics. The account was checked using the proof assistant Coq and it was shown that it produces the correct results as regards to its inferential properties based on the semantic treatment. We have also studied the use of indexed types in dealing with advanced linguistic issues like gradability and temporal sensitivity, showing the potential of the CNs-as-types approach from this particular respect.

Acknowledgements Thanks go to Koji Mineshima for very helpful email communications with the second author, often with thought provoking questions and examples.

A. Coq: A Gentle Introduction

Coq is an interactive theorem prover, i.e. a proof-assistant, that implements the Calculus of Inductive Constructions (pCIC, see The Coq Team 2007), in effect an MTT.[14] The basic idea behind Coq is as follows: it helps one to see whether propositions

[14]The calculus pCIC is very similar to UTT (Luo 1994), the MTT we have used in MTT-semantics. This is the case especially after the universe Set became predicative in the version 8 of Coq in 2004 (The Coq Team 2007).

based on statements previously pre-defined or user defined (definitions, parameters, variables) can be proven or not. To give a very short example on how Coq operates, imagine that we want to prove the following propositional tautology:

(73) $((P \vee Q) \wedge (P \rightarrow R) \wedge (Q \rightarrow R)) \rightarrow R$

Given Coq's typed nature we have to introduce the variables P, Q, R as being of type *Prop* ($P, Q, R : Prop$). Using the command *Theorem*, we put Coq into proof mode:

(74) *Theorem A* : $((P \vee Q) \wedge (P \rightarrow R) \wedge (Q \rightarrow R)) \rightarrow R$

```
Theorem A:((P\/Q)/\(P -> R)/\(Q->R))->R.
1 subgoal
============================
(P\/Q)/\(P -> R)/\(Q -> R)->R
```

The idea from this point onwards is to guide the prover into the completion of the proof using proof tactics that are either already predefined or can be defined by the user using the tactical language *Ltac*. For the case interested, we first introduce the antecedent as an assumption using *intro*:

```
A < intro.
1 subgoal
H :(P \/ Q)/\(P -> R)/\(Q -> R)
============================
R
```

We split the hypothesis into individual hypotheses using *destruct*

```
destruct H0.
1 subgoal
H : P \/ Q
H0 : P -> R
H1 : Q -> R
============================
R
```

Now, we can apply the elimination rule for disjunction (*elim*) which will basically result in two subgoals:

```
elim H.
2 subgoals
H : P \/ Q
H0 : P -> R
H1 : Q -> R
============================
```

```
P -> R
subgoal 2 is:
Q -> R
```

The two subgoals are already hypotheses. We can use the *assumption* tactic that matches the goal with an identical hypothesis and the proof is completed:

```
assumption. assumption.
1 subgoal
H : P \/ Q
H0 : P -> R
H1 : Q -> R
============================
Q -> R
Proof completed.
```

Now, what we need to do in order to reason about NL is to implement our theoretical accounts we have provided in Coq. For example, one must introduce the universe CN and further declare the types of the universe. For the moment, since universe construction is not an option in Coq, we define CN to be Coq's predefined *Set* universe. We show these, plus the introduction of individuals *John* and *Mary* and quantifier *some*. We also introduce the subtyping relations $Man < Human < Animal < Object$:

```
Definition CN:=Set.
Parameters Man Human Animal Object:CN.
Parameter John:Man.
Parameter Mary:Woman.
Axiom mh: Man->Human. Coercion mh: Man>->Human.
Axiom ha: Human->Animal. Coercion ha: Human>->Animal.
Axiom ao: Animal->Object. Coercion ao: Animal>->Object.
Definition some:= fun A:CN, fun P:A->Prop=> exists x:A, P(x).
Parameter: Animal->Prop.
```

With the above definitions one can perform reasoning tasks, for example one might want to check whether *some man walks* follows from *John walks.* We formulate this as a theorem named *EX*:

```
Theorem EX:  (walk) John-> some Man (walk).
```

The next step here is to unfold the definition for the quantifier. This is done using the *cbv* command, which basically performs any reduction possible. Then, we move the antecedent as a hypothesis using intro:

```
EX < intro.
1 subgoal
H : walk John
============================
exists x : Man, walk x
```

At this point, we use the *exists* tactic to substitute *John* for *x*. Using *assumption* the theorem is proven.

B. Coq Data and Assumptions

The following gives all of the data and assumptions made in Coq in order to deal with all the examples discussed in this paper.

```
Definition CN:=Set.
Parameters Man Human Animal Object Logician Linguist Table:CN.
Parameter John:Man.
Axiom mh: Man->Human.Coercion mh: Man>->Human.
Axiom ha: Human->Animal.Coercion ha: Human>->Animal.
Axiom ao: Animal->Object.Coercion ao: Animal>->Object.
Definition some:= fun A:CN,fun P:A->Prop=>exists x:A, P(x).
Definition pr := fun(A : CN)(a : A)=>True.
Set Implicit Arguments.
Parameter NOT: forall A:CN,(A->Prop)->(Object->Prop).
Parameter Linguist Logician:CN.
Axiom lo:Linguist->Human.Coercion lo:Linguist>->Human.
Axiom lo1:Logician->Human.Coercion lo1:Logician>->Human.
Unset Implicit Arguments.
Definition PR_o:= fun A:CN,fun x:Object=>not(NOT(pr A)x).
Definition PR_m:= fun A:CN,fun x:Man=>not(NOT(pr A)x).
Definition PR_h:= fun A:CN,fun x:Human=>not(NOT(pr A)x).
Definition PR_l:= fun A:CN,fun x:Linguist=>not(NOT(pr A)x).
Definition PR_lo:= fun A:CN,fun x:Logician=>not(NOT(pr A)x).
Parameter Table: CN.
Axiom to: Table->Object. Coercion to: Table>->Object.

Section NEG.
Variable Classical_Negation:forall P:Prop,not(not P)<->P.
Variable NOT_2hm:forall(a:Human),NOT(pr Human)a->
NOT(pr Man) a.
Variable NOT_2ha:forall(a:Animal),NOT(pr Animal)a->
NOT(pr Human) a.
Variable NOT_2ao:forall(a:Object),NOT (pr Object)a->
```

```
NOT(pr Animal) a.
Variable NOT_2hl: forall (a : Human),  NOT  (pr Human) a ->
NOT  (pr Logician) a.
Variable NOT_1m:forall x:Man,NOT(pr Man)x->not(pr Man x).
Variable NOT_1h:forall x:Human,NOT(pr Human)x->not(pr
 Human x).
Variable NOT_1a:forall x:Animal,NOT(pr Animal)x->
not(pr Animal x).
Variable NOT_1l:forall x:Linguist,NOT(pr Linguist)x->
not(pr Linguist x).
Variable NOT_1lo:forall x:Logician,NOT(pr Logician)x->
not(pr Logician x).
Parameter Red: Object->Prop.
Record redtable:CN:=mkRedtable{rt:>Table;_ :Red rt}.
```

We further assume that all the types in the universe D are totally ordered and dense; i.e., they respect the following axioms:

$$\begin{aligned}
(\textit{reflexivity}) \quad & \forall A : D.\, \forall d : A.\, d \leq d \\
(\textit{anti-symmetry}) \quad & \forall A : D.\, \forall d, d_1 : A.\, d \leq d_1 \wedge d_1 \leq d \rightarrow d = d_1 \\
(\textit{transitivity}) \quad & \forall A : D.\, \forall d, d_1, d_2 : A.\, d \leq d_1 \wedge d_1 \leq d_2 \rightarrow d \leq d_2 \\
(\textit{density}) \quad & \forall A : D.\, \forall d, d_1 : A.\, d \leq d_1 \rightarrow \exists d_2 : A. d \leq d_2 \leq d_1
\end{aligned}$$

Remark There is a pending question w.r.t the formal properties of grades. One of the common assumptions in the literature is that grades are dense, total, transitive, anti-symmetric and reflexive. The question is whether these assumptions are enough or different weaking might be needed depending on the grade in each case. What is rather difficult to tackle is the formal properties of abstract grades like *idiocy*.

References

Asher, N. (2012). *Lexical meaning in context: A web of words*. Cambridge: Cambridge University Press.

Asher, N., Abrusan, M., & van de Cruys, T. (2016). Types, meanings and co-composition in lexical semantics. In this volume.

Asher, N., & Luo, Z. (2012). In *Formalisation of coercions in lexical semantics, Sinn und Bedeutung 17, Paris*.

Beeson, M. J. (1985). *Foundations of constructive mathematics*. Heidelberg: Springer.

Bekki, D. (2014). Representing anaphora with dependent types. *Logical aspects of computational linguistics* (Vol. 8535), LNCS Heidelberg: Springer.

Bekki, D., & Mineshima, K. (2016). Context-passing and underspecification in dependent type semantics. In this volume.

Chatzikyriakidis, S., & Luo, Z. (2013). Adjectives in a modern type-theoretical setting. In G. Morrill & J. M. Nederhof (Eds.), *Proceedings of Formal Grammar 2013. LNCS 8036* (pp. 159–174).

Chatzikyriakidis, S., & Luo, Z. (2014). Natural language reasoning in Coq. *Journal of Logic, Language and Information*, *23*, 441.
Chatzikyriakidis, S., & Luo, Z. (2015). Individuation criteria, dot-types and copredication: A view from modern type theories. *Proceedings of the 14th International Conference on Mathematics of Language, Chicago*, (Vol. 23).
Chatzikyriakidis, S., & Luo, Z. (2016). Adjectival and adverbial modification in modern type theories. Manuscript.
Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, *5*(1), 56–68.
Constantinescu, C. (2013). In *Big eaters and real idiots: evidence for adnominal degree modification? Sinn und Bedeutung*, (Vol. 17, pp. 183–200).
Curry, H. B., & Feys, R. (1958). *Combinatory logic* (Vol. 1). Amsterdam: North Holland.
Davidson, D. (1967). The logical form of action sentences. In S. Rothstein (Ed.), *The logic of decision and action*. Pittsburgh: University of Pittsburgh Press.
De Groote, P., & Maarek, S. (2007). Type-theoretic extensions of abstract categorial grammars. In *Proceedings of Workshop on New Directions in Type-theoretic Grammars*.
Geach, P. (1962). *Reference and generality*. New York: Cornell University Press.
Howard, W. A. (1980). The formulae-as-types notion of construction. In J. Hindley & J. Seldin (Eds.), *To H. B. Curry: Essays on combinatory logic*. Cambridge: Academic Press.
Krahmer, E., & Piwek, P. (1999). Presupposition projection as proof construction. In H. Bunt & R. Muskens (Eds.), *Computing meaning*, Studies in Linguistics and Philosophy Netherlands: Springer.
Luo, Z. (1994). *Computation and reasoning: A type theory for computer science*. Oxford: Oxford University Press.
Luo, Z. (1999). Coercive subtyping. *Journal of Logic and Computation*, *9*(1), 105–130.
Luo, Z. (2010). Type-theoretical semantics with coercive subtyping. In *Semantics and Linguistic Theory 20 (SALT20), Vancouver*.
Luo, Z. (2011). Contextual analysis of word meanings in type-theoretical semantics. *Logical aspects of computational linguistics (LACL'2011)* (Vol. 6736), LNAI New York: Springer.
Luo, Z. (2012). Common nouns as types. In D. Bechet & A. Dikovsky (Eds.), *Logical aspects of computational linguistics (LACL'2012)* (Vol. 7351), LNCS Heidelberg: Springer.
Luo, Z. (2012a). Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophy*, *35*(6), 491–513.
Luo, Z., & Soloviev, S. (2016) Dependent event types. Manuscript.
Luo, Z., Soloviev, S., & Xue, T. (2012). Coercive subtyping: Theory and implementation. *Information and Computation*, *223*, 18–42.
Martin-Löf, P. (1984). *Intuitionistic type theory*. Italy: Bibliopolis.
Moltmann, F. (2003). Nominalizing quantifiers. *Journal of Philosophical Logic*, *32*(5), 445–481.
Moltmann, F. (2007). Events, tropes, and truthmaking. *Philosophical Studies*, *134*(3), 363–403.
Montague, R. (1973). The proper treatment of quantification in ordinary English. In J. Hintikka, J. Moravcsik, & P. Suppes (Eds.), *Approaches to natural languages*. Netherlands: Springer.
Montague, R. (1974). In R. Thomason (Ed.), *Formal philosophy*. Connecticut: Yale University Press. Collected papers.
Morzycki, M. (2009). Degree modification of gradable nouns: Size adjectives and adnominal degree morphemes. *Natural Language Semantics*, *17*(2), 175–203.
Nordström, B., Petersson, K., & Smith, J. (1990). *Programming in Martin-Löf's type theory: An Introduction*. Oxford: Oxford University Press.
Parsons, T. (1990). *Events in the semantics of english*. Cambridge: MIT Press.
Quine, W. V. (1986). *Philosophy of logic* (2nd ed.). Cambridge: Harvard University Press.
Ranta, A. (1994). *Type-theoretical grammar*. Oxford: Oxford University Press.
Retoré, C. (2013). The montagovian generative lexicon Tyn: A type theoretical framework for natural language semantics. In R. Matthes & A. Schubert (Eds.), *Proceedings of TYPES2013*.

Solt, S. (2009). The semantics of adjectives of quantity. *Ph.D. Thesis, The City University of New York*.

Tanaka, R., Mineshima, K., & Bekki, D. (2015). Factivity and presupposition in dependent type semantics. In *Type Theories and Lexical Semantics Workshop*.

The Coq Team. (2007). *The Coq Proof Assistant Reference Manual (Version 8.1), INRIA*.

Tonhauser, J. (2002). A dynamic semantic account of the temporal interpretation of noun phrases. *Semantics and linguistic theory* (pp. 286–305). New York: Cornell University.