Journal of Logic, Language and Information manuscript No.
(will be inserted by the editor)

Propositional Forms of Judgemental Interpretations

Tao Xue · Zhaohui Luo · Stergios Chatzikyriakidis

Abstract In formal semantics based on modern type theories, some sentences may be interpreted as judgements and some as logical propositions. When interpreting composite sentences, one may want to turn a judgemental interpretation or an ill-typed semantic interpretation into a proposition in order to obtain an intended semantics. For instance, an incorrect judgement $a : A$ may be turned into its propositional form $\textsc{is}(A, a)$ and an ill-typed application $p(a)$ into $\textsc{do}(p, a)$, so that the propositional forms can take part in logical compositions that interpret composite sentences, especially those that involve negations and conditionals.

In this paper, we propose an operator NOT that facilitates such a transformation. Introducing NOT axiomatically, with five axiomatic laws to govern its behaviour, we shall use it to define IS and DO and give examples to illustrate its use in semantic interpretation. The introduction of NOT into type theories is logically consistent – this is justified by showing that NOT can be defined by means of the heterogeneous equality JMeq so that all of the axiomatic laws for NOT become provable. Therefore, since the extension with JMeq preserves logical consistency, so does the extension with NOT. We shall also study conditions under which IS and DO operators can be used safely without the risk of over-generation.

Keywords Formal Semantics · Type Theory · Judgemental Interpretation

Tao Xue
Huazhong University of Science and Technology. Wuhan, 430074, China.
E-mail: xuetao@hust.edu.cn

Zhaohui Luo
Royal Holloway, University of London, Surrey TW20 0EX, U.K.
E-mail: zhaohui@cs.rhul.ac.uk

Stergios Chatzikyriakidis
CLASP, University of Gothenburg, Dicksonsgatan 4, 41256 Göteborg, Sweden.
E-mail: stergios.chatzikyriakidis@gu.se

1 Introduction

In recent years, the virtue of rich type structures in constructing semantic interpretations has been recognised by many researchers and various rich type systems have been successfully employed in formal semantics including, for example, (Luo, 2012b; Retoré, 2014; Bekki, 2014). In some of these approaches, common nouns are interpreted as types, rather than predicates as in the traditional Montagovian approach. For example, the sentence (1) is interpreted as (2), where $Teacher$ that interprets the CN 'teacher' is a type, $talk$ that interprets the verb 'talk' is a predicate whose domain is type $Human$ of humans, and $Teacher$ is a subtype of $Human$ (and hence $talk(x)$ is well-typed when $x : Teacher$). The interpretation (2) is different from the traditional Montagovian semantics (3) where $teacher$, $talk_1 : \mathbf{e} \to \mathbf{t}$ are predicates with domain $\mathbf{e}$ of all entities.[1]

(1) Every teacher talks.

(2) $\forall x : Teacher.\ talk(x)$

(3) $\forall x : \mathbf{e}.\ teacher(x) \wedge talk_1(x)$

This CNs-as-types paradigm (Mönnich, 1985; Sundholm, 1986; Ranta, 1994; Luo, 2012a) has several merits including, for instance, desirable treatment of some linguistic features such as copredication that have been found difficult to be dealt with in traditional settings. It is adopted and further developed in formal semantics based on Modern Type Theories (MTT-semantics for short) (Luo, 2012b; Chatzikyriakidis and Luo, 2020), where the rich type structure has been used effectively and shown to provide powerful means in semantic constructions.[2]

The CNs-as-types paradigm has led to the use of type-theoretic *judgements*, as well as logical propositions, as semantic interpretations of NL sentences. In type theory, a most basic form of judgement is that of membership judgements: $a : A$ is a judgement stating that a is of type A. As an example, (4) can be interpreted as the judgement (5), where $Student$ is a type. Note that this is different from the Montagovian interpretation (6) where (4) is interpreted as a logical formula with 'Bob' as an entity $b : \mathbf{e}$ and 'student' as a predicate $student : \mathbf{e} \to \mathbf{t}$.

(4) Bob is a student.

(5) $bob : Student$

(6) $student(b)$

There are several notable advantages when CNs are interpreted as types. A straightforward one is that selectional restriction can naturally be enforced

[1] In the literature on linguistic semantics, the formula in (3) is usually written as $\forall x.\ teacher(x) \wedge talk_1(x)$ without the domain $\mathbf{e}$ because, in most of the cases in the Montagovian setting, quantifications are always over $\mathbf{e}$ of all entities.

[2] For example, the rich type structure in MTTs has been used to interpret a wide range of modifications – see (Chatzikyriakidis and Luo, 2020) (especially its §3.3) for a recent account based on earlier treatments as reported in (Chatzikyriakidis and Luo, 2013, 2017a).

automatically by means of decidable type-checking. For example, consider the following sentence (9), which contains a category error and, in normal circumstances, most people would regard as meaningless. One may consider that meaningfulness is captured semantically as being well-typed, while meaninglessness as being ill-typed (Asher, 2012). For example, for $talk : Human \to Prop$, the semantic interpretation (10) of the sentence (9) would not be well-typed, since table t is not a human. As another example, the interpretation (12) of (11) is also ill-typed because $eat : Animal \to Food \to Prop$ requires its first argument to be an animal, while the table t_0 is not and, therefore, there is a semantic type mismatch and the function application is impossible.[3]

(9) (#) Tables talk.

(10) (#) $\forall t : Table.\ talk(t)$

(11) (#) The table ate the egg.

(12) (#) $eat(t_0, e_0)$

In a type theory, it is decidable whether an expression such as (10) or (12) is well-typed (i.e., it can be checked by the computer) and, therefore, linking meaningfulness with well-typedness gives a nice solution to selectional restriction and, therefore, modern type theories with their rich type structures provide a powerful framework in this respect. It is necessary to note that, to adopt the CNs-as-types approach, 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 (2) of (1), $Teacher$ is a subtype of $Human$ and this subtyping relationship makes the application $talk(x)$ in (2) well-typed. Fortunately, there is a subtyping mechanism called coercive subtyping (Luo, 1999; Luo et al., 2013; Xue, 2013) that is suitable for modern type theories and makes the CNs-as-types approach viable (Luo, 2010, 2012b).

It is important to emphasise the fact that a judgement in a type theory is not a logical proposition. When interpreting some sentences as judgements, one would want to turn a judgemental interpretation into a proposition so that composite sentences can be interpreted properly by logical compositions, especially when it occurs negatively in negative sentences or conditionals. For instance, the conjunction connective 'and' is used in (13) and one might use (14) to interpret (13). But (14) is not a well-formed formula (or, equivalently, ill-typed as a proposition) because the judgement $bob : Student$ is not a logical proposition and cannot be used as a conjunct.

(13) Bob is a student and he is happy.

[3] Note that this is different from Montague semantics: the Montagovian interpretations (7) and (8) are well-formed formulas of type $\mathbf{t}$ because $table$, $talk_1$ and eat_1 are all predicates with domain $\mathbf{e}$ and t_1 and e_1 are entities of type $\mathbf{e}$. Put in another way, they are legal formulas, although their intended interpretations should usually be false.

(7) $\forall x : \mathbf{e}.\ table(x) \wedge talk_1(x)$

(8) $eat_1(t_1, e_1)$

(14)(#) $(bob : Student) \wedge happy(bob)$

As another example, although (9) is usually regarded as meaningless, its negation (15) is meaningful. However, (16), the intuitive interpretation of (15), is ill-typed, just like (10), because table t is not a human.

(15)Tables do not talk.
(16)(#) $\forall t : Table.\ \neg talk(t)$

In order to give proper semantic interpretations to such sentences as (13) and (15), one would need to turn those interpretations like (14) and (16) that involve judgemental or ill-typed interpretations into acceptable propositional forms so that they can occur meaningfully in logical compositions or negative contexts. In this paper, we shall introduce such propositional forms, show that their addition to the underlying type theory preserves the good properties such as logical consistency, and study how to prevent potential over-generation because of such extra propositions. For example, a propositional form corresponding to a membership judgement $a : A$ may be the proposition $\textsc{is}(A, a)$ which intuitively expresses that a is of type A. If so, the conjunctive sentence (13) can be interpreted as (17). Similarly, the ill-typed $talk(t)$ would correspond to a proposition $\textsc{do}(talk, t)$ and the negative sentence (15) can be interpreted as (18).

(17)$\textsc{is}(Student, bob) \wedge happy(bob)$
(18)$\forall t : Table.\ \neg\textsc{do}(talk, t)$

We shall propose an operator NOT for constructing propositional forms of judgemental or ill-typed interpretations. In particular, both operators IS and DO can be defined by means of NOT to deal with negative sentences and conditionals in satisfactory ways. NOT and its associated axiomatic laws can be justified by means of the heterogeneous equality for type theories (McBride, 2002) and, in particular, it is shown that the extension by NOT is logically consistent. Furthermore, we'll also study how to avoid over-generation by means of the operators IS and DO. This shows that our proposal offers a satisfactory solution so that the semantics of negative sentences and conditionals can be properly considered in the extended type theory. Logical expressions are provided together with Coq codes in this work.

The rest of the paper is organised as follows. In the following subsection §1.1, we shall give a summary of the related work, especially about formal semantics based on type theories. In §2, we informally discuss the issue of judgemental and ill-typed interpretations and the idea of introducing their propositional forms. Formally proposing the solution, we introduce NOT in §3, where its axiomatic rules are studied and NL examples are given to illustrate how the operators IS and DO, defined by means of NOT, can be used in semantic interpretations. §4 uses the heterogeneous equality JMeq to justify the introduction of NOT (and hence IS and DO) and, in §5, we study how to avoid over-generation when IS and DO are used for semantic interpretations.

1.1 Related Work: Formal Semantics Based on Type Theories

The application of type theory to formal semantics has been initiated by Montague's pioneering work (Montague, 1974). Montague employed Church's simple type theory STT (Church, 1940) (and Henkin's model theory of STT (Henkin, 1950)) as the foundational language for formal semantics. This has since become the dominant approach in this field. An enormous amount of work based on Montague's original system IL, or its variations, have been produced since then. One such work is Gallin's study of TY_2, a reformulation of Church's STT with an extra base type (concerning intensions), and his translation of Montague's IL into TY_2 which establishes a solid foundation for Montague's semantics (Gallin, 1975).There are many other related research on NL semantics in type theory including, for instance, research relating the dynamic frameworks such as dynamic predicate logic (Groenendijk and Stokhof, 1991) and Discourse Representation Theory (Kamp and Reyle, 1993) to simple type theory (Groenendijk and Stokhof, 1990; Muskens, 1996).

In the last two decades or so, researchers have worked on employing rich type theories for formal semantics. In his seminal work, Ranta (Ranta, 1994) proposed to study various aspects of NL semantics using Martin-Löf's intensional type theory (a typical Modern Type Theory[4]). Many other researchers have also recognised the potential advantages of rich type structures for formal semantics including, for example, (Mönnich, 1985; Sundholm, 1989; Luo and Callaghan, 1998; Boldini, 2000; Cooper, 2005; Dapoigny and Barlatier, 2009; Bekki, 2014; Retoré, 2014). More recently, there has been a move to develop Modern Type Theories as a full-blown setting for formal semantics (MTT-semantics – see (Luo, 2012b; Chatzikyriakidis and Luo, 2020) among other papers). It has been argued (Luo, 2014, 2019a) that MTT-semantics has the advantage of being both proof-theoretic and model-theoretic: being proof-theoretic as discussed in (Kahle and Schroeder-Heister, 2006), it provides direct support of practical applications of computer-assisted reasoning with proof assistants; and being model-theoretic, their rich type structures deliver a wide semantic coverage of various linguistic features.[5]

The fact that MTTs are proof systems with proof-theoretic semantics has a significant and practical consequence in NL reasoning based on MTT-semantics. In particular, this makes it possible for MTTs to be implemented

[4] By Modern Type Theories (MTTs), we refer to the family of formal systems such as Martin-Löf's intensional type theory (MLTT) (Martin-Löf, 1975; Nordström et al., 1990), the Calculus of Inductive Constructions (pCIC) (Coq, 2010) and the Unifying Theory of dependent Types (UTT) (Luo, 1994). Martin-Löf's type theory MLTT can be employed as an adequate foundational language for MTT-semantics when it is extended with h-logic – see (Luo, 2019b) for more details.

[5] Another important property of MTTs is *decidability*, which is necessary for an internal logical system based on the propositions-as-types principle (Curry and Feys, 1958; Howard, 1980). Note that, given a proof candidate a and a proposition A, it should be decidable whether $a : A$ is correct (i.e., type checking should be decidable), because we are dealing with a finite proof system and, in particular, both a and A are finite. Please also note that this is different from *provability* of a proposition, which is undecidable because, without a proof candidate, one has to try to find out whether a proof exists.

in proof assistants such as Agda (Agda 2008, 2008), Coq (Coq, 2010) and Lego/Plastic (Luo and Pollack, 1992; Callaghan and Luo, 2001) – computer-assisted reasoning systems that computer scientists have developed and successfully used for the formalisation of mathematics and verification of computer programs. Therefore, MTT-semantics can be directly implemented in proof assistants that implement MTTs: for example, the MTT-semantics in type theory UTT has been implemented in Coq (Chatzikyriakidis and Luo, 2014; Luo, 2011) and Plastic (Callaghan and Luo, 2001; Xue and Luo, 2012) and used for NL reasoning (Chatzikyriakidis and Luo, 2016). In this paper, we shall use the Coq system in checking the proposed semantics etc. The interested reader can also find a short introduction to Coq in Appendix A.

2 Judgemental Interpretations and Their Propositional Forms

In Modern Type Theories (MTTs, see Footnote 4), judgements and their derivability are basic concepts: for example, a judgement $a : A$ asserts that object a is of type A. In MTT-semantics, we allow *judgemental interpretations*: sentences can be interpreted by means of judgements as well as logical propositions. For example, (19) can be given the judgemental interpretation (20), while (21) can be interpreted as the logical proposition $talk(j)$ of type $Prop$.

(19) John is a human.

(20) $j : Human$

(21) John talks.

(22) $talk(j)$

Some judgements are *derivable*, representing correct assertions. For instance, the above interpretation (20) is derivable with $j : Man$ and $Man \leq Human$. Some judgements are *non-derivable* and, as judgemental interpretations, they indicate that the interpreted NL sentences are meaningless in usual situations. For example, (23) is usually considered meaningless (unless in some fictional or special setting) and its semantic interpretation (24) is non-derivable.

(23) (#) John is a table.

(24) $j : Table$

Although it is straightforward to see that (20) is derivable, the non-derivability of (24) is not: with $j : Man$, it assumes that the types Man and $Table$ have no common objects (formally, they are *disjoint*, i.e., they do not have non-empty common subtypes – see §5).

Besides derivable and non-derivable judgements, some judgements are only *potentially derivable* because one does not have the full information to make the determination, or put in another way, whether it is derivable depends on the environment to be fixed. For example, for the sentence (25), its judgemental interpretation (26) is potentially derivable since, intuitively, John may be a

male student or may not be a student at all: (26) is derivable in the former situation but non-derivable in the latter.[6]

(25) John is a student.

(26) $j : \mathit{Student}$

Since judgements are not propositions, they cannot be directly used in semantic compositions with other propositional interpretations. For instance, consider the sentence (27) with the conjunction connective 'and'. The expression (28), that one might use to interpret (27), is not a well-formed formula (or, equivalently, ill-typed as a proposition) because the judgement $j : \mathit{Human}$ is not a logical proposition and cannot be used as a conjunct.

(27) John is a human and he is happy.

(28) (#) $(j : \mathit{Human}) \wedge \mathit{happy}(j)$

How can a judgemental interpretation be turned into a proposition in order to compose with other propositional interpretations? For instance, can we find a proposition P that corresponds to the judgement $j : \mathit{Human}$ so that (27) can be interpreted as $P \wedge \mathit{happy}(j)$? The answer to this question is positive – let's consider the different categories of judgements in turn, starting with derivable judgements.

When derivable, a judgemental interpretation has a simple corresponding proposition: if $a : A$ is derivable, its corresponding proposition is $p_A(a)$, where $p_A : A \to \mathit{Prop}$ is defined as in (29), the constant predicate of type $A \to \mathit{Prop}$ that is always true for any object of type A. For instance, the judgemental interpretation $j : \mathit{Human}$ corresponds to $p_{\mathit{Human}}(j)$ and, therefore, the composite sentence (27) can be interpreted as (30). In other words, we obtain (30) from (28) by using the proposition $p_{\mathit{Human}}(j)$ to replace the judgement $(j : \mathit{Human})$.

(29) $p_A(x) = \mathbf{true}$, for any $x : A$.

(30) $p_{\mathit{Human}}(j) \wedge \mathit{happy}(j)$

Note that, although p_A is a simple constant predicate, it is non-trivial in that the well-typedness of $p_A(a)$ *presupposes* that a is of type A. In other words, the truth of $p_A(a)$ is (meta-theoretically) equivalent to the correctness (derivability) of the judgement $a : A$. For example, $p_{\mathit{Human}}(j)$ is true if, and only if, $j : \mathit{Human}$ is derivable. That is why we can use $p_A(a)$ as the propositional form of a derivable judgement $a : A$:

$$a : A \text{ (derivable)} \leftrightsquigarrow p_A(a)$$

A non-derivable judgemental interpretation indicates that the interpreted sentence is meaningless, as exemplified by (23-24). However, the negation of

[6] Formally, (26) is derivable if, for example, $j : \Sigma x{:}\mathit{Student}.\mathit{male}(x)$, which is a subtype of both $\mathit{Student}$ and Man with $\mathit{Student} \leq \mathit{Human}$ and $\mathit{Man} = \Sigma x{:}\mathit{Human}.\mathit{male}(x)$. Under many other environments, the judgement $j : \mathit{Student}$ is non-derivable.

such a sentence is usually meaningful, as shown in (31), which negates (23). If such a sentence is used as a premise of a conditional, the whole sentence is meaningful as well: see (32) for an example.

(31) John is not a table.

(32) If John were a table, Mary would be happy.

How to interpret such sentences? [7] For instance, consider how to interpret the negative sentence (31). First, we cannot use the logical connective $\neg$ to negate the judgemental interpretation $j : Table$ – the resulting expression (33) is illegitimate because $j : Table$ is not a proposition. Secondly, we cannot simply negate the judgement $j : Table$ meta-theoretically: what it would result in, (34), is only a meta-level statement. Thirdly, we cannot use the predicate p_{Table}, either: if we did so as in (35), we would have assumed the well-typedness of $p_{Table}(j)$, which is equivalent to the derivability of $j : Table$.

(33) (#) $\neg(j : Table)$

(34) (#) $(j : Table)$ is not derivable.

(35) (#) $\neg p_{Table}(j)$

In other words, in order to interpret composite sentences like (31) and (32), we'd need to have a propositional form of the judgement $j : Table$. Fortunately, such a proposition exists: we have a proposition $\textsc{is}(Table, j)$ which intuitively means that 'j is of type $Table$'. So, (31) and (32) can now be interpreted as (36) and (37), respectively.

(36) $\neg\textsc{is}(Table, j)$

(37) $\textsc{is}(Table, j) \Rightarrow happy(m)$

In general, for any type T and any object $a : A$, if $a : T$ is non-derivable, its propositional form is $\textsc{is}(T, a)$ and can be used in a negative context adequately in a semantic interpretation (see §5):

$$a : T \text{ (non-derivable and occurring negatively)} \leftrightsquigarrow \textsc{is}(T, a)$$

When a judgemental interpretation is only potentially derivable, the IS-proposition may tentatively be used as its propositional form. For instance, the propositional form of $j : Student$ of the sentence (25) may be $\textsc{is}(Student, j)$ and the composite sentences (38-40) may be interpreted as (41-43), respectively.

(38) John is a student and he is happy.

[7] The second author is grateful to several researchers who have discussed with him about how to interpret negative sentences in MTT-semantics, including Glyn Morrill (during ESSLLI 2011), Nicholas Asher (in an email communication about a paper in LACL 2014) and Koji Mineshima (during ESSLLI 2014 and subsequent communications with the second author when he was writing (Chatzikyriakidis and Luo, 2017b), where a preliminary NOT-operator was studied – see Footnote 8 in §3.2.)

(39) John is not a student.

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

(41) $\textsc{is}(\mathit{Student}, j) \wedge \mathit{happy}(j)$

(42) $\neg\textsc{is}(\mathit{Student}, j)$

(43) $\textsc{is}(\mathit{Student}, j) \Rightarrow \mathit{happy}(j)$

But note that such interpretations with the IS-operator are only *tentative* because, whence the environment is fully fixed, some of them might be inadequate. If $j : \mathit{Student}$ is derivable, $\textsc{is}(\mathit{Student}, j)$ is logically equivalent to $p_{\mathit{Student}}(j)$ (by an axiomatic law of NOT – see §3). If $j : \mathit{Student}$ is non-derivable, the interpretations (42) and (43) are OK, but (41) is inadequate (or 'over-generates', as some would put it) because sentence (38) is usually regarded as meaningless.

Besides judgemental interpretations, some NL sentences may also be difficult to be interpreted straightforwardly because their logical semantics would seem to involve ill-typed applications (or potentially ill-typed expressions). Such situations often occur for negative sentences or conditionals. For instance, consider (15), repeated below as (44), which is a negative sentence about a general verb 'talk'. First note that, unlike the positive sentence 'Tables talk' which is regarded as meaningless in usual situations, its negation (44) is meaningful. However, to interpret it, one cannot use (45) since it involves $\mathit{talk}(x)$ which is ill-typed (talk's arguments are humans and it cannot be applied to a table). Fortunately, we have a proposition $\textsc{do}(\mathit{talk}, x)$ that intuitively means that 'x talks'. Therefore, the negative sentence (44) can be interpreted as (46). Similarly, the conditional (47) cannot be interpreted as (48), but rather (49).

(44) Tables do not talk.

(45) (#) $\forall x{:}\mathit{Table}.\ \neg\mathit{talk}(x)$

(46) $\forall x{:}\mathit{Table}.\ \neg\textsc{do}(\mathit{talk}, x)$

(47) If a table talked, Mary would be surprised.

(48) (#) $\exists x{:}\mathit{Table}.\ \mathit{talk}(x) \ \Rightarrow\ \mathit{surprised}(m)$

(49) $\exists x{:}\mathit{Table}.\ \textsc{do}(\mathit{talk}, x) \ \Rightarrow\ \mathit{surprised}(m)$

In general, when an ill-typed application $p(a)$ occurs negatively, its corresponding propositional form $\textsc{do}(p, a)$ can be used in semantic interpretations:

$$p(a) \text{ (ill-typed and occurring negatively)} \leftrightsquigarrow \textsc{do}(p, a)$$

Technically, both propositional operators IS and DO can be defined by means of a more generic operator NOT, which has been proposed and studied in (Xue et al., 2018) (and preliminarily in (Chatzikyriakidis and Luo, 2017b)). Adding NOT to a modern type theory amounts to a proper extension and needs be justified to show, for example, that logical consistency is preserved. NOT (and IS and DO) will be studied below: we shall show how they can be introduced, how they may be employed in semantic interpretations, how their introduction to type theory can be justified and how to avoid over-generation in their use.

3 Propositional Forms: A Formal Treatment

As explained above, in MTT-semantic interpretations, one may need to turn a judgemental interpretation or an ill-typed application into a well-typed proposition. This can be illustrated by examples whose interpretations involve the following, where the examples are all from the above §2:

- Derivable judgements – exemplified by (20) that interprets (19).
- Non-derivable judgements (in negative occurrences) – when interpreting sentences like (31-32).
- Ill-typed applications (in negative occurrences) – when interpreting sentences like (44) and (47).

There are propositional forms corresponding to such judgemental or ill-typed interpretations and, in this section, we'll study them in a more formal and precise way. We shall first introduce the true predicate p_A, used to give propositional forms for derivable membership judgements, and then the operator NOT that can be used to define IS and DO that provide propositional forms for non-derivable judgements or ill-typed applications, respectively, when used in negative contexts. The NOT-operator will be introduced axiomatically and examples are given to show how these operators can be used in semantic constructions. In §4, it is shown that the axiomatic laws for NOT can all be justified by means of the heterogeneous equality in type theory (McBride, 2002; McBride and McKinna, 2004) and in §5 we shall study conditions under which IS and DO can be used without resulting in over-generation.

In this section, we shall work in the impredicative type theory UTT (Luo, 1994), one of the modern type theories. For a predicative type theory, one may replace the totality $Prop$ by means of some predicative universe (see, for example, the system MLTT_h in (Luo, 2018)) and then similar results can be obtained.

3.1 Propositional Forms of Derivable Judgements

As described in §2, when a judgemental interpretation of the form $a : A$ is derivable, its propositional form is given as $p_A(a)$ where, for any type A, p_A is the 'true predicate' defined as in (29), repeated as follows: for any type A,

(50) $p_A : A \to Prop$

(51) $p_A(x) = \mathbf{true}$, for any $x : A$.

As exemplified in the above §2, the composite sentence (27) can be interpreted as (30), where $p_{Human}(j)$ acts as the propositional form of $j : Human$. Note that $p_A(a)$ is only equivalent to **true** if a is of type A – that is, the well-typed $p_A(a)$ carries more information than **true**: it presumes that a is of type A. This is why $p_A(a)$ can play the role of being the propositional form of the derivable judgement $a : A$.

3.2 The NOT-Operator and Definitions of IS and DO

Non-derivable judgemental interpretations and ill-typed applicational interpretations may be meaningfully used when they occur negatively – see §2 for examples such as (31)(32) for the former and (44)(47) for the latter. Their propositional forms can be represented by means of the IS and DO operators, as in (36)(37) for the former and (46)(49) for the latter. Both operators IS and DO can be defined by means of a more general operator NOT.[8]

The type of NOT is given in (52): it takes four arguments to form a proposition $\text{NOT}(A, p, B, b)$, where A and B are common nouns in CN, $p : A \to Prop$ is a predicate over A, and b is an object of type B.

(52) $\text{NOT} : \Pi X{:}\text{CN}\ \Pi p : X \to Prop\ \Pi Y{:}\text{CN}\ \Pi y : Y.\ Prop$

Intuitively, $\text{NOT}(A, p, B, b)$ means that 'b does not p' and, in particular, when p is the true predicate p_A as defined above, it means that 'b is not an A'. Abbreviations can be made to define the operators IS and DO: [9]

(53) $\text{IS}_B : \text{CN} \to B \to Prop$, for any $B : \text{CN}$;
(54) $\text{IS}_B(A, b) = \neg\text{NOT}(A, p_A, B, b)$, for any $A, B : \text{CN}$ and $b : B$,

where p_A is the true predicate defined in (51), and

(55) $\text{DO}_{A,B} : (A \to Prop) \to B \to Prop$, for any $A, B : \text{CN}$;
(56) $\text{DO}_{A,B}(p, b) = \neg\text{NOT}(A, p, B, b)$, for $A, B : \text{CN}$, $p : A \to Prop$ and $b : B$.

Sometimes (and usually in examples), we omit the subscripts to write $\text{IS}(A, b)$ and $\text{DO}(p, b)$ for $\text{IS}_B(A, b)$ and $\text{DO}_{A,B}(p, b)$, respectively. Intuitively, $\text{IS}(A, b)$ and $\text{DO}(p, b)$ mean that 'b is an A' and 'b does p', respectively. Examples of how to use the operators IS and DO include those mentioned above in §2 – to repeat: (36) for (31), (37) for (32), (46) for (44) and (49) for (47).

3.3 Axiomatic Laws for NOT

The operator NOT is introduced axiomatically. In particular, it should satisfy the laws (A_1-A_5) in Figure 1, where $A, B, C : \text{CN}$ and $A \preceq B$ means that $A \leq_c B$ for some injective coercion c (i.e., in coercive subtyping (Luo, 1999; Luo et al., 2013), A is a subtype of B via the coercion c from A to B that is injective.[10])

Using the operator DO defined in (55-56), we can rewrite the laws (A_1-A_5) in Figure 1 as (A_1^d-A_5^d) in Figure 2, and it is straightforward to verify that each (A_i^d) is logically equivalent to (A_i), for $i = 1, ..., 5$: Arguably, for $i = 1, ..., 5$,

[8] A preliminary NOT-operator was studied by the second and third authors in (Chatzikyriakidis and Luo, 2017b), which is a special case for the NOT-operator proposed above. The study of the preliminary NOT-operator was limited in its scope – it assumed that there be a top type for all types in CN, its axiomatic laws were not general enough, their justification

(A_1) $\forall p{:}A \to Prop.\forall x{:}A.\ \neg\textsc{not}(A,p,A,x) \Leftrightarrow p(x)$.
(A_2) $\forall p,q{:}A \to Prop.\ [\forall x{:}A.\ p(x) \Rightarrow q(x)] \Rightarrow [\forall y{:}B.\ \textsc{not}(A,q,B,y) \Rightarrow \textsc{not}(A,p,B,y)]$.
(A_3) If $A \preceq B$, then $\forall p{:}B \to Prop.\forall z{:}C.\ \textsc{not}(B,p,C,z) \Rightarrow \textsc{not}(A,p,C,z)$.
(A_4) If $A \preceq B$, then $\forall p{:}C \to Prop.\ [\forall y{:}B.\textsc{not}(C,p,B,y)] \Rightarrow [\forall x{:}A.\textsc{not}(C,p,A,x)]$.
(A_5) If $A \preceq B$, then $\forall p{:}C \to Prop.\ [\exists x{:}A.\textsc{not}(C,p,A,x)] \Rightarrow [\exists y{:}B.\textsc{not}(C,p,B,y)]$.

Fig. 1 Axiomatic Laws for NOT.

(A_1^d) $\textsc{do}_{A,A}(p,x) \Leftrightarrow p(x)$.
(A_2^d) $[\forall x{:}A.\ p(x) \Rightarrow q(x)] \Rightarrow [\forall y{:}B.\ \textsc{do}(p,y) \Rightarrow \textsc{do}(q,y)]$.
(A_3^d) If $A \preceq B$, then $\textsc{do}_{A,C}(p,z) \Rightarrow \textsc{do}_{B,C}(p,z)$.
(A_4^d) If $A \preceq B$, then $\forall y{:}B.\neg\textsc{do}_{C,B}(p,y) \Rightarrow \forall x{:}A.\neg\textsc{do}_{C,A}(p,x)$.
(A_5^d) If $A \preceq B$, then $\exists x{:}A.\neg\textsc{do}_{C,A}(p,x) \Rightarrow \forall y{:}B.\neg\textsc{do}_{C,B}(p,y)$.

Fig. 2 Rewritten Axiomatic Laws for NOT.

each (A_i^d) gives an intuitively more understandable reading of the corresponding law (A_i).

Natural language examples and their semantics interpretations will be given in the following subsection §3.4 to illustrate the laws. Before then, we shall first give some remarks.

Remark 1 Some remarks are in order to explain the laws informally and elaborate on the restriction on injectivity in $(A_3$-$A_5)$.

1. *Informal readings.* In order to understand the laws $(A_1$-$A_5)$ in Figure 1, one may first rewrite them as $(A_1^d$-$A_5^d)$ by means of DO or use IS to instantiate their special cases, where one can informally read $\textsc{is}(A,b)$ as 'b is of type A' and $\textsc{do}(p,b)$ as 'b does p'. For instance, the first law, (A_1^d) or (A_1), says that, when $x : A$ (and hence $p(x)$ is well-typed), $\textsc{do}(p,x) = \neg\textsc{not}(A,p,A,x)$ is logically equivalent to $p(x)$ and, as a special case when $p = p_A$, $\textsc{is}(A,x) = \neg\textsc{not}(A,p_A,A,x)$ is logically equivalent to $p_A(x)$. As another example, the second law (A_2) or (A_2^d) says that, if predicate p is stronger than q, then 'b does p' implies 'b does q', for any b.
2. *Injectivity.* In laws (A_3), (A_4) and (A_5), we require that A be a subtype of B not just in general (injectivity is not necessarily required in coercive subtyping), but must be due to some injective coercion, which intuitively implies that the 'size' of A is not bigger than that of B. For example, consider a Σ-type $\Sigma x{:}A.P$, where P is a proposition. Then, with proof

was not given, and the issue of over-generation was not studied (for the study of these for the NOT-operator proposed here, see §3.3, §4 and §5).

[9] In (Xue et al., 2018), we have defined the proposition $P_{A,B} : B \to Prop$. In the current notation, we have $P_{A,B}(t) = \textsc{is}_B(A,t)$.

[10] A function $c : A \to B$ is injective if, for all $x_1, x_2 : A$, $c(x_1) = c(x_2)$ implies that $x_1 = x_2$. For instance, the identity function that maps any object to itself is injective. In a coercive subtyping relation $A \leq_c B$, if c from A to B is an injection function, then c is called an injective coercion.

irrelevance (Luo, 2012a, 2019b), as represented by the following rule $(*)$:

$$(*) \qquad \frac{\Gamma \vdash P : Prop \quad \Gamma \vdash p_1 : P \quad \Gamma \vdash p_2 : P}{\Gamma \vdash p_1 = p_2 : P}$$

the first projection $\pi_1 : (\Sigma x{:}A.P) \to A$ is injective because, for any (a_1, p_1) and (a_2, p_2) of type $\Sigma x{:}A.P$, if $a_1 = a_2$, then $(a_1, p_1) = (a_2, p_2)$ since we have $p_1 = p_2$ by proof irrelevance.[11]

3.4 Examples and Their Semantic Interpretations

To explain our laws more clearly and intuitively, we are going to provide some examples for each law. As we have mentioned above, $\text{DO}(p, b)$ means that 'b does p' and, in particular, when p is p_A, it means 'b is an A'. Hence, the examples in this subsection will cover both special cases 'b is an A' and general cases 'b does p' for each law. For each example, we will provide logical expressions as well as the Coq codes for each expression[12].

Before presenting the examples, we recall the type of NOT (57/58) and the definitions of predicate p_A (59/60), $\text{IS}_B(A, b)$ (61/62) and $\text{DO}_{A,B}(p, b)$ (63/64), together with the corresponding codes in the Coq proof assistant.

(57) $\text{NOT} : \Pi A{:}\text{CN}\ \Pi p{:}A \to Prop\ \Pi B{:}\text{CN}\ \Pi b{:}B.\ Prop$

(58) `NOT : forall A:CN, (A->Prop) -> forall B:CN, B->Prop`

(59) $p_A(x) = true$, for any $x : A$.

(60) `pr (A:CN)(a:A) := True`

(61) $\text{IS}_B(A, b) = \neg\text{NOT}(A, p_A, B, b)$, for any $b : B$.

(62) `IS(B:CN)(A:CN)(b:B) := ~(NOT A (pr A) B b)`

(63) $\text{DO}_{A,B}(p, b) = \neg\text{NOT}(A, p, B, x)$, for any $p : A \to Prop$ and $b : B$.

(64) `DO(A:CN)(B:CN)(p:A->Prop)(b:B) := ~(NOT A p B b)`

3.4.1 Examples for Law (A_1/A_1^d)

Law (A_1^d) states that, when A and B are the same, $\text{DO}_{A,B}(p, b)$ is logically equivalent to $p(b)$. It is repeated here together with its Coq code:

(65) $\forall p{:}A \to Prop.\forall x{:}A.\ \text{DO}_{A,A}(p, x) \Leftrightarrow p(x)$

(66) `(p:A->Prop)(x:A),(DO A A p x)<->(p x).`

In the special case when p is p_A, it says that $\text{IS}_A(A, b)$ is equivalent to $p_A(b)$.

[11] We have proved this in the proof assistant Coq (Coq, 2010): the Coq code for this can be found in Appendix B, where we have assumed the propositional version of proof irrelevance, which can be proved by means of the above $(*)$-rule.

[12] In Coq ',' is used as a 'separator' instead of '.'

Example 1

(67) John is not a man.

(68) $\neg p_{Man}(j)$, where $j : Man$.

(69) `~(pr Man j)`

By (A_1) and the definition of IS, we have $\neg p_{Man}(j) \Leftrightarrow \neg\neg\text{NOT}(Man, p_{Man}, Man, j) = \neg\text{IS}(Man, j)$.

Example 2

(70) It is not the case that the animal does not eat.

(71) $\neg\neg eat(a)$, where $eat : Animal \to Prop$ and $a : Animal$.

(72) `~~(eat a)`

By (A_1^d) and the definition of DO, we have $\neg\neg eat(a) \Leftrightarrow \neg\neg\neg\text{NOT}(Animal, eat, Animal, a) = \neg\neg\text{DO}(eat, a)$.

3.4.2 Example for Law (A_2/A_2^d)

Law (A_2/A_2^d) states that, 'if p is stronger than q, then that y does p implies that y does q'. The law (A_2^d) is repeated here together with its Coq code:

(73) $\forall p, q{:}A \to Prop.\ [\forall x{:}A.p(x) \Rightarrow q(x))] \Rightarrow [\forall y{:}B.\text{DO}(p, y) \Rightarrow \text{DO}(q, y)]$

(74)
```
forall(p q:A->Prop),
  (forall(x:A),(p x)->(q x))->forall(y:B),(DO A B p y)->(DO A B q y).
```

Example 3

(75) If a table doesn't talk, then it doesn't talk loudly.

(76) $\forall y{:}Table.\ \neg\text{DO}(talk, y) \Rightarrow \neg\text{DO}(talk\_loudly, y)$

(77) `(y:Table)(~(DO Human Table talk y))->(~(DO Human Table talk_loudly y))`

To explain the above example, we have: if we assume $\forall x{:}Human.\ talk\_loudly(x) \Rightarrow talk(x)$, then (76) (and (77) in Coq) can be proved by means of (A_2/A_2^d).

3.4.3 Example for Law (A_3/A_3^d)

Law (A_3/A_3^d) concerns with subtyping and states: if A is a subtype of B via an injective coercion and if $p : B \to Prop$ and $z : C$, then that z does p (with $p : A \to Prop$) implies that z does p (with $p : B \to Prop$). The law (A_3^d) is repeated here together with its Coq code:

(78) If $A \preceq B$, $\forall p{:}B \to Prop.\forall z{:}C.\ \text{DO}_{A,C}(p, z) \Rightarrow \text{DO}_{B,C}(p, z)$

(79)
```
Variable cAB:A->B. Coercion cAB:A>->B.
   Inj cAB -> (forall (p:B->Prop)(z:C), DO A C p z -> DO B C p z).
```

In the special case for the IS-operator, the above law becomes: if $A \preceq B$, then $\text{IS}(A, z) \Rightarrow \text{IS}(B, z)$ for any $z : C$.

Example 4 In this example, we consider a toy bear, Teddy, whose semantics is $Teddy : Toy$. Then, the semantics (81) of sentence (80) is intuitively true with this law since we usually have $Man \preceq Human$. To elaborate, we may define Man as $Man = \Sigma(Human, male)$, where $male : Human \to Prop$. Then, we have $Man = \Sigma(Human, male) \leq_{\pi_1} Human$, where the coercion π_1 is injective because of proof irrelevance (see the remark on injectivity in §3.3) and, therefore, $Man \preceq Human$. As a consequence, (81) can be easily proved by means of (A_3/A_3^d).

(80) If Teddy is a man, then Teddy is a human.

(81) $\textsc{is}(Man, Teddy) \Rightarrow \textsc{is}(Human, Teddy)$, where $Teddy : Toy$.

(82)
```
(IS Toy Man Teddy) -> (IS Toy Human Teddy)
```

Remark 2 Note that, when considering (A_3^d) (and (A_4^d) and (A_5^d) below), one needs to show that the subtyping relation is through an injective coercion. For instance, the above example uses the first projection as the coercion which can be shown to be injective when the second parameter of the Σ-type is a proposition (a subtyping relation often occurring in MTT-semantics). For the sake of simplicity, we shall skip the injectivity proof in the rest of this section.

3.4.4 Examples for Law (A_4/A_4^d)

Law (A_4/A_4^d) concerns with universal quantification and negation and states: if A is a subtype of B via an injective coercion and $p : C \to Prop$, then that all objects of type B do not do p implies that all objects of type A do not do p. The law (A_4^d) is repeated here together with its Coq code:

(83) If $A \preceq B, \forall p{:}C \to Prop.[\forall y{:}B.\neg \textsc{do}_{C,B}(p, y)] \Rightarrow [\forall x{:}A.\neg \textsc{do}_{C,A}(p, x)]$

(84)
```
Variable cAB:A->B. Coercion A>->B.
    Inj cAB -> forall(p:C->Prop),
    (forall(y:B),~(DO C B p y)) -> (forall(x:A),~(DO C A p x))
```

In the special case for the IS-operator, the above law says: if $A \preceq B$, we have $\forall y{:}B.\neg \textsc{is}(C, y) \Rightarrow \forall x{:}A.\neg \textsc{is}(C, x)$.

Example 5

(85) If women are not men, beautiful women are not men either.

(86) $\forall y{:}Woman.\neg \textsc{is}(Man, y) \Rightarrow \forall x{:}BWoman.\neg \textsc{is}(Man, x)$

(87)
```
(x:Woman)(~(IS Woman Man x))->(y:BWoman)(~(IS BWoman Man y))
```

where, in (86), $BWoman = \Sigma(Woman, Beautiful) \preceq Woman$ and, in (87), we use the following record type in Coq to represent the Σ-type $BWoman$:

```
Record BWoman : CN := mkBWoman {bw :> Woman; _ : Beautiful bw}
```

Example 6

(88) If tables do not talk, then red tables do not talk, either.

(89) $\forall x{:}Table.\neg \mathrm{DO}_{Human,Table}(talk, x) \Rightarrow \forall y{:}RTable.\mathrm{NOT}_{Human,RTable}(talk, y)$, where $RTable = \Sigma(Table, red)$.

(90)
```
(x:Table)(~(DO Human Table talk x))->(y:Redtable)(~(DO Human Redtable talk y))
```

where, in (89), $RTable = \Sigma(Table, Red) \preceq Table$ and, in (90), we use the following record type in Coq to represent the Σ-type $RTable$:

```
Record Redtable : CN := mkRedtable{rt :> Table; _ : Red rt}.
```

3.4.5 Examples for Law (A_5/A_5^d)

Law (A_5/A_5^d) concerns with existential quantification and negation and states: if A is a subtype of B via an injective coercion and $p : C \to Prop$, then that some object of type A does not do p implies that some object of type B does not do p. The law (A_5^d) is repeated here together with its Coq code:

(91) If $A \preceq B, \forall p{:}C \to Prop.[\exists x{:}A.\neg \mathrm{DO}_{C,A}(p, x)] \Rightarrow [\exists y{:}B.\neg \mathrm{DO}_{C,B}(p, y)]$

(92)
```
Variable cAB:A->B. Coercion A>->B.
Inj cAB -> forall(p:C->Prop),
(exists x:A,~(DO C A p x))->(exists y:B,~(DO C B p y))
```

In the special case for the IS-operator, the above law says: if $A \preceq B$, we have $\exists x{:}A.\neg \mathrm{IS}(C, x) \Rightarrow \exists y{:}B.\neg \mathrm{IS}(C, y)$.

Example 7

(93) Since not every linguist is a logician, not every human is a logician.

(94) $\neg\forall l{:}Linguist.(\mathrm{IS}(Logician, l)) \Rightarrow \neg\forall h{:}Human.(\mathrm{IS}(Logician, h))$

(95)
```
not (forall l:Linguist, IS Linguist Logician l)
-> not (forall h:Human, IS Human Logician h)
```

By using (A_5^d) in the special case for IS, (94) can be proved to be true.

Example 8

(96) It is not the case that every man works, so it is not the case that every human works.

(97) $\neg\forall l{:}Man.(\mathrm{DO}_{Human,Man}(work, l)) \Rightarrow \neg\forall l{:}Human.(\mathrm{DO}_{Human,Human}(work, l))$.

(98)
```
not (forall l:Man,DO Human Man work l)
-> not (forall l:Human,DO Human Human work l)
```

By using (A_5^d), (97) can be proved to be true.

4 Justification of NOT

In the above, NOT has been introduced axiomatically to extend the underlying modern type theory. Is such an extension OK (for instance, is it logically consistent)? The answer to this question is positive; put in another way, this extension can be justified. There can be rather different ways to provide such a justification. For instance, one could go ahead to prove the meta-theoretic properties of the extended system, showing directly that the extension is consistent, among other things. However, this would have been an overkill, as anyone who has done such meta-theoretic proofs knows that they are rather tedious and require a lot of effort. Another solution is to consider a known consistent extension E of the underlying type theory T and perform the following:

1. Known: $T + E$ is consistent.
2. Define NOT (by means of E) in $T + E$.
3. Using the definition, prove that (A_i) $(i = 1, ..., 5)$ are theorems in $T + E$.
4. Then, we know that $T +$ NOT is consistent.

The question is: what is a suitable notion E for the above? Luckily, there is such a notion E: we can use JMeq, the heterogeneous equality that has been studied by McBride (McBride, 2002; McBride and McKinna, 2004). NOT can be defined by means of JMeq and the extension by means of JMeq is logically consistent, so is our extension by the NOT-operator, as to be studied in this section.

4.1 Heterogeneous Equality JMeq

Usually, in a type theory, equalities are homogeneous one can only form an equality proposition between two objects of the same type. The equality judgements, written as $a = b : A$, presumes that a and b are of type A. Similarly, a logical proposition, like the Leibniz equality $a =_A b$ in UTT or the identity type $Id(A, a, b)$ in Martin-Löf's type theory, that states that a and b are the same, also assumes that a and b are of the same type A. If $a : A$, $b : B$, and A and B are different types, we usually cannot talk about whether a and b are equal – they cannot even be compared with each other.

A heterogeneous equality, however, allows one to talk about equality between arguments of different types. JMeq is such a heterogenous equality,[13] whose type is given in (99). Intuitively, JMeq(A, a, B, b) means that a and b are equal, although their types A and B may be different.

(99) JMeq : $\Pi A{:}Type\ \Pi x{:}A\ \Pi B{:}Type\ \Pi y{:}B.\ Prop.$

Reflexivity holds for JMeq: for any $x : A$, we always have JMeq(A, x, A, x). One can prove the lemmas that JMeq satisfies symmetry and transitivity and,

[13] JMeq is proposed and named 'John Major equality' by McBride (McBride, 2002).

therefore, it is an equivalence relation. The extension of the underlying MTT (e.g., UTT) with JMeq has been proved to be logically consistent (McBride, 2002).

In the standard library of the Coq proof assistant, JMeq is defined as follows:

```
Inductive JMeq (A:Type)(x:A) : forall B:Type, B->Prop
       := JMeq_refl : JMeq x x .
```

We shall use it in our Coq development below.

4.2 Justification of NOT by JMeq

NOT can be defined by means of JMeq as (100), which intuitively defines 'b does not p' as meaning that, for any x in A, if x equals b, then $p(x)$ is not true. Note that, although $p(b)$ may be ill-typed (because b's type B may not be the same as the domain A of p), $p(x)$ is well-typed since $x : A$.

(100) $\text{NOT}(A, p, B, b) = \forall x{:}A.\ \text{JMeq}(A, x, B, b) \Rightarrow \neg p(x)$.

We shall prove a theorem to show that the axiomatic laws for NOT are all theorems in an MTT with JMeq and, as a corollary, since the extension with JMeq is logically consistent (McBride, 2002), so is the extension by NOT.

To justify NOT by JMeq, we will prove the axiomatic laws (A_1-A_5) (and their equivalent (A_1^d-A_5^d)) as theorems when NOT is defined by JMeq as in (100). To prove (A_3), (A_4) and (A_5), we need to employ an axiom about injective coercions: for any $A, B : \text{CN}$ such that $A \preceq B$, we have $\forall x{:}A, \text{JMeq}(A, x, B, x)$. Note that this last formula actually stands for $\forall x{:}A.\text{JMeq}(A, x, B, c(x))$ for some injective coercion c. Then, for any $x_1, x_2{:}A$, if $c(x_1) = c(x_2)$, we have $\text{JMeq}(B, c(x_1), B, c(x_2))$. Hence we can derive that $\text{JMeq}(A, x_1, A, x_2)$ with symmetry and transitivity rules of JMeq, which gives us $x_1 = x_2$ in JMeq. This matches the injectivity of c.

Theorem 1 *If* NOT *is defined as in (100),* (A_i) *in Figure 1 or* (A_i^d) *in §3.3* $(i = 1, ..., 5)$ *are all provable in the type theory extended by* JMeq.

Proof With the definition (100) of NOT and properties of JMeq, we can prove the theorem without any difficulty.[14] □

5 Avoiding Over-Generations

Usually, a sentence whose semantic interpretation is a non-derivable judgements or involves ill-typed applications can be regarded as meaningless. But, as illustrated above in §2, when they occur as sub-sentences and *negatively*, such

[14] The proof of Theorem 1 has been done in the Coq proof assistant as well – see Appendix C for the Coq statements for (A_1^d-A_5^d) and their proofs.

sentences may be used meaningfully. Therefore, we have introduced and studied the IS/DO-operators (which can be defined by means of NOT) to interpret such sentences. However, it is obvious that we cannot use them without any restriction for, otherwise, over-generation may occur. For instance, usually $Table$ and $Human$ do not share objects and, therefore, (103) is non-derivable (since $j : Human$) and $talk(t)$ in (104) is ill-typed (since $talk : Human \to Prop$). We do not use (105) and (106) to interpret (101) and (102).

(101) (#) John is a table.

(102) (#) Tables talk.

(103) (#) $j : Table$

(104) (#) $\forall t : Table.\ talk(t)$

(105) $\text{IS}(Table, j)$

(106) $\forall t : Table.\ \text{DO}(talk, t)$

In this section, conditions are studied so that the IS/DO-operators can be employed without risking over-generation. We shall study two notions: the first being *type disjointness* and the second *negative occurrence* (Luo and Xue, 2020), both important for adequate use of IS and DO so that over-generation would not occur.

5.1 Type Disjointness: Judgemental Derivability and Ill-Typed Applications

When $j : Man$, the judgements such as $j : Table$ and $j : Woman$ are non-derivable. This non-derivability is not trivial: it assumes that the types Man be disjoint with $Table$ and $Woman$. When non-derivable, a judgemental interpretation signals that the NL sentence is meaningless (at least usually so, unless in some fictional or special settings). Similarly, some propositional applications are ill-typed. For instance, with $talk : Human \to Prop$ and $t : Table$, $talk(t)$ is ill-typed. Such ill-typed propositions usually represent meaningless sentences as well.

Here is a definition of the notion of type disjointness.

Definition 1 (disjointness) *Let A and B be types without free variables. A and B are* disjoint *if there is no non-empty type C such that $C \leq A$ and $C \leq B$ (i.e., if $\vdash c : C$, then either $\not\vdash C \leq A$ or $\not\vdash C \leq B$).* □

It is important to find out whether two types are disjoint in order to determine (in)correctness of judgements and well-typedness of terms, especially those involving IS/DO-operators. Put in another way, the disjointness of A and B is a necessary condition for the use of $\text{IS}_B(A, b)$ or $\text{DO}_{A,B}(p, b)$ in semantic interpretations:

1. If b is of type B and A and B are disjoint, then the judgement $b : A$ is incorrect; therefore, it may be possible to use $\text{IS}_B(A, b)$ to describe the semantics of $b : A$.

2. If b is of type B and A is the domain of predicate p, and A and B are disjoint, then $p(b)$ is ill-typed; therefore, it may be possible to use $\text{DO}_{A,B}(p, b)$ to describe the semantics of $p(b)$.

For example, usually $Table$ and $Human$ are disjoint and this is a necessary condition for us to use $\text{IS}(Table, j)$ and $\text{DO}(talk, t)$ to interpret the incorrect judgement $j : Table$ and the ill-typed term $talk(t)$, where $j : Human$ and $t : Table$.

5.2 Negative Occurrences

In semantic interpretations, the type disjointness as discussed above, is only a necessary condition, but not a sufficient condition, to use IS/DO-operators. Even when A and B are disjoint, it may not be reasonable to use $\text{IS}_B(A, b)$ ($\text{DO}_{A,B}(p, b)$) to interpret the incorrect judgement $b : A$ or the ill-typed $p(b)$ – we also need them to '*occur negatively*' (or to *occur in a negative context*), for otherwise, over-generations can occur.

The notion of negative occurrence is defined using an auxiliary notion of NOT-*expression*, inductively defined as follows:

1. $\text{IS}_B(A, b)$ and $\text{DO}_{A,B}(p, b)$ are NOT-expressions.
2. $A \wedge B$ and $A \vee B$ are NOT-expressions, if either A or B is an NOT-expression.
3. $\forall x{:}T.A$ and $\exists x{:}T.A$ are NOT-expressions, if A is an NOT-expression.

Definition 2 (negative occurrence) [15] *Let A, B and C be of the form $P_1 \oplus ... \oplus P_n$, where P_i's are atomic, $\oplus \in \{\wedge, \vee\}$ and $n \geq 1$. Then,*

1. *A and its subformulas* occur negatively *in $\neg A$.*
2. *A and its subformulas* occur negatively *in $A \Rightarrow B$.*
3. *B and its subformulas* occur negatively *in $A \Rightarrow B$ if A is a* NOT-*expression.*
4. *A subformula of A* occurs negatively *in $\forall x{:}T.A$ and $\exists x{:}T.A$ if it occurs negatively in A.*
5. *A formula* occurs negatively *in $A \Rightarrow B \Rightarrow C$ if it occurs negatively in $A \wedge B \Rightarrow C$.*

The operators IS/DO can be used to occur negatively in semantic constructions, without causing over-generation. Here are some examples.

(107) Women are not men.

(108) Tables do not talk.

(109) It is not the case that tables don't talk.

(110) If tables talk, so do chairs.

The semantics of the above sentences are as follows, with explanations:

(111) $\forall x{:}Woman.\ \neg\text{IS}(Man, x)$
This is semantics of (107) in which, according to (1) and (4) in Definition 2, $\text{IS}(Man, x)$ occurs negatively.

[15] Please note that the notion of 'negative occurrence' here is different from that of the term usually used in the literature.

(112) $\forall x{:}Table.\ \neg \textsc{do}(talk, x)$
This is semantics of (108) in which, according to (1) and (4) in Definition 2, $\textsc{do}(talk, x)$ occurs negatively.

(113) $\neg\forall x{:}Table.\ \neg \textsc{do}(talk, x)$
This is semantics of (109). The only difference with the above semantics (112) of (108) is that we have one more negation in front. According to (1) and (4) in Definition 2, $\textsc{do}(talk, x)$ occurs negatively.

(114) $[\forall x{:}Table.\ \textsc{do}(talk, x)] \Rightarrow [\forall y{:}Chair.\ \textsc{do}(talk, y)]$
This is semantics of (110) in which, according to (2) and (3) in Definition 2, $\textsc{do}(talk, x)$ and $\textsc{do}(talk, y)$ both occur negatively.

In the above, the semantics (113) of (109) shows that, different from the usual notion of negative occurrence in the literature (c.f., Footnote 15), even 'double negation' may not change the status of being a negative occurrence.

Also, we remark that the last clause in Definition 2 is saying that, in the world of semantics, the negative occurrences in the formula $A \Rightarrow B \Rightarrow C$ are the same as those in $A \wedge B \Rightarrow C$ (that is, they have the same negative occurrences).

6 Conclusion

In this paper, we have proposed several operators for constructing propositional forms of judgemental interpretations and ill-typed applicational semantic interpretations. It has been shown that the introduction of these operators can be justified by means of the heterogeneous equality in type theory. Furthermore, we have studied conditions that these operators may be employed in semantic constructions without risking over-generation. Future work includes further studies of more extensive uses of such operators in semantic constructions.

Conflict of interest

The authors declare that they have no conflict of interest.

References

Agda 2008 (2008) The Agda proof assistant (version 2). Available from the web page: `http://appserv.cs.chalmers.se/users/ulfn/wiki/agda.php`

Asher N (2012) Lexical Meaning in Context: a Web of Words. Cambridge University Press

Bekki D (2014) Representing anaphora with dependent types. LACL 2014, LNCS 8535

Boldini P (2000) Formalizing context in intuitionistic type theory. Fundamenta Informaticae 42(2):1–23

Callaghan P, Luo Z (2001) An implementation of LF with coercive subtyping and universes. Journal of Automated Reasoning 27(1):3–27
Chatzikyriakidis S, Luo Z (2013) Adjectives in a modern type-theoretical setting. In: Morrill G, Nederhof J (eds) Proceedings of Formal Grammar 2013. LNCS 8036, Springer, pp 159–174
Chatzikyriakidis S, Luo Z (2014) Natural language reasoning in Coq. J of Logic, Lang and Info 23(4)
Chatzikyriakidis S, Luo Z (2016) Proof assistants for natural language semantics. Logical Aspects of Computational Linguistics 2016, Nancy LNCS 10054
Chatzikyriakidis S, Luo Z (2017a) Adjectival and adverbial modifications: The view from modern type theories. Journal of Logic, Language and Information 26(1)
Chatzikyriakidis S, Luo Z (2017b) On the interpretation of common nouns: Types v.s. predicates. In: Modern Perspectives in Type-Theoretical Semantics, Springer
Chatzikyriakidis S, Luo Z (2020) Formal Semantics in Modern Type Theories. Wiley/ISTE
Church A (1940) A formulation of the simple theory of types. J Symbolic Logic 5(1)
Cooper R (2005) Records and record types in semantic theory. J Logic and Compututation 15(2)
Coq (2010) The Coq Proof Assistant Reference Manual (Version 8.3), INRIA. The Coq Development Team
Curry H, Feys R (1958) Combinatory Logic, vol 1. North Holland Publishing Company
Dapoigny R, Barlatier P (2009) Modeling contexts with dependent types. Fundamenta Informaticae 21
Gallin D (1975) Intensional and higher-order modal logic: with applications to Montague semantics. North-Holland, Amsterdam
Groenendijk J, Stokhof M (1990) Dynamic Montague grammar. Proceedings of the Second Symposium on Logic and Language
Groenendijk J, Stokhof M (1991) Dynamic predicate logic. Linguistics and Philosophy 14(1)
Henkin L (1950) Completeness in the theory of types. J Symbolic Logic 15:81–91
Howard W (1980) The formulae-as-types notion of construction. In: Hindley J, Seldin J (eds) To H. B. Curry: Essays on Combinatory Logic, Academic Press
Kahle R, Schroeder-Heister P (eds) (2006) Proof-Theoretic Semantics. Special Issue of Synthese, 148(3)
Kamp H, Reyle U (1993) From Discourse to Logic. Kluwer
Luo Z (1994) Computation and Reasoning: A Type Theory for Computer Science. 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. 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) LNAI 6736
Luo Z (2012a) Common nouns as types. In: Bechet D, Dikovsky A (eds) Logical Aspects of Computational Linguistics (LACL'2012). LNCS 7351
Luo Z (2012b) Formal semantics in modern type theories with coercive subtyping. Ling & Phil 35(6)
Luo Z (2014) Formal Semantics in Modern Type Theories: Is It Model-theoretic, Proof-theoretic, or Both? Invited talk at Logical Aspects of Computational Linguistics 2014 (LACL 2014), Toulouse LNCS 8535 pp 177–188
Luo Z (2018) Formal semantics in modern type theories (and event semantics in mtt-framework). Invited talk at LACompLing18
Luo Z (2019a) Formal Semantics in Modern Type Theories (MTT-semantics is both model/proof-theoretic). Proof-Theoretic Semantics: Assessment and Future Perspectives, Proc of the Third Tuebingen Conference on Proof-Theoretic Semantics, Tuebingen
Luo Z (2019b) Proof irrelevance in type-theoretical semantics. Logic and Algorithms in Computational Linguistics 2018 (LACompLing2018), Studies in Computational Intelligence (SCI) Springer
Luo Z, Callaghan P (1998) Coercive subtyping and lexical semantics (extended abstract). Logical Aspects of Computational Linguistics (LACL'98)
Luo Z, Pollack R (1992) LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Dept of Computer Science, Univ of Edinburgh
Luo Z, Xue T (2020) Disjointness of types and negative occurrences. Notes, February 2020
Luo Z, Soloviev S, Xue T (2013) Coercive subtyping: theory and implementation. Information and Computation 223:18–42
Martin-Löf P (1975) An intuitionistic theory of types: predicative part. In: HRose, JCShepherdson (eds) Logic Colloquium'73
McBride C (2002) Elimination with a motive. In: Callaghan P, Luo Z, McKinna J, Pollack R (eds) Types for Proofs and Programs, Proc. of TYPES 2000, LNCS 2277, Springer
McBride C, McKinna J (2004) The view from the left. Journal of Functional Programming 14(1):69–111
Mönnich U (1985) Untersuchungen zu einer konstruktiven Semantik fur ein Fragment des Englischen. Habilitation. University of Tübingen
Montague R (1974) Formal Philosophy: Selected Papers of Richard Montague. Yale University Press
Muskens R (1996) Combining Montague semantics and discourse representation. Linguistics and philosophy 19(2):143–186
Nordström B, Petersson K, Smith J (1990) Programming in Martin-Löf's Type Theory: An Introduction. Oxford University Press
Ranta A (1994) Type-Theoretical Grammar. Oxford University Press

Retoré C (2014) The Montagovian Generative Lexicon Ty_n: a Type Theoretical Framework for Natural Language Semantics. Proofs and Programs (TYPES 2013), LIPIcs(26)
Sundholm G (1986) Proof theory and meaning. D Gabbay and F Guenthner (eds) Handbook of Philosophical Logic, Vol III
Sundholm G (1989) Constructive generalized quantifiers. Synthese 79(1):1–12
Xue T (2013) Theory and implementation of coercive subtyping. PhD thesis, Royal Holloway, University of London
Xue T, Luo Z (2012) Dot-types and their implementation. Logical Aspects of Computational Linguistics (LACL 2012) LNCS 7351
Xue T, Luo Z, Chatzikyriakidis S (2018) Propositional forms of judgemental interpretations. Proc of Workshop on Natural Language and Computer Science Oxford

A A Short Introduction to Coq

The rationale behind Coq, and proof-assistants in general, can be roughly summarized as follows: you use Coq in order to check whether propositions based on statements previously pre-defined or user-defined (definitions, parameters, variables) can be proven or not. Coq is a dependently typed proof assistant and implements the Calculus of Inductive Constructions (pCIC) (Coq, 2010), which is a modern type theory, very similar to the type theory UTT (Luo, 1994); thus, Coq 'speaks' so to say the language we use to interpret linguistic semantics in this paper. Coq is a reasoning engine and there are at least ways that can be used in studying linguistic semantics, to an extent overlapping with each other: a) as a formal checker for the semantic validity of proposed accounts in NL semantics and b) Natural Language Inference (NLI), i.e. reasoning with NL.

We proceed by formulating a simple example of NLI in Coq. In effect, NLI is seen as a theorem proving task, where 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 (115), will be formulated as the following theorem (named *ex*) in Coq (116):

(115) John walks ⇒ some man walks

(116) Theorem ex: John walks → some man walks

Then, depending on the semantics of the individual lexical items one may or may not be able to prove the theorem in question. Inferences like the one shown in (115) are easy cases to prove. Assuming a semantics of *some* specifying that given any A of type CN and a predicate of type $A \to Prop$, there exists an $x : A$ such that $P(x) : Prop$, such cases are straightforwardly proven.

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 Coq's universe Set and interpret common nouns like *man, human* as being of type CN (thus we have for example $Man, Human : CN$). Verbs are defined as predicates requiring arguments of type $A : CN$ (the choice of A depends on the verb itself.) Coercive subtyping is supported in Coq. Adjectives are defined as predicates, and adjectival modification as Σ types. Quantifiers and VP adverbs are defined as types ranging over the universe CN. For the example we are interested in, the following are declared:

```
CN:=Set.
Parameter Man Human Animal: CN.
Parameter John: Man.
```

```
Axiom mh: Man->Human. Coercion mh: Man>->Human.
Axiom ha: Human->Animal. Coercion mh:Human>->Animal.
Definition some:= fun A:CN, fun P:A->Prop=> exists x:A, P(x).
Definition walk: Animal->Prop
```

We have introduced CN as Coq's universe Set, declared Man, $Human$ and $Animal$ to be of type CN, further introduced the relevant subtyping relations and lastly $walk$. With $walk$ as being of type $Human \to Prop$ and $John$ as being of type Man with $Man < Human$, we can prove the theorem in (116) easily. We first use the proof tactic $intro$ to move the premises to be hypotheses. Then, we apply the tactic $unfold$ to $some$ ($unfold\ some$). $Unfold$ does exactly what it promises: it unfolds the definition associated with a lexical entry (if there is one):

```
ex < unfold some.
1 subgoal
============================
walk John -> exists x : Man, walk x
```

The $intro$ tactic is then used, moving the antecedent to the list of premises. Now, one can existentially instantiate $x : Man$ with $John : Man$:

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

Finally, the tactic $assumption$ finishes the proof.

B Coq Code for a Proof of Injectivity

Assuming proof irrelevance, one can show the injectivity of the first projection for a Σ-type whose second component is a proposition.

```
Definition CN := Set.
Definition Inj{X Y:Type}(f:X->Y) := forall(x y:X), (f x)=(f y) -> x=y.

Axiom proof_irrelevance : forall (P:Prop) (p1 p2:P), p1 = p2.

Variable A : CN. Variable B : A -> Prop.
Record Sigma : CN := mkSigma {p1 :> A; _ : B p1}.

Lemma inj_p1 : Inj p1.
Proof.
cbv.
intros. destruct x. destruct y.
subst. cut (b=b0).
intros. subst. auto.
apply proof_irrelevance.
Qed.
```

C Justification of NOT by JMeq: Coq Proofs

Here are the Coq proofs of the laws (A_1^d-A_5^d), where NOT is defined by means of the heterogeneous equality JMeq and DO by means of NOT.

```
Require Import Coq.Logic.JMeq.
Require Import Classical_Prop.
Definition CN := Set.

(* NOT defined by means of JMeq and DO by NOT *)
Definition NOT (A:CN)(p:A->Prop)(B:CN)(b : B) := forall x:A, JMeq x b -> not (p x).
Definition DO (A:CN)(B:CN)(p: A -> Prop)(b:B) := not (NOT A p B b).

(* A1d: if x:A & p:A->Prop, DO(p,x) iff p(x) *)
Definition A1d := forall (A:CN)(x:A)(p:A->Prop), DO A A p x <-> (p x).
Lemma law1 : A1d.
unfold A1d. unfold DO. unfold NOT. intros A x p.
split.
(* proof -> *)
cut (~ (p x) -> forall x0 : A, JMeq x0 x -> ~ (p x0)).
intros.
cut (~ p x -> False).
apply NNPP. intros. apply (H0 (H H1)).
unfold not. intros. apply (H (JMeq_ind p H1 H0)).
(* proof <- *)
unfold not. intros.
apply (H0 x). apply JMeq_refl. apply H.
Qed.

(* A2d: if p is stronger than q then DO(p,b) => DO(q,b) *)
Definition A2d :=
forall (A B:CN)(p q:A->Prop),
(forall (x:A), (p x)->(q x)) -> forall (y:B), (DO A B p y) -> (DO A B q y).
Lemma law2 : A2d.
Proof.
cbv. intros. apply H0. intros.
apply (H1 x). apply H2. apply (H x H3).
Qed.

(* injectivity *)
Definition Inj{X Y:Type}(f:X->Y) := forall(x y:X), (f x)=(f y) -> x=y.
Variables A B C : CN.
Variable cAB : A->B. Coercion cAB : A >->B.
Axiom JMeq_inj : Inj cAB -> forall (x:A), (@JMeq A x B x ).

(* A3d: If A <=_c B with Inj(c) and z:C, then DO[A,C](p,z) => DO[B,C](p,z) *)
Definition A3d := Inj cAB -> (forall (p:B->Prop)(z:C), DO A C p z -> DO B C p z).
Lemma law3 : A3d.
Proof.
cbv. intros. apply H0. intros. apply (H1 x).
apply (JMeq_trans (JMeq_sym (JMeq_inj H x))). apply H2. apply H3.
Qed.

(* A4d: If A <=_c B with Inj(c), then
   forall y:B.~DO[C,B](p,y) => forall x:A.~DO[C,A](p,x) *)
Definition A4d :=
   Inj cAB ->
   forall(p:C->Prop), (forall(y:B), ~(DO C B p y)) -> (forall(x:A), ~(DO C A p x)).
```

```
Lemma law4 : A4d.
Proof.
cbv. intros. apply (H0 x). intros. apply H1. intros.
apply (H2 x0). apply (JMeq_trans H3). apply (JMeq_inj H). apply H4.
Qed.

(* A5d: If A <=_c B with Inj(c), then
   (Exsits x:A. ~DO[C,A](p,x)) => Exists y:B. ~DO[C,B](p,y) *)
Definition A5d :=
   Inj cAB ->
   forall(p:C->Prop), (exists x:A, ~(DO C A p x))->(exists y:B, ~(DO C B p y)).
Lemma law5 : A5d.
Proof.
cbv. intros. destruct H0. exists x. intros.
apply H0. intros. apply H1. intros.
apply (H2 x0). apply (JMeq_trans H3). apply (JMeq_sym).
apply (JMeq_inj H). apply H4.
Qed.
```