Propositional Forms of Judgemental Interpretations

Tao Xue, Zhaohui Luo and Stergios Chatzikyriakidis

June 2, 2018

Propositional Forms of Judgemental Interpretations

Tao Xue* Zhaohui Luo† Stergios Chatzikyriakidis‡

Abstract

In type-theoretical semantics, sentences may often be interpreted as judgements, rather than propositions. When interpreting composite sentences such as those involving negations and conditionals, one may want to turn a judgemental interpretation into a proposition in order to obtain an intended semantics. In this paper, we propose a new negation operator NOT for constructing propositional forms of judgemental interpretations. NOT is introduced axiomatically, with five axiomatised laws to govern its behaviour, and several examples are given to illustrate its use in semantic interpretation. In order to justify NOT, we employ a heterogeneous equality to prove its laws and, since the addition of heterogeneous equality to type theories is consistent, so is our introduction of the NOT operator.

1 Introduction

In recent years, rich type systems have been successfully employed in formal semantics including, for example, [1, 11, 16]. In some of these approaches common nouns (or some of them) are interpreted as types, rather than as predicates. For example, in formal semantics in Modern Type Theories (MTT-semantics for short) [15, 11, 7], the rich type structure has been used effectively to interpret a wide range of modifications [5].

Interpreting CNs as types has led to the interpretation of some sentences as judgements, rather than logical propositions. For instance, the following sentence (1):

(1) Bob is a student.

is interpreted as the judgement $bob : Student$, where $Student$ is a type, rather than the proposition $student(bob)$, where $student$ is a predicate. There are some advantages with such an interpretation, one of them being that, with CNs interpreted as types, selectional restriction can naturally be enforced automatically by means of typing. For example, consider the following sentence (2):

(2) (#) Tables talk.

Most people would say that, in the normal circumstances, (2) is meaningless. This can be captured by typing: for example, for $talk : Human \to Prop$, the semantic interpretation (3) of the sentence (2) would not be well-typed, since a table t is not a human:

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

Interpreting some sentences as judgements, one would want to turn a judgemental interpretation into a proposition so that composite sentences can be interpreted by logical compositions, especially that negative sentences and conditionals can be properly considered semantically. For example, unlike (2), the following sentence (4) is meaningful. However, if we took the intuitive interpretation of (4), it would be the untyped (5) since, again, a table is not a human.[1]

(4) Tables do not talk.

(5) (#) $\forall t : Table.\ \neg talk(t)$

How should the sentences like (4) be interpreted in such a setting? In MTT-semantics, proposals have been made in [6] to use presuppositions of logical formula to capture non-hypothetical judgements and to extend

*Huazhong Univ of Science and Technology, Wuhan, China. xuetao@hust.edu.cn

†Royal Holloway, University of London, U.K. zhaohui.luo@hotmail.co.uk

‡CLASP, University of Gothenburg, Sweden. stergios.chatzikyriakidis@gu.se

[1] Several people have raised this issue of how to interpret negated sentences in MTT-semantics to the second author including Glyn Morrill (during ESSLLI 2011), Nicholas Asher (in an email communication about a paper in LACL 2014) and Koji Mineshima (in ESSLLI 2014). We are grateful for their interesting discussions and comments.

the underlying type theory with a negation operator, called $\textsc{not}_{old}$ in this paper, to deal with judgemental negations for negative sentences and hypothetical judgements for conditionals. However, although it delivers intended semantical treatments, the correctness of $\textsc{not}_{old}$ has not been justified: for example, it has not been proven that the extension by the $\textsc{not}_{old}$ operator is logically consistent.

In this paper, we propose a new negation operator NOT, which is more general than $\textsc{not}_{old}$ ($\textsc{not}_{old}$ is a special case of NOT). Unlike [6], to introduce NOT, we do not need to assume the existence of a top type (a super type of all other types that interpret CNs). Besides being capable of dealing with negative sentences and conditionals in satisfactory ways, NOT and its associated laws can be justified by means of the heterogeneous equality for type theories [14] and, in particular, it is shown that the extension by NOT is logically consistent. This shows, we contend, that our proposal offers a satisfactory solution with an adequate justification so that negative sentences and conditionals can be properly considered in the extended type theory.

2 Judgemental Interpretations

2.1 CNs-as-Types

Interpreting common nouns as types (CNs-as-types) is a major approach when we consider formal semantics with Modern Type Theories. This CNs-as-types approach was first studied in Ranta's seminal work on using Martin-Löf type theory in formal semantics [15]. For instance, the sentence (6) is interpreted as (7):

(6) Every teacher talks.

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

where $Teacher$ is a type that interprets the CN 'teacher' and $talk : Human \to Prop$ interprets the verb 'talk'. The CNs-as-types approach, has several advantages as compared with other approaches such as the CNs-as-predicates approach. For example, it has been used successfully to deal with selectional restrictions and copredication [9] and has been applied to practical reasoning by implementing these ideas in the Coq proof assistant ([3, 4]). 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 table ate the egg.

where the predicate $eat : Animal \to Food \to Prop$ needs an argument of type $Animal$ while a table is not of type $Animal$.

To adopt the CNs-as-types approach, it is important to note that there must be a compatible subtyping mechanism in the type-theoretical framework, otherwise, the approach would not be viable. For instance, in the semantics (7) of (6), $Teacher$ is a subtype of $Human$ and this subtyping relationship makes the application $talk(x)$ in (7) well-typed. Fortunately, there is a subtyping mechanism called coercive subtyping [8, 13, 17] that is suitable for modern type theories and makes the CNs-as-types approach viable [9, 11]. We also have studied this approach systematically to show how various classes of CNs with adjectival and adverbial modifications can be interpreted as types in MTTs [2, 5].

2.2 Propositional forms of non-hypothetical judgements

In MTT-semantics, some of the sentences are interpreted with non-hypothetical judgements. For instance, we can consider the following sentence (9) which is usually interpreted as (10), where $Doctor$ is a type:

(9) John is a doctor

(10) $j : Doctor$

The 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 $\Gamma \vdash a : A$ where the context Γ is empty.

As proposed in [6], the propositional form of a non-hypothetical judgement $a : A$ can be the proposition $p_A(a)$, where $p_A(x) = True$ is the constant predicate that returns true for all $x : A$. Note that $p_A(a)$ is well-typed (and equals True) if, and only if, $a : A$, because the well-typedness of $p_A(a)$ presupposes that $a : A$.

Definition 2.1 *(predicate p_A) Let A be a* CN. *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.*

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 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.

2.3 Judgemental Interpretations: a Problem with Negative Sentences and Conditionals

Consider the following sentences:

(11) Bob is a student.

(12) If Carl is a student, he is happy.

(13) Carl is not a student.

If we use judgemental interpretations, intuitively, we would get the semantics of (11), (12) and (13) as (14), (15) and (16), respectively.

(14) $Bob : Student$

(15) $Carl : Student \vdash happy(Carl)\ true$

(16) $(\#) \nvdash Carl : Student$

Note that, interpreting (12), we use the hypothetical judgement (15), where $Carl : Student$ that interprets the if-part of (12) is formalised as a contextual entry on the left of the turn-style symbol.[2] Also note that (16) is not a judgement at all (it is only a meta-level statement that the judgement $\vdash Carl : Student$ is not derivable.)

A question arises: how to interpret the negative sentences such as (13)? A solution would be to turn a judgemental interpretation into a proposition so that composite sentences can be interpreted by logical compositions. As a first attempt, we could think that the sentences (11), (12) and (13) could be interpreted as $p_{Student}(Bob)$, $p_{Student}(Carl) \Rightarrow happy(Carl)$ and $\neg p_{Student}(Carl)$, respectively, and then we would be able to use logical compositions to construct semantics of more complicated composite sentences. However, we should be very careful with such interpretations. In fact, the above is not quite right. For example, it would not be always correct to interpret (13) as $\neg p_{Student}(Carl)$, because the well-typedness of the formula already presupposes that $Carl : Student$. Put in another way, we should realize that, in (13), the interpretation of *Carl* may not always be of type *Student*; instead, it could well be some object which is not a student. In the next section, we will propose an operator to interpret such cases of negation sentences in NL like (13).

3 NOT: a Negation Operator

A judgemental interpretation can be 'negated' when one considers the semantics of negative sentences or conditionals, such as the following examples:

(17) John is not a doctor.

(18) If John is a doctor, he works hard.

The judgemental interpretation for (17) would be the negation of the judgement $j : Doctor$ and that for (18) would be the hypothetical judgement $j : Doctor \vdash [\![work\ hard]\!](j)\ true$, where $[\![work\ hard]\!] : Human \to Prop$. However, one cannot use $\neg p_{Doctor}(j)$ or $p_{Doctor}(j) \Rightarrow [\![work\ hard]\!](j)$ as the propositional forms of these judgemental interpretations, because both of them (i.e., their well-typedness) would have already presumed that $j : Doctor$, which may not be the case in either (17) or (18).

A solution to the above problem is to extend the underlying type theory with a negation operator that can play the role of capturing propositional forms of judgemental negations and hypothetical judgements. In [6], a negation operator $\textsc{not}_{old}$ has been proposed,[3] which is of type

$$\Pi A : \textsc{cn}.(A \to Prop) \to (Obj \to Prop)$$

[2] It would be even better if *Carl* is a constant, not a variable. This requires us to introduce signatures [12].

[3] The operator is simply called NOT in [6], but we shall call it $\textsc{not}_{old}$ to distinguish it from the NOT operator to be introduced in the current paper.

where Obj is the top type in the universe $\textsc{cn}$ of common nouns. Although it provided a nice way to solve the problems in the negative sentences or conditionals, it has some drawbacks. First, the definition of $\textsc{not}_{old}$ requires a super type of all other types that interpret common nouns, which is unnecessary. One may even argue that the existence of such a top type is unreasonable to be assumed. Second, the justification of $\textsc{not}_{old}$ was not given in [6] – we will do that in the current paper.

We propose a new negation operator $\textsc{not}$:

$$\textsc{not} : \Pi A : \textsc{cn}\ \Pi p : A \to Prop.\Pi B : \textsc{cn}.\Pi b : B.Prop$$

Intuitively, $\textsc{not}(A, p, B, b)$ means that 'b does not p' and, in particular, when p is p_A, it means 'b is not an A' – see the following definition, where the proposition $P_{A,B}(t)$ means that for the term t of type B, t can behave as a term of A without assuming that t is of type A.

Definition 3.1 *(predicate $P_{A,B}$) Assume that $A, B : \textsc{cn}$. Then, predicate $P_{A,B} : B \to Prop$ is defined as: for any $x : B$, $P_{A,B}(x) = \neg\textsc{not}(A, p_A, B, x)$, where p_A is the predicate defined in Definition 2.1.*

For example, the propositional forms of the judgemental interpretations of (17) and (18) are (19) and (20), respectively:

(19) $\textsc{not}(Doctor, p_{Doctor}, Human, j)$

(20) $P_{Doctor,Human}(j) \Rightarrow [\![work\ hard]\!](j)$

where, with P being capital, $P_{Doctor,Human}(j) = \neg\textsc{not}(Doctor, p_{Doctor}, Human, j)$ is the propositional form of the judgemental premise $j : Doctor$ in $j : Doctor \vdash [\![work\ hard]\!](j)\ true$.

Similarly, we can give some more examples for the use of $\textsc{not}$, covering both scenarios. (21)(23) are examples for 'b is not p', (22)(24) are for 'b does not p'.

(21) Women are not men.

(22) Tables do not talk.

(23) Some logicians are not linguists.

(24) Some logicians don't talk.

The above examples can be interpreted as:

(25) $\forall x : Woman.\ \textsc{not}(Man, p_{Man}, Woman, x)$

(26) $\forall x : Table.\ \textsc{not}(Human, talk, Table, x)$

(27) $\exists x : Logician.\ \textsc{not}(Linguist, p_{Linguist}, Logician, x)$

(28) $\exists x : Logician.\ \textsc{not}(Human, talk, Logician, x)$

Note that, usually, we would interpret (24) as $\forall x : Logician.\ \neg talk(x)$ which, according to the law (A_1) below, is equivalent to (28).

3.1 Laws for NOT

The negation operator $\textsc{not}$ is introduced axiomatically. In particular, it should satisfy the laws below that govern the behaviour of the negation operator. Before presenting them, we shall first define the notion of injectivity which will be used in their formulations.

Definition 3.2 (injectivity) *$c : A \to B$ 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 x any type A to itself is injective.

Definition 3.3 *$A \preceq B$ means that $A \leq_c B$ for some injective c.*

Formally, for all $A, B, C : \textsc{cn}$, the operator $\textsc{not}$ satisfies the following rules:

(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 $A \preceq B, \forall p : B \to Prop. \forall z : C.\textsc{not}(B, p, C, z) \Rightarrow \textsc{not}(A, p, C, z)$

(A_4) If $A \preceq B, \forall p : C \to Prop.(\forall y : B.\textsc{not}(C, p, B, y)) \Rightarrow \forall x : A.\textsc{not}(C, p, A, x)$

(A_5) If $A \preceq B, \forall p : C \to Prop.(\exists x : A.\textsc{not}(C, p, A, x)) \Rightarrow \exists y : B.\textsc{not}(C, p, B, y)$

It is straightforward to see that the negation operator proposed in [6], called $\textsc{not}_{old}$ here, is a special case: $\textsc{not}_{old}(A, p, o)$ can be defined as $\textsc{not}(A, p, Obj, o)$, where Obj is the top type in $\textsc{cn}$ (i.e., $A \leq Obj$ for all $A : \textsc{cn}$). We note that, in (A_3), p is also of type $A \to Prop$ which is a supertype of $B \to Prop$. When specialised to p_A, the laws (A_1) and (A_3) are those called (L_1) and (L_2) in [6], which relate the operator $\textsc{not}$ with the predicate p_A with (A_1) saying that, if A and B are the same types, $\neg\textsc{not}(A, p_A, B, b)$ is logically true and (A_3) stating that, if A is a subtype of B and $z : C$, then z is not a B implies that z is not an A. To introduce $\textsc{not}$, we do not have to assume the existence of Obj anymore, which allows more flexibility in semantic interpretations.

In intuitionistic (and classical) logic, if $p \Rightarrow q$ we can derive that $\neg q \Rightarrow \neg p$. Comparatively, ($A_2$) says that in our notion with $\textsc{not}$, if 'b does p' implies 'b does q', we should have 'b does not q' implies 'b does not p'.

(A_4) and (A_5) are focusing on the type of object b for 'b is p' or 'b does p'. In law (A_4), if all terms of type B 'does not p', then for any term of B's subtype A, it 'does not p' as well. Law (A_5) means that, if there is a term $x : A$ which satisfies 'x does not p', then for A's supertype B, there exists a term $y : B$, which satisfies 'y does not p'. Apparently, x is an evidence of such a claim.

Discussion on Injectivity. The injectivity condition has been added when we have coercions in laws (A_3), (A_4) and (A_5). Generally in coercive subtyping, *injectivity* is not a necessary condition. However, when we use coercive subtyping to model subsumptive subtyping, the coercions should not be non-injective. More precisely, in subsumptive subtyping with rule (sub), the 'size' of A is not bigger than that of B. But a non-injective surjection $c : A \to B$ implies that the size of A is not smaller than that of B.

$$\frac{a : A \quad A \leq B}{a : B} \ (\text{sub})$$

If we think that in the applications to NL semantics, subtyping means something like subsumptive subtyping, we should then take that injectivity always holds. Particularly, we should also assume proof irrelevance [10],

$$\frac{\Gamma \vdash A : Prop \quad \Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash a = b : A}$$

In particular, for a Σ-type $\Sigma x : A.B(x)$ where $B(x)$ is a proposition, the first projection π_1 is injective because of proof irrelevance.

Lemma 3.4 *Let A be a type and $B : A \to Prop$. If proof irrelevance holds, then the projection $\pi_1 : (\Sigma x : A.B(x)) \to A$ is injective.*

3.2 Examples to illustrate the laws

To explain our laws more clearly and intuitively, we are going to provide some examples for each law. As we have mentioned above, $\textsc{not}(A, p, B, b)$ means that 'b does not p' and. In particular, when p is p_A, it means 'b is not an A'. Hence, the examples in this subsection will cover both special cases 'b is not an A' and general cases 'b does not p' for each law. For each example, we will provide logical expressions as well as the Coq codes for each expression.

Before presenting the examples, we recall the type of $\textsc{not}$ (29/30), predicate p_A (31/32) and predicate $P_{A,B}$ (33/34) with their Coq code.

(29) $\textsc{not} : \Pi A : \textsc{cn}\ \Pi.p : A \to Prop.\Pi B : \textsc{cn}.\Pi b : B.Prop$

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

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

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

(33) for any $x : B, P_{A,B}(x) = \neg\textsc{not}(A, p_A, B, x)$

(34) `PR(A:CN){B:CN}(b:B) := not(NOT A (pr A) B b)`

3.2.1 Examples for Law (A_1)

Law (A_1) is repeated here together with its Coq code:

(35) $\forall p : A \to Prop.\forall x : A.\neg\text{NOT}(A, p, A, x) \Leftrightarrow p(x)$

(36)
```
(p:A->Prop)(x:A),not(NOT A p A x)<->(p x).
```

With 'double negation' strategy, $\neg\text{NOT}(A, p, B, b)$ intuitively means 'b does p' or 'b is an A' without preassumption $b : A$. Specially, when $B = A$, this double negation should be the same with our usual description.

Example 3.5

(37) It is not the case that John is not a man.

(38) $\neg\text{NOT}(Man, p_{Man}, Man, John)$*, where* $John : Man$*.*

(39)
```
not (NOT Man (pr Man) Man John)
```

By using (A1), we can show that $p_{Man}(John)$ is true when we have $John : Man$, it trivially conforms with the definition of p_A (Definition 2.1)

$$\neg\text{NOT}(Man, p_{Man}, Man, John) \Leftrightarrow p_{Man}(John)$$

Please note that, usually, we shall interpret (37) as $\neg\neg p_{Man}(John)$, instead of (38), although these two propositions are equivalent. (This applies to The following Example 3.6 and Example 3.9 as well.)

Example 3.6

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

(41) $\neg\text{NOT}(Animal, eat, Animal, a)$,
where $eat : Animal \to Prop$ *and* $a : Animal$*, with the latter interpreting 'the animal'.*

(42)
```
not (NOT Animal eat Animal a)
```

With (A_1), we have the following theorem. It shows a special case for P_A in Definition 3.1, where B and A are the same (both *Animal*).

$$\neg\text{NOT}(Animal, eat, Animal, a) \Leftrightarrow eat(a)$$

3.2.2 Examples for Law (A_2)

Law (A_2) is repeated here together with its Coq code:

(43) $\forall p, q : A \to Prop.(\forall x : A.p(x) \Rightarrow q(x)) \Rightarrow$
$\forall y : B.\text{NOT}(A, q, B, y) \Rightarrow \text{NOT}(A, p, B, y)$

(44)
```
forall(p q:A->Prop),(forall(x:A),(p x)->(q x))->
forall(y:B),(NOT A q B y)->(NOT A p B y).
```

Example 3.7

(45) If tables don't talk, then tables don't talk loudly.

(46) $\forall x : Human.talk(x) \Rightarrow talk\_loudly(x)$
$\forall y : Table.\text{NOT}(Human, talk, Table, y) \Rightarrow \text{NOT}(Human, talk\_loudly, Table, y)$

(47)
```
(y:Table)(NOT Human talk Table y)->(NOT Human talk_loudly Table y)
```

In this example, we are focusing on the explanation of the laws. We simply consider 'talk loudly' as a phrase that can be derived from 'talk'.

3.2.3 Examples for Law (A_3)

Law (A_3) is repeated here together with its Coq code:

(48) If $A \preceq B, \forall p : B \to Prop.\forall z : C.\text{NOT}(B, p, C, z) \Rightarrow \text{NOT}(A, p, C, z)$

(49)
```
Variable c:A->B. Coercion c:A>->B. Lemma l:injective(c).
forall (p:B->Prop)(z:C), NOT B p C z -> NOT A p C z.
```

We should pay more attention to the law (A_3), also (A_4) and (A_5). In these laws, we need to prove injectivity for coercion c. In the previous subsection, we have shown, besides identity, the projection π_1 is injective when the second parameter of a Σ-type is a predicate – this is a common coercion in NL semantics. For the sake of simplicity, we shall skip the injectivity proof in the rest of this section.

Example 3.8

(50) *If John is not a human, then John is not a man.*

(51) $\text{NOT}(Human, p_{Human}, Man, John) \Rightarrow \text{NOT}(Man, p_{Human}, Man, John)$

(52) `NOT Human (pr Human) Man John -> NOT Man (pr Man) Man John`

We can define Man as $Man = \Sigma(Human, male)$, where $male : Human \to Prop$. So, we have

$$\Sigma(Human, Male) \leq_{\pi_1} Human$$

where π_1 is injective because of proof irrelevance. So, $Man \preceq Human$ and (51) can be easily proved by (A_3).

Example 3.9

(53) *It is not the case that John does not work.*

(54) $work : Human \to Prop$,
$\neg\text{NOT}(Human, work, Man, John)$

(55) `not NOT Human work Man John`

By using (A_3) and (A_1), we can show.

$$work(John) \Rightarrow \neg\text{NOT}(Man, work, Man, John) \Rightarrow \neg\text{NOT}(Human, work, Man, John)$$

3.2.4 Examples for Law (A_4)

Law (A_4) is repeated here together with its Coq code:

(56) If $A \preceq B, \forall p : C \to Prop.(\forall y : B.\text{NOT}(C, p, B, y)) \Rightarrow \forall x : A.\text{NOT}(C, p, A, x)$

(57)
```
Variable c:A->B. Coercion A>->B. Lemma l:injective(c).
forall(p:C->Prop),(forall(y:B),NOT C p B y)->(forall (x:A),NOT C p A x)
```

Similarly, we have injective projection π_1 of Σ-type for coercion

$$\Sigma(Woman, Beautiful) \leq_{\pi_1} Woman$$

$$\Sigma(Table, red) \leq_{\pi_1} Table$$

Hence, we can give the following two examples

Example 3.10

(58) *If women are not men, beautiful women are not men either.*

(59) $\forall x : Woman.\text{NOT}(Man, p_{Man}, Woman, x) \Rightarrow \forall y : BWoman.\text{NOT}(Man, p_{Man}, BWoman, y)$, *where* $BWoman = \Sigma(Woman, Beautiful)$.

(60) `(x:Woman)(NOT Man (pr Man) Woman x)->(y:BWoman)(NOT Man (pr Man) BWoman y)`

In Coq, we use record to represent Σ-type, `BWoman` in (60) for beautiful woman could be formally defined as:

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

Example 3.11

(61) *If tables do not talk, then red tables do not talk.*

(62) $\forall x : Table.\text{NOT}(Human, talk, Table, x) \Rightarrow \forall y : RTable.\text{NOT}(Human, talk, RTable, y)$, *where* $RTable = \Sigma(Table, red)$.

(63) `(x:table)(NOT Human talk Table x)-> (y:Redtable)(NOT Human talk Redtable y)`

Similarly, in (62) we use Σ-type for red table, `Redtable` in (63) is defined with record as:

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

3.2.5 Examples for Law (A_5)

Law (A_5) is repeated here together with its Coq code:

(64) If $A \preceq B, \forall p : C \rightarrow Prop.(\exists x : A.\textsc{not}(C, p, A, x)) \Rightarrow \exists y : B.\textsc{not}(C, p, B, y)$

(65)
```
Variable c:A->B. Coercion A>->B. Lemma l:injective(c).
forall (p:C->Prop),(exists x:A,NOT C p A x )->(exists y:B,NOT C p B y)
```

With De Morgen Law in classical logic, $\exists x : A.\textsc{not}(C, p, A, x)$ is equivalent to $\neg\forall x : A.\neg\textsc{not}(C, p, A, x)$. It means that if there's $x : A$ such that 'x does not p', then it is not the case that forall $x : A$ such that 'x does p'. Moreover, we have the following two examples:

Example 3.12

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

(67) $\neg\forall l : Linguist.(\neg\textsc{not}(Logician, p_{Logician}, Linguist, l)) \Rightarrow$
$\neg\forall l : Human.(\neg\textsc{not}(Logician, p_{Logician}, Human, l))$

(68)
```
not(forall l:Linguist, PR Logician l)->not(forall h:Human, PR Logician h)
```

By definition 3.1 and type inference, `PR Logician l` in (68) is $P_{Logician,Linguist}(l)$ which equals to $\textsc{not}(Logician, p_{Logician}, Linguist, l)$ with $l : Linguist$. Similarly, `PR Logician h` is $P_{Logician,Human}(h)$ which equals to $\textsc{not}(Logician, p_{Logician}, Human, h)$ with $h : Human$.

Example 3.13

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

(70) $\neg\forall l : Man.(\neg\textsc{not}(Human, work, Man, l)) \Rightarrow$
$\neg\forall l : Human.(\neg\textsc{not}(Human, work, Human, l))$.

(71)
```
not(forall l:Man,not(NOT Human work Man l)->
not(forall l:Human,not(NOT Human work Human l)
```

With the subtype relation $Man \leq Human$ and law (A_5), (69) can also be easily described and reasoned.

4 Justification of NOT

We have introduced a negation operator NOT with laws for negative sentences or conditionals, and shown several examples corresponding to each law. However, from a logical aspect, we still need to show that the operator is provided in a reasonable way. In another word, we need to show whether the introduction of such negation operator can be justified. For example, is the extension by NOT logically consistent? Such a justification has not been provided for $\textsc{not}_{old}$ proposed in [6]. In this section, we present one method to justify NOT, the justification is given by means of the heterogenous equality JMeq, called John Major equality [14].

4.1 Heterogeneous equality JMeq

In type theory, equality propositions such as the Leibniz equality are usually considered only between objects of the same type. For examples, if $a : A$, $b : B$, A and B are different types, we usually cannot talk about whether a and b are equal. However a heterogenous equality allows us to talk about equality on arguments of different types.

JMeq (named as "John Major Equality"), proposed by Conor McBride [14], is a heterogenous equality, which allows us to apply equality on arguments of different types.

$$\text{JMeq} : \ \Pi A : Type.\Pi x : A.\Pi B : Type.\Pi y : B.Prop$$

We have the following rules for JMeq

$$\frac{\Gamma \vdash A : Type \quad \Gamma \vdash x : A}{\Gamma \vdash \text{JMeq } A\ x\ A\ x : Prop}$$

Axiom 1 *For all $A : Type$, if $x, y : A$ and* JMeq $A\ x\ A\ y$*, then $x = y : A$.*

It is trivial to prove the following lemma for symmetry and transitivity properties of JMeq, hence it is a equivalent relation.

Lemma 4.1

1. *(Symmetry) For all* $A, B : Type$*,* $x : A$*,* $y : B$*,* JMeq $A\ x\ B\ y$*, the* JMeq $B\ y\ A\ x$*.*
2. *(Transitivity) For all* $A, B : Type$*,* $x : A$*,* $y : B$*,* $z : C$*,* JMeq $A\ x\ B\ y$*,* JMeq $B\ y\ C\ z$*, the* JMeq $A\ x\ C\ z$*.*

With JMeq, we can form proposition $\mathrm{JMeq}(A, a, B, b)$ to describe the equality between a and b, even if A and B are different. In Coq standard library, JMeq is defined as:

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

4.2 Justification of not by JMeq

With the identifier JMeq, we can define NOT as follows:

$$\textsc{not}(A, p, B, b) = \forall x : A.\mathrm{JMeq}(A, x, B, b) \Rightarrow \neg p(x) \quad (**)$$

Informally, it says that 'b does not p' means that, for any x in A, if x equals b, then $p(x)$ is not true.

The extension of the underlying MTT (e.g., UTT) with JMeq was proved to be consistent [14]. If the laws (A_i) (i = 1 to 5) can be proved (and done in the proof assistant Coq) with NOT defined as in (**), the consistency of extending MTT with JMeq will imply that our extension with NOT is logically consistent as well.

To justify NOT by JMeq, we need to prove (A_1)-(A_5) with the notion of JMeq. However, to prove (A_3) (A_4) and (A_5) with JMeq, we need to employ a further axiom with the notion of injectivity and coercion.

Axiom 2 $\forall A, B : \textsc{cn}$*,* $A \preceq B$*, then we have*

$$\forall x : A, \mathrm{JMeq}(A, x, B, x)$$

Remark 3 $\forall x : A.\mathrm{JMeq}(A, x, B, x)$ *actually stands for* $\forall x : A.\mathrm{JMeq}(A, x, B, c(x))$ *with some injective coercion c. Then, for any* $x_1, x_2 : A$*, if* $c(x_1) = c(x_2)$*, we have* $\mathrm{JMeq}(B, c(x_1), B, c(x_2))$*. Hence we can derive that* $\mathrm{JMeq}(A, x_1, A, x_2)$ *with symmetry and transitivity rules of JMeq, which gives us* $x_1 = x_2$ *in JMeq. This matches the injectivity of c.*

Theorem 4.2 *For any* $A, B, C : \textsc{cn}$*, if* NOT *is the notion for* $\textsc{not}(A, p, B, b) = \forall x : A.\mathrm{JMeq}(A, x, B, b) \Rightarrow \neg p(x)$*, then we can prove the following (the laws for* NOT*):*

1. $\forall p : A \to Prop.\forall x : A.\neg\textsc{not}(A, p, A, x) \Leftrightarrow p(x)$
2. $\forall p, q : A \to Prop.(\forall x : A.p(x) \Rightarrow q(x)) \Rightarrow \forall y : B.\textsc{not}(A, q, B, y) \Rightarrow \textsc{not}(A, p, B, y)$
3. *if* $A \preceq B$*,* $\forall p : B \to Prop.\forall z : C.\textsc{not}(B, p, C, z) \Rightarrow \textsc{not}(A, p, C, z)$
4. *if* $A \preceq B$*,* $\forall p : C \to Prop.(\forall y : B.\textsc{not}(C, p, B, y)) \Rightarrow \forall x : A.\textsc{not}(C, p, A, x)$
5. *if* $A \preceq B$*,* $\forall p : C \to Prop.(\exists x : A.\textsc{not}(C, p, A, x)) \Rightarrow \exists y : B.\textsc{not}(C, p, B, y)$

Proof With the definition of NOT and properties of JMeq, we can prove the theorem with no difficulty. The proofs have been done in the Coq proof assistant as well – see Appendix A for their Coq statements (the proof codes are omitted to save the space.)

Acknowledgement. We are very much grateful to many people who have had helpful discussions with us on related topics including Nicholas Asher, Koji Mineshima, Glyn Morrill and Christian Retorè, among other.

References

[1] D. Bekki. Representing anaphora with dependent types. *LACL 2014, LNCS 8535*, 2014.

[2] S. Chatzikyriakidis and Z. Luo. Adjectives in a modern type-theoretical setting. In G. Morrill and J.M Nederhof, editors, *Proceedings of Formal Grammar 2013. LNCS 8036*, pages 159–174. Springer, 2013.

[3] S. Chatzikyriakidis and Z. Luo. Natural language reasoning in Coq. *J. of Logic, Lang. and Info.*, 23, 2014.

[4] S. Chatzikyriakidis and Z. Luo. Individuation criteria, dot-types and copredication: A view from modern type theories. *Proc of the 14th Inter. Conf. on Mathematics of Language, Chicago*, 23, 2015.

[5] S. Chatzikyriakidis and Z. Luo. Adjectival and adverbial modifications: The view from modern type theories. *Journal of Logic, Language and Information*, 26(1), 2017.

[6] S. Chatzikyriakidis and Z. Luo. On the interpretation of common nouns: Types v.s. predicates. In *Modern Perspectives in Type-Theoretical Semantics*. Springer, 2017.

[7] S. Chatzikyriakidis and Z. Luo. *Formal Semantics in Modern Type Theories*. Wiley & ISTE, 2018. (to appear).

[8] Z. Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(1):105–130, 1999.

[9] Z. Luo. Type-theoretical semantics with coercive subtyping. *SALT20, Vancouver*, 2010.

[10] Z. Luo. Common nouns as types. In D. Bechet and A. Dikovsky, editors, *Logical Aspects of Computational Linguistics (LACL'2012). LNCS 7351*, 2012.

[11] Z. Luo. Formal semantics in modern type theories with coercive subtyping. *Ling & Phil*, 35(6), 2012.

[12] Z. Luo. 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*, pages 177–188, 2014.

[13] Z. Luo, S. Soloviev, and T. Xue. Coercive subtyping: theory and implementation. *Information and Computation*, 223:18–42, 2012.

[14] C. McBride. Elimination with a motive. In P. Callaghan, Z. Luo, J. McKinna, and R. Pollack, editors, *Types for Proofs and Programs, Proc. of TYPES 2000, LNCS 2277*. Springer, 2002.

[15] A. Ranta. *Type-Theoretical Grammar*. Oxford University Press, 1994.

[16] C. Retoré. The Montagovian Generative Lexicon Ty_n: a Type Theoretical Framework for Natural Language Semantics. *Proofs and Programs (TYPES 2013), LIPIcs(26)*, 2014.

[17] T. Xue. *Theory and Implementation of Coercive Subtyping*. PhD thesis, Royal Holloway, Univ of London, 2013.

A Justification of NOT by JMeq: Coq Proofs in Brief

Here are the Coq statements of the laws (A_1) to (A_5), which have all been proved in Coq.

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

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

(* A1: if x:A & p:A->Prop, NOT(A,p,A,x) iff p(x) *)
Definition A1 := forall (A:CN)(x:A)(p:A->Prop), not (NOT A p A x) <-> (p x).
(* A2: if p=>q then NOT(q,b) => NOT(p,b) *)
Definition A2 :=
  forall (A B:CN)(p q:A->Prop),
  (forall (x:A), (p x)->(q x)) -> forall (y:B), (NOT A q B y) -> (NOT A p B y).
(* injectivity: A functional operation f:(X)Y is injective if for all x,y:X, f(x)=f(y) implies x=y*)
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 ).
(* A3: If A <_c B, where c is injective, then
          forall p:B->Prop. forall z:C. NOT(B; p; C; z)=>NOT(A; p; C; z). *)
Definition A3 := Inj cAB ->  (forall (p:B->Prop)(c:C), NOT B p C c->NOT A p C c).
(* (A4) For A,B,C : CN, A <=_c B with Inj(c) and p : C->Prop, we have
         (forall y:B. NOT(C,p,B,y)) => forall x:A. NOT(C,p,A,x) *)
Definition A4 := Inj cAB->forall(p:C->Prop),(forall(y:B),NOT C p B y)->(forall(x:A),NOT C p A x).

(* (A5) For A,B,C : CN, A <=_c B with p : C->Prop, we have
         (Exsits x:A. NOT(C,p,A,x)) => Exists y:B. NOT(C,p,B,y) *)
Definition A5 :=Inj cAB->forall(p:C->Prop),(exists x:A,NOT C p A x)->(exists y:B,NOT C p B y).
```