Proof Assistants for Natural Language Semantics

Stergios Chatzikyriakidis[1⋆] and Zhaohui Luo[2⋆⋆]

[1] Dept of Philosophy, Linguistics and Theory of Science
University of Gothenburg
stergios.chatzikyriakidis@gu.se,
[2] Dept of Computer Science, Royal Holloway, Univ of London
zhaohui@hotmail.ac.uk

Abstract. In this paper we discuss the use of interactive theorem provers (also called proof assistants) in the study of natural language semantics. It is shown that these provide useful platforms for NL semantics and reasoning on the one hand, and allow experiments to be performed on various frameworks and new theories, on the other. In particular, we show how to use Coq, a prominent type theory based proof assistant, to encode type theoretical semantics of various NL phenomena. In this respect, we can encode the NL semantics based on type theory for quantifiers, adjectives, common nouns, and tense, among others, and it is shown that Coq is a powerful engine for checking the formal validity of these accounts as well as a powerful reasoner about the implemented semantics. We further show some toy semantic grammars for formal semantic systems, like the Montagovian Generative Lexicon, Type Theory with Records and neo-Davidsonian semantics. It is also explained that experiments on new theories can be done as well, testing their validity and usefulness. Our aim is to show the importance of using proof assistants as useful tools in natural language reasoning and verification and argue for their wider application in the field.

Keywords: Type Theory · Proof Assistants · Reasoning · Formal Semantics · Coq

1 Introduction

Interactive theorem provers (also called proof assistants) have come a long way since they were first introduced in the late 60's as tools to formalise mathematics (cf., the AUTOMATH project [3]). Today, a number of state-of-the-art proof assistants exist and their uses have been proven fruitful both in formalisation of mathematics and software verification, among other things; see, for example,

⋆ Partially supported by Centre for Linguistic Theory and Studies in Probability, University of Gothenburg.
⋆⋆ Partially supported by EU COST Action CA15123 and CAS/SAFEA Inter. Partnership Program.

[13] for the proof of the four colour theorem in the proof assistant Coq[3]. The importance and usefulness of proof assistants have also been further proven by some recent research projects, including the very attractive research on Univalent Foundations [28] that aims to develop alternative foundations of mathematics, where the proof assistants Coq and Agda [1] play a crucial role to the whole endeavour (see [34] for an example of formalization of part of the project in Coq).

The use of constructive type theories for the study of NL semantics has also seen a revival in the last decade.[4] A number of approaches that directly employ constructive type theories or are inspired by them have been put forth by various researchers in the recent years and have provided interesting accounts on classic problems of formal semantics (see [9, 23, 26, 30, 16, 2, 33] for examples, although this is not a complete list). In this context, it is worth noting the following:

- Some of the proof assistants, like Coq and Agda, implement constructive type theories;
- The proof assistants are extremely powerful reasoning engines; and
- Constructive type theories have been shown to be a nice alternative to the simple type theory usually in formal semantics.

It seems that the time is right to look at the combination of these three in order to use proof assistants as natural language reasoners and as checkers of the formal validity of formal semantics accounts. Indeed, we have taken the first step in this direction and have used Coq as a natural language reasoner [6, 5]. In this paper, we extend this work and create a number of small Coq libraries to show that proof assistant like Coq can provide useful platforms for:

- Formalising NL semantics and, based on it, formally describe various NL phenomena, including co-predication, individuation, common nouns, adjectives and tense, among others. (These libraries are based on earlier theoretical work using Luo's Type Theory with Coercive Subtyping (TTCS for short) [20, 21, 23].)
- Experimenting with various semantic frameworks: we show how to use Coq to formalise them by implementing some small examples in Rétore's Montagovian Generative Lexicon [30], Cooper's Type Theory with Records (TTR) [9], and neo-Davidsonian event semantics [27].
- Experimenting with new theories: we formalise in Coq a newly developed theory [8] of predicational forms to give semantics to negative sentences and conditionals in constructive type theory. We also look at the issue of individuation and its interaction with copredication from the same perspective.

[3] The proof assistant Coq implements a constructive type theory in the tradition of Martin-Löf. The type theory is an impredicative type theory called the Calculus of Inductive Constructions (pCIC) [11], which is similar to the type theory UTT (or TTCS as called in this paper) [18].

[4] The use of constructive type theories has been initiated by the pioneering work of Aarne Ranta [29].

The current paper is structured as follows: in §2, we provide an introduction to TTCS and the implementation of some of the ideas casted in TTCS with respect to NL semantics in the Coq proof assistant, especially its use in formalising NL semantics in TTCS. In §3, we present several small libraries: first the one based on our work in type theory, introducing the relevant formal features of TTCS when needed, then several small libraries for other semantic frameworks and, finally, the library for the theory of predicational forms and individuation criteria. In the conclusion, some future work is discussed.

2 Type Theoretical Semantics for NL in Coq

In this section, we shall first introduce formal semantics in a constructive type theory and then how we will discuss the use of Coq to implement the semantics for various features in natural language.

2.1 Formal Semantics in Type Theory with Coercive Subtyping

Type Theory with Coercive Subtyping (TTCS) is a constructive type theory based on Luo's UTT [18] with the addition of an effective subtyping mechanism, that of coercive subtyping [19, 26]. TTCS has been effectively used in the study of NL semantics for a range of phenomena including common nouns, adjectives, adverbs and belief intensionality among other things [20, 21, 23, 5, 7]. TTCS is a dependent type theory with rich type structures which are exploited for the study of NL semantics. We will refer to this type of semantics in this paper as Modern Type Theoretical (MTT) semantics.[5] In MTT-semantics, some of the major linguistic categories and their interpretation are shown below:

1. A common noun (CN) can be interpreted as a type.
2. A verb (IV) can be interpreted as a predicate over the type D that interprets the domain of the verb (i.e., a function of type $D \to Prop$, where $Prop$ is the type of logical propositions
3. An adjective (ADJ) can be interpreted as a predicate over the type that interprets the domain of the adjective (i.e., a function of type $D \to Prop$).
4. Modified common nouns (MCNs) can be interpreted by means of Σ-types, types of (dependent) pairs.
5. A sentence (S) is interpreted as a proposition of type $Prop$.

See Figure 1 for a summary with examples.

[5] The formal semantics based on Modern Type Theories such as Martin-Löf's type theory or TTCS is usually called MTT-semantics. In the current paper, we shall still talk about MTT-semantics although, if taken seriously, it means formal semantics in TTCS because the Coq implementation of the NL semantics is based on TTCS.

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

Fig. 1. Examples in formal semantics.

2.2 NL Semantics in Coq

Coq [11] implements pCIC, a type theory whose major part is essentially[6] TTCS (UTT with coercive subtyping), based on which the formal semantics briefly described in the previous subsection has been implemented. The encoding of NL semantics based on TTCS is quite straightforward in most of the cases. Let us see some basics of how this can be done.

Starting with the type of logical propositions, nothing needs to be encoded, since Coq already involves a universe of logical propositions, *Prop*. The next step, is to see what the universe of entities would be taken to be. In MG, a coarse-grained type of entities exists, i.e. the type e of all entities. In MTT-semantics, the common nouns constitute a universe, denoted as CN; the type CN contains the (interpretations of) CNs, each of which is further interpreted as a type that contains entities belonging to them. CNs are interpreted as types rather than predicates. However, since universe construction (i.e., defining new universes) is not an option in Coq, we equate CN with Coq's predefined universe *Set*.

Σ-types (types of dependent pairs), which are used to give semantics to some modified common nouns among other things, are encoded using Coq's dependent record type mechanism[7] and adjectives and verbs are defined as predicates (objects of type $A \to Prop$). Subsective adjectives like *large* are encoded as polymorphic predicates (see [4]), extending over the universe CN.[8] Subtyping is encoded using Coq's coercion mechanism and the proper names are given suitable domain types: e.g., *John* is assumed to be of type Man.

The Coq codes for this basic set up are as follows.

```
Definition CN := Set.
Parameters Man Woman Human Animal Object : CN.
Axiom mh : Man->Human.   Coercion mh : Man >-> Human.
Axiom wh : Woman->Human. Coercion wh : Woman >-> Human.
Axiom ha : Human-> Animal. Coercion ha : Human>->Animal.
Axiom ao : Animal->Object. Coercion ao : Animal>->Object.
```

[6] Coq has co-inductive types which are not present in TTCS.

[7] Coq's record types are just Σ-types with global names associated with them.

[8] This is encoded using Π-types as follows: $[\![Adj_{subs}]\!] : \Pi A : \text{CN}.\ A \to Prop$. The 'forall' part in the code corresponds to Π.

```
Parameter Black : Object->Prop.
Parameter Large : forall A:CN, A->Prop.
Parameter walked: Human->Prop.
Parameter John : Man.
```

Quantifiers can be given polymorphic types as well: a quantifier takes a CN argument A : CN and returns a function of type $(A \to Prop) \to Prop$. Thus, if A is Man the type for the quantified NP will be $(Man \to Prop) \to Prop$ and, if A is $Object$, it is of type $(Object \to Prop) \to Prop$, and so on. As examples, we define the quantifiers *some, all, no* as follows:

```
Definition some := fun A:CN => fun P:A->Prop => exists x:A, P(x).
Definition all  := fun A:CN => fun P:A->Prop => forall x:A, P(x).
Definition no   := fun A:CN => fun P:A->Prop => forall x:A, not(P(x)).
```

Note that the typing is the one we have been describing, taking an A:CN argument, an $A \to Prop$ argument and returning a proposition.

Now, let us see how one can exploit Coq in order to reason with NL sentences based on the implemented semantics. First of all, if one wants to check typing, the command *Check* followed by the element we want to check can be used. Note that Coq is a strongly typed language, so by definition ill-typed constructs cannot be defined, since they will be blocked by Coq. Let us see an NL reasoning example, the one shown below:

(1) John walked ⇒ Some man walked

Formalizing this example in Coq, we consider the following 'theorem' whose name is `JOHN` (to be proved):

```
Theorem JOHN : walked John -> (some Man) walked.
```

This will put Coq into proof-mode. We unfold the definition for *some* using cbv and use the tactic *intro*, which will introduce the antecedent as a hypothesis:[9]

```
JOHN < cbv. intro. subgoal
H : walked John
============================
exists x : Man, walked x
```

What we need to do is substitute John for x and using the tactic assumption, which matches a goal in case there is an identical premise in the context of the proof, the proof is completed and we can save the proof using *Qed*. The whole proof then consists of the steps:

1. cbv (unfolding definitions (in our case the one for *some*))[10]
2. intro (moving the antecedent as a hypothesis)

[9] The tactic *cbv* performs all possible reductions.

[10] In general the tactic *cbv* performs all possible reductions. For more information, see [11].

3. exists John (substituting x for John)
4. assumption (matching the goal with a hypothesis)

Remark 1. The MTT-semantics has proved to be a viable alternative to Montague Grammar, with several notable advantages. Here, we think it is worth mentioning one of them: that is, MTT-semantics is both model-theoretic and proof-theoretic, as argued in [24]. It is model-theoretic because, in an MTT-semantics, an MTT is employed as a representational language and it can do so because of its rich representational structures as well as its internal logic. Therefore, it has a wide coverage of linguistic features and can be compared to Montague semantics in this respect. It is also proof-theoretic, in the sense of [14], because MTTs are specified proof-theoretically and the meanings of MTT-judgements, that are used to give semantics to NL sentences, can be understood by means of their inferential roles. Therefore, reasoning with NL can be directly performed in proof assistants like Coq that implement MTTs. This is unique for MTTs and MTT-semantics: such a possibility of having a semantics which is both model-theoretic and proof-theoretic is not available to us until we have the MTT-semantics (for example, if one considers the traditional model-theoretic semantics in set theory, we simple would not have a proof-theoretic representational language: set theory is not proof-theoretic.)

3 Libraries for NL Semantics

We have created a number of small libraries in Coq, encoding NL semantics. They may be classified as follows:

- *MTT-semantics and reasoning*: We have studied various NL phenomena using MTT-semantics and formalised them in Coq.
- *Platform for other semantic frameworks*: We have looked at several semantic frameworks and provided some examples including, for example, Rétore's Montagovian Generative Lexicon [30], Cooper's Type Theory with Records (TTR) [9], and a toy semantic grammar for neo-Davidsonian event semantics [27].
- *Experiments on new semantic theories*: We have done interesting experiments in Coq about some new semantic theories, including that about predicational forms in MTT-semantics [8], as reported here.

The libraries can be found at *https://github.com/StergiosCha/CoqLACL*.

3.1 MTT semantics for NL in Coq

The main file for MTT-semantics is MainCoq.v. This includes the Coq implementation of a number of ideas in MTT-semantics. The universe CN includes a number of types (e.g., $Man, Human, Delegate, Woman, Animal, Object$) and subtyping relations between them. Synonym relations are encoded via the *let*-command in Coq. Adjectives are defined in the way specified in the previous

section and sometimes some added lexical semantics are inserted. For example, *small* is defined as the opposite of *large*, and both are polymorphically defined as follows:

```
Parameter Large Normalsized: forall A:CN, A->Prop.
Definition Small :=
  fun A:CN => fun a:A => not (Large A a) /\ not (Normalsized A a).
```

Basically the idea here is that small is defined as being not large but furthermore not of normal size. This reflects the idea that something which is not large is not necessarily small.[11] This is needed in order to get the relevant inferences right (see [5]).

In MTT-semantics, there is also a widespread use of Σ-types for factive verbs, adverbs and comparatives. We have not the space here to go in full detail but the idea can be briefly described as follows, taking the case of veridical sentence adverbs as an example. What we need to capture is that the proposition without the adverb is implied by the proposition including the adverb. In order to do this, we first define an auxiliary object:

```
Parameter ADVS : forall (v:Prop), sigT (fun p:Prop => p->v).
```

This basically takes a proposition v and returns a pair whose first component is a proposition p and whose second component is the proposition that p implies v. Then, veridical sentence adverbs (we use *fortunately* as an example) are defined as the first projection of this auxiliary pair:

```
Definition fortunately := fun v:Prop => projT1 (ADVS v).
```

Similar uses of Σ-types can be found for VP adverbs, comparatives as well as factive verbs in the library (see [5] for more details.)

For comparatives, we introduce indexed types for common nouns; for example, humans of type $Human$ may be indexed by a height parameter. Then, a comparative adjective takes two $Human_i$ arguments with $i::Height$.

```
Inductive HUMAN : nat->Type := HUMAN1:forall n:nat,HUMAN n.
```

A simple model of tense is defined and an attempt to deal with some aspects of tense exists. There is a type $Time$ and a date is defined as triple, taking year, month and day arguments and returning a result in $Time$. A default date is defined which consists of the defaults for year, month and day. Then, verbs are defined with an extra time argument. Present, past and future are then defined using the *precedes* relation with respect to the default time. For example, an adverb like currently is defined as identifying the time argument with the default time:

```
Definition currently := fun P : Time -> Prop => P default_t.
```

[11] The level of fine-grainedness with respect to size, i.e. whether sizes between these proposed three will be used, will not bother us here.

The next file is adjectives.v, which involves some more fine-grained issues in adjectival semantics. In particular it deals with multidimensional adjectives and introduces a hack in order to take care of the fact that Coq does not allow subtyping to propagate through constructors (as it is the case in TTCS).[12] *Multidimensional adjectives* do not just involve one dimension (e.g., the dimension of height in the case of *tall*), but more than one. Classical cases are the adjectives like *healthy* and *sick* or even adjectives like *big*. The idea is that an adjective like *healthy* quantifies over a number of dimensions, e.g., blood pressure, cholesterol etc. [32]. Similarly, *big* may involve different dimensions like *height*, *width* etc. For an adjective like healthy, we define health as an enumerated type including all the relevant dimensions. Then, *Healthy* is defined as taking an argument of type *Human* and assuming that this human is healthy in all dimensions. For sick, the assumption is that the argument is not healthy w.r.t. to at least one dimension. This follows the ideas set out in [32]:

```
Inductive Health:CN:=Heart|Blood|Cholesterol.
Parameter Degree:R. Parameter healthy:Health->Human->Prop.
Definition Sick:=fun y:Human=>~(forall x:Health,healthy x y).
Definition Healthy:=fun y:Human=>forall x:Health,healthy x y.
```

The files FracasCoq.v and test.v are meant to be used in conjunction. Actually FracasCoq loads test.v. FracasCoq.v contains a number of FraCaS test suite examples formalized in Coq along with their proofs. The FraCaS Test Suite [10] arose out of the FraCaS Consortium, a huge collaboration with the aim to develop a range of resources related to computational semantics. The FraCaS test suite is specifically designed to reflect what an adequate theory of NL inference should be able to capture. It comprises NLI examples formulated in the form of a premise (or premises) followed by a question and an answer. Here is a typical example from the suite:

(2) Some Irish delegates finished the survey on time.
Did any delegate finish the report on time [Yes, FraCaS 055]

The modified CN Irish delegates is defined as a Σ type. Given that π_1 is defined as a coercion, the inference will go through easily. Please see [5] for more details and the code for the actual

3.2 Other Semantic Frameworks

Proof assistants can be used as platforms to experiment with different semantic frameworks. In this respect, there are three files that have some very small toy

[12] Some remark on subtyping propagation in Coq is needed. If $A < B$, then we should have $\Sigma(A, C) < \Sigma(B, C)$ (which follows in TTCS). But this does not follow in Coq. In order to remedy this we have introduced a sort of a hack by overloading the type using unit types (see the actual code and consult [21] for the use of unit types).

semantic grammars of other frameworks that have been used in the study of linguistic semantics. Note that these implementations are shallow implementations in the sense that no deep implementation of the underlying formal systems is done. In other words, we are not doing a faithful implementation of a semantic framework; instead, we emphasise the quick return so that examples can be done. For instance, Retoré's Generative Montagovian Lexicon [30] is based on system F [12, 31], but no implementation of system F is done on our part.

In MontagovianLexiconToy.v, we encode some of the ideas in presented in Generative Montagovian Lexicon as presented in [30]. Note that the idea that, representing the interpretation of a common noun, each type has its corresponding predicate cannot be implemented since it is not clear how such correspondence will be formally defined.[13] We, however, encode the idea that a word like *book* has a principal lambda term and then a number of coercions that take care of its dot-type status. This is done by using type overloading via unit types. We further formalize the polymorphic conjunction of [30] and prove that it is equivalent to the semantics of regular conjunction. For example, the definition of polymorphic conjunction is given as follows:

```
Definition PAND := fun a:e => fun b:e => fun P:a->t => fun Q:b->t =>
                   fun x:e => fun y:x => fun f:x->a => fun g:x->b =>
                   and (P(f(y))) (Q(g(y))).
```

Records.v has some very simple experimentations on encoding ideas from Cooper's TTR [9]. For example, the record for *a man owns a donkey* is encoded as:

```
Record amanownsadonkey : Type :=
    mkamanownsadonkey{ x : Ind;
                       c1 : man x;
                       y  : Ind;
                       c2 : donkey y;
                       c3 : own x y}.
```

From this record type in Coq, one can prove any of the individual fields. For example, one can show that a man exists, that a donkey exists (*man* and *donkey* are defined here as predicates), and that the man owns the donkey.

Lastly, Davidson.v contains a typed neo-Davidsonian toy semantic grammar. It has some simple examples and the welcoming inferential properties of neo-Davidsonian semantics where each modifier adds a conjunct. The grammar presents a typed version of neo-Davidsonian semantics[14]. Similarly, a transitive

[13] For example, one can define both a type book and a predicate book* but linking the two and defining such a process for every common noun is something that we do not know how can be done, without leading to formal difficulties such as undecidability of type-checking [8]. There is not a formal proposal on how to do this in [30] either.

[14] See [25] for a theory of dependent event types which extends Church's simple type theory with dependent event types. This is an initial step towards a theory of events with dependent types.

verb like *stabs* is defined as taking an event argument e and two arguments x and y of type Ind and returning a proposition which specifies that there is a stabbing event e1 such that $stabs(x)(y)(e1)$, x is the agent, y is the theme and $e = e1$. This toy semantic grammar can take care of inferences like the following (proofs are in the file):

(3) Brutus stabbed Caesar with a knife in Rome ⇒ Brutus stabbed Caesar with a knife

(4) Brutus stabbed Caesar with a knife in Rome ⇒ Brutus stabbed Caesar

(5) Brutus stabbed Caesar with a knife in Rome ⇒ the agent of the stabbing was Brutus

Remark 2. As we have already mentioned, the above implementations are shallow implementations of fragments of other semantic theories.[15] Coq implements an MTT, which in itself is a very powerful language to represent NL semantics. In a sense, one way of using Coq would be to use this very powerful language in order to embed different semantic theories as kind of modules within Coq's MTT. For example, one might want to define a Natural Logic component (as for example [17] has done), or a neo-Davidsonian fragment as we have very briefly done here. We believe that this is a nice way of looking at how the systems like Coq can be used for NL semantics. Different comparisons can then be performed as regards the different frameworks based e.g. on the predictions they make as regards inference.

3.3 Experiments with New Semantic Theories

Systems like Coq can play a useful role in verifying newly proposed theories in semantics. Here, we consider two cases. The first concerns the theory of predicational forms as studied in [8]. The theory is to deal with negated sentences or conditionals in a type theory where some CNs are interpreted as types in a multi-sorted type system (e.g., the MTT-semantics) and the file predhyp.v contains the experiments done in Coq that formalizes the theory of predicational forms and considered many relevant examples.

Consider the simplest example, where (7) is the (judgemental) interpretation of (6):

(6) John is a man.

(7) $j : Man$

Note that $j : Man$ is a judgment and not a proposition. How do we give semantics to its negation like (8)?

(8) John is not a man.

[15] See [15] for an informal explanation of shallow and deep embeddings.

Similarly, a negated sentence like (9) needs to be given semantics, but it would be simply negating the semantics of 'Tables talk' since the latter is meaningless (i.e., ill-typed)[16].

(9) Tables do not talk.

Also, some conditionals correspond to hypothetical judgements and require a treatment as well (we omit the details here).

The theory of predicational forms [8] is a logical theory to deal with the above issues. Based on it, suitable semantic interpretations can be given to negated sentences and conditionals as intended.

The formalisation of the theory (and examples) can be found in predhyp.v.[17] For instance (just showing one example), the following sentences and inferences have been done:

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

(11)It is not the case that every human is a logician

(12)Some red tables do not talk ⇒ Some tables do not talk

Another theory is to consider how to deal with inferences concerning CNs. Individuation.v contains an account of how individuation criteria should be decided within an MTT. The general idea is that every common noun is associated with its own identity criteria (IC) which can be inherited by other common nouns (see [22] for the theory on this and more detailed discussions on ICs.) For example, one can assume that Man inherits its IC from $Human$. Given this assumption, common nouns are not simple types but setoids whose first component is a type (the domain of the CN), in DomCN (which is the old CN universe) and whose second component is its IC. So under this view, the common noun Human will be represented by the following (we use capitals to denote the new formalization and retain the first letter with uppercase notation to denote the type in DomCN):

(13) $HUMAN = \Sigma(Human, =_H)$

Several IC criteria are defined for different common nouns and dot.types like book are given two different IC criteria depending on whether their physical or informational aspect is individuated. Thus, we have:

(14) $BOOK_1 = \Sigma(Book, =_P)$

(15) $BOOK_2 = \Sigma(Book, =_I)$

A number of proofs then follow including, for example, a proof of the following:

(16) John picked up and mastered three books ⇒
John picked up three physical objects and mastered three informational objects

[16] Note that it is not given false as in MG.

[17] The files FracasCoq.v and test.v are meant to be used in conjunction. Actually FracasCoq loads test.v. FracasCoq.v contains a number of FraCaS test suite examples formalized in Coq along with their proofs.

Remark 3. One issue that is worth mentioning here, is that of automation. Coq is an interactive theorem prover, which means that the user guides the prover to the proof. However, Coq has a very powerful tactic language that can be used in order to construct composite tactics that can automate part of or whole proofs. We have defined a number of tactics that can automate proofs. The interested reader can check for example the automated tactic AUTO in the files Davidson.v (for example BRUTUS1 to BRUTUS4 are proven using AUTO only) and MontagovianLexicon.v. AUTO can prove all theorems in these two files. A more advanced automatic tactic is needed for the proofs found in the FracasCoq.v file. Such a tactic is AUTOa (this tactic also solves all the goals in the previous files solved by AUTO) [6, 5]. All proofs can be automated with this tactic except one that is semiautomated (see FracasCoq.v file).

4 Conclusions and Future Work

In this paper, we have argued for the use of the proof assistant technology for natural language semantics. In particular, we have argued, that the time is mature for such an endeavor given the progress made in both the proof technology itself as well as the use of constructive type theories for natural language semantics. We have prepared a number of small libraries for NL semantics using the proof assistant Coq based on Luo's TTCS and have shown the benefits of such an endeavor by exemplifying the use of proof assistants as natural language reasoners or as checkers of the formal validity of proposals in formal semantics. We have lastly shown how experiments with semantic accounts proposed in several semantic frameworks can also be implemented in Coq.

As future work, we are envisaging the extension of work as regards inference by endorsing a system where a tight correspondence between syntax and semantics exists, in the same way such a correspondence is found in categorial grammar. This builds on theoretical work of second author, where a proposal for extending the Lambek calculus with dependent types can be found. Given such a development one can then define a parser based on this extended Lambek calculus with dependent types, which will automatically give us MTT-semantics as output. These semantics will then be used by Coq to perform reasoning tasks. The ultimate goal is to develop a wide-coverage, robust parser that will then be able to output semantics for larger pieces as well as open text. Similar work using multi-modal categorial grammars or combinatory categorial grammar has been shown to be feasible. If this is the case, this is a great chance of using a more structured semantic framework as well as a specific purpose reasoning device (Coq) in order to deal with NLI.

References

1. Agda proof assistant. `http://appserv.cs.chalmers.se/users/ulfn/wiki/agda.php` (2008)

2. Bekki, D.: Representing anaphora with dependent types. Logical Aspects of Computational Linguistics 2014, LNCS 8535 (2014)
3. de Bruijn, N.: A survey of the project AUTOMATH. In: Hindley, J., Seldin, J. (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press (1980)
4. Chatzikyriakidis, S., Luo, Z.: Adjectives in a modern type-theoretical setting. In: Morrill, G., Nederhof, J. (eds.) Proceedings of Formal Grammar 2013. LNCS 8036. pp. 159–174 (2013)
5. Chatzikyriakidis, S., Luo, Z.: Natural language inference in Coq. J. of Logic, Language and Information. 23(4), 441–480 (2014)
6. Chatzikyriakidis, S., Luo, Z.: Natural language reasoning using proof-assistant technology: Rich typing and beyond. In: Proceedings of EACL2014 (2014)
7. Chatzikyriakidis, S., Luo, Z.: Using signatures in type theory to represent situations. Logic and Engineering of Natural Language Semantics 11. Tokyo (2014)
8. Chatzikyriakidis, S., Luo, Z.: On the interpretation of common nouns: Types v.s. predicates. In: Chatzikyriakidis, S., Luo, Z. (eds.) Modern Perspectives in Type Theoretical Semantics. Studies of Linguistics and Philosophy, Springer (2016 (to appear))
9. Cooper, R.: Records and record types in semantic theory. J. Logic and Compututation 15(2) (2005)
10. Cooper, R., Ginzburg, J.: A compositional situation semantics for attitude reports. In: Selignmann, J., Westerstahl, D. (eds.) Logic, language and computation,. CSLI (1996)
11. The Coq Team: The Coq Proof Assistant Reference Manual (Version 8.1), INRIA (2007)
12. Girard, J.Y.: Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur. Ph.D. thesis, Université Paris VII (1972)
13. Gonthier, G.: A computer-checked proof of the Four Colour Theorem (2005), http://research.microsoft.com/~gonthier/4colproof.pdf
14. Kahle, R., Schroeder-Heister, P. (eds.): Proof-Theoretic Semantics. Special Issue of Synthese, 148(3) (2006)
15. Keller, C., Werner, B.: Importing HOL light into Coq. In: International Conference on Interactive Theorem Proving. LNCS, vol. 6172, pp. 307–322. Springer (2010)
16. Krahmer, E., Piwek, P.: Presupposition projection as proof construction. In: H. Bunt and R. Muskens (eds.) Computing Meaning. Studies in Linguistics and Philosophy 73 (1999)
17. Lungu, G.E., Luo, Z.: Monotonicity Reasoning in Formal Semantics Based on Modern Type Theories, LNCS, vol. 8538, pp. 138–148. Springer Berlin Heidelberg (2014)
18. Luo, Z.: Computation and Reasoning: A Type Theory for Computer Science. Oxford University Press (1994)
19. Luo, Z.: Coercive subtyping in type theory. CSL'96, LNCS'1258 (1997)
20. Luo, Z.: Type-theoretical semantics with coercive subtyping. Semantics and Linguistic Theory 20 (SALT20), Vancouver (2010)
21. Luo, Z.: Contextual analysis of word meanings in type-theoretical semantics. Logical Aspects of Computational Linguistics (LACL'2011). LNAI 6736 (2011)
22. Luo, Z.: Common nouns as types. In: LACL'2012, LNCS 7351 (2012)
23. Luo, Z.: Formal semantics in modern type theories with coercive subtyping. Linguistics and Philosophy 35(6), 491–513 (2012)

24. Luo, Z.: 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 (2014)
25. Luo, Z., Soloviev, S.: Dependent event types (abstract). LACL 2016 (2016)
26. Luo, Z., Soloviev, S., Xue, T.: Coercive subtyping: theory and implementation. Information and Computation 223, 18–42 (2012)
27. Parsons, T.: Events in the Semantics of English. MIT Press (1990)
28. Program, T.U.F.: Homotopy type theory: Univalent foundations of mathematics. Tech. rep., Institute for Advanced Study (2013)
29. Ranta, A.: Type-Theoretical Grammar. Oxford University Press (1994)
30. Retoré, C.: The montagovian generative lexicon Tyn: a type theoretical framework for natural language semantics. In: Matthes, R., Schubert, A. (eds.) 19th International Conference on Types for Proofs and Programs (TYPES 2013). vol. 26, pp. 202–229 (2013)
31. Reynolds, J.: Towards a theory of type structure. Lecture Notes in Computer Science 19 (1974)
32. Sassoon, G.: A typology of multidimensional adjectives. Journal of semantics 30 (3), 335–380 (2013)
33. Tanaka, R., Mineshima, K., Bekki, D.: Factivity and presupposition in dependent type semantics. In: Type Theories and Lexical Semantics Workshop (2015)
34. Voevodsky, V.: Experimental library of univalent formalization of mathematics. Mathematical Structures in Computer Science 25, 1278–1294 (2015)