A Type-Theoretical system for the FraCaS test suite: Grammatical Framework meets Coq

Jean-Philippe Bernardy
University of Gothenburg
jean-philippe.bernardy@gu.se

Stergios Chatzikyriakidis
University of Gothenburg
stergios.chatzikyriakidis@gu.se

Abstract

We present a type-theoretical framework for formal semantics, which leverages two existing well established tools: Grammatical Framework (GF) and Coq. The framework is the semantic equivalent of GF's resource grammars: every syntactic construction is mapped to a (compositional) semantics. Our tool thus extends the standard GF grammar with a formal semantic backbone. We evaluated our framework on 5 sections of the FraCaS test-suite (174 examples) and significantly improved on the state of the art by 14 percentage points, obtaining 83% accuracy. Our semantics is free software and available at this url: http://github.com/GU-CLASP/FraCoq

1 Intro

Roughly put, Natural Language Inference (NLI) is the task of determining whether a Natural Language (NL) hypothesis can be inferred from an NL premise. NLI is central within a theory of formal semantics for Natural Language (NL), given that humans do not only have the ability to understand infinitely many NL sentences, but can further reason about them. In effect, understanding a NL sentence amounts (among others) to knowing what can be inferred or not from such a sentence.

Natural Language Inference has been also central in the field of computational semantics. As Cooper et al. aptly put it 'inferential ability is not only a central manifestation of semantic competence but is in fact centrally constitutive of it' (Cooper et al., 1996). A number of test suites have been proposed throughout the years and different computational approaches have been put forth in order to deal with NLI. The most well-known platforms are the FraCaS test suite (Cooper et al., 1996), the Pascal Recognizing Textual Entailment tasks (RTE) (Dagan et al., 2006), and recently the Stanford Natural Language Inference platform (SNLI) (Bowman et al., 2015). At least until a few years ago, a bipartite classification of NLI accounts depending on whether a translation to an intermediate logical language was performed or not existed. On the one hand, one got logical approaches where such a translation, usually to first-order or even weaker logics like Natural Logic, is performed (Blackburn et al., 2006; Bos and Markert, 2005; Pulman, 2013; Mineshima et al., 2015) and on the other, approaches where this is not the case (e.g. bag of words approaches or other approaches using various machine learning techniques) (Romano et al., 2006; Glickmann et al., 2005; Hickl et al., 2005; MacCartney et al., 2008). Recently, Mineshima et al. (2015) have argued that using higher order logic can significantly increase accuracy of systems using logic, compared to those where weaker logics are used.

In this paper, we show how to construct a formal system for NLI that bridges the gap between two well-studied systems, the Grammatical Framework (GF) (Ranta, 2011) on the one hand, and the proof assistant Coq on the other. Concretely, we map the GF parse trees for NLI problems to Coq propositions. The NLI solutions then become Coq proofs. We thus obtain a complete system which leverages GF for parsing and Coq for reasoning. We evaluate against 5 sections of the FraCaS (almost half of it, 174

examples in total) and show that our approach can outperform the state of the art in logical approaches by 14 percentage points. The structure of the paper is as follows: in Section 2, we present the background to our approach, GF, Type Theoretical (TT) semantics and the proof assistant Coq. In Section 3, we describe the FraCoq system concentrating on the most important and linguistically relevant aspects of the system. In Section 4, we first briefly present the FraCaS test suite. Then, we evaluate our system against it and present the results. We further discuss the issue of proof automation. In Section 5 we conclude and further point to future research directions.

2 Background: GF, TT semantics and Coq

2.1 GF

In GF, abstract syntax is comprised of: a) a number of syntactic categories, and b) a number of syntactic construction functions, which provide the means to compose basic syntactic categories into more complex ones. For example, the constructor $AdjCN : AP \rightarrow CN \rightarrow CN$ expresses that one can append an adjectival phrase to a common noun and obtain a new common noun. Additionally GF comes with a library of mappings from abstract syntax to concrete natural language syntaxes. These mappings can be inverted by GF, thus offering parsers from natural text into abstract syntax. Yet in this project we skip the parsing phase and use the parse trees constructed by Ljunglöf and Siverbo (2011), thereby avoiding any syntactic ambiguity.

2.2 TT semantics

In the tradition of formal semantics, we interpret abstract syntax into logic. The type of logics that we are employing here have been dubbed as modern or rich type theories (MTTs) by researchers like Luo (2012); Chatzikyriakidis and Luo (2014); Cooper et al. (2015) and are constructive TTs within the tradition of Martin-Löf (1971); Martin-Löf (1984). There is a considerable body of work on doing semantics using MTTs going back at least to Sundholm (1989) and Ranta (1994). Recent approaches of this kind can be found in Luo (2012); Retoré (2013); Chatzikyriakidis and Luo (2014); Tanaka et al. (2015); Cooper et al. (2015); Bekki and Mineshima (2017); Chatzikyriakidis and Luo (2017); Cooper (2017); Grudzińska and Zawadowski (2017) among many others. Compared to simple type theories (e.g. Church's TT), MTTs have been claimed to offer a number of advantages. The most important of these are summarized below:

1. Type many-sortedness. This usually refers to the possibility of using a more structured domain for the monolithic domain of individuals (e) one finds in systems based on simple type-theory, e.g. Montague Semantics (Montague, 1973). Even though we do not make use of this feature here, it remains an attractive feature that brings fine-grainedness to semantic interpretations.

2. Dependent Typing. Here we mention two instances of dependent typing:

   (a) Dependent sum types, or Σ-types, often written $\sum_{x:A} B[x]$ and which have product types $A \times B$ as a special case when B does not depend on x.

   (b) Dependent product, Π-types, often written ($\prod_{x:A} B[x]$), and which have arrow-types $A \rightarrow B$ as a special case. Dependent Π are very useful because they generalise function types and universal quantification. Additionally they offer type polymorphism, a very useful feature for semantic interpretation (see for example our treatment of VP based on Π).

3. Proof-theoretical specification and support for effective reasoning. The latter is evident from the fact that the most powerful interactive theorem provers (proof assistants) are based on type-theory. That is, the machinery *par excellence* for reasoning with formal structures implement MTTs (for example, the proof assistants Coq and Agda).

We will not delve here into further details: a complete discussion of MTTs for formal semantics is out of the scope of this paper. We will instead introduce any relevant features in discussing the transition from GF trees to semantic representations in Coq in the next section.

2.3 Coq

Coq is a proof assistant based on the calculus of inductive constructions (CiC), which is itself a lambda calculus with dependent types. Coq is arguably one of the leading proof assistants. Indeed, it has famously been instrumental in formalising a proof of the four-color theorem (Gonthier, 2008), a proof of the odd order theorem (Gonthier et al., 2013), as well as developing CompCert, a formally verified compiler for C (Leroy, 2013). Faithful to our actual development, our semantics will be presented in Coq syntax. While the reader should turn to a proper textbook Bertot and Castéran (2013) for reference, the main two features that we use are Π types (in Coq syntax $\prod_{x:A} B[x]$ is written `forall (x:A), B` or (simply `A → B` when `B` does not depend on `x`) and record types, which generalise Σ-typed and are encoded as (trivial) inductive types with a single constructor. In particular the type $\sum_{x:A} B[x]$ is encoded as follows:

```
Inductive Sigma : Type := mkPair : forall (x:A) (y:B) → Sigma.
```

3 The FraCoq system

As already mentioned, we do not perform parsing, but use the existing treebank of Ljunglöf and Siverbo (2011). The bulk of the work is thus to take these trees to their type-theoretical counterparts.

3.1 From GF trees to semantic representations in Coq

The structure of our semantic representation is the following:

1. Every GF syntactic category C is mapped to a Coq Set, noted $[\![C]\!]$.
2. GF Functional types are mapped compositionally : $[\![A \to B]\!] = [\![A]\!] \to [\![B]\!]$
3. Every GF syntactic construction function $f : X$ is mapped to a function $[\![f]\!]$ such that $[\![f]\!] : [\![X]\!]$.
4. GF function applications are mapped compositionally: $[\![t(u)]\!] = [\![t]\!]([\![u]\!])$.

Theorem 1 (Semantics interpretations are type sound)**.** *For every applicative expression* e *in GF, if* $e : X$ *then* $[\![e]\!] : [\![X]\!]$.

Proof. By induction over the GF typing relation, resting on the fact that the GF type-system is a subset of that of Coq. □

We proceed and describe the details of the interpretation $[\![\cdot]\!]$ for several key syntactic constructions.

Sentences As customary, we interpret sentences as propositions: $[\![S]\!] = Prop$, or in Coq syntax:

```
Definition S := Prop.
```

Consequently, checking entailment can be done by checking logical entailment of the interpretations. Namely, to verify that P entails H, we prove the proposition $[\![P]\!] \to [\![H]\!]$.

Common nouns We represent common nouns as customary in typed formal semantics, namely as predicates over an abstract object type:

```
Parameter object : Set.
Definition CN := object→ Prop.
```

Another option would be to follow the common nouns as Types paradigm, an approach found in Ranta (1994); Luo (2012); Chatzikyriakidis and Luo (2017) among others. This would allow common nouns to be basic types and every common noun to be associated with a base type in a type theoretic CN universe. A subtyping relation could then be used to encode semantic relations between the types. Indeed, this is the approach taken in Chatzikyriakidis and Luo (2014) in dealing with inference in Coq. However, for the current task, the need for subtyping is not crucial. In contrast it is needed to test whether an object belongs to a CN as a proposition, which is tricky to encode compositionally we we simply have $[\![CN]\!] = Type$, so a fully-fledged universe would be required.[1]

Verb phrases Like common nouns, verb phrases are traditionally represented as predicates over the subject. Here, we additionally parameterize over the *noun* of the subject (using Π types). This allows in particular for comparative copulas to know which class they refer to, which is often needed in FraCas.

```
Definition VP := forall (subjectClass : CN) (subject : object), Prop.
```

Adjectival phrases and adjectives Adjectives and adjectival phrases are represented as modifiers of common nouns.

```
Definition A := CN → CN.
```

The test suite further requires to refine adjectives as intersective, subsective, extensional subsective, privative and non-commital, as it is standard in the formal semantics literature (Kamp, 1975; Partee, 2007, 2010). For a full discussion of adjectives and in general modification within a constructive setting, see Chatzikyriakidis and Luo (2017). For concision we show how to deal with intersective and extensional subsective only.

An intersective adjective (`IntersectiveA`) is fully defined by a predicate over objects. The adjectival meaning is the conjunction of such predicate and the bare noun (`wkIntersectiveA`). Additionally, to relieve the user from calling this semantic function in in many places, we declare it as an implicit coercion.

```
Definition IntersectiveA := object → Prop.
Definition wkIntersectiveA : IntersectiveA → A
          := fun a cn (x:object) ⇒ a x ∧ cn x.
Coercion wkIntersectiveA : IntersectiveA >→ A.
```

Extensional subsective adjectives are characterised by a noun-modifier a, and the property that if two noun classes p and q are extensionally equivalent, then $a(q)$ implies $a(p)$ (again, note the use of a dependent type):

```
Inductive ExtensionalSubsectiveA : Type :=
  mkExtensionalSubsective :
    forall (a : (object → Prop) → (object → Prop)),
    forall (ext : forall (p q:object → Prop),
                  (forall x, p x → q x) → (forall x, q x → p x) → forall x, a p x → a q x),
    ExtensionalSubsectiveA.
```

The adjectival semantics is that of a subsective adjective: the bare noun holds in conjunction with the modified noun:

[1] For example, a sentence like "John is a man" would be interpreted as a typing judgment in the common nouns as Types paradigm, $John : [\![Man]\!]$. Then, one would want some extra mechanism to turn this into a proposition in order to reason about it in Coq. Indeed, this is feasible as shown in Chatzikyriakidis and Luo (2017) but this is something that we have not pursued in this paper. Working on the common nouns as Types approach is left for future work, and we hope that results will shed light on the theoretical advantages of the two approaches.

```
Definition apExtensionalSubsectiveA
          : ExtensionalSubsectiveA → A
          := fun a cn (x:object) ⇒ let (aa,_) := a in
               aa cn x ∧ cn x .
Coercion apExtensionalSubsectiveA : ExtensionalSubsectiveA >→ A.
```

In the same way we treated intersective adjectives, we add the semantics as a coercion for subsectives as well. It should be stressed that it suffices to declare an adjective as extensional subsective for Coq to remember the extensional property, even though it does not appear in the interpretation as a coerced general adjective.

Adverbs Adverbs are similar to adjectives, except that they modify verbal predicates or propositions instead of nouns. In the FraCaS test suite we only find VP adverbs (verbal predicates). Furthermore, the test suite does not require the noun class to be passed to the adverb semantics, and thus we omit it. *A contrario*, the test suite requires many adverbs to be veridical and covariant. Thus, as for adjectives, we define a refined subclass to capture these properties.

```
Definition ADV := (object → Prop) → (object → Prop).
Definition Adv:= ADV.
Definition VeridicalAdv :=
  { adv : (object → Prop) → (object → Prop)
    & (forall (x : object) (v : object → Prop), (adv v) x → v x) *
      (forall (v w : object → Prop),
        (forall x, v x → w x) → forall (x : object), adv v x → adv w x)
    }.
```

The plain adverbial semantics are recovered by extracting the adv component. Similarly to extensional adjectives, veridical adverbs the additional properties are made available solely by declaring lexical entries as belonging to the correct class. A coercion between $VeridicalAdv$ and Adv is further defined (in effect we define veridical adverbs to be subtypes of adverbs). To give an example, consider the veridical adverb on_time. This will have the entry:

```
Parameter on_time_Adv : VeridicalAdv .
```

Noun phrases and predeterminers The literature usually defines the semantics of noun-phrases as a predicate over verb phrases:

```
Definition NP0 := VP → Prop.
```

Unfortunately, such a clean definition cannot work with GF's abstract syntax. The main issue is that predeterminers, which include "most", "at least", "all", etc. are parsed as modifiers of noun phrases: $PredetNP : Predet \to NP \to NP$. A moment of thought suffices to be convinced that predicates over verb-phrases are too rigid to be "pre-determined" in this way. Therefore the semantics that we use is a tuple of the components of noun-phrases: number, quantifier, and common noun:

```
Inductive NP : Type := mkNP : Num → Quant → CN → NP.
```

Predeterminers can then update the quantifier part of the NP. For example, the "all" and "most" predeterminers replace the quantifier part by the corresponding quantifier:

```
Definition Predet := NP → NP.
Definition all_Predet : Predet := fun np ⇒ let (num,qIGNORED,cn) := np
                                    in mkNP num all_Quant cn.
Definition most_Predet : Predet := fun np ⇒ let (num,qIGNORED,cn) := np
                                     in mkNP num MOST_Quant cn.
```

One may wonder if it is justified to simply ignore the quantifier in such a way. We answer positively, because in practice, a noun-phrase with a useful quantifier is never equipped with the "all"

predeterminer — in fact, in this case GF introduces a dummy indefinite article which *must* be overwritten in the semantics.

Numerals, cardinals An important part of the NP is the number. Here we record as precisely as possible the information given by the syntax; which can be a singular, a plural, a precise cardinality or even the "more than" modifier.

```
Inductive Num : Type   :=
  singular : Num       |
  plural : Num         |
  unknownNum : Num     |
  moreThan : Num → Num |
  cardinal : nat → Num .
```

Generalised quantifiers and articles Generalised quantifiers turn a number and a common noun into a (usual) noun-phrase (which we call $NP0$).

```
Definition Quant := Num → CN → NP0.
```

Certain quantifiers ignore the number, and are thus given usual definitions:

```
Definition all_Quant : Quant :=fun (num:Num) (cn : CN) (vp : VP) ⇒ forall x, cn x→ vp cn x.
```

Some others, such as "at most" make essential use of the number:

```
Definition atMost_quant : Quant
  := fun num cn vp ⇒ interpAtMost num (CARD (fun x ⇒ cn x ∧ vp cn x))
```

In the above, `interpAtMost` checks that the given number is less than the given cardinality. The function `CARD` is a context-dependent abstract function which turns a predicate into a natural number. We equip `CARD` with common-sense axioms of set cardinality, such as monotonicity:

```
Parameter CARD : (object → Prop) → nat.
Variable CARD_monotonous : forall a b:CN, (forall x, a x → b x) → CARD a <= CARD b.
```

The `CARD` variable is used to interpret several other quantifiers, including "most":

```
Definition MOST_Quant : Quant :=
    fun num (cn : CN) (vp : VP) ⇒ CARD (fun x ⇒ cn x ∧ vp cn x) >= MOSTPART (CARD cn).
```

where `MOSTPART` is another context-dependent abstract function from natural to natural. To support FraCas examples, it is sufficient to equip it with a monotonicity axiom:

```
Parameter MOSTPART: nat → nat.
Variable MOST_mono : forall x, MOSTPART x <= x.
```

As usual, articles are special cases of quantifiers. When a useful number is provided by the NP, the indefinite article enforces it. Otherwise it generates an existential quantification.

```
Definition IndefArt:Quant:= fun (num : Num) (P:CN)⇒ fun Q:VP⇒ match num with
cardinal n ⇒ CARD (fun x ⇒ P x ∧ Q P x) = n                  |
moreThan n ⇒ interpAtLeast n (CARD (fun x ⇒ P x ∧ Q P x))   |
_          ⇒ exists x, P x∧ Q P x end                        .
```

The definite article checks for plural noun phrases, in which case it implements definite plurals (universal quantification). Otherwise, it looks up the object of discourse in an abstract *environment*, which is a function which turns a common noun into an object: $environment : CN \to object$.

```
Definition DefArt:Quant:= fun (num : Num) (P:CN)⇒ fun Q:VP⇒ match num with
plural ⇒ (forall x, P x → Q P x) ∧ Q P (environment P) ∧ P (environment P) |
_ ⇒ Q P (environment P) ∧ P (environment P) end.
```

Prepositions Prepositions are interpreted as values transforming simplified noun phrases (1) to predicates. This transformation is veridical (2) and covariant (3). These three aspects are captured in three fields of a record, as follows.

```
Definition NP1 := (object → Prop) → Prop.
Inductive Prep : Type :=
  mkPrep : forall
  (prep : NP1 → (object → Prop) → (object → Prop)),              (* 1 *)
  (forall (prepArg : NP1) (v : object → Prop) (subject : object),    (* 2 *)
    prep prepArg v subject → v subject) →
    (forall (prepArg : NP1) (v w : object → Prop),
    (forall x, v x → w x) → forall x, prep prepArg v x → prep prepArg w x) (* 3 *)
     → Prep.
```

Comparatives We interpret comparatives as functions from adjective and NP into an adjectival phrase. We use the common noun (class) provided by the NP as a common reference class for the quality (or adjective, denoted a below) under comparison. Note that we additionally obtain the class of the object, but we ignore it. (Future work may want to unify those two classes).

```
Definition ComparA : A → NP → AP
 := fun a np cn x ⇒ apNP np (fun yCN y ⇒ (a cn y → a cn x)
                                        ∧ (not (a cn x) → not (a cn y))).
Definition ComparAsAs : A → NP → AP
 := fun a np cn x ⇒ apNP np (fun _class y ⇒ a cn x ↔ a cn y).
```

We define 'as ... as' as an equivalence of the quality a between the subject and the object. We define 'more ... than' as an implication of the quality a in the appropriate direction, and the converse implication in the opposite direction. Note that the treatment of comparatives does not involve any reference to scales or degrees: this was not needed at least for the examples at hand, even though many fine-grained treatments of comparatives take these into consideration.[2]

Relative clauses Relative clauses are interpreted as verb phrases and used intersectively when building noun phrases:

```
Definition RS := VP.
Definition RelNPa : NP → RS → NP
  := fun np rs ⇒ let (num,q,cn) := np
  in mkNP num q (fun x ⇒ cn x ∧ rs cn x).
```

4 Evaluation

4.1 The FraCaS test suite

The FraCaS test suite is a test suite for natural language inference (NLI) (Cooper et al., 1996). It arose out of the FraCas Consortium, a huge effort with the aim to develop a range of resources related to computational semantics. The goal of the suite is to reflect what an adequate theory of NL inference should be capable of capturing. It contains 346 NLI examples in the form of one or more premises followed by a question along with an answer to that question. There are three potential answers to the question: a) YES, indicating that the declarative sentence formed out of the question follows from the premise(s), b) NO, indicating that the negation of the declarative sentence follows from the premise(s), and c) UNK, indicating that neither the declarative sentence nor its negation follow from the premise(s). The suite is structured according to the semantic phenomena involved in the inference process for each

[2]For a first treatment in MTTs, the interested reader is directed to Chatzikyriakidis and Luo (2014, 2017).

example, and contains 9 sections. As such, there are sections on quantifiers, adjectives, comparatives, plurals etc. Some representative examples from the suite are shown below:[3]

(1) An Irishman won the Nobel prize for literature.
An Irishman won a Nobel prize.
Did an Irishman win a Nobel prize? [Yes, FraCaS 017]

(2) No delegate finished the report.
No delegate finished the report on time.
Did any delegate finished the report on time? [No, FraCaS 038]

(3) Smith, Jones or Anderson signed the contract.
Jones signed the contract.
Did Jones sign the contract? [UNK, FraCaS 083]

The FraCaS test suite has considerable weaknesses, for example its small size as well as the artificial nature of the examples.[4] However, FraCaS covers a wide range of phenomena associated with NLI and it remains a good suite to test logical approaches as regards NLI.

4.2 Evaluating FraCoq against the FraCaS

We evaluated FraCoq against 5 sections of the FraCaS test suite, a total of 174 examples. We excluded the sections where a whole lot of context-dependency has to be taken into consideration (sections on anaphora, ellipsis and temporal reference), plus the very short section dubbed as 'verbs', including aspectual class and collective predication test cases.[5] We classify as YES if a proof can be constructed from the premises to the hypothesis, NO if a proof of the negated hypothesis can be constructed and UNK otherwise.

	Section	# examples	Ours	MINE	Nut
1	Quantifiers	75	.96	.77	.53
2	Plurals	33	.76	.67	.52
3	Adjectives	22	.95	.68	.32
4	Comparatives	31	.56	.48	.45
5	Attitudes	13	.85	.77	.46
6	Total	174 (181)	0.83	0.69	0.50

The above result reveals a considerable improvement over earlier approaches in terms of accuracy. Still there are some cases that present difficulties. For example, the section on comparatives was quite challenging. Mostly, this was due to examples like the following, where one needs not only to provide adequate semantics for *more* but also to make sure that the elliptical fragment is also correctly reconstructed:

(4) ITEL won more orders than APCOM.
ITEL won some orders.
Did ITEL win some orders? [Yes, FraCaS 233]

Another example case concerns the semantics of definite plurals. The problem there is that definite plurals sometimes give rise to a universal interpretation, and sometimes to an existential one. We have

[3]The examples are taken from Bill Maccartney's xml conversion of the suite. Note that in the xml conversion, the declarative hypothesis formed out of the question is also included.

[4]These weaknesses have given rise to the creation of other NLI platforms, like RTE and SNLI. For more information on these platforms the interested reader should consult Dagan et al. (2006) and Bowman et al. (2015) respectively. The discussion on what type of NLI platform is better suited for computational semantics is out of the scope of this paper.

[5]However see the discussion in section 5 for relevant ideas for future research.

used the universal interpretation for definite plurals, so the examples involving the existential interpretation were not predicted correctly. The example below is a case where our system will find the proof, contrary to what we would want:

(5) The inhabitants of Cambridge voted for a Labour MP.
very inhabitant of Cambridge voted for a Labour MP.
Did every inhabitant of Cambridge vote for a Labour MP? [UNK, FraCaS 094]

As can be seen from the table, our system outperforms both Mineshima et al. (2015) and the Nutcracker system. We evaluate on 7 fewer examples than Mineshima et al. (2015), 174 instead of 181 (we have not evaluated against the "verbs" section), and get an overall accuracy of 0.83 compared to 0.69 by Mineshima et al. (2015), an overall improvement of 14 percentage points.

4.3 Automation

One issue to be discussed is automation. So far, our proofs are not automated, which means that there are a number of steps (usually very few), that are needed in order to produce a proof. In the earlier approaches using the proof-assistant Coq, i.e. Chatzikyriakidis and Luo (2014); Mineshima et al. (2015), the authors use Coq's tactical language LTAC to create macros of proof tactics that automate the proofs. However, we believe that such automation says little of the system's ability to provide a general automated procedure for examples outside the ones tested. Indeed, one can trivially automate all the examples at hand by just going through all the proof tactics or observe the tactics that are used in the proofs to create a macro that will automate the proofs. Yet, can that macro of tactics generalize outside the suite? Unfortunately, only to a limited extent: when exactly the same set of tactics yields a proof. For this reason, we have not automated proof search to obtain the results presented in this paper. Yet if someone were to automate the proof search for the examples looked at, in the same sense as Mineshima et al. (2015), this is not difficult to do.

Another issue with automating the work is that this would make an unprincipled use of higher-order logic (HOL): indeed, in HOL, there is no algorithm which can decide if a proposition has a proof or not. This means that we must use heuristics both to search for proofs and to decide when to give up searching. Fortunately, in FraCas, most problems have either obvious proofs or obviously lack a proof. From a logical perspective, the most difficult inferences involve veridicality and covariance, yet the premises involving those are sufficiently clear to guide the search in a clear direction. Still, due to its heuristic nature the proof search necessarily contains a human component, which is problematic to make a statement about the suitability of FraCoq outside FraCas from the percentages observed in the above table. This issue compounds with the small size of FraCas and the lack of separation between a development and a testing subset. (At the limit, one can construct a tool with 100% success rate by simply using a lookup table.)

A related shortcoming is that we sometimes use specialised semantics for specific entries in the lexicon — which may themselves trigger the need for more heuristics in proof search.

5 Conclusions and Future work

A first improvement to this work would be to address the issue of automation. We could take the unprincipled approach outlined above. A more principled approach would be to define a decidable fragment of the logic and only work within such fragment. Then, we would be able to concisely characterize how our approach generalises. In the meantime, users of our framework must understand the semantics defined above in order to grasp their scope.

Another area of improvement is at the GF level. An obvious advance would be to improve the syntax of predeterminers, to make it more suitable for compositional semantics, as we have hinted at above.

Similarly, the GF syntax for "more than" makes it very hard to recover the elliptical component of the phrase (see example (4) above). Indeed, in phrases such as "more orders than APCOM", "more"

is often in a different syntactic subtree than "than APCOM", which renders a compositional semantics infeasible. (When "than APCOM" is attached to the verb phrase the connection to "more" is not naturally recoverable.) In cases where "than APCOM" is in the correct subtree, it is construed as an adjectival phrase built from a prepositional phrase. Again, unsuitable for compositional semantics.

Lastly, a mid-term improvement to our work would be to support anaphora. A straightforward, if tedious, way to do so would be to thread an environment of mentioned objects throughout phrases.

In summary, we have connected two well-defined systems based on type-theory by providing a resource semantics for GF. We have underlined how approaches based on higher-order (and to a lesser extent even first-order) logic are inherently limited in how they generalize. Despite this limitation, we have shown that it is possible to achieve very precise semantics for specific domains. This translates into high accuracy for NLI, as exemplified by our results on the FraCas test suite. We hope that in the future such approaches will be used in performing inference tasks on controlled natural language domains and/or applications having such a component.

References

Bekki, D. and K. Mineshima (2017). Context-passing and underspecification in dependent type semantics. In S. Chatzikyriakidis and Z. Luo (Eds.), *Modern Perspectives in Type-Theoretical Semantics*, pp. 11–41. Springer.

Bertot, Y. and P. Castéran (2013). *Interactive theorem proving and program development: Coq?Art: the calculus of inductive constructions*. Springer Science & Business Media.

Blackburn, P., J. Bos, M. Kohlhase, and H. d. Nivelle (2006). separate performative account of the german right dislocation,. In *Proc. of Sinn und Bedeutung 10*.

Bos, J. and K. Markert (2005). Recognising textual entailment with logical inference. In *Proc. of the 2005 Conference on Empirical Methods in Natural Language Processing (EMNLP),*, pp. 98–103.

Bowman, S. R., G. Angeli, C. Potts, and C. D. Manning (2015). A large annotated corpus for learning natural language inference. In *Proceedings of EMNLP*, pp. 632–642.

Chatzikyriakidis, S. and Z. Luo (2014). Natural language inference in coq. *Journal of Logic, Language and Information 23*(4), 441–480.

Chatzikyriakidis, S. and Z. Luo (2017). Adjectival and adverbial modification: The view from modern type theories. *Journal of Logic, Language and Information 26*(1), 45–88.

Cooper, R. (2017). Adapting type theory with records for natural language semantics. In S. Chatzikyriakidis and Z. Luo (Eds.), *Modern Perspectives in Type-Theoretical Semantics*, pp. 71–94. Springer International Publishing.

Cooper, R., D. Crouch, J. van Eijck, C. Fox, J. van Genabith, J. Jaspars, H. Kamp, D. Milward, M. Pinkal, M. Poesio, and S. Pulman (1996). Using the framework. Technical report lre 62-051r, The FraCaS consortium. http://www.cogsci.ed.ac.uk/ fracas/.

Cooper, R., S. Dobnik, S. Larsson, and S. Lappin (2015). Probabilistic type theory and natural language semantics. *LiLT (Linguistic Issues in Language Technology) 10*, 1–43.

Dagan, I., O. Glickman, and B. Magnini (2006). The pascal recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pp. 177–190. Springer.

Glickmann, O., I. Dagan, and M. Koppel (2005). Web based probabilistic textual entailment. In *Proceedings of the PASCAL Challenges Workshop on Recognizing Textual Entailment*.

Gonthier, G. (2008). Formal proof–the four-color theorem. *Notices of the AMS 55*(11), 1382–1393.

Gonthier, G., A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. OConnor, S. O. Biha, et al. (2013). A machine-checked proof of the odd order theorem. In *Interactive Theorem Proving*, pp. 163–179. Springer.

Grudzińska, J. and M. Zawadowski (2017). Generalized quantifiers on dependent types: A system for anaphora. In S. Chatzikyriakidis and Z. Luo (Eds.), *Modern Perspectives in Type-Theoretical Semantics*, pp. 95–131. Cham: Springer International Publishing.

Hickl, A., J. Williams, J. Bensley, K. Roberts, B. Rink, and Y. Shi (2005). Recognizing textual entailment with lCC's groundhog system. In *Proc. of Second PASCAL Challenges Workshop on Recognizing Textual Entailment*, pp. 80–85.

Kamp, H. (1975). Two theories about adjectives. In E. Keenan (Ed.), *Formal Semantics of Natural Language*. Cambridge Univ Press.

Leroy, X. (2013). The compcert c verified compiler: Documentation and users manual. http://compcert.inria.fr/man/manual.pdf.

Ljunglöf, P. and M. Siverbo (2011). A bilingual treebank for the FraCas test suite. Clt project report, University of Gothenburg.

Luo, Z. (2012). Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophy 35*(6), 491–513.

MacCartney, B., M. Galley, and i. C.D. Manning (2008). A phrase-based alignment model for natural language inference. In *Proceedings of EMNLP-08*, pp. 802–811.

Martin-Löf, P. (1971). An intuitionistic theory of types. manuscript.

Martin-Löf, P. (1984). *Intuitionistic Type Theory*. Bibliopolis.

Mineshima, K., Y. Miyao, and D. Bekki (2015). Higher-order logical inference with compositional semantics. In *Proceedings of EMNLP*.

Montague, R. (1973). The proper treatment of quantification in ordinary english. In *Approaches to natural language*, pp. 221–242. Springer.

Partee, B. (2007). Compositionality and Coercion in Semantics: The Dynamics of Adjective Meaning. In *Cognitive Foundations of Interpretation*. Royal Netherlands Academy of Arts and Sciences.

Partee, B. (2010). Privative Adjectives: Subsective plus Coercion. In R. Bauerle, U. Reyle, and T. Zimmermann (Eds.), *Presuppositions and Discourse: Essays Offered to Hans Kamp*, Volume 21 of *Current Research in Semantics/Pragmatics Interface*. Emerald Group Publishing Ltd.

Pulman, S. (2013). Second order inference in NL semantics. Talk given at the KCL Language and Cognition seminar, London.

Ranta, A. (1994). *Type-Theoretical Grammar*. Oxford University Press.

Ranta, A. (2011). *Grammatical framework: Programming with multilingual grammars*. CSLI Publications.

Retoré, C. (2013). The montagovian generative lexicon Tyn: a type theoretical framework for natural language semantics. In R. Matthes and A. Schubert (Eds.), *Proc of TYPES2013*.

Romano, L., M. Kuylekov, I. Szpektor, I. Dagan, and A. Lavelli (2006). Investigating a generic paraphrase-based approach for relation extraction. In *Proceedings of EACL 2006.*, pp. 409–416.

Sundholm, G. (1989). Constructive generalized quantifiers. *Synthese 79*(1), 1–12.

Tanaka, R., K. Mineshima, and D. Bekki (2015). Factivity and presupposition in dependent type semantics. In *Proceedings of TyTLeS, ESSLLI2015*.