Transformational Semantics on a Tree Bank

Oleg Kiselyov(✉)

Tohoku University, Sendai, Japan
oleg@okmij.org

Abstract. Recently introduced Transformational Semantics (TS) formalizes, restraints and makes rigorous the transformational approach epitomized by QR and Transformational Grammars: deriving a meaning (in the form of a logical formula or a logical form) by a series of transformations from a suitably abstract (tecto-) form of a sentence. Unlike QR, each transformation in TS is rigorously and precisely defined, typed, and deterministic. The restraints of TS and the sparsity of the choice points (in the order of applying the deterministic transformation steps) help derive negative predictions and control over-generation.

The rigorous nature of TS makes it easier to carry analyses mechanically, by a computer. We report on such a mechanical, fully automatic application of TS to a tree bank of FraCAS text entailment problems (generalized quantifier section). Set-theoretic logical formulas derived by TS as meanings for input sentences are submitted to an automatic *first-order* theorem prover to decide entailment. A characteristic feature of our approach is the exhaustive enumeration of quantifier and other such ambiguities.

Overall TS proved just as capable as natural logic in inferences involving a variety of generalized quantifiers. Still open is the problem of mechanically dealing with bare plurals.

1 Introduction

We report on an automatic application of Transformational Semantics (TS) [4,5] to the FraCAS bank of textual inference problems [2].

TS, in a word, is a rigorous, rigid and restrained version of the familiar QR [3], well-expressing covert movements and other 'logical form' transformations. A transformation from a (logical) abstract form to a semantic set-theoretic formula is composed from several steps. Each step, such as raising a single quantifier, is deterministic, with no wiggle room. The order of the steps, on the other hand, is non-deterministic. TS has been applied to quantifier ambiguity, scoping islands and binding, crossover, topicalization, inverse linking, and non-canonical coordination [4,5].

Because TS is precisely specified, its transformations can be carried out mechanically, by a computer. The current implementation takes the form of a domain-specific language embedded in Haskell. It was originally intended as a semantic theory design aid: to interactively try various transformations, observe their results or failures. This paper reports on a different, more real-life application: to fully automatically derive meanings of tree bank sentences and their entailments.

The application to the real-world tree bank is not as simple as it may appear. The input is not a clean, abstract (tecto-grammatical) formula; rather it is a Penn-treebank–like annotated tree. It proved rather messy to transform the latter to the former. Another problem is dealing with quantifier and other ambiguities. In TS, they manifest as choices in the order of applying primitive transformation steps. We resolve to systematically enumerate all possible choices, hence, to expose all ambiguities licensed by TS. The final challenge is expressing the meaning of various generalized quantifiers, modal operators and intensional constructions as a *first-order* formula – so to interface with an automatic theorem prover.

Section 2 describes the whole process in detail: the annotated FraCAS and its annotations in Sect. 2.1; the conversion to the tecto-grammatical form in Sect. 2.2; verifying (type-checking) the form in Sect. 2.3; applying the Transformational Semantics to turn the form to the logical formula in Sect. 2.4; and deciding the entailment in Sect. 2.5. The conversion in Sect. 2.2 from an annotated tree to a (tecto-grammatical) form suitable for semantic analyses is not particular to TS and can be used independently, with other semantic theories. Section 3 details the handling of generalized quantifiers like "several", "many", "few", "at least three" – in particular their representation in *first-order* theories. Section 4 describes other challenges: definite descriptions and intensional verbs like "want". The challenges not yet overcome are discussed in Sect. 5. We then present the results of applying the TS to the generalized quantifier section of FraCaS.

The complete source code is available at http://okmij.org/ftp/gengo/transformational-semantics/.

2 From a Sentence to an Entailment Decision

We illustrate the TS process of deriving the meaning formula (and then, entailments) on a simple example, problem 049 from the FraCaS textual inference problem set [2] (actually, from Bill MacCartney's modified and converted to XML set[1]):

A Swede won a Nobel prize.
Every Swede is a Scandinavian.
A Scandinavian won a Nobel prize.

The goal is to decide if the third sentence is entailed by the first two.

We chose this running example because it is so straightforward: it lets us focus on the mechanics of the transformational framework without getting distracted with theoretical semantic problems. The problems are present, in abundance: see Sects. 3, 4 and 5.

[1] http://www-nlp.stanford.edu/wcmac/downloads/fracas.xml.

2.1 Parsing

Fortunately, parsing is not our problem. We take as input not raw text but a tree, annotated according to the Penn Historical Corpora system[2] (also known as the annotation system for the Penn-Helsinki Parsed Corpus of Early Modern English). The FraCAS corpus in this annotated form has been very kindly provided by Butler [1]. The first two sentences of problem 049 look as follows.

```
( (IP-MAT (NP-SBJ (D A) (ADJ Swede))
     (VBD won) (NP-OB1 (D a) (NPR Nobel) (N prize))
     (PU .))
  (ID 86_JSeM_beta_150530))

( (IP-MAT (NP-SBJ (Q Every) (ADJ Swede))
          (BEP is) (NP-OB1 (D a) (ADJ Scandinavian))
          (PU .))
  (ID 87_JSeM_beta_150530))
```

Fig. 1. The first two problem 049 sentences in the tree bank form

All words including punctuation are tagged with their part of speech: N for singular common noun, NPR for singular proper noun, VBD for verb in past tense, D for determiner, ADJ for adjective, Q for quantifier, etc. Special words "be", "do" and "have" have their own tags, such as BEP for "be" in present tense. Significantly, the syntactic structure is also annotated: e.g., subject noun phrases as NP-SBJ and object noun phrases as NP-OB1. Phrasal structure, however, is only partly annotated: either because the detailed bracketing is easily derivable or less practically useful, or because phrase boundaries are difficult to determine. (This is the case for VP: even in Modern English the attachment of verbal adjuncts is ambiguous). Dropping all annotations gives the original sentence as it was.

The main attraction of using the Penn Historical annotated data as input is that there is a wealth of such data available: not just for Modern and Historical English but also for many other languages; not just FraCAS but also the Bible, "Moby Dick", newswires, and many other texts [1]. These annotated sources have high quality. This is a treasure trove of empirical material for TS to work on.

2.2 From the Treebank to the Abstract Form

The input to TS is a (tecto-grammatical) *abstract form*, to be described in Sect. 2.3. Looking ahead, Fig. 2 shows the abstract form for the first two sentences in Fig. 1.

[2] http://www.ling.upenn.edu/ppche/ppche-release-2016/annotation/index.html.

```
cl (a_x (swede entity))              cl (every_x (swede entity))
   (won (a_y nobel_prize))              (is_cn (scandinavian entity))
```

Fig. 2. Abstract-form terms

The first task hence is to convert FraCaS problems to this abstract form. Starting from the treebank annotated data saves us parsing – but not completely. The phrasal structure in treebank trees is only partly exposed: the trees are generally rather flat. The trees have to be pre-processed first to fill in the missing structural annotations and make them binary branching. The preprocessing step also removes punctuation and metadata and normalizes parts of speech annotations eliminating irrelevant for semantic analyses distinctions.

The comparison of Figs. 1 and 2 hints that the conversion from the former to the latter is messy. For example, the NP-SBJ branch in Fig. 1 adjoins the `ADJ "Swede"` to a determiner. In the abstract form, a determiner has the type $N \to NP$ and cannot take adjective as the argument. Since "Swede" can also be a common noun (category N), one could regard Fig. 1 as mis-annotated. Butler has suggested, however, to assume that such phrases omit the trivial noun, "entity". We have to put it in.

Thus the input annotated tree requires significant preprocessing:

– Normalize all strings to lower case and remove punctuation;
– Abstract away tense: replace both `VBP` and `VBD` tags (for present- and past-tense verbs) with just `VB`;
– Insert the dummy `N "entity"` as explained earlier;
– Treat "a few", "one of the", etc. as atomic quantifiers: `Q "afew"`;
– Take "The world's greatest" to be the intensified "greatest" rather than a possessive phrase per the original annotation;
– Regard abstract common nouns with the null article (e.g., "charity") as proper-name–like NPs;
– Treat the indefinite, `D "a"`, as a quantifier `Q "a"`;
– Collect noun clusters and mark them as a compound noun: e.g., `"nobel_prize"`;
– In the same manner, turn idiomatic-like phrases such as "really great" and "travel freely" into compounds `ADJ "really_great"` and `VB "travel_freely"`;
– Simple definite descriptions like "the world" and "the report" function quite like proper nouns, so tag them as such (Sect. 4 discusses definite descriptions in detail);
– Regard "There is NP" as "NP exists" and "There are NP" as "Several NP exist";
– Simplify subject relative clauses by removing traces;
– Collect all noun complements within an NP and introduce new nodes `nc` (for singular common noun with a list of complements) and `ncs` (for plural common noun). Unlike the original Penn Historical annotation tags, ours are in lowercase;

– Recognize copular clauses and tag them with `cop`;
– Regard “some” with a plural restrictor as “several”;
– Introduce nodes `tv` and `tv-app` for a transitive verb with an argument, so to make the tree binary.

All these transformation steps are programmed as top-down macro-tree transducers and applied in the order described. The Haskell code[3] closely follows the macro-tree transducer notation and is hence easy to modify and extend. Figure 3 displays the result of the preprocessing: binary branching trees with the minimum needed information for our analyses. The preprocessed trees are straightforward to convert to the abstract terms in Fig. 2.

```
(IP-MAT
  (NP-SBJ (Q a) (nc (adj swede) (N entity)))
  (tv-app (tv won) (NP (Q a) (N nobel_prize))))

(IP-MAT
  (NP-SBJ (Q every) (nc (adj swede) (N entity)))
  (cop (Q a) (nc (adj scandinavian) (N entity))))
```

Fig. 3. Preprocessed treebank sentences

The transformations from the Penn Historical treebank trees to the “tecto-grammatized” trees in Fig. 3 and then to the terms in Fig. 2 are not specific to TS and can be used independently, with other semantic theories.

2.3 From Ad Hoc to Meaningful Transformations

The end result of the preprocessing transformations just described, and the starting point of TS transformations in Sect. 2.4 is the abstract form, a tecto-grammatized form of a sentence. The abstract form for the first two sentences of the running example was shown in Fig. 2, repeated below as Fig. 4. The preprocessing transformations are ad hoc; little is guaranteed about them. TS transformations in Sect. 2.4, as we soon see, have firmer foundations and a certain degree of correctness. Naturally they demand their input be ‘sane’, that is, well-typed. Hence the next step in our processing chain is type checking the produced abstract form.

The abstract form is defined (see [5]) to be a term in a multisorted algebra, whose sorts are familiar categories. The type of a composite term is determined from the type of its constituents using the typing rule of functional application (i.e., modus ponens). As for the primitives in our example, `entity` and `nobel_prize` have the type N, `swede` and `scandinavian` have the type ADJ (equivalent to $N \to N$), the transitive verb `won` is typed as $NP \to VP$. Determiners (quantifiers) such as `a_x` and `a_y` have the type $N \to NP$. They are

[3] http://okmij.org/ftp/gengo/transformational-semantics/Treebank.hs.

```
cl (a_x (swede entity))                cl (every_x (swede entity))
   (won (a_y nobel_prize))                (is_cn (scandinavian entity))
```

Fig. 4. Abstract-form terms

indexed for identification. The predicational copula constant `is_cn` has the type $N \rightarrow VP$; the constant `cl` of the type $NP \rightarrow VP \rightarrow S$ forms a clause.

Implementation-wise, the preprocessing transformations in Sect. 2.2 eventually produce an abstract-form term in Haskell notation, which is then 'loaded' into the Haskell interpreter – at which point it is type-checked and verified to have the type S. Although the type checking does not say that the initial treebank trees or our preprocessing steps are 'correct', it does check they are not obviously wrong.

2.4 TS Transformations

Even with syntactic details abstracted away, the terms in Fig. 4 are difficult to immediately interpret as logical formulas. The problem is the embedded quantifiers whose scope is not apparent. Making the scope explicit is the job of TS transformations.

Befitting their name, TS transformations transform an abstract term to a form in which the quantifier scope is clearly marked: specifically, by pulling an embedded quantifier (along with its restrictor) in front of the subterm over which it takes scope. TS transformations hence do quantifier raising. Unlike QR [3], however, TS transformations are rigidly and rigorously defined (and also typed). A transformation that raises a quantifier into an inappropriate place or produces an ill-formed logical formula cannot even be written (i.e., accepted by the Haskell type checker). Thus TS assures a degree of correctness. Formally, TS transformations are described in [5]; Fig. 5 illustrates them on our running example.

Input	`cl (a_x (swede entity)) (won (a_y nobel_prize))`
Raising `a_x`	`Ex (swede entity) (cl x (won (a_y nobel_prize)))`
Raising `a_y`	`Ex (swede entity) (Ey nobel_prize (cl x (won y)))`

Fig. 5. Steps of TS transformations for the first sentence of the running example

TS transformations are composed from primitive, deterministic steps, such as raising a single quantifier. The raised quantifier like `a_x`, leaving behind the constant `x`, is represented at its new place by the constant `Ex`, applied to the restrictor and the term over which the quantifier takes scope. The landing place is rigidly fixed: right over the closest `cl` subterm. In the final result, the last line of Fig. 5, all quantifier scopes are explicit; such term is straightforward to interpret as a logical (set-theoretic) formula, as detailed in Sect. 2.5. The two transformation steps can be applied in the opposite order, giving a different but logically equivalent result.

Our interactive Haskell implementation lets the users choose the steps and their order. The automatic implementation is programmed to try all possible steps in every order. Not every sequence of steps is successful (that is, ends in a logical formula): some analyses can be blocked.

2.5 Deciding Entailment

The end result of TS transformations is the abstract form that can be easily interpreted logically – or set-theoretically. Figure 6 shows that interpretation, for all three sentences in our running example (FraCaS problem 049).

The meaning of a sentence is given as a set-theoretic proposition written in first-order logic. For example, "Every Swede is a Scandinavian" is interpreted as the proposition that the intersection of `swede` and `entity` is included in the intersection of `scandinavian` and `entity` (that is, `swede` is a subset of `scandinavian`). Likewise, "A Swede won a Nobel prize" states that the intersection of `swede` and `entity` has an element in relation `won` with `nobel_prize`.

The logical formulas are written in the TPTP format supported by almost all automated theorem provers (http://www.tptp.org/). The premises (the first two sentences of FraCaS problem 049) are marked as axioms and the putative entailment as a conjecture. The TPTP formulas are written in Prolog-like syntax, with the variable names capitalized. The universal quantification $\forall x$ is notated as `![X]:` and the existential $\exists x$ as `?[X]:`; `&` stands for conjunction and `=>` for implication.

```
fof(s1,axiom,
  ?[Y]: (in(Y,nobel_prize) &
          (?[X]: ((in(X,swede) & in(X,entity)) & rel(Y,won,X))))).

fof(s2,axiom,
  ![X]: ((in(X,swede) & in(X,entity)) =>
         (in(X,scandinavian) & in(X,entity)))).

fof(c,conjecture,
  ?[Y]: (in(Y,nobel_prize) &
          (?[X]: ((in(X,scandinavian) & in(X,entity)) & rel(Y,won,X))))).
```

Fig. 6. Problem 049 in TPTP

We submit the Fig. 6 code to E, the automatic theorem prover for the first-order logic with equality [6]. It finds and displays the entailment proof.

3 Generalized Quantifiers in First Order

The story so far of transforming an annotated tree to a set-theoretic proposition has been (save for the preprocessing step, perhaps) easy-going – too easily. We now describe the pitfalls and problems, some of which are still open. The first problem is the generalized quantifiers.

The FraCAS corpus exhibits quantifier phrases that go beyond the mere existence and universality: "several", "a few", "few", "at least three", "five", "most". We describe our treatment of such generalized quantifiers – in particular, their representation in a *first-order* theory. Such a representation lets us use off-the-shelf, mature first-order theorem provers to *automatically* decide entailment. To be sure, there are sentences whose meaning cannot be represented in a first-order theory, but they are not common (in particular, they do not occur in FraCaS).

Expressing the meaning of the whole variety of quantified English phrases in a first-order theory may seem like a hard problem; yet it turned out clear-cut. As far as TS transformations are concerned, generalized quantifiers pose no difficulty: they are raised, along with their restrictors, just like the plain existential and universal quantifiers.

As an example, consider problem 076:

Few committee members are from southern Europe.
Few female committee members are from southern Europe.

The first sentence has the following abstract form:

```
cl (few_x committee_member) (is_pp (from southern_europe))
```

where `committee_member` is of type N, `from southern_europe` has the type PP and `is_pp` has the type $PP \to VP$. A TS transformation raises `few_x` as usual, giving:

```
Few_x committee_member (cl x (is_pp (from southern_europe)))
```

The problem comes from trying to express the raised generalized quantifiers in a first-order theory. The problem is not just writing a formula but also being able to decide its entailments.

Set-theoretically, we interpret `few` as *an uninterpreted relation* – in our case, between the set of committee members and the set of people from southern Europe:

```
fof(s1,axiom,
  (![X]: (in(X,sks11) <=> rel(southern_europe,from,X))) &
  few(committee_member,sks11)).
```

The former set is denoted by the constant `committee_member`; for the latter, we introduce the fresh constant `sks11` accompanied by the postulate that all members of the set `sks11` are also in the relation `from` with `southern_europe`. Likewise, the second sentence of problem 076 is represented by the conjecture:

```
fof(c,conjecture,
 (((![X]: (in(X,skc3) <=> rel(southern_europe,from,X))) &
  (![Xc1]: (in(Xc1,skc2) <=> (in(Xc1,female) & in(Xc1,committee_member)))))
  => few(skc2,skc3)).
```

where the constant `skc2` is the name for the intersection of `female` and `committee_member`, and `skc3` is another name for the set of people from southern Europe (`skc3` and the earlier `sks11` are hence extensionally equivalent).

The relation `few` is uninterpreted: the E prover knows nothing about it. To check if the conjecture holds given the premise `s1`, some properties of `few` are required – such as downward monotonicity in its first argument[4]:

```
 fof(few1,axiom,![P,P1,Q,Q1]:
   ((few(P,Q) & seteq(Q,Q1) & imply(P1,P)) => few(P1,Q1))).

 fof(imp,axiom,   ![P,Q]: (imply(P,Q) <=> (![X]: (in(X,P) => in(X,Q))))).
 fof(seteq,axiom, ![P,Q]: (seteq(P,Q) <=> (![X]: (in(X,P) <=> in(X,Q))))).
```

That is, if `few(P,Q)` holds for some sets `P` and `Q`, and `Q1` is extensionally equivalent to `Q` and `P1` is a subset of `P`, we assert that `few(P1,Q1)` also holds. The axioms `imp` and `seteq` define the subset relation and the extensional equivalence. With these axioms, the E prover easily determines that the conjecture of problem 076 holds.

Other generalized quantifiers are treated similarly, as uninterpreted relations with axioms defining their monotonicity and other properties. As another, somewhat surprising example, let's consider problem 002:

Every Italian man wants to be a great tenor.
Some Italian men are great tenors.
There are Italian men who want to be a great tenor.

When transcribing the original FraCaS corpus to XML, Bill MacCartney added: "Note that second premise is unnecessary and irrelevant". At first glance, who can doubt it: the fact that some Italian men are great tenors has no bearing on wanting to be one. It is only when we set out to prove the conjecture that we see the subtlety.

The set-theoretic meaning of the three sentences in problem 002 is as follows:

```
fof(s1,axiom,
 ![X]: ((in(X,italian) & in(X,man)) => in(X,want_greattenor))).

fof(s2,axiom,
 (((![X]:   (in(X,sks22) <=> (in(X,great) & in(X,tenor)))) &
  (![Xs21]: (in(Xs21,skitalianman) <=>
               (in(Xs21,italian) & in(Xs21,man))))) &
  several(skitalianman,sks22))).
```

[4] Bill MacCartney, the author of the XML-annotated FraCaS, noted that the original FraCaS authors must have taken "few" to mean a small absolute number – downward-monotone in the first argument.

```
fof(c,conjecture,
 (((![X]:  (in(X,skc2) <=> in(X,exist))) &
  (![Xc1]: (in(Xc1,skitalianmanwant_greattenor) <=>
      ((in(Xc1,italian) & in(Xc1,man)) & in(Xc1,want_greattenor)))))
  => several(skitalianmanwant_greattenor,skc2))).
```

The axiom for several

```
 fof(several1,axiom,![P,P1,Q,Q1]:
   ((several(P,Q) &
    (![X]: ((in(X,P) & in(X,Q)) => ((in(X,P1) & in(X,Q1))))))
   => several(P1,Q1))).
```

states that if `several(P,Q)` holds for two sets `P` and `Q`, then it holds for any other sets `P1` and `Q1` that have at least the same intersection. Without the second sentence of problem 002, the prover was unable to make the inference. A moment of thought tells why: the premise "Every Italian man wants to be a great tenor." does not actually imply that there exist at least several Italian men (even if we are to assume that "every" has existential import). It is the seemingly irrelevant "Some Italian men are great tenors." that asserts their existence, which is needed to reach the conclusion in the problem.

4 Further Challenges

Generalized quantifiers are not the only and not the biggest challenge that we have encountered so far. We now describe other problems in trying to represent the meaning of FraCaS sentences as first-order set-theoretic propositions.

Problem 002 discussed in Sect. 3 has another complication. Here its first sentence again:

Every Italian man wants to be a great tenor.

showing off the intensional "want" with the infinitival complement. Intensionality is a thorny subject. We tackle it rather unconventionally: syntactically, without resorting to possible worlds. To wit, we represent the predicate of our sentence as an uninterpreted constant `want_greattenor`, spelling out the wanted property. Clearly it does not imply (or implied by) `want_great` and `want_tenor`, exhibiting the desired referential opacity.

Definite descriptions are common among FraCaS problems as they are in English in general. A simple definite description like "the report" is regarded as an uninterpreted constant `the_report` – quite like a proper name. There are many more complicated cases in FraCaS, for example, problem 017:

An Irishman won the Nobel prize for literature.
An Irishman won a Nobel prize.

which we understand as

```
fof(s1,axiom, ?[X]:
 ((in_prominent(the_sks12,sks12) &
  (![Xs11]: (in(Xs11,sks12) <=>
            (rel(literature,for,Xs11) & in(Xs11,nobel_prize))))) &
  (in(X,irishman) & rel(the_sks12,won,X)))).

fof(c,conjecture,
  ?[X]: (in(X,irishman) & (?[Y]: (in(Y,nobel_prize) & rel(Y,won,X))))).
```

introducing two uninterpreted constants `sks12` and `the_sks12`. The first sentence of problem 017 asserts that `sks12` is extensionally equivalent to the intersection of `for-literature` and `nobel_prize`, and that `the_sks12` is the prominent element of that set. The prominent membership has the expected properties of unique membership:

```
fof(prominent1,axiom,![X,P]: (in_prominent(X,P) => in(X,P))).
fof(prominent2,axiom,![X,Y,P]:
  ((in_prominent(X,P) & in_prominent(Y,P)) => (X=Y))).
```

5 Open Questions

Although the generalized quantifier section of FraCaS contains rather simple sentences, they already pose problems that we are yet to solve. The most prominent, and the frequent, is plurality – especially bare plurals. In phrases like "there are Italian men" one may assume an implicit quantifier "several". However, such assumption fails, for example, for problem 013:

Both leading tenors are excellent.
Leading tenors who are excellent are indispensable.
Both leading tenors are indispensable.

If we are to treat "leading tenors" as "several leading tenors", the third sentence clearly does not follow from the other two – yet native speakers report the entailment. The example illustrates the well-known (and well-argued about) ambiguity of bare plurals between generic and existential readings. Literature has no shortage of proposals for analyzing bare plurals. What is lacking is the robust method that can let us reliably do textual inferences without any human intervention.

6 Evaluation

We have applied TS to the entailment problems in the generalized quantifier section of FraCaS, which has 80 problems. So far[5] we have handled 36 of them – in all the cases obtaining the agreement with the FraCaS entailment results. Specifically, out of 16 problems of FraCaS Subsect. 1.1 (conservativity) we successfully handled 12.

Currently we cannot deal with many cases of mass nouns, bare plurals or definite plural descriptions, as discussed in Sect. 5. For the remaining problems, it is just the matter of writing axioms for generalized quantifiers.

[5] As of February 2018.

7 Conclusions and Reflections

We have described the transformational-semantics–based approach to text entailment. It transforms an input Penn treebank-like annotated tree to a tecto-grammatical form, and then to a set-theoretic logical formula submitted to a first-order theorem prover. The approach is fully automatic and exhaustively enumerates all ambiguities licensed by TS.

We have applied TS to the generalized quantifier problems of FraCaS. For all the problems we can currently apply TS to, we have obtained the expected entailments or the lack of entailments. Thus, TS does work on a tree bank, and is capable of natural logic inferences with generalized quantifiers.

Applying TS to the FraCaS corpus has been a humbling experience. Facing a bank of sentences rather than a few examples – and analyzing them automatically, and being able to perform inferences – turns out quite a bigger challenge than I could imagine. Disappointingly, the quantifier ambiguity, the original motivation for TS, was conspicuously absent in the FraCaS fragment at hand. Quantifier puzzles are indeed uncommon. Plurals, on the other hand, appear all the time.

Plurals and mass nouns, hence, are the most pressing problems for the future work. We would also like to extend TS to event semantics, so to handle tense and aspect-related parts of FraCaS.

Acknowledgments. I am very grateful to Alastair Butler for providing the FraCaS data in the treebank form and explaining the annotation system – and for his many encouragements.

References

1. Butler, A.: The treebank semantics parsed corpus (2017). http://www.compling.jp/tspc/
2. Cooper, R., Crouch, D., van Eijck, J., Fox, C., van Genabith, J., Jaspars, J., Kamp, H., Milward, D., Pinkal, M., Poesio, M., Pulman, S., Briscoe, T., Maier, H., Konrad, K.: Using the framework. Deliverable D16, FraCaS Project (1996)
3. Heim, I., Kratzer, A.: Semantics in Generative Grammar. Blackwell Publishers, Oxford (1997)
4. Kiselyov, O.: Applicative abstract categorial grammars in full swing. In: Otake, M., Kurahashi, S., Ota, Y., Satoh, K., Bekki, D. (eds.) JSAI-isAI 2015. LNCS, vol. 10091, pp. 66–78. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-50953-2_6
5. Kiselyov, O.: Non-canonical coordination in the transformational approach. In: Kurahashi, S., Ohta, Y., Arai, S., Satoh, K., Bekki, D. (eds.) JSAI-isAI 2016. LNCS (LNAI), vol. 10247, pp. 33–44. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61572-1_3
6. Schulz, S.: System description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 735–743. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45221-5_49