1 Description of the Task

We propose a targeted textual entailment task designed to train and evaluate parsers. The typical parser training and evaluation methodology uses a gold treebank which raises several issues: (1) The treebank is built around a particular linguistic representation which makes parsers that use different representations (e.g. phrase structure vs. dependency) difficult to compare. (2) The parsers are evaluated based on how much of the linguistic structure dictated by the treebank they can replicate, some of which may be irrelevant for downstream applications. (3) The annotators that create the treebank not only have to understand the sentences in the corpus, but master the particular linguistic representation used, which makes their training difficult and leads to inconsistencies (see [1] for a review and [5] for more recent work on parser evaluation).

In the proposed method, simple textual entailments like the following will be used to fine-tune and evaluate different parsers:

• Final-hour trading accelerated to 108.1 million shares, a record for the Big Board.
  
  – 108.1 million shares was a record. – YES
  – Final-hour trading accelerated a record. – NO

• Earlier the company announced it would sell its aging fleet of Boeing Co. 707s because of increasing maintenance costs.
  
  – It would sell the fleet because of increasing costs. – YES
  – Selling the fleet would increase maintenance costs. – NO

• Persistent redemptions would force some fund managers to dump stocks to raise cash.
  
  – The managers would dump stocks to raise cash. – YES
  – The stocks would raise cash. – NO

The entailment examples will be generated based on the following criteria: (1) It should be possible to automatically decide which entailments are implied based on the parser output only, i.e. there should be no need for lexical semantics, anaphora resolution etc. (2) It should be easy for a non-linguist annotator to decide which entailments are implied, reducing the time for training and increasing inter-annotator agreement. (3) The entailments should be non-trivial, i.e. they should focus on areas of disagreement between current state of the art parsers. The above examples satisfy all three criteria.

Training and evaluating parsers based on targeted textual entailments address each of the issues listed in the first paragraph regarding treebank based methods: The evaluation is representation independent, therefore there is no difficulty in comparing the performance of parsers from different frameworks. By focusing on the parse differences that result in different entailments, we ignore trivial differences that stem from the conventions of the underlying representation which should not matter for downstream applications. Finally our annotators will only need a good understanding of the English language and no expertise on any linguistic framework.
2 Generating Data

The example entailment questions can be generated by considering the differences between the outputs of different state of the art parsers and their gold datasets. Some of the detected parse differences can be turned into different entailments about the sentence. The example entailments in the previous section were generated comparing the outputs of two dependency parsers, which are included in the appendix. The generated entailments will then be annotated by multiple annotators and the differences will be resolved using standard techniques.

Generating entailment questions out of parser differences allow us to satisfy conditions 1 and 3 easily: the entailments can be judged based on parser output because that is how they were generated, and they are non-trivial because some state of the art parsers disagree on them.

In our experience the most difficult condition to satisfy is 2: that it should be easy for a non-linguist annotator to decide which entailments are implied. In most of the example sentences we looked at, the differences between the parsers were trivial, e.g. different conventions on how to tag coordinating conjunctions, or whether to label a particular phrase with ADVP vs. ADJP etc. These differences are trivial in the sense that it is impossible to generate different entailments from them, thus it is hard to see how they would matter in a downstream application.

The trivial differences between parsers make example generation using our process difficult. The efficiency of example generation may be improved by pre-filtering candidate sentences which contain structures that involve non-trivial decisions by the parser such as prepositional phrase attachments. In addition some types of entailment generation can be automated. On the other hand the requirement of expressing differences in entailments will hopefully focus the training and the evaluation on non-trivial differences that actually matter in applications.

3 Evaluation Methodology

The participants will be provided with training and test sets of entailments and they will be evaluated using the standard tools and methodology of the RTE challenges [2]. The main difference is that our entailment examples focus exclusively on parsing. This should make it possible to write simple tree matching modules that decide on the entailments based on parser output alone. Example tree matching modules for standard formats (Penn Treebank [3] format for phrase structure parsing, and CoNLL [4] format for dependency parsing) will be provided as examples which should make preparing an existing parser for evaluation relatively easy.

The training part will be more parser specific, so individual participants will have to decide how to best make use of the provided entailment training set. It is unlikely that we will be able to generate enough entailment examples to train a parser from scratch. Therefore the task will have to be open to using other resources. The participants will be free to use standard resources such as treebanks to train their parsers. We can also consider restricting the outside training resources (e.g. Penn Treebank only) and the domain of the entailments (e.g. finance only). The entailment training set can then be used to fine tune the parser by focusing the evaluation on important parser decisions that effect downstream applications.

References


Example Sentence 1

GoldParse

MaltParser

MSTParser
Example Sentence 2

GoldParse

MaltParser

MSTParser
Example Sentence 3

GoldParse

MaltParser

MSTParser