Extraction of Entailed Semantic Relations Through Syntax-based Comma Resolution

Vivek Srikumar [1] **Roi Reichart**[2] **Mark Sammons**[1] **Ari Rappoport**[2] **Dan Roth**[1]

[1]University of Illinois at Urbana-Champaign
{vsrikum2|mssammon|danr}@uiuc.edu

[2]Institute of Computer Science, Hebrew University of Jerusalem
{roiri|arir}@cs.huji.ac.il

Abstract

This paper studies textual inference by investigating comma structures, which are highly frequent elements whose major role in the extraction of semantic relations has not been hitherto recognized. We introduce the problem of comma resolution, defined as understanding the role of commas and extracting the relations they imply. We show the importance of the problem using examples from Textual Entailment tasks, and present *A Sentence Transformation Rule Learner (ASTRL)*, a machine learning algorithm that uses a syntactic analysis of the sentence to learn sentence transformation rules that can then be used to extract relations. We have manually annotated a corpus identifying comma structures and relations they entail and experimented with both gold standard parses and parses created by a leading statistical parser, obtaining F-scores of 80.2% and 70.4% respectively.

1 Introduction

Recognizing relations expressed in text sentences is a major topic in NLP, fundamental in applications such as Textual Entailment (or Inference), Question Answering and Text Mining. In this paper we address this issue from a novel perspective, that of understanding the role of the commas in a sentence, which we argue is a key component in sentence comprehension. Consider for example the following three sentences:

1. *Authorities have arrested John Smith, a retired police officer.*
2. *Authorities have arrested John Smith, his friend and his brother.*
3. *Authorities have arrested John Smith, a retired police officer announced this morning.*

Sentence (1) states that John Smith is a retired police officer. The comma and surrounding sentence structure represent the relation 'IsA'. In (2), the comma and surrounding structure signifies a list, so the sentence states that three people were arrested: (i) John Smith, (ii) his friend, and (iii) his brother. In (3), a retired police officer announced that John Smith has been arrested. Here, the comma and surrounding sentence structure indicate clause boundaries.

In all three sentences, the comma and the surrounding sentence structure signify relations essential to comprehending the meaning of the sentence, in a way that is not easily captured using lexical- or even shallow parse-level information. As a human reader, we understand them easily, but automated systems for Information Retrieval, Question Answering, and Textual Entailment are likely to encounter problems when comparing structures like these, which are lexically similar, but whose meanings are so different.

In this paper we present an algorithm for *comma resolution*, a task that we define to consist of (1) disambiguating comma type and (2) determining the relations entailed from the sentence given the commas' interpretation. Specifically, in (1) we assign each comma to one of five possible types, and in (2) we generate a set of natural language sentences that express the relations, if any, signified by each comma structure. The algorithm uses information extracted from parse trees. This work, in addition to having immediate significance for natural language processing systems that use semantic content, has potential applications in improving a range of automated analysis by decomposing complex sentences into a set of simpler sentences that capture the same meaning. Although there are many other widely-used structures that express relations in a similar way, commas are one of the most commonly used symbols[1]. By addressing comma resolution, we offer a promising first step toward resolving relations in sentences.

To evaluate the algorithm, we have developed annotation guidelines, and manually annotated sentences from the WSJ PennTreebank corpus. We present a range of experiments showing the good performance of the system, using gold-standard and parser-generated parse trees.

In Section 2 we motivate comma resolution through Textual Entailment examples. Section 3 describes related work. Sections 4 and 5 present our corpus annotation and learning algorithm. Results are given in Section 6.

2 Motivating Comma Resolution Through Textual Entailment

Comma resolution involves not only comma disambiguation but also inference of the arguments (and argument boundaries) of the relationship represented by the comma structure, and the relationships holding between these arguments and the sentence as a whole. To our knowledge, this is the first paper that deals with this problem, so in this section we motivate it in depth by showing its importance to the semantic inference task of Textual Entailment (TE) (Dagan et al., 2006), which is increasingly recognized as a crucial direction for improving a range of NLP tasks such as information extraction, question answering and summarization.

TE is the task of deciding whether the meaning of a text T (usually a short snippet) can be inferred from the meaning of another text S. If this is the case, we say that S entails T. For example[2], we say that sentence (1) entails sentence (2):

1. *S*: *Parviz Davudi was representing Iran at a meeting of the Shanghai Co-operation Organization (SCO), the fledgling association that binds two former Soviet republics of central Asia, Russia and China to fight terrorism.*

2. *T*: *SCO is the fledgling association that binds several countries.*

To see that (1) entails (2), one must understand that the first comma structure in sentence (1) is an apposition structure, and does not indicate the beginning of a list. The second comma marks a boundary between entities in a list. To make the correct inference one must determine that the second comma is a list separator, not an apposition marker. Misclassifying the second comma in (1) as an apposition leads to the conclusion that (1) entails (3):

3. *T*: *Russia and China are two former Soviet republics of central Asia* .

Note that even to an educated native speaker of English, sentence 1 may be initially confusing; during the first reading, one might interpret the first comma as indicating a list, and that 'the Shanghai Co-operation Organization' and 'the fledgling association that binds...' are two separate entities that are meeting, rather than two representations of the same entity.

From these examples we draw the following conclusions: 1. Comma resolution is essential in comprehending natural language text. 2. Explicitly representing relations derived from comma structures can assist a wide range of NLP tasks; this can be done by directly augmenting the lexical-level representation, e.g., by bringing surface forms of two text fragments with the same meaning closer together. 3. Comma structures might be highly ambiguous, nested and overlapping, and consequently their interpretation is a difficult task. The argument boundaries of the corresponding extracted relations are also not easy to detect.

The output of our system could be used to augment sentences with an explicit representation of entailed relations that hold in them. In Textual Entailment systems this can increase the likelihood of correct identification of entailed sentences, and in other NLP systems it can help understanding the shallow lexical/syntactic content of a sentence. A similar approach has been taken in (Bar-Haim et al., 2007; de Salvo Braz et al., 2005), which augment the source sentence with entailed relations.

[1]For example, the WSJ corpus has 49K sentences, among which 32K with one comma or more, 17K with two or more, and 7K with three or more.

[2]The examples of this section are variations of pairs taken from the Pascal RTE3 (Dagan et al., 2006) dataset.

3 Related Work

Since we focus on extracting the relations represented by commas, there are two main strands of research with similar goals: 1) systems that directly analyze commas, whether labeling them with syntactic information or correcting inappropriate use in text; and 2) systems that extract relations from text, typically by trying to identify paraphrases.

The significance of interpreting the role of commas in sentences has already been identified by (van Delden and Gomez, 2002; Bayraktar et al., 1998) and others. A review of the first line of research is given in (Say and Akman, 1997).

In (Bayraktar et al., 1998) the WSJ PennTreebank corpus (Marcus et al., 1993) is analyzed and a very detailed list of syntactic patterns that correspond to different roles of commas is created. However, they do not study the extraction of entailed relations as a function of the comma's interpretation. Furthermore, the syntactic patterns they identify are unlexicalized and would not support the level of semantic relations that we show in this paper. Finally, theirs is a manual process completely dependent on syntactic patterns. While our comma resolution system uses syntactic parse information as its main source of features, the approach we have developed focuses on the *entailed relations*, and does not limit implementations to using only syntactic information.

The most directly comparable prior work is that of (van Delden and Gomez, 2002), who use finite state automata and a greedy algorithm to learn comma syntactic roles. However, their approach differs from ours in a number of critical ways. First, their comma annotation scheme does not identify arguments of predicates, and therefore cannot be used to extract complete relations. Second, for each comma type they identify, a new Finite State Automaton must be hand-encoded; the learning component of their work simply constrains which FSAs that accept a given, comma containing, text span may co-occur. Third, their corpus is preprocessed by hand to identify specialized phrase types needed by their FSAs; once our system has been trained, it can be applied directly to raw text. Fourth, they exclude from their analysis and evaluation any comma they deem to have been incorrectly used in the source text. We include all commas that are present in the text in our annotation and evaluation.

There is a large body of NLP literature on punctuation. Most of it, however, is concerned with aiding syntactic analysis of sentences and with developing comma checkers, much based on (Nunberg, 1990).

Pattern-based relation extraction methods (e.g., (Davidov and Rappoport, 2008; Davidov et al., 2007; Banko et al., 2007; Pasca et al., 2006; Sekine, 2006)) could in theory be used to extract relations represented by commas. However, the types of patterns used in web-scale lexical approaches currently constrain discovered patterns to relatively short spans of text, so will most likely fail on structures whose arguments cover large spans (for example, appositional clauses containing relative clauses). Relation extraction approaches such as (Roth and Yih, 2004; Roth and Yih, 2007; Hirano et al., 2007; Culotta and Sorenson, 2004; Zelenko et al., 2003) focus on relations between Named Entities; such approaches miss the more general apposition and list relations we recognize in this work, as the arguments in these relations are not confined to Named Entities.

Paraphrase Acquisition work such as that by (Lin and Pantel, 2001; Pantel and Pennacchiotti, 2006; Szpektor et al., 2004) is not constrained to named entities, and by using dependency trees, avoids the locality problems of lexical methods. However, these approaches have so far achieved limited accuracy, and are therefore hard to use to augment existing NLP systems.

4 Corpus Annotation

For our corpus, we selected 1,000 sentences containing at least one comma from the Penn Treebank (Marcus et al., 1993) WSJ section 00, and manually annotated them with comma information[3]. This annotated corpus served as both training and test datasets (using cross-validation).

By studying a number of sentences from WSJ (not among the 1,000 selected), we identified four significant types of relations expressed through commas: SUBSTITUTE, ATTRIBUTE, LOCATION, and LIST. Each of these types can in principle be expressed using more than a single comma. We define the notion

[3]The guidelines and annotations are available at `http://L2R.cs.uiuc.edu/~cogcomp/data.php`.

of a *comma structure* as a set of one or more commas that all relate to the same relation in the sentence.

SUBSTITUTE indicates an IS-A relation. An example is 'John Smith, a Renaissance artist, was famous'. By removing the relation expressed by the commas, we can derive three sentences: 'John Smith is a Renaissance artist', 'John Smith was famous', and 'a Renaissance artist was famous'. Note that in theory, the third relation will not be valid: one example is 'The brothers, all honest men, testified at the trial', which does not entail 'all honest men testified at the trial'. However, we encountered no examples of this kind in the corpus, and leave this refinement to future work.

ATTRIBUTE indicates a relation where one argument describes an attribute of the other. For example, from 'John, who loved chocolate, ate with gusto', we can derive 'John loved chocolate' and 'John ate with gusto'.

LOCATION indicates a LOCATED-IN relation. For example, from 'Chicago, Illinois saw some heavy snow today' we can derive 'Chicago is located in Illinois' and 'Chicago saw some heavy snow today'.

LIST indicates that some predicate or property is applied to multiple entities. In our annotation, the list does not generate explicit relations; instead, the boundaries of the units comprising the list are marked so that they can be treated as a single unit, and are considered to be related by the single relation 'GROUP'. For example, the derivation of 'John, James and Kelly all left last week' is written as '[John, James, and Kelly] [all left last week]'.

Any commas not fitting one of the descriptions above are designated as OTHER. This does not indicate that the comma signifies no relations, only that it does not signify a relation of interest in this work (future work will address relations currently subsumed by this category). Analysis of 120 OTHER commas show that approximately half signify clause boundaries, which may occur when sentence constituents are reordered for emphasis, but may also encode implicit temporal, conditional, and other relation types (for example, 'Opening the drawer, he found the gun.'). The remainder comprises mainly coordination structures (for example, 'Although he won, he was sad') and discourse markers indicating inter-sentence relations (such as 'However, he soon cheered up.'). While we plan to develop an annotation scheme for such relations, this is beyond the scope of the present work.

Rel. Type	Avg. Agreement	# of Commas	# of Rel.s
SUBSTITUTE	0.808	243	729
ATTRIBUTE	0.687	193	386
LOCATION	0.929	71	140
LIST	0.803	230	230
OTHER	0.949	909	0
Combined	0.869	1646	1485

Table 1: Average inter-annotator agreement for identifying relations.

Four annotators annotated the same 10% of the WSJ sentences in order to evaluate inter-annotator agreement. The remaining sentences were divided among the four annotators. The resulting corpus was checked by two judges and the annotation corrected where appropriate; if the two judges disagreed, a third judge was consulted and consensus reached. Our annotators were asked to identify comma structures, and for each structure to write its relation type, its arguments, and all possible simplified version(s) of the original sentence in which the relation implied by the comma has been removed. Arguments must be contiguous units of the sentence and will be referred to as *chunks* hereafter. Agreement statistics and the number of commas and relations of each type are shown in Table 4. The Accuracy closely approximates Kappa score in this case, since the baseline probability of chance agreement is close to zero.

5 A Sentence Tranformation Rule Learner (ASTRL)

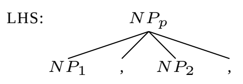
In this section, we describe a new machine learning system that learns Sentence Transformation Rules (STRs) for comma resolution. We first define the hypothesis space (i.e., STRs) and two operations – substitution and introduction. We then define the feature space, motivating the use of Syntactic Parse annotation to learn STRs. Finally, we describe the ASTRL algorithm.

5.1 Sentence Transformation Rules

A **Sentence Transformation Rule (STR)** takes a parse tree as input and generates new sentences. We formalize an STR as the pair $l \rightarrow r$, where l is a tree fragment that can consist of non-terminals, POS tags and lexical items. r is a set $\{r_i\}$, each element of which is a template that consists of the non-terminals of l and, possibly, some new tokens. This template is used to generate a new sentence, called a *relation*.

The process of applying an STR $l \rightarrow r$ to a parse tree T of a sentence s begins with finding a match for l in T. A match is said to be found if l is a subtree of T. If matched, the non-terminals of each r_i are instantiated with the terminals that they cover in T. Instantiation is followed by generation of the output relations in one of two ways: *introduction* or *substitution*, which is specified by the corresponding r_i. If an r_i is marked as an introductory one, then the relation is the terminal sequence obtained by replacing the non-terminals in r_i with their instantiations. For substitution, firstly, the non-terminals of the r_i are replaced by their instantiations. The instantiated r_i replaces all the terminals in s that are covered by the l-match. The notions of introduction and substitution were motivated by ideas introduced in (Bar-Haim et al., 2007).

Figure 1 shows an example of an STR and Figure 2 shows the application of this STR to a sentence. In the first relation, NP_1 and NP_2 are instantiated with the corresponding terminals in the parse tree. In the second and third relations, the terminals of NP_1 and NP_2 replace the terminals covered by NP_p.



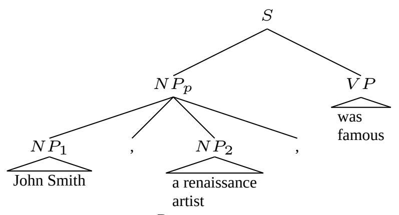
RHS:

1. NP_1 be NP_2 (introduction)
2. NP_1 (substitution)
3. NP_2 (substitution)

Figure 1: Example of a Sentence Transformation Rule. If the LHS matches a part of a given parse tree, then the RHS will generate three relations.

5.2 The Feature Space

In Section 2, we discussed the example where there could be an ambiguity between a list and an apposition structure in the fragment *two former Soviet republics, Russia and China*. In addition, simple surface examination of the sentence could also identify the noun phrases *'Shanghai Co-operation Organization (SCO)'*, *'the fledgling association that binds*



RELATIONS:

1 [John Smith]/NP_1 be [a renaissance artist]/NP_2

2 [John Smith] /NP_1 [was famous]

3 [a renaissance artist]/NP_2 [was famous]

Figure 2: Example of application of the STR in Figure 1. In the first relation, an introduction, we use the verb 'be', without dealing with its inflections. NP_1 and NP_2 are both substitutions, each replacing NP_p to generate the last two relations.

two former Soviet Republics', *'Russia'* and *'China'* as the four members of a list. To resolve such ambiguities, we need a nested representation of the sentence. This motivates the use of syntactic parse trees as a logical choice of feature space. (Note, however, that semantic and pragmatic ambiguities might still remain.)

5.3 Algorithm Overview

In our corpus annotation, the relations and their argument boundaries (chunks) are explicitly marked. For each training example, our learning algorithm first finds the smallest valid STR – the STR with the smallest LHS in terms of depth. Then it refines the LHS by specializing it using statistics taken from the entire data set.

5.4 Generating the Smallest Valid STR

To transform an example into the smallest valid STR, we utilize the augmented parse tree of the sentence. For each chunk in the sentence, we find the lowest node in the parse tree that covers the chunk and does not cover other chunks (even partially). It may, however, cover words that do not belong to any chunk. We refer to such a node as a *chunk root*. We then find the lowest node that covers all the chunk roots, referring to it as the *pattern root*. The initial LHS consists of the subtree of the parse tree rooted at the pattern root and whose leaf nodes are all either chunk roots or nodes that do not belong to any chunk. All the nodes are labeled with the corresponding labels in the augmented parse tree. For example, if we consider the parse tree and relations shown in Figure 2, then doing the above procedure gives us the initial LHS as $S\ (NP_p(NP_1, NP_2,)\ VP)$. The three relations gives us the RHS with three elements 'NP_1 be NP_2', '$NP_1\ VP$' and '$NP_1\ VP$', all three being introduction.

This initial LHS need not be the smallest one that explains the example. So, we proceed by finding the lowest node in the initial LHS such that the subtree of the LHS at that node can form a new STR that covers the example using both introduction *and* substitution. In our example, the initial LHS has a subtree, $NP_p(NP_1, NP_2,)$ that can cover all the relations with the RHS consisting of 'NP_1 be NP_2', NP_1 and NP_2. The first RHS is an introduction, while the second and the third are both substitutions. Since no subtree of this LHS can generate all three relations even with substitution, this is the required STR. The final step ensures that we have the smallest valid STR at this stage.

5.5 Statistical Refinement

The STR generated using the procedure outlined above explains the relations generated by a single example. In addition to covering the relations generated by the example, we wish to ensure that it does not cover erroneous relations by matching any of the other comma types in the annotated data.

Algorithm 1 ASTRL: A Sentence Transformation Rule Learning.

```
1: for all $t$: Comma type do
2:     Initialize $STRList[t] = \emptyset$
3:     $\mathbf{p}$ = Set of annotated examples of type $t$
4:     $\mathbf{n}$ = Annotated examples of all other types
5:     for all $x \in \mathbf{p}$ do
6:         $r$ = Smallest Valid STR that covers $x$
7:         Get fringe of $r.LHS$ using the parse tree
8:         $S = Score(r, \mathbf{p}, \mathbf{n})$
9:         $S_{prev} = -\infty$
10:        while $S \neq S_{prev}$ do
11:            if adding some fringe node to $r.LHS$ causes a significant change in score then
12:                Set $r$ = New rule that includes that fringe node
13:                $S_{prev} = S$
14:                $S = Score(r, \mathbf{p}, \mathbf{n})$
15:                Recompute new fringe nodes
16:            end if
17:        end while
18:        Add $r$ to $STRList[t]$
19:        Remove all examples from $\mathbf{p}$ that are covered by $r$
20:    end for
21: end for
```

For this purpose, we specialize the LHS so that it covers as few examples from the other comma types as possible, while covering as many examples from the current comma type as possible. Given the most general STR, we generate a set of additional, more detailed, candidate rules. Each of these is obtained from the original rule by adding a single node to the tree pattern in the rule's LHS, and updating the rule's RHS accordingly. We then score each of the candidates (including the original rule). If there is a clear winner, we continue with it using the same procedure (i.e., specialize it). If there isn't a clear winner, we stop and use the current winner. After finishing with a rule (line 18), we remove from the set of positive examples of its comma type all examples that are covered by it (line 19).

To generate the additional candidate rules that we add, we define the *fringe* of a rule as the siblings and children of the nodes in its LHS in the original parse tree. Each fringe node defines an additional candidate rule, whose LHS is obtained by adding the fringe node to the rule's LHS tree. We refer to the set of these candidate rules, plus the original one, as the rule's *fringe rules*. We define the score of an STR as

$$Score(Rule, \mathbf{p}, \mathbf{n}) = \frac{R_p}{|\mathbf{p}|} - \frac{R_n}{|\mathbf{n}|}$$

where $\mathbf{p}$ and $\mathbf{n}$ are the set of positive and negative examples for this comma type, and R_p and R_n are the number of positive and negative examples that are covered by the STR. For each example, all examples annotated with the same comma type are positive while all examples of all other comma types are negative. The score is used to select the winner among the fringe rules. The complete algorithm we have used is listed in Algorithm 1. For convenience, the algorithm's main loop is given in terms of comma types, although this is not strictly necessary. The stopping criterion in line 11 checks whether any fringe rule has a significantly better score than the rule it was derived from, and exits the specialization loop if there is none.

Since we start with the smallest STR, we only need to add nodes to it to refine it and never have to delete any nodes from the tree. Also note that the algorithm is essentially a greedy algorithm that performs a single pass over the examples; other, more complex, search strategies could also be used.

6 Evaluation

6.1 Experimental Setup

To evaluate ASTRL, we used the WSJ derived corpus. We experimented with three scenarios; in two of them we trained using the gold standard trees and then tested on gold standard parse trees (*Gold-Gold*), and text annotated using a state-of-the-art statistical parser (Charniak and Johnson, 2005) (*Gold-Charniak*), respectively. In the third, we trained and tested on the Charniak Parser (*Charniak-Charniak*).

In gold standard parse trees the syntactic categories are annotated with functional tags. Since current statistical parsers do not annotate sentences with such tags, we augment the syntactic trees with the output of a Named Entity tagger. For the Named Entity information, we used a publicly available NE Recognizer capable of recognizing a range of categories including Person, Location and Organization. On the CoNLL-03 shared task, its f-score is about 90%[4]. We evaluate our system from different points of view, as described below. For all the evaluation methods, we performed five-fold cross validation and report the average precision, recall and f-scores.

6.2 Relation Extraction Performance

Firstly, we present the evaluation of the performance of ASTRL from the point of view of relation extraction. After learning the STRs for the different comma types using the gold standard parses, we generated relations by applying the STRs on the test set once. Table 2 shows the precision, recall and f-score of the relations, without accounting for the comma type of the STR that was used to generate them. This metric, called the *Relation metric* in further discussion, is the most relevant one from the point of view of the TE task. Since a list does not generate any relations in our annotation scheme, we use the commas to identify the list elements. Treating each list in a sentence as a single relation, we score the list with the fraction of its correctly identified elements.

In addition to the Gold-Gold and Gold-Charniak settings described above, for this metric, we also present the results of the Charniak-Charniak setting, where both the train and test sets were annotated with the output of the Charniak parser. The improvement in recall in this setting over the Gold-Charniak case indicates that the parser makes systematic errors with respect to the phenomena considered.

Setting	**P**	**R**	**F**
Gold-Gold	86.1	75.4	80.2
Gold-Charniak	77.3	60.1	68.1
Charniak-Charniak	77.2	64.8	70.4

Table 2: ASTRL performance (precision, recall and f-score) for relation extraction. The comma types were used only to learn the rules. During evaluation, only the relations were scored.

6.3 Comma Resolution Performance

We present a detailed analysis of the performance of the algorithm for comma resolution. Since this paper is the first one that deals with the task, we could not compare our results to previous work. Also, there is no clear baseline to use. We tried a variant of the most frequent baseline common in other disambiguation tasks, in which we labeled all commas as OTHER (the most frequent type) except when there are list indicators like *and*, *or* and *but* in adjacent chunks (which are obtained using a shallow parser), in which case the commas are labeled LIST. This gives an average precision 0.85 and an average recall of 0.36 for identifying the comma type. However, this baseline does not help in identifying relations.

We use the following approach to evaluate the comma type resolution *and* relation extraction performance – a relation extracted by the system is considered correct only if both the relation and the type of the comma structure that generated it are correctly identified. We call this metric the *Relation-Type metric*. Another way of measuring the performance of comma resolution is to measure the correctness of the relations per comma type. In both cases, lists are scored as in the Relation metric. The performance of our system with respect to these two metrics are presented in Table 3. In this table, we also compare the performance of the STRs learned by ASTRL with the smallest valid STRs without further specialization (i.e., using just the procedure outlined in Section 5.4).

[4]A web demo of the NER is at `http://L2R.cs.uiuc.edu/~cogcomp/demos.php`.

Type	Gold-Gold Setting						Gold-Charniak Setting					
Relation-Type metric												
	Smallest Valid STRs			ASTRL			Smallest Valid STRs			ASTRL		
	P	R	F	P	R	F	P	R	F	P	R	F
Total	66.2	76.1	70.7	81.8	73.9	**77.6**	61.0	58.4	59.5	72.2	59.5	**65.1**
Relations Metric, Per Comma Type												
ATTRIBUTE	40.4	68.2	50.4	70.6	59.4	**64.1**	35.5	39.7	36.2	56.6	37.7	**44.9**
SUBSTITUTE	80.0	84.3	81.9	87.9	84.8	**86.1**	75.8	72.9	74.3	78.0	76.1	**76.9**
LIST	70.9	58.1	63.5	76.2	57.8	**65.5**	58.7	53.4	55.6	65.2	53.3	**58.5**
LOCATION	93.8	86.4	**89.1**	93.8	86.4	**89.1**	70.3	37.2	**47.2**	70.3	37.2	**47.2**

Table 3: Performance of STRs learned by ASTRL and the smallest valid STRs in identifying comma types *and* generating relations.

There is an important difference between the Relation metric (Table 2) and the Relation-type metric (top part of Table 3) that depends on the semantic interpretation of the comma types. For example, consider the sentence 'John Smith, 59, went home.' If the system labels the commas in this as both ATTRIBUTE and SUBSTITUTE, then, both will generate the relation 'John Smith is 59.' According to the Relation metric, there is no difference between them. However, there is a semantic difference between the two sentences – the ATTRIBUTE relation says that being 59 is an attribute of John Smith while the SUBSTITUTE relation says that John Smith is the number 59. This difference is accounted for by the Relation-Type metric.

From this standpoint, we can see that the specialization step performed in the full ASTRL algorithm greatly helps in disambiguating between the ATTRIBUTE and SUBSTITUTE types and consequently, the Relation-Type metric shows an error reduction of 23.5% and 13.8% in the Gold-Gold and Gold-Charniak settings respectively. In the Gold-Gold scenario the performance of ASTRL is much better than in the Gold-Charniak scenario. This reflects the non-perfect performance of the parser in annotating these sentences (parser F-score of 90%).

Another key evaluation question is the performance of the method in identification of the OTHER category. A comma is judged to be as OTHER if no STR in the system applies to it. The performance of ASTRL in this aspect is presented in Table 4. The categorization of this category is important if we wish to further classify the OTHER commas into finer categories.

Setting	P	R	F
Gold-Gold	78.9	92.8	85.2
Gold-Charniak	72.5	92.2	81.2

Table 4: ASTRL performance (precision, recall and f-score) for OTHER identification.

7 Conclusions

We defined the task of comma resolution, and developed a novel machine learning algorithm that learns Sentence Transformation Rules to perform this task. We experimented with both gold standard and parser annotated sentences, and established a performance level that seems good for a task of this complexity, and which will provide a useful measure of future systems developed for this task. When given automatically parsed sentences, performance degrades but is still much higher than random, in both scenarios. We designed a comma annotation scheme, where each comma unit is assigned one of four types and an inference rule mapping the patterns of the unit with the entailed relations. We created annotated datasets which will be made available over the web to facilitate further research.

Future work will investigate four main directions: (i) studying the effects of inclusion of our approach on the performance of Textual Entailment systems; (ii) using features other than those derivable from syntactic parse and named entity annotation of the input sentence; (iii) recognizing a wider range of implicit relations, represented by commas and in other ways; (iv) adaptation to other domains.

Acknowledgement

The UIUC authors were supported by NSF grant ITR IIS-0428472, DARPA funding under the Bootstrap Learning Program and a grant from Boeing.

References

M. Banko, M. Cafarella, M. Soderland, M. Broadhead, and O. Etzioni. 2007. Open information extraction from the web. In *Proc. of IJCAI*, pages 2670–2676.

R. Bar-Haim, I. Dagan, I. Greental, and E. Shnarch. 2007. Semantic inference at the lexical-syntactic level. In *Proc. of AAAI*, pages 871–876.

M. Bayraktar, B. Say, and V. Akman. 1998. An analysis of english punctuation: The special case of comma. *International Journal of Corpus Linguistics*, 3(1):33–57.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. of the Annual Meeting of the ACL*, pages 173–180.

A. Culotta and J. Sorenson. 2004. Dependency tree kernels for relation extraction. In *Proc. of the Annual Meeting of the ACL*, pages 423–429.

I. Dagan, O. Glickman, and B. Magnini, editors. 2006. *The PASCAL Recognising Textual Entailment Challenge.*, volume 3944. Springer-Verlag, Berlin.

D. Davidov and A. Rappoport. 2008. Unsupervised discovery of generic relationships using pattern clusters and its evaluation by automatically generated sat analogy questions. In *Proc. of the Annual Meeting of the ACL*.

D. Davidov, A. Rappoport, and M. Koppel. 2007. Fully unsupervised discovery of concept-specific relationships by web mining. In *Proc. of the Annual Meeting of the ACL*, pages 232–239.

R. de Salvo Braz, R. Girju, V. Punyakanok, D. Roth, and M. Sammons. 2005. An inference model for semantic entailment in natural language. In *Proc. of AAAI*, pages 1678–1679.

T. Hirano, Y. Matsuo, and G. Kikui. 2007. Detecting semantic relations between named entities in text using contextual features. In *Proc. of the Annual Meeting of the ACL*, pages 157–160.

D. Lin and P. Pantel. 2001. DIRT: discovery of inference rules from text. In *Proc. of ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2001*, pages 323–328.

M. P. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

G. Nunberg. 1990. *CSLI Lecture Notes 18: The Linguistics of Punctuation*. CSLI Publications, Stanford, CA.

P. Pantel and M. Pennacchiotti. 2006. Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *Proc. of the Annual Meeting of the ACL*, pages 113–120.

M. Pasca, D. Lin, J. Bigham, A. Lifchits, and A. Jain. 2006. Names and similarities on the web: Fact extraction in the fast lane. In *Proc. of the Annual Meeting of the ACL*, pages 809–816.

D. Roth and W. Yih. 2004. A linear programming formulation for global inference in natural language tasks. In Hwee Tou Ng and Ellen Riloff, editors, *Proc. of the Annual Conference on Computational Natural Language Learning (CoNLL)*, pages 1–8. Association for Computational Linguistics.

D. Roth and W. Yih. 2007. Global inference for entity and relation identification via a linear programming formulation. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press.

B. Say and V. Akman. 1997. Current approaches to punctuation in computational linguistics. *Computers and the Humanities*, 30(6):457–469.

S. Sekine. 2006. On-demand information extraction. In *Proc. of the Annual Meeting of the ACL*, pages 731–738.

I. Szpektor, H. Tanev, I. Dagan, and B. Coppola. 2004. Scaling web-based of entailment relations. In *Proc. of EMNLP*, pages 49–56.

S. van Delden and F. Gomez. 2002. Combining finite state automata and a greedy learning algorithm to determine the syntactic roles of commas. In *Proc. of ICTAI*, pages 293–300.

D. Zelenko, C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106.