In Proceedings of the Eighth Conference of the Pacific Association for Computational Linguistics 2009.

Constructing a Semantic Interpreter Using Distributional Analysis

Michael Glass, Ken Barker, Rekha Kumar, Guhan Ravi and Bruce Porter
Department of Computer Science
University of Texas at Austin
{mrglass,kbarker,rekha,guhanceg,porter}@cs.utexas.edu

Abstract

Extracting a formal representation from text that can be used to reason and answer questions has long been a goal of Artificial Intelligence research. We demonstrate a method for knowledge engineers to construct a semantic interpeter that requires little natural language processing expertise. The resulting semantic interpreter is also able to extend its coverage using semi-supervised learning. We compare the performance of an existing semantic interpretation system to our resulting semantic interpreter. Our semantic interpreter shows considerably superior performance on the two of the three test documents.

1 Introduction

There is a wealth of information available in text. However, attempting to use the text directly to answer questions means only shallow, direct questions can be answered (Etzioni et al., 2008). Knowledge representation and reasoning (KR&R) systems can answer questions that require inference, solve problems and even provide explanations (Barker et al., 2004). But constructing knowledge bases by hand requires considerable skill and is time consuming. We are working towards a semantic interpretation system to use the wealth of information in text to augment knowledge bases. By approximating the text using existing, high level concepts and relations, we funnel the complexity and ambiguity of text into a form that retains the core of the meaning and supports reasoning.

This work builds on the Möbius prototype system (Barker et al., 2007). The system consists of three major components:

- A parser, which can be combined with other natural language processing components.
- A semantic interpeter that converts the dependency tree into a set of triples using the concepts and relations of the target knowledge base.
- A knowledge integration component that combines the interpretations of the sentences and relates them to the existing knowledge (Kim and Porter, 2009).

The KR&R system used in this research is The Knowledge Machine (KM) (Clark and Porter, 1998) and the ontology is the Component Library (Barker et al., 2001). The Component Library is a richly axiomatized knowledge base of basic concepts.

2 Existing Semantic Interpretation

Below is an example of a function used in the Möbius prototype semantic interpreter.

HANDLE-CONSIST-OF(*Triples*)
  ▷ (*consist* NSUBJ *X*)(*consist* PREP_OF *Y*) ⇒
  ▷ (*X* DS_PART *Y*)
  consist ← NIL, *X* ← NIL, *Y* ← NIL
  for each $t \in Triples$
    if $t_{relation}$ = INSTANCE-OF **and** t_{tail} = "consist"
      then $consist \leftarrow t_{head}$
  for each $t \in Triples$
    if $t_{relation}$ = NSUBJ **and** $t_{head} = consist$
      then $X \leftarrow t_{head}$
    if $t_{relation}$ = PREP_OF **and** $t_{head} = consist$
      then $Y \leftarrow t_{head}$
  if $consist \neq$ NIL **and** $X \neq$ NIL **and** $Y \neq$ NIL
    then $Triples \leftarrow Triples \cup \{(X \text{ DS\_PART } Y)\}$

This function interprets a parse tree containing a "X consists of Y" subtree into an intermediate representation (X ds_part Y). The ds_part relation will later be transformed into a "part-like" relation such as has-part or subevent depending on whether X is an *Entity* or *Event*.

It should be clear that writing an interpreter this way is quite difficult. It requires knowledge

of the parser's output, knowledge of the intermediate representations used by the semantic interpreter, knowledge of the order in which interpretation steps will be attempted and testing to confirm that the function is working as intended.

The function is also not terribly general. It only applies when the sentence contains a form of the word "consist". The function will not interpret similar sentences such as "An insect *is made up of* three segments". The function could be made to apply to more cases, at a cost in complexity. However, the increase in complexity is only part of the problem. Another problem is that although any speaker of English will know many alternative ways of expressing "consists of", they can not easily list these alternations.

3 Increasing the Coverage of Interpretation

The goal of our research is to address these two key problems in interpretation coverage: the expertise and time required by development and the identification of alternative ways semantic relationships can be expressed. To address these problems we built three systems.

- A rule creation tool allows knowledge engineers with no background in parsing and no knowledge of the semantic interpreter to write useful, parser independent interpretation rules. The construction of the rules is tightly integrated with their testing, to confirm that the rules are working as intended.
- A paraphrase acquisition system finds equivalent ways of expressing the semantics covered by existing rules.
- A new semantic interpreter, SINDA (Semantic INterpretation with Distributional Analysis), uses these declarative rules for interpretation, automatically ordering the rules and selecting among alternative interpretations.

3.1 Rule Creation

The knowledge engineer begins with a sentence they would like to have interpreted. Consider, for example, "The virus attaches to the receptors." The knowledge engineer then creates the closest representation using the Component Library ontology. In this case the representation is:

(attach instance-of *Attach*)
(virus instance-of *Living-Entity*)
(receptors instance-of *Region*)
(attach object virus) (attach base receptors)

To create the interpretation rule the knowledge engineer generalizes the sentence and its interpretation by marking some of the words as slots or **variables**.

"The **virus** attaches to the **receptors**."

The interpretation is also generalized by removing or generalizing some of the instance-of triples, yielding:

(attach instance-of *Attach*)
(attach object **virus**) (attach base **receptors**)

The tool then parses the marked sentence (Figure 1) and finds the spanning tree that connects all the marked words. The result is the parse tree pattern in Figure 2.

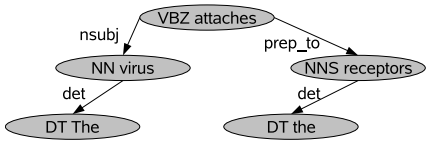


Figure 1: Parse for "The virus attaches to the receptors."

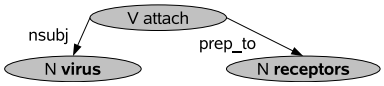


Figure 2: Parse pattern for "The **virus** attaches to the **receptors**."

The parsers used in this work are Minipar (Lin, 1993) and the Stanford parser (Klein and Manning, 2003); any parser that produces labeled dependency trees could be used. Parse tree patterns are morphologically normalized so that a pattern with "attaches" will also match "attach", "attached" and "attaching". Also, part of speech (POS) tags are grouped together in very general categories with all the different verb POS tags grouped under a single umbrella and another umbrella all for nouns.

To confirm that the interpretation rule works as intended the tool then searches a corpus of preparsed text for sentences that match the pattern. Each sentence, its parse and the resulting interpretation is then displayed to the knowledge engineer. The knowledge engineer marks the interpretation as correct or incorrect. After several such trials an estimate of the rule's value in semantic interpretation can be made. If the rule generally results in

improved interpretations, the rule is saved to the database.

To make clear the semantics of these rules an example may be useful. The *slot*s are the part of the parse pattern that are variable, in this case **virus** and **receptors** are the slots. When the parse pattern matches a sentence, such as "In a signal whip, the cracker attaches directly to the body of the whip.", the slots match *filler*s, in this case cracker and body.

By transferring the fillers from the pattern to the interpretation the sentence is partially interpreted to mean:

(attach instance-of *Attach*)
(attach object cracker) (attach base body)

3.2 Paraphrase Acquisition

The rule creation process simplifies and accelerates the process of adding coverage to a semantic interpreter. However, enumerating all of the ways a particular semantic relationship may be expressed in natural language is still too cumbersome a task. We instead use the knowledge engineer written patterns as seeds to discover alternations and variations. Essentially we find paraphrases for the patterns written by knowledge engineers.

Previous work on paraphrase acquisition has shown it is possible to use an unsupervised algorithm operating on unannotated text to discover alternate means of expressing the same semantics. In a seminal work, Hearst (1992) demonstrated a means of finding hyponyms from text using both an initial set of regular-expression-like syntactic patterns and a semi-automatically discovered set of additional syntactic patterns. In Discovering Inference Rules from Text (DIRT) (Lin and Pantel, 2001), parse tree paths, similar to our parse tree patterns but limited to two noun slots connected by a verb, are extracted from a corpus and compared for similarity. The result is a set of parse path to parse path inference rules. We combine and generalize these approaches by searching for arbitrary semantic relationships, as specified by seed parse tree patterns with potentially more than two slots.

In order to explain our method of paraphrase discovery, the following example traces a single seed pattern through the process. The process (and the semantic relation of interest) is similar to the one presented by Hearst.

Consider the knowledge engineer authored pattern below.

"A **car** is a **vehicle**" ⇒ (**car** superclass **vehicle**)

1) FIND ALL THE FILLERS FOR THE SEED PATTERN, EXCLUDING FILLERS THAT OCCUR IN THE CORPUS AT A HIGH FREQUENCY.

These two matching sentences are chosen to highlight two key filler tuples that will become important in the next step.

- In North America, a flyover is a high-level overpass, built above main overpass lanes, or a bridge built over what had been an at-grade intersection .
- A group automorphism is a group isomorphism from a group to itself.

The result of the first step is a set of word tuples (pairs in this case). These words are related according to the interpretation of the pattern, in this case by hyponymy.

2) SEARCH THE CORPUS FOR OTHER SENTENCES THAT CONTAIN ONE OF THESE TUPLES OF WORDS.

The two sentences below are found by the (automorphism, isomorphism) and (overpass, flyover) pairs.

- An order isomorphism from (S ,) to itself is called an order automorphism
- An overpass (In UK , most Commonwealth countries flyover) is a bridge , road , railway or similar structure that crosses over another road.

From each of these sentences we extract a "possible paraphrase" by finding the portion of the parse tree that spans the key words. The "possible paraphrase" for (automorphism, isomorphism) is shown in Figure 3.

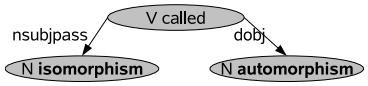


Figure 3: Parse pattern for "An order **isomorphism** from (S ,) to itself is called an order **automorphism**."

3) FOR EACH "POSSIBLE PATTERN" FIND ALL THE FILLERS.

The third step is to repeat the first step of filler extraction for each of the "possible" syntactic patterns. The result is a set of word tuples for each pattern.

4) FOR EACH "POSSIBLE PATTERN" COMPARE ITS FILLERS TO THE SEED PATTERN'S FILLERS

Following DIRT, we consider the similarity of each slot separately. This is necessary because the data is far too sparse to consider slots jointly.

The table below shows the overlap between the **car** slot in the seed pattern with the **automorphism** slot in the discovered pattern. The last line gives the total number of distinct words found in each slot as well as the sum of all the frequencies. Note that although the patterns are indeed similar, most slot fillers do not overlap. For brevity, only words that occur at least five times are included.

car	freq.	**automorphism**	freq.
area	7	area	5
function	10	function	7
group	20	group	10
number	17	number	8
point	9	point	9
process	8	process	5
space	10	space	14
Total fillers	Total freq.	Total fillers	Total freq.
76	860	21	137

Table 1: The overlap between the fillers of the **car** slot and the **automorphism** slot.

Although this table shows raw frequency counts, the similarity function uses a log-likelihood adjustment to give more weight to words that occur infrequently in the corpus since these words are less likely to occur in two different patterns by chance. In order to find the similarity of two slots we view them as vectors with a dimensionality equal to the number of distinct words that occur as fillers. The value of each dimension is the log-likelihood adjusted frequency. Now we may use traditional measures of vector similarity to assess the similarity of slots in syntactic patterns. For simplicity we use the cosine measure. Following DIRT, to find the similarity of syntactic patterns given the similarity of each slot considered separately, we take the harmonic mean of the slot similarities.

5) RETURN THE MOST SIMILAR n "POSSIBLE PATTERNS" AS SIMILAR PATTERNS.

The number n is chosen to select the precision/recall tradeoff. The similar patterns are then evaluated by a knowledge engineer to determine if they do indeed have the same interpretation as the seed pattern. The process for this is identical to the process for confirming that a human created interpretation rule is working as intended.

3.3 SINDA

So far we have focused on semantic interpretation rules in isolation. However, the interaction between the rules may be complex. Certain verbs require specialized interpretations depending on their argument structure. Compare the sentence, "The man moved." to "The man moved the plate." In both sentences "the man" is the syntactic subject but his semantic role is different. The semantic interpreter can be made aware of this distinction by two rules.

The **ball** moved. ⇒ (moved instance-of *Move*) (moved object **ball**)

The **man** moved the **ball**. ⇒ (moved instance-of *Move*) (moved object **ball**) (moved agent **man**)

Notice however, that the first pattern will match any sentence that the second pattern matches, since it is strictly less specific. We address this problem by creating a partial ordering by specificity. And we add the constraint that a more specific rule blocks the application of all less specific rules.

Additionally, there may be multiple possible interpretations for a given syntactic construct. For example, "**engine**'s **piston**" indicates partonomy while "**John**'s **boat**" indicates ownership. These two phrases have the same parse pattern.

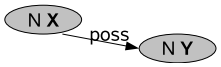


Figure 4: Parse pattern for "**engine**'s **piston**" and "**John**'s **boat**"

So phrases like "the instrument's stereo speakers" match both semantic interpretation rules. In order to address this conflict we examine the instance-of triples in each competing interpretation. The instance-of triples for "engine's piston" are (engine instance-of *Physical-Object*) (piston instance-of *Physical-Object*) and in the rule for "John's boat" the triples are (John instance-of *Person*) (boat instance-of *Tangible-Entity*). We use these instance-of triples to limit the application of rules, particularly when more than one rule may be applied.

Another component to semantic interpretation, the WORD2CONCEPT function helps to choose the best interpretation. WORD2CONCEPT maps words first to a WordNet (Fellbaum, 1998) synset and then uses a synset to Component Library concept

mapping to determine the nearest concept for a given word.

By applying WORD2CONCEPT on "instrument" and "speaker" we can determine that "instrument" is a *Musical-Instrument* and "speaker" is a *Device*. By climbing the Component Library taxonomy we determine that *Musical-Instrument* is a type of *Physical-Object* but certainly not a type of *Person*. Therefore, the instance-ofs returned by WORD2CONCEPT suggest that the partonomy rule is the correct interpretation.

4 Evaluation

In order to evaluate both the method for constructing the semantic interpreter and the resulting interpreter, two knowledge engineers used the rule creation tool to create 40 seed rules. The seed rules focused on a fragment of the most basic Component Library concepts, dealing with *Communicate*, *Create* and *Move* events as well as common relations such as "has-part", "causes" and "superclass". The paraphrase acquisition component was then run twice, once using Minipar and once using the Stanford parser. The corpus used was approximately 500MB of the Wikipedia XML corpus (Denoyer and Gallinari, 2006) with XML tags removed. The similar patterns returned were then screened to find approximately 25 valid patterns from each parser's run.

We then chose three passages, each from a different domain, in order to test the semantic interpreter. Each passage was approximately 20 sentences long with the sentences varied in length from 5 to 40 words. The passages were taken from Wikipedia entries on trade, chemical reactions and a musical instrument and were not modified prior to processing.

The two knowledge engineers, working separately, each constructed a gold standard representation for the passages. A single, unified, gold standard was then prepared by consensus. The human agreement for the passages was calculated as the size of the intersection of the gold triples divided by the size of their union. This resulted in figures of 0.88 for the chemical reaction passage, 0.86 for the piano passage and 0.75 for the trade passage. A single, unified, gold standard was then prepared by consensus.

The passages were then run through an existing semantic interpretation system, KLEO (Kim and Porter, 2009), as well as both SINDA when combined with Minipar and when combined with the Stanford parser. The output of each system was compared to the gold standard to determine precision and recall. Partial credit (the $0.5 \cdot partial$ term in the formulas) was given for instance-of triples that gave a closely related concept such as an immediate superclass or subclass of the concept in the gold standard and for triples with semantic relations similar, but not identical, to those in the gold standard.

$$Recall = \frac{correct + 0.5 \cdot partial}{Total\ gold\ standard\ triples}$$

$$Precision = \frac{correct + 0.5 \cdot partial}{Total\ semantic\ interpreter\ triples}$$

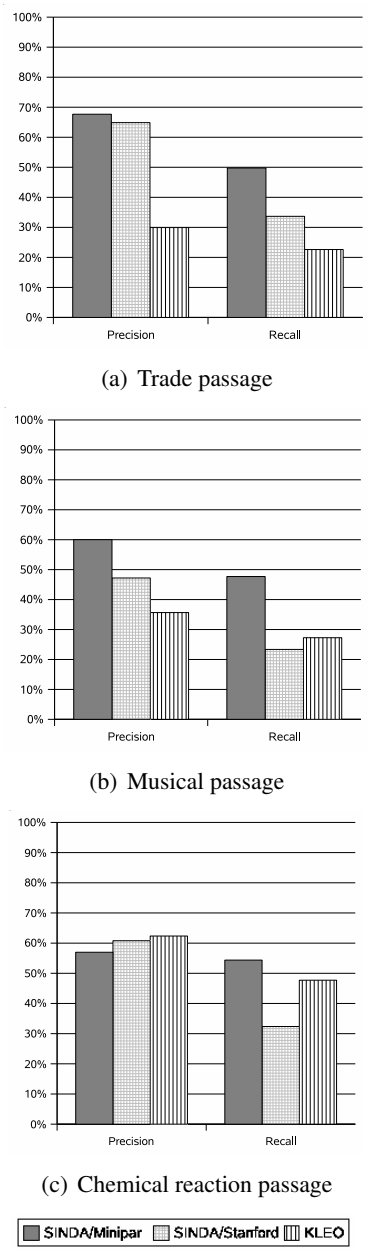
4.1 Results

Figure 5 shows the precision and recall values for each interpretation system. The performance of SINDA was, as hoped, better than its predecessor KLEO. A more suprising result is that SINDA using Minipar fared significantly better than SINDA using Stanford. This may be because the Stanford parser has more fine grained dependency relations. Minipar has a single dependency relation for the semantic subject of a sentence regardless of whether that sentence is in the active or passive voice. The Stanford parser distinguishes between these cases. This tends to make patterns from marked sentences parsed by the Stanford parser less general.

4.2 Conclusion

We attempted to achieve broad coverage combined with a rich representation by interpreting sentences using the concepts and relations of a high level, richly axiomatized knowledge base. By improving the interpretations of texts from three diverse domains we have moved closer to this goal. Additionally, we hoped to reduce the expertise and time required during the development of interpretation rules. We did not track total development time but the expertise required was limited primarily to KR&R expertise. The interpretation rules were created by knowledge engineers with no prior exposure to the parsers used and with no knowledge of SINDA's internal operation. Finally we attempted to automatically identify alternative ways semantic relationships can be expressed. From 40 seed rules we discovered approximately 25 additional rules. If scaled to other concepts and semantic relations in the Component

Figure 5: Performance of Semantic Interpretation Systems



(a) Trade passage

(b) Musical passage

(c) Chemical reaction passage

Library we may expect to see a similar degree of rule discovery as well as further improvements in performance.

Acknowledgements

Support for this research was provided by a 2008 IBM Open Collaborative Faculty Award.

References

Ken Barker, Bhalchandra Agashe, Shaw Yi Chaw, James Fan, Michael Glass, Jerry Hobbs, Eduard Hovy, David Israel, Doo Soon Kim, Rutu Mulkar, Sourabh Patwardhan, Bruce Porter, Dan Tecuci, and Peter Yeh. 2007. Learning by reading: A prototype system, performance baseline and lessons learned. In *Proceedings of Twenty-Second National Conference on Artificial Intelligence*.

Ken Barker, Shaw Yi Chaw, James Fan, Bruce Porter, Dan Tecuci, Peter Yeh, Vinay K. Chaudhri, David Israel, Sunil Mishra, Pedro Romero, and Peter E. Clark. 2004. A question-answering system for AP chemistry: Assessing KR&R technologies. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference*.

Ken Barker, Bruce Porter, and Peter Clark. 2001. A library of generic concepts for composing knowledge bases. In *Proceedings of the international conference on Knowledge capture*, pages 14–21. ACM Press.

Peter Clark and Bruce Porter. 1998. KM - The Knowledge Machine: Reference manual. Technical report, University of Texas at Austin. Http://www.cs.utexas.edu/users/mfkb/km.html.

Ludovic Denoyer and Patrick Gallinari. 2006. The Wikipedia XML Corpus. *SIGIR Forum*.

Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. 2008. Open information extraction from the web. *Commun. ACM*, 51(12):68–74.

C. Fellbaum. 1998. *WordNet – An Electronic Lexical Database*. MIT Press.

Marti A. Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, pages 539–545. Association for Computational Linguistics, Morristown, NJ, USA.

Doo Soon Kim and Bruce Porter. 2009. KLEO: A bootstrapping learning-by-reading system. *AAAI Spring Symposium*.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430. Association for Computational Linguistics, Morristown, NJ, USA.

Dekang Lin. 1993. Principle-based parsing without overgeneration. In *ACL-93*, pages 112–120.

Dekang Lin and Patrick Pantel. 2001. DIRT discovery of inference rules from text. In *Knowledge Discovery and Data Mining*, pages 323–328.