Integrating Natural Language, Knowledge Representation and Reasoning, and Analogical Processing to Learn by Reading

Kenneth D. Forbus, Christopher Riesbeck, Lawrence Birnbaum,
Kevin Livingston, Abhishek Sharma, Leo Ureel

EECS Department, Northwestern University
2133 Sheridan Road, Evanston, IL, 60208
forbus@northwestern.edu

Abstract

Learning by reading requires integrating several strands of AI research. We describe a prototype system, Learning Reader, which combines natural language processing, a large-scale knowledge base, and analogical processing to learn by reading simplified language texts. We outline the architecture of Learning Reader and some of system-level results, then explain how these results arise from the components. Specifically, we describe the design, implementation, and performance characteristics of a natural language understanding model (DMAP) that is tightly coupled to a knowledge base three orders of magnitude larger than previous attempts. We show that knowing the kinds of questions being asked and what might be learned can help provide more relevant, efficient reasoning. Finally, we show that analogical processing provides a means of generating useful new questions and conjectures when the system ruminates off-line about what it has read.

Introduction

Learning by reading requires the integration of multiple AI techniques. Natural language processing must be used to transform the text into candidate internal representations. Knowledge representation and reasoning techniques must be used to test this new information, and determine how it is to be integrated into the system's evolving models so that it can be used for effective problem solving. While tremendous strides have been made in individual research areas, few efforts have attempted to integrate them to achieve learning by reading. For example, many researchers are exploring how to mine facts matching particular patterns from the web (e.g., Etzioni *et al* 2005) or to answer specific queries (e.g., Matuszek *et al* 2005).

Working with open texts is certainly an important challenge. However, we are focusing on a different, but equally important, problem: How to handle learning a wide range of conceptual knowledge via reading, including using that knowledge to update a large knowledge base in ways that support effective reasoning. Consequently, we are less concerned with handling a wide variety of surface forms of text, exploiting instead simplified English. This is based on the observation that, for centuries, human cultures have taught children using simplified language: Less complex grammatical constructions, shorter texts, more scaffolding as to what they are about. We focus on attempting to learn as much as possible from each piece of text, while placing as few *a priori* limitations on the structure of what is to be learned as possible.

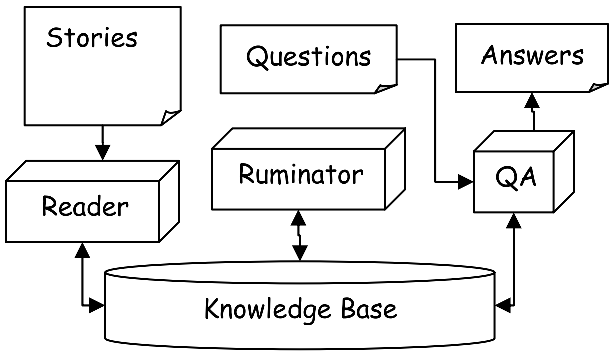

Figure 1: Learning Reader Architecture

We start by describing the architecture of Learning Reader, and some recent results obtained with the whole system. Then we discuss the ways in which we integrated several AI techniques to achieve this performance. We describe how the *Direct Memory Access Parsing* model (DMAP; Martin & Riesbeck, 1986) tightly integrates natural language processing with the KB and reasoning, unlike the usual pipeline model that splits them up. We outline how knowledge about what is already known and what can be generated by the NL system guides the extraction of axioms for question-answering from the knowledge base. We describe how analogical processing is used to construct generalizations, generate questions, and produce new conjectures during rumination. Finally, we discuss related work and plans for future work.

Learning Reader: The System

Figure 1 shows the architecture of Learning Reader. The starting endowment of the knowledge base has been extracted from ResearchCyc[1]. The Reader processes text, producing cases that are stored back into the knowledge base. The Ruminator subsequently examines these cases, asking itself questions to improve its understanding of what it has read. The Q/A system has a parameterized questions interface that enables trainers to quiz the system.

To drive our effort, we have chosen an area which is extremely broad: world history. Our corpus currently consists of 62 stories (956 sentences) about the Middle East, including its geography, history, and information about current events. All the examples and experiments described in this paper are based on this corpus.

Stories were simplified in a two step process. First, complex sentences were rewritten, typically into multiple simpler sentences. Second, the story contents were broken up into *snippets*, each with an identifiable topic category. Examples of topic categories include *geography*, *history*, *person*, *organization*, *terrorist-attacks*. The manually identified topic is stored with the text snippet and is available during processing. (Currently, the only component which uses this topic information is the Ruminator, during its search for retrievals and generalizations, as explained below.) The corpus of 62 stories used in all the experiments below, for instance, was translated into 186 text snippets via this process.

Condition	#Answers	%	# Wrong	Accuracy
Before Reading	87	10%	0	100%
Reading only	320	37%	1	99.7%
Reading + Deductive Rumination	434	50%	3	99.3%
Reading + Deductive + PCA	525	60%	48	90.8%

Table 1: Summary of System-Level Experiments

Table 1 summarizes some system-level experiments. The set of questions asked was generated by using the templates associated with the parameterized questions in the Q/A system and the knowledge base resulting from reading the entire corpus, to ensure that questions were asked about new entities appearing in the text. These questions were asked before any processing was done (the "before reading" row), after the entire corpus had been processed by the Reader, but without any rumination (the "Reading only" row), and after rumination in two conditions, one where the Ruminator operated purely deductively, and the other where it was allowed to make conjectures (the PCA condition, described below). The improvement from 10% without reading to 37% after reading demonstrates that the system did indeed learn by reading. The further improvement to 50% after deductive rumination, and 60% after rumination with PCA, demonstrates that rumination does indeed help the system learn more from the texts that it has read. However, it does come at some cost: Even deductive rumination introduces a few new errors, and of the 91 new questions that the system can answer after rumination with PCA, 47 of them are incorrect, meaning the new knowledge is leading to mistakes roughly half the time.

Next we examine each of the components, showing how they integrate multiple AI techniques in order to achieve this performance.

The Reader

The primary goal of the Reader is to identify quickly and accurately what pre-existing knowledge an input text is referring to, creating new knowledge only when none can be found. For this reason, we use the Direct Memory Access Parsing (DMAP) model of natural language understanding (Martin and Riesbeck, 1986). DMAP treats understanding as a recognition process, rather than as a semantic composition process. A DMAP system sees an input as a stream of references to concepts. It incrementally matches those references against *phrasal patterns*. When patterns are completely matched, they generate additional higher-order conceptual references.

For example, the lexical items in "an attack occurred in Baghdad" initially generate references to the concepts for `AttackOnObject` and `CityOfBaghdad`, These concepts plus the original lexical items in turn match the phrasal pattern `((isa ?event Event) Occur-TheWord In-TheWord (isa ?location GeographicalRegion))`, because `AttackOnObject` is a type of `Event` and `CityOfBaghdad` is a `GeographicalRegion`. Matching this phrasal pattern identifies a reference to the conceptual assertion `(eventOccursAt ?event ?location)`, where `?event` and `?location` are known to be the attack and Baghdad concepts already seen. The Reader then queries the KB for existing instances. Thus, in this example, the Reader will query memory for known instances of attacks that have occurred in Baghdad, to provide a specific value(s) for `?event`. If none are found, a Skolem constant will be generated. For example, given the text snippet:

> "An attack occurred in Al Anbar. The bombing occurred on August 3, 2005. The attack killed 14 soldiers."

DMAP produces the following output:

```
(isa Bombing-653 AttackOnTangible)
(isa Bombing-653 Bombing)
(eventOccursAt Bombing-653 AlAnbar-ProvinceIraq)
(dateOfEvent Bombing-653
    (DayFn 3 (MonthFn August (YearFn 2005))))
(deathToll Bombing-653 ArmyPersonnel 14)
```

Since DMAP did not know of an attack that satisfied what it was reading, it created a new instance (`Bombing-653`), but it was careful to use entities that it already understood (e.g., `AlAnbar-ProvinceIraq`) rather than, for instance,

[1] http://research.cyc.com/

creating a new entity and being forced to resolve it later, as many NLU systems do.

The research goal for the DMAP-based Reader is to develop scalable techniques for knowledge-rich lexically-based language understanding in large realistically-sized knowledge bases. The challenges boil down to *scale* and *variability*. In terms of scale, the Reader has to manage over 30,000 phrasal patterns, and avoid queries like "an event in a location" that can retrieve thousands of instances. In terms of variability, the Reader has to deal with a KB that was developed by a number of knowledge engineers over time. This leads inevitably to variations in *detail*, e.g., some event descriptions omit critical information like specific time and place, *specificity*, e.g., an agentive assertion might use `doneBy` or `perpetrator` or some other related but not identical relationships, and *representational choice*, e.g., over time, increasing use has been made of structured non-atomic terms (NATs) rather than named entities. The Reader cannot simply ask for "all attacks in Baghdad." It has to look for all events that are consistent with being an attack in Baghdad, without being overwhelmed with irrelevant results.

To evaluate DMAP's performance on the corpus, a preliminary answer key was created representing some of the primary assertions we expect from each story. Currently the Reader reproduces 87% of this key. However, this answer key does not represent all the assertions that should be produced, and coverage may decrease as the answer key is made more complete.

DMAP currently uses over 30,000 phrasal patterns. A small subset (50) of these were hand-generated, the rest were automatically translated from linguistic knowledge in the ResearchCyc KB contents. For DMAP, an important metric is how much of the KB is potentially accessible through its phrasal patterns. DMAP can find instances in memory for 99% (27453/27649) of the collections in the Cyc ontology. DMAP can produce assertions for new instances for 57% of the collections. With respect to predicates in the Cyc ontology, DMAP has phrasal patterns that can produce and access 13% (1175/8892) of all possible predicates. While 13% may seem small, this still enables it to access 43% of the 1.2 million assertions presently made in ResearchCyc, including `isa` expressions. If we only consider predicates other than `isa`, DMAP can access 24.8% of ResearchCyc.

Unlike traditional syntactic parsers, DMAP is tightly integrated with the knowledge base. Every component of DMAP interacts with the KB. Names are translated into existing instances using assertions in the Cyc KB. Similarly, text is translated into lexical concepts and then from lexical concepts into semantic concepts using other assertions in the KB. Pattern matching is done using patterns extracted from other linguistic knowledge in Cyc.

Several processes in DMAP leverage the knowledge in the KB directly to provide cues and biases for building interpretations. (See Livingston & Riesbeck (2007) for details.) After patterns are matched, the KB is queried to identify potentially relevant instances. Interpretations that reference known instances are preferred. This produces a bias to understand things in terms of that which it already knows.

Second, DMAP is also biased to interpretations that include predicates it has seen co-occur with predicates used in the interpretations of earlier sentences. By co-occur we mean the number of times statements using these predicates have shared an argument in the knowledge base.

A third way DMAP leverages the underlying memory is to perform coreference resolution. DMAP will allow references from two different sentences (e.g. "terrorist attack" and "bombing") to refer to the same entity if one is a generalization of the other. If a generalization does not hold, then DMAP will use a case-based resolution strategy. DMAP will query the knowledge base to see if there exists a known instance that belongs to both classes. In this example, because the KB has examples of terrorist bombings, DMAP would allow the coreference.

Since DMAP is intrinsically tied to its underlying knowledge base, in this case ResearchCyc (which is over three orders of magnitude larger than has been used with previous DMAP systems) problems of scale and managing ambiguity arise. Our original implementation tracked understanding ambiguities as they arose at the word level. This approach could only read 35% of the sentences at a rate of one second or better, and did not scale, asymptotically approaching processing only 63% of the test corpus, even when given hours per sentence. Switching to a Reader that tracked ambiguity at the sentence level provided significant improvement, allowing 63% of the corpus to be read at a rate of one second per sentence or better, and reaching 99% of the corpus when allowed to take as much as 8.7 minutes per sentence. Leveraging the heuristics mentioned above to prioritize a best-first search provided even better scaling, reaching 78% of the corpus in under a second per sentence, and 99% when allowed to take 1.1 minutes.

The Q/A System

The purpose of the current Q/A system is to provide a means of examining what the system has learned. We use a parameterized question template scheme (cf. Cohen *et al*, 1998) to ask types of questions that are particularly appropriate for the domain we are dealing with. The current templates are: (1) Who is <*Person*>?, (2) Where did <*Event*> occur?, (3) Where might <*Person*> be?, (4) What are the goals of <*Person*>?, (5) What are the consequences of <*Event*>?, (6) When did <*Event*> occur?, (7) Who is involved in <*Event*>?, (8) Who is acquainted with (or knows) <*IntelligentAgent*>?, (9) Why did <*Event*> occur?, and (10) Where is <*SpatialThing*>?

In each template, the parameter (e.g., <*Person*>) indicates the kind of thing for which the question makes sense (specifically, a collection in the Cyc ontology). Each template expands into a set of formal queries, all of which are attempted in order to answer the question. The minimum number of formal queries per template is one, the maximum is 13 (location), with a mean of 5. For example, question 3 uses queries involving `hasBeenIn`, `citizens`, and `objectFoundInLocation`.

One problem with large knowledge bases is that, as they grow, the cost of inference can become astronomical, with failed queries taking hours or even days[1]. Our solution to this problem is to restrict the set of axioms used for reasoning. In the FIRE reasoning engine, backchaining is restricted to small sets of axioms called *chainers*. A chainer is a single partition within the KB, used for reasoning, in the sense of Amir & McIlraith (2005). While their algorithm for partitioning assumes a fixed KB, ours must deal with a KB that grows as the system reads. We exploit two sources of knowledge in our extraction process. The first is common, i.e., the kinds of predicates that will be used in queries. The second is information about the kinds of statements that are already in the KB plus the kinds of statements that the natural language system is capable of generating. That is, if the DMAP phrasal patterns are capable of inferring statements containing a predicate *P*, then we know that we could learn such statements via reading, and otherwise we cannot.

Our axiom extraction algorithm creates a set of Horn clauses (for tractability). The KB axioms are typically not Horn, so we translate them into clauses to extract a Horn clause subset (cf. Peterson *et al* 1998). The antecedents of each Horn clause are examined to see if they are potentially available in the KB, or if they are obtainable by the Reader. If they are, the Horn clause is added to the set of axioms for the chainer. Otherwise, the failed antecedents are examined to see if there are Horn clauses that could prove them. This process continues for a maximum depth (default = 3), filtering out any rules that have antecedents that will not be derivable within that boundary. The details are described in Sharma & Forbus (in preparation); what is important here is that we are exploiting the structure of the natural language system to make deductive inference more efficient, another advantage of creating an integrated system.

Two chainers are created by this process. The chainer for Q/A must be efficient, because it is an interactive process. Consequently, we limit it to creating Horn clauses from the `specPred` hierarchy in the Cyc ontology, but with unlimited depth. For example, if the KB contained

```
  (genlPreds explosiveDeviceUsed deviceUsed)
```

the following Horn clause would be added:

```
  (<== (deviceUsed ?x ?y)
       (explosiveDeviceUsed ?x ?y))
```

The QA chainer contains 787 axioms. The Ruminator chainer is more complex, containing 1,978 axioms, since it can operate off-line. It includes rules that map from `specPreds` to the query predicates, up to a depth of 6, while other rules are limited to the default depth of 3. Other recursive clauses are eliminated to improve performance. Further automatic static analysis is done to eliminate reasoning bottlenecks, which can speed inference by a factor of 129 on average, with a worst-case improvement of a factor of 4, with only an 8.5% loss of completeness.

[1] cf. www.projecthalo.com/content/docs/

The Ruminator

The Reader does focused forms of inference, to retrieve, filter, combine, and create descriptions of the text. But this does not capture the human tendency to learn by later reflecting upon what they have read, connecting it more deeply to what they already know and pondering its implications. The Ruminator models this kind of off-line learning. The operation of the Ruminator can be divided into three phases: *Elaboration*, *question generation*, and *question processing*. We discuss each in turn.

Elaboration: The Reader's output is a case, representing its understanding of a text snippet. The first step is to enrich the case with information about the entities and events involved from the knowledge base. We do this by using dynamic case construction techniques (Mostek *et al* 2000) to extract KB facts that are directly linked to the entities and events of the story. This elaboration serves two purposes. First, it reduces the amount of work needed for subsequent inferences. Second, it "primes the pump" for analogical processing in the next phase. We call these descriptions *conceptual models*. For example, in the snippet used earlier, this process adds facts indicating that Al Anbar is a province, in the country of Iraq.

Question Generation: A key process in rumination is generating interesting questions to consider. We use three strategies for generating questions. The simplest uses a form of *knowledge patterns* (Clark *et al* 2000), canonical questions that one asks about a kind of entity. Given our current focus on world history, we use formalized versions of the standard Journalist's Questions (who, what, when, where, why, how) as defined in the Q/A system. In the Al Anbar example, for instance, one question the Ruminator generates in this way is, paraphrased, "Who is involved in the Al Anbar attack?"

The second strategy, analogical retrieval, is based on the insight that if two things are similar in some ways, they might be similar in others. We use the MAC/FAC model of similarity-based retrieval (Forbus *et al* 1994) to retrieve cases. The retrieval probe is the conceptual model for the story. The case library used for a story is based on the topic given for the text snippet. It includes all instances of that concept from both the KB and the system's prior reading. The second stage of MAC/FAC uses SME (Falkenhainer *et al* 1989; Forbus *et al* 1994), which models analogical matching, to construct candidate inferences about the probe using the retrieved case. These candidate inferences serve as the basis for another set of questions. For example, based on an analogy with a terrorist attack in Farah, Afghanistan, one question the Ruminator generated about the Al Anbar example used above is, paraphrasing, "Was the device used in the Al Anbar attack something like a rocket?"

The third strategy is to compare the new story with generalizations made about the topic. The generalizations are automatically constructed via analogical processing, using SEQL (Kuehne *et al* 2000) over all of the instances of that topic in the KB and the system's prior reading. As a new case comes in, we use SME to compare it with every generalization, creating candidate inferences which are then used as new questions. This gives us a source of questions that reflect the system's experience with that topic[1]. We use an extension of SEQL due to Halstead & Forbus (2005) that provides probabilities for statements in generalizations. This provides us with information that can be used for prioritizing questions: Candidate inferences generated from a more likely statement are more likely to be interesting, however they turn out.

The strategies so far are entirely deductive: Even if a question was generated via analogy with a prior example or with a generalization, deductive reasoning with the Ruminator's chainer generated a proof in terms of the facts in the case plus the KB contents. The Promiscuous Conjecture Acceptance (PCA) strategy moves beyond deductive rumination. Recall that candidate inferences represent what might hold in a new situation (the target), based on the way that it corresponds with some prior experience (the base). Here the base is a prior case or a generalization, and the target is the newly read case. Not all snippets provide exactly the same information, so candidate inferences provide a form of pattern completion. For example, one story might mention that an attack involving a Sunni insurgent group occurred in Iraq, whereas another might not. When PCA is used, candidate inferences that pass certain tests are accepted as true in the case. The tests are (1) the candidate inference cannot contain any analogy skolems. An analogy skolem represents the projection into the target of an entity in the base which was not mapped to anything. For example, the "something like a rocket" in the analogy question about the Al Anbar attack is the English rendition of an analogy skolem, which is used as a variable in the query. Such unbound existentials are not very useful for subsequent reasoning, hence they are not introduced. (2) The inference must not be obviously contradictory. By this we mean that it must not violate disjointness constraints in the KB and simple spatial constraints (e.g., a conjectured location of an event must not be spatially disjoint with what is already known about where it occurred).

In the experiment described above, 186 text snippets gave rise to 871 knowledge pattern questions and 1,238 analogical questions, for a total of 2,109 questions. The average number of questions/snippet is 11.3, 6.6 (58%) of which on average are from analogies.

Question Processing: Two of the three sources of questions we use are non-deductive, so it is possible to generate questions that simply don't make sense, given what the system already knows. (e.g., "Is it true that the City of San Antonio's spouse is Chile?") We use type inference with argument restrictions to eliminate questions that are clearly internally inconsistent[2]. As with Q/A, we use restricted inference to attempt to answer the questions that seem to make sense. The chainer used for rumination was described above. Answers, when found, are stored in the conceptual model.

As the statistics above indicate, the Ruminator can generate a huge number of questions. Those questions that it cannot answer are stored in the KB, as a queue of open questions for future consideration. When a new story is read, it reconsiders these questions to see if the new knowledge enables it to now answer them.

Noise: A system-level issue

Our experiments to date suggest that the presence of noise in learned knowledge is one of the key issues in learning by reading. For example, in one run we ended up with the Sudan being viewed as a military person, and the assertion that, up to 1920, Iraq was a definite NL attribute. There are three sources of noise: Errors in the initial knowledge base, imperfect understanding in the Reader, and conjectures inappropriately accepted during rumination. When rumination is purely deductive, only the first two sources of noise are possible. But, as the errors about the Sudan and Iraq illustrate, they do indeed occur. We have recently modified the elaboration stage of Rumination to scrutinize incoming facts more cautiously, to seek out contradictions on its own. The provenance of all information in a case is recorded, providing the potential to track down such misunderstandings and correct them.

The problem of noise raises another fundamental issue for learning by reading systems: How does noise in the KB change as a function of learning by reading? Under what conditions does the feedback loop provided by the read/ruminate cycle act to dampen noise in the KB over time, versus amplify it? This will be investigated in future experiments, as outlined below.

Related Work

Most systems that learn by reading are aimed at extracting particular kinds of facts from the web. For example, KnowItAll (Etzioni *et al* 2005) extracts named entities and OPINE (Popescu & Etzioni, 2005) extracts properties of products. While impressive in the quantity of information they can acquire, they do not attempt to understand a story as a whole, nor do they attempt to integrate it into a large pre-existing knowledge base. Closer to Learning Reader is Cycorp's "Factovore" (Matuszek *et al* 2005), which uses web searches to find candidate answers to queries generated by using a hand-generated set of templates. Their question generation process is similar to our use of knowledge patterns in the Ruminator, but they do not have the equivalent of our analogy-based question generation

[1] We speculate that such questions eventually become new knowledge patterns, but we have not experimented with this yet.

[2] Unfortunately this process is imperfect, because many argument restrictions in the KB are weak, e.g. `SpatialThing` or even `Thing`.

strategies. For us, questions are generated based on what we have read, whereas for them information extraction is done in order to answer specific questions. Cycorp also uses a human editorial staff to validate knowledge as part of their cycle. Our goal is that trainers should never know the underlying representations that Learning Reader is creating. We hope to enable people to extend it as long as they can use simplified English, without being AI experts.

We know of no other system that integrates analogical processing into the understanding process.

Discussion

We have described Learning Reader, a prototype system that integrates natural language processing, deductive reasoning over large knowledge bases, and analogical processing in order to learn by reading simplified texts. While Learning Reader is in its early stages, we believe the results shown here indicate great promise. We have shown that a system can "close the loop", with natural language processing producing representations of text that can be assimilated into a large knowledge base, and used to answer questions and to improve subsequent understanding (via being retrieved through DMAP during reading and being retrieved via MAC/FAC during rumination).

There are several directions we plan to pursue next. First, we plan to expand our corpus. Our original corpus will be doubled in size to test breadth, and a further expansion will be done by systematically building up stories about a particular area, so that we can explore the impact of noise on learning from a large body of interrelated material. Second, we intend to use DMAP for question-parsing instead of parameterized questions. This will expand coverage and provide the basis for implementing an interactive dialogue system, to allow trainers to ask follow-up questions, and to allow the Ruminator to ask its trainers a limited number of questions, with answers being interpreted also via DMAP. Third, the process we added to the Ruminator which scrutinizes newly learned knowledge for inconsistencies detects errors, but it does not yet propose or implement repairs to that knowledge. We expect that techniques of model-based diagnosis, applied to a model of the system's own processing (cf. de Koning *et al* 2000) will be useful in this regard. Finally, we plan to expand the role of evidential reasoning in the Ruminator, exploiting the probabilities generated via SEQL to help decide what action to take when a misunderstanding is diagnosed.

Acknowledgements

This research was conducted as part of the Learning by Reading Project, supported by DARPA IPTO.

References

Amir, E. and McIlraith, S. 2005 Partition-Based Logical Reasoning for First-Order and Propositional Theories, *Artificial Intelligence* **162** (1-2), pp. 49-88

Clark, P., Thompson, J. and Porter, B. 2000. Knowledge Patterns. *Proceedings of KR2000*.

Cohen, P., Schrag, R., Jones, E., Pease, A., Lin, A., Starr, B., Gunning, D., and Burke, M. 1998. The DARPA High-Performance Knowledge Bases Project. *AI Magazine*, 19(4), Winter, 1998, 25-49

de Koning, K., Bredeweg, B., Breuker, J., and Wielinga, B. 2000. Model-based reasoning about learner behaviour. *Artificial Intelligence* 117, 173-229.

Etzioni, O., Cafarella, M., Downey, D., Popescu, A., Shaked, T., Soderland, S., Weld, D., and Yates, A. Unsupervised Named-Entity Extraction from the Web: An Experimental Study. *Artificial Intelligence*.

Falkenhainer, B., Forbus, K. and Gentner, D. 1989. The Structure-Mapping Engine: Algorithms and Examples. *Artificial Intelligence*.

Forbus, K., Ferguson, R., and Gentner, D. 1994. Incremental structure-mapping. *Proceedings of CogSci94*.

Forbus, K., Gentner, D. and Law, K. 1994. MAC/FAC: A model of similarity-based retrieval. *Cognitive Science*

Halstead, D. and Forbus, K. 2005. Transforming between Propositions and Features: Bridging the Gap. *Proceedings of AAAI05*.

Kuehne, S., Forbus, K., Gentner, D. and Quinn, B. 2000. SEQL: Category learning as progressive abstraction using structure mapping. *Proceedings of CogSci2000*

Livingston, K., and Riesbeck, C.K. 2007. Using Episodic Memory in a Memory Based Parser to Assist Machine Reading, *Working notes, AAAI Spring Symposium on Machine Reading*, AAAI Press.

Martin, C.E. and Riesbeck, C.K. Uniform Parsing and Inferencing for Learning. Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, PA, August 11 - 15, 1986, pp 257-261.

Matuszek, C., Witbrock, M., Kahlert, R., Cabral, J., Schneider, D., Shah, P., and Lenat, D. 2005. Searching for Common Sense: Populating Cyc from the Web. *Proceedings of AAAI05*

Mostek, T., Forbus, K. and Meverden, C. 2000. Dynamic case creation and expansion for analogical reasoning. *Proceedings of AAAI-2000*.

Peterson, B., Andersen, W., and Engel, J. 1998. Knowledge Bus: Generating Application-focused Databases from Large Ontologies. *Proceedings of the 5th KRDB Workshop*, Seattle, WA.

Popescu, A., and Etzioni, O. 2005. Extracting Product Features and Opinions from Reviews. *Proceedings of HLT-EMNLP 2005*

Sharma, A. and Forbus, K. (in preparation) Automatic Extraction of Efficient Axiom Sets from Large Knowledge Bases.