A Prototype System that Learns by Reading Simplified Texts

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

Systems that could learn by reading would radically change the economics of building large knowledge bases. This paper describes Learning Reader, a prototype system that extends its knowledge base by reading. Learning Reader consists of three components. The Reader, which converts text into formally represented cases, uses a Direct Memory Access Parser operating over a large knowledge base, derived from ResearchCyc. The Q/A system, which provides a means of quizzing the system on what it has learned, uses focused sets of axioms automatically extracted from the knowledge base for tractability. The Ruminator, which attempts to improve the system's understanding of what it has read by off-line processing, generates questions for itself by several means, including analogies with prior material and automatically constructed generalizations from examples in the KB and its prior reading. We discuss the architecture of the system, how each component works, and some experimental results.

Introduction

One of the long-term dreams of Artificial Intelligence is to create systems that can bootstrap themselves, learning from the world in the ways that people do. Learning by reading is a particularly attractive version of this vision, since that is a powerful source of knowledge for people, and there are now massive amounts of text available on-line. Indeed, many researchers are working to directly use such texts to extract facts matching particular patterns (e.g., Etzioni *et al* 2005) or to answer specific queries (e.g., Matuszek *et al* 2005). Such efforts focus on scale, at the cost of limiting the kinds of information they seek. We are focusing on a different set of tradeoffs. In the Learning Reader project, we are starting with simplified text. 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. Similarly, human children do a lot of learning in homes and schools, where the sources of information are controlled for accuracy, only later learning "on the street" when they are more experienced. We believe a similar sequence of learning could be beneficial for AI systems that do large-scale learning on their own. By starting with simplified texts of known quality, we think we can improve their comprehension skills to the point where they can successfully learn from "street" sources, like the Wikipedia.

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 provides a parameterized questions interface that enables trainers to quiz the system.

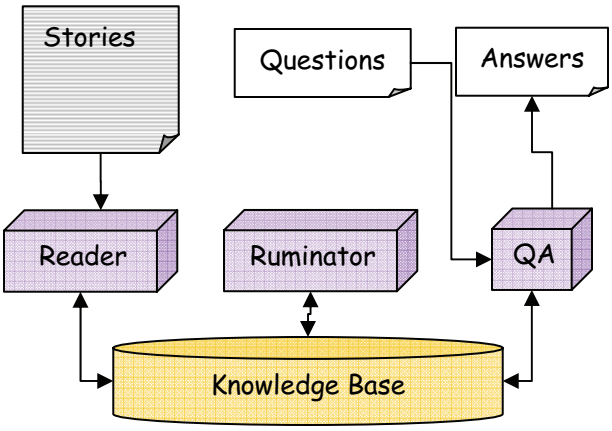

Figure 1: Learning Reader Architecture

While our goal is open-domain learning by reading, to drive our effort we have chosen an area which is extremely broad and reasonably subtle: world history. Our corpus currently consists of 62 stories (956 sentences) about the Middle East, including its geography, history, and some information about current events. All the examples and experiments described in this paper are based on system performance with this corpus.

We begin by describing Learning Reader, starting with the overall architecture and how texts were simplified. We

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

then describe how each component works and provide quantitative data concerning their performance. Then we discuss two system-level experiments, which provide evidence that the Learning Reader prototype can indeed learn by reading. Finally, we discuss some closely related efforts and future work.

Learning Reader: The System

We have architected Learning Reader to be run either as a stand-alone system on a single CPU, or as a set of agents on a cluster, each on its own CPU. The agents are implemented using the agent software developed for Companion cognitive systems (Forbus & Hinrichs, 2004), which communicate via KQML. In both configurations, the knowledge base is used as a blackboard for communication between components. In the stand-alone configuration, it is literally a shared KB in the same Lisp environment. In the multi-agent configuration, each agent has a copy of the KB, and all relevant changes made by an agent to its copy of the KB are propagated to the other agents. A web-based interface enables interactive use in either configuration, and scripting facilities support batch-mode experiments. This flexibility has been invaluable in facilitating development and performing experiments.

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.

We describe each of the components in turn next.

The Reader

The primary goal of the Reader is to identify quickly and accurately what knowledge an input text is commenting on. For any given text, the rest of the system needs to know what's old knowledge and what's new. 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 an `Event` and `CityOfBaghdad` is a `GeographicalRegion`. Matching this phrasal pattern identifies a reference to the conceptual assertion

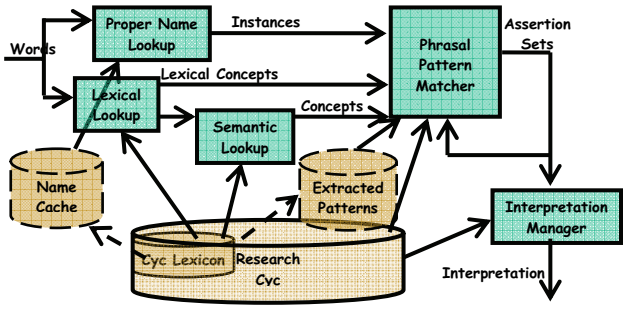

Figure 2: The Reader in Learning Reader

`(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 for `?event`. If none is found, a Skolem constant will be generated. Figure 2 shows how DMAP-based reading is tightly integrated with the central LR knowledge base and reasoning engine at every step in the process.

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:

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

Average results / text	1K Patterns	28K Patterns
Avg. # sentences	15.4	15.4
Processing time	10.84 s	147.69 s
# Sentences w/ assertions	6.6	7.2
#Assertions found	16.9	22.5

Table 1: DMAP Statistics

```
(isa (DayFn 3 (MonthFn August (YearFn 2005)))
Date)
(eventOccursAt Bombing-653 AlAnbar-ProvinceIraq)
```

```
(isa AlAnbar-ProvinceIraq GeographicalRegion)
(isa Bombing-653 Event)
(isa Bombing-653 Attack)
```

Since DMAP did not know of any 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, 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 28,000 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 `perpretatedBy` 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 can not 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.

The current DMAP-based Reader has been tested on the entire LR corpus of stories, with the complete set of over 28,000 plus phrasal patterns, and a more focused set of a little over 1,000 patterns, to see which variables contribute most to speed and accuracy. (A small subset (50) of these were hand-generated, the rest were automatically translated from linguistic knowledge in the ResearchCyc KB contents.) A summary of its performance so far appears in Table 1. Details on these experiments are being reported elsewhere, and more experiments are needed to identify key bottlenecks and accuracy levels. But this data already provides some interesting information. Increasing the number of phrasal patterns from 1,000 to 28,000 resulted in only a linear degradation in processing speed. We saw, however, a less than linear improvement in corpus coverage. The average number of sentences per story for which one or more assertions were recognized went from 6.6 out of 15.4 to 7.2. The average number of assertions per story went from 16.9 to 22.5. This is likely due to the fact that the original set of 1,000 patterns was selected to provide the most coverage on our particular test corpus, and the remaining 27,000 provide a smaller contribution to covering this corpus. There was however a significant increase in the types of predicates in the KB reachable by phrasal patterns. Of the approximately 9,000 predicates in the Research Cyc ontology (including the bookkeeping and structural predicates), 3% are reachable with the 1,000 phrasal pattern set, while 13% are reachable using the 28,000 phrasal pattern set. A preliminary answer key was also created representing some of the primary assertions we expect from each story. Currently the Reader reproduces 84% of this key (using the 28,000 phrasal pattern set). 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.

The Q/A System

The purpose of the current Q/A system is to provide a means of quizzing the system, to examine what it has learned. Consequently, we are using a simple parameterized question template scheme (cf. Cohen *et al*, 1998) to ask a selection of questions that seem particularly appropriate for the domain we are dealing with. (Planned extensions are described below.) 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?, (10) Where is *<SpatialThing>*?, and (11) What are the goals of *<Country>*?

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[2]. 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). We further restrict axioms to Horn clauses. These restrictions reduce completeness, but help ensure that reasoning remains tractable. We have experimented with two types of axioms in the partitions. The first is a simple exploitation of inheritance among relations (i.e., the `specPred` hierarchy in the Cyc ontology). This is the chainer used during Q/A, consisting of 787 axioms. A more complex chainer is used during rumination, as described below.

The Ruminator

Inference during reading tends to be focused, following the most probable paths required to make sense of the text. But people seem to also learn by later reflecting upon what they have read, connecting it more deeply to what they

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

already know and pondering its implications. The Ruminator models this kind of off-line mulling of new material. The operation of the Ruminator can be divided into three phases: *Elaboration, question generation,* and *question processing.* We discuss each in turn.

Elaboration: The input to the Ruminator is a case representing a snippet as understood by the Reader. 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 from the 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 about the story. 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 query templates that are applied to each event to generate one set of questions. These are the same queries that are used for Q/A, as noted above. 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 chosen based on the topic given for the text snippet, and 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), a model of 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. By running SME on the generalizations constructed about a topic, we get questions that reflect the system's experience with that topic[3]. 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.

For example, in the experiment described below, 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: Recall that the chainer used for Q/A is designed to provide rapid, on-line performance. Rumination, being an off-line process, can expend more resources, so the chainer for rumination is much larger, drawing on a larger portion of the KB. However, rumination must still be relatively efficient, in order to handle large bodies of text. Consequently, we also developed techniques for automatically extracting Horn clauses from relevant subsets of the knowledge base, and automatically analyzing them for efficiency.

The extraction process is guided by the structure of Learning Reader. That is, we ideally want to prove any and all queries that can be generated via Q/A, using whatever kinds of statements appear in the KB currently, as well as whatever kinds of statements can be generated by the Reader and by the Ruminator. Thus an analysis of the KB contents, to ascertain what is currently available, plus a structural analysis of the phrasal patterns of the Reader, to ascertain the kinds of statements that could be later learned via reading, guides the process of what axioms to extract. In essence, for each query pattern, we start with the set of axioms that can prove that pattern. The KB axioms are typically not Horn, so we translate them into clausal form 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. If they are not, then the failed antecedents are examined to see there are Horn clauses that could prove them. This process continues for a maximum depth (currently 3), filtering out any rules that have antecedents that will not be derivable within that boundary. Rules that map from `specPred`s to the query predicates are included, up to a depth of 6, as opposed to the unlimited `specPred` recursion used in the Q/A chainer. Other recursive clauses are eliminated for performance reasons. Further automatic static analysis is done to eliminate reasoning bottlenecks[4], which can speed inference by as much as a factor of 70, while dropping completeness by only a few percent.

Two of the three sources of questions we use are non-deductive, so it is possible to generate questions that

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

[4] This process is described in a separate paper in preparation.

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[5]. As with Q/A, we use restricted inference to attempt to answer the questions that seem to make sense. Answers, when found, are stored in the conceptual model.

As can be seen from the statistics above, 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 they have been answered. We plan to allow the Ruminator to ask its trainers a limited number of questions per story, so we have been exploring how to prioritize questions.

System-Level Experiments

Each of the components in Learning Reader has novel aspects, but how well do they all work together?

Does Reading lead to better Q/A results? To examine this question, we used the Reader in batch mode, reading all 62 stories, one snippet at a time, as discussed above. The Ruminator was not used, in order to focus on what was gained by Reader over what was in the KB originally. Questions relevant to the stories were generated by finding all entities in the KB post-reading that could be used in the parameterized questions. Before reading, Q/A answered 87 questions (10%), and after reading, 320 questions (37%). This indicates that Reader is indeed generating new facts which can be used by Q/A to answer questions. This is especially impressive given that the Q/A system cannot even ask about most of the kinds of facts that the system can potentially produce (e.g., `deathToll` in the earlier example).

Does Rumination lead to better Q/A results? We have examined this question by running the Ruminator in batch mode, over all of the stories that had been read by the Reader in the prior experiment. We used the same Q/A rules to ask the same questions. Two conditions were tested: *Deductive Rumination* (DR) only included facts derived via deductive inference from the Ruminator's chainer. *Promiscuous Conjecture Acceptance* (PCA) also included inferences derived via analogy with prior cases, which were simply accepted as true if they did not introduce new individuals (i.e., no analogy skolems). In the DR condition, the number of Q/A questions answered rose to 434 (50% of total, an increase of 35%). In PCA, 525 (60%) of the questions were answered, an increase of 64% over no rumination, and an increase of 21% over the DR condition.

It is crucial, of course, to consider accuracy. We manually scored all answers to determine this. Q/A before reading was 100% accurate. Only one error occurred in Q/A after reading, leading to an accuracy of 99.69%. The DR condition was similarly precise, increasing the number of errors to only 3, or 99.31% accuracy. The PCA condition, as one might expect, had substantially more errors, 48, dropping accuracy to 90.84%. Put another way, of the 90 additional answers provided by PCA over DR, 50% of them were incorrect. Given that the analogical inferences were simply accepted without further scrutiny in the PCA condition, this is actually a higher accuracy than we expected. These results are summarized in Table 2.

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 + Promiscuous Conjecture Acceptance	525	60%	48	90.8%

Table 2: Summary of System-Level Experiments

System-level issues: The presence of noise in learned knowledge is perhaps one of the key issues in learning by reading. There are three sources of noise: Errors in the initial knowledge base, imperfect understanding in the Reader, and conjectures inappropriately accepted during Rumination. While not all of the first two kinds of problems are necessarily caught by Q/A, given the limited number of patterns used, it can be a very useful filter. 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. These inconsistencies were caught when filtering questions to detect if they were inappropriate. This detection of inappropriate self-questions could be used as evidence of an earlier misunderstanding, and we plan on modifying 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 with it, providing the potential to track down such misunderstandings and correct them.

These results bring into sharp relief a fundamental problem 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.

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

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 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.

Discussion

We have described Learning Reader, a prototype system that learns by reading simplified texts. While Learning Reader is very much in its early stages, we believe the results shown here indicate great promise, both as a method for extending knowledge bases and for a potential computational model for how people learn by reading.

There are several directions we plan to pursue next. First, we plan to greatly 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, we are adding a process to the Ruminator which scrutinizes newly learned knowledge for errors, in order to detect both KB errors and reading errors, and to more safely use non-deductive rumination strategies. We also 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. Finally, as the capabilities of the system grow, we plan on comparing its behavior to that of people, by for example giving them both the same sequence of texts and comparing the conclusions (and misconceptions) that arise from each.

Acknowledgements

This research was supported by a seedling grant from the Information Processing Technology Office of DARPA.

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

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*

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*