Collecting Semantics in the Wild: The Story Workbench

Mark Alan Finlayson
markaf@mit.edu
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
32 Vassar Street, Cambridge, MA 02139 USA

Abstract

Analogical reasoning is crucial to robust and flexible high-level cognition. However, progress on computational models of analogy has been impeded by our inability to quickly and accurately collect large numbers (100+) of semantically annotated texts. The Story Workbench is a tool that facilitates such annotation by using natural language processing techniques to make a guess at the annotation, followed by approval, correction, and elaboration of that guess by a human annotator. Central to this approach is the use of a sophisticated graphical user interface that can guide even an untrained annotator through the annotation process. I describe five desiderata that govern the design of the Story Workbench, and demonstrate how each principle was fulfilled in the current implementation. The Story Workbench enables numerous experiments that previously were prohibitively laborious, of which I describe three currently underway in my lab.

Analogical reasoning underlies many important cognitive processes, including learning, categorization, planning, and natural language understanding (Gentner, Holyoak, and Kokinov 2001). It is crucial to robust and flexible high-level cognition. Despite great strides early in the computational understanding of analogical reasoning (Gick and Holyoak 1980; Winston 1980; Gentner 1983; Falkenhainer, Forbus, and Gentner 1989; Forbus, Gentner, and Law 1994), recent progress has been slow. Most computational models of analogy require semantic knowledge as input, supplied as semantically annotated texts. Historically, as the models became more complex, vetting them required ever larger sets of annotations, of greater detail and complexity. It is the assembly of these sets that has become a major bottleneck to progress. The sets should contain hundreds of annotations of sufficient richness, must have high inter-annotator agreement, and need to be collected quickly and without prohibitive expense. Manual assembly of such sets is costly, time-consuming, and error-prone. Automatic annotation systems also provide no relief: they lack coverage, are often imprecise or inaccurate, and are in general unable to provide the full scope of annotations required. This data-collection bottleneck has seriously impaired progress in the computational understanding of analogical reasoning, and a solution is needed if progress is to resume at a reasonable pace.

The Story Workbench

The Story Workbench is a tool that facilitates the collection of semantic annotations of texts. It is similar in appearance to a word processing program: it has an editor area where one can enter and modify text, menus and buttons for performing operations on that text, and a variety of side views showing supplementary information. In great contrast to a word processor, however, it allows the user (whom I will refer to as the *annotator*) to specify what the text 'means' to them, i.e., to annotate it. (What 'means' means will be treated in more detail in the next section.) This annotation is not done from scratch. Rather, the Story Workbench uses off-the-shelf natural language processing (NLP) technology to make a best guess as to the annotations, presenting that guess (if necessary) to the human annotator for approval, correction, and elaboration. This is neither fully manual, nor fully automatic, and thus will be called *semi-automatic* annotation.

If semi-automatic annotation were all the Story Workbench provided, we would still be in a substantially improved position in regard to the data collection bottleneck. We could expect that training professional annotators would be easier, and they would see significant improvements in speed and agreement of their annotations. Nevertheless, the Story Workbench as so far described does not take us all the way to our goal, namely, the ability to collect semantic annotations from *untrained* subjects. This ability cannot be overemphasized, and is what separates the Story Workbench from other annotation tools. Collecting annotations from untrained subjects realizes at least three important benefits. First, the annotations are better insulated from the charge that they have been tailored to produce the desired experimental result – see (Chalmers, French, and Hofstadter 1992). Second, the work of creating the annotations is transferred from a few (the experimenters) to many (the untrained subjects), and so large gains in throughput can be achieved by merely adding more subjects, rather than engaging in expensive and time-consuming training. Finally, given only the tool, annotations can be easily repeated by others, allowing the experimental results to be examined for replicability.

Using a programmed tool to assist in annotation is not

a new idea, and it is worthwhile to briefly mention several intellectual precursors to this work. The speech community has long used annotation tools to assemble their corpora (Bird and Harrington 2001). There are freely available annotation tools such as GATE (Cunningham et al. 2002), Ellogon (Petasis et al. 2002), and VisualText[1] that allow unrestricted text annotation. The most important difference between these platforms and the Story Workbench is that they are geared for researchers, and are not suitable for use by untrained subjects. They are complicated tools that take a significant amount of training and background knowledge to use. The intent of the Story Workbench is to allow nearly any subject to provide annotations, by hiding all the complex machinery behind an intuitive and user-friendly GUI.

Motivating the Story Workbench

Let us look at some specific examples of semantic annotations so we can better understand where the difficulty lies. As previously stated, I strive for the assembly of a reasonably large (100+) set of semantically annotated texts, produced by untrained annotators. My use of the word *annotation* is the same as in corpus linguistics, in that it covers "any descriptive or analytic notations applied to raw language data" (Bird and Liberman 2001). By *semantically annotated*, I mean the following: Given a natural language text, which is a specific sequence of characters, including all symbols and whitespace, a semantic annotation will be the assignment to a subset of those characters another string of characters that conforms to a defined format.[2] The format will be called the *representation*, and the second string will be called the *annotation* or the *description*. Thus, the semantic annotation of a text requires (1) the definition of a set of representations (the aspects of the meaning we are interested in annotating) along with (2) the creation of a set of descriptions in those representations that are attached to specific points in the text (the pieces of meaning).

Consider the following text:

```
John kissed Mary.
```

There are 17 characters, including two spaces and one period. I refer to subsets of characters by referring to their indices, starting at 0. The first letter, *J*, spans indices [0,1], and the first word, *John*, spans indices [0,4].[3] Consider a few different aspects of this text that must be made explicit. Each open class word can be annotated with the appropriate definition from the WordNet electronic dictionary, using the WordNet sense key to identify the definition (Fellbaum 1998). Thus the following string might be attached to the word *kissed* spanning `[5,11]`:

```
kiss%2:35:00::
```

Each sentence in the text can have a Penn-Treebank-style syntactic analysis(Marcus, Marcinkiewicz, and Santorini 1993). Thus we might attach the following description to the text span `[0,17]`:

```
(S (NP (NN John)) (VP (VBD kissed) (NN
Mary)) (.  .))
```

For analogy research, we usually need a logical-form-like annotation of the text, which marks events with their causal and temporal relationships, along with the people, places, and things involved. Figure 1 shows such an annotation, the 1979 Chinese invasion of Vietnam, written as a graph.[4] Annotations such as this can be instantiated in a set of strings much like the syntactic parse tree above, using three different representations, one to list the the different entities and where they appear in the text, one to list the events and their participants, and one to list the causal relationships between the events. From the size and complexity of this graph, which was produced from a mere 13 sentences of controlled English using a rule-based semantic parser, one can begin to appreciate the scope of the problem when each sentence is of a normal complexity, each text is hundreds of sentences long, and the size of corpus runs into the hundreds of texts. Manually annotating such a corpus, and doing it well, is daunting to say the least.

Automatic Annotation

Automatically extracting meanings such as those illustrated in Figure 1 is beyond the reach of the current state of the art. Happily, the automatic extraction of certain specialized aspects of meaning are significantly advanced. For example, assigning a part of speech to each token in a text can be done with extremely high accuracy by a statistical part of speech tagger (Toutanova et al. 2003). Similarly, statistical parsers can provide syntactic analyses of sentences at a lesser, but still good, accuracy (Klein and Manning 2003). And recently, there has been encouraging work on the use of statistically-trained taggers to assign word senses to words (Agirre and Edmonds 2007) and identify and assign arguments to verbs (Pradhan et al. 2005).

The advantages of these automatic techniques are that they are fast and consistent. Even the most complex sentence can be parsed on a fast computer in a few seconds, and almost all techniques are deterministic, in that the same input produces the same output.[5] They also excel at producing well-formatted results (even if the results are not *well-formed*). Unfortunately, they usually lack coverage, are prone to significant error, and cannot produce all the sorts of annotations needed.[6] Furthermore, automatic techniques currently do not allow us to investigate the variation of human understanding of the same text. Given a text on, say,

[1] http://www.textanalysis.com

[2] It is tempting to think that annotations could be assigned to subsets of *tokens* rather than subsets of characters. This, however, is problematic because tokens themselves are annotations, and could potentially be ambiguous. Is the word *don't* one token or two? How about the proper noun *New York City*? Character subsets are unambiguous, and thus form the foundation on which all other annotations are built.

[3] Where `[x,y]` has the natural interpretation, namely, the span of characters in the text that starts at index `x` and ends at index `y`.

[4] See (Finlayson and Winston 2006) for full details.

[5] With caveats; see (Fong and Berwick 2008).

[6] For example, while there are wide-coverage, high-accuracy syntactic parsers, available logical-form parsers have poor coverage and even worse accuracy.

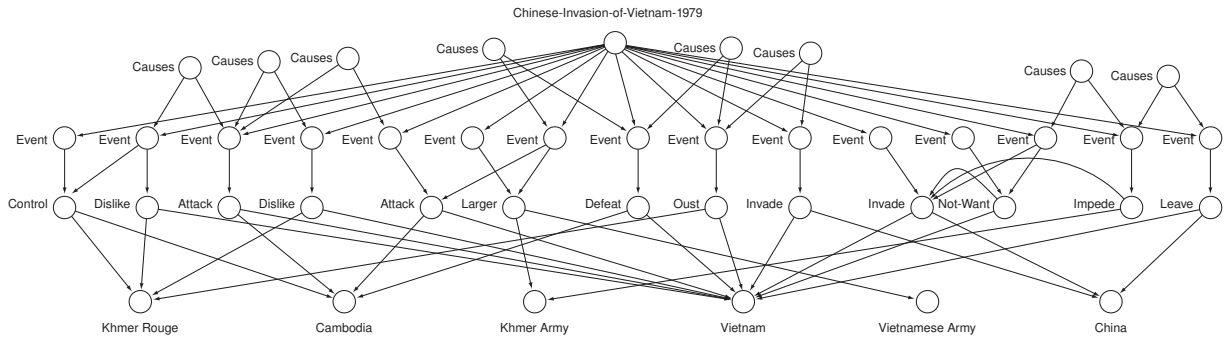

Figure 1: Graphical interpretation of a Logical-Form-like Semantic Annotation of the Chinese-Vietnamese War of 1979. Events are ordered in temporal sequence from left to right.

the ecology of a forest, suppose we want to annotate the understanding of a trained Ecologist, a layman, and perhaps a Native American living on a nearby reservation? Automatic techniques currently have no purchase on this problem.

Manual Annotation

Manual annotation, in contrast, relies on highly-trained human annotators' own natural intuition to understand a text. The benefit is that humans can create annotations that we cannot yet create automatically. But highly-trained human annotators have numerous problems. They are expensive. Training is time-consuming and complicated: they must understand the representations they will be annotating and the constraints those representations obey, and they must be trained to recognize and handle edge-cases. They must translate their understanding into the formal structures needed by the computer, a task at which people are notoriously bad.[7] Finally, if we restrict ourselves to highly-trained annotators, we significantly reduce our subject pool and potentially introduce significant biases into our data.

Semi-Automatic Annotation

The Story Workbench's semi-automatic approach combines the best features of both manual and automatic annotation. It uses automatic techniques where they are available. If the techniques have high-accuracy, these can be used with little to no supervision; otherwise, their results can be presented to the human annotator for correction. Elsewhere, automation is used to tackle the tasks that are hard for people but easy for computers: checking long lists of formatting constraints, or generating possible answers by searching large solution spaces. Where the computer has absolutely no purchase on a problem, the human can take over completely, using special editors to specify the meaning.

Central to this approach is a sophisticated graphical user interface that gives annotators the right information at the right times and does so in an intuitive and clear manner. Careful engineering is required to get the right "impedance match" between annotators and the computer. Many users of the modern computer are familiar with the devices that constitute a user-friendly graphical interface: a wide range of graphical modalities for conveying information, such as color, highlighting, movement, shape, and size; evocative icons; automatic correction of formats and well-known errors; quick feedback loops that show errors and warnings as they occur; wizards and dialogs that guide and constrain repetitive or formulaic tasks; solutions to common problems encapsulated in their own functions and offered for execution at the appropriate times. With such a tool, we could collect data from untrained subjects. With little more than literacy and the most basic of computer skills, almost anyone can give us the data we need.

Case Studies

For those still not convinced of the utility of the Story Workbench, I briefly review three research programs that would have benefited from such a tool, or suffered for lack of it. The first is drawn from the computational study of analogy, the second is an example of how to design a representation, and third concerns the process of constructing a corpus for computational linguistics. These examples are not meant as criticisms, but rather illustrations of how, if these researchers had had access to a tool like the Story Workbench, certain aspects of the work could have been improved.

Karla the Hawk Studies In the 1980's and early 90's, Gentner, Forbus, and co-workers conducted a series of studies using the so-called Karla the Hawk story set (Gentner and Landers 1985; Rattermann and Gentner 1987; Forbus, Gentner, and Law 1994). These studies examined people's ability to do *analogical* retrieval, that is, retrieve from long-term memory an analogically-related precedent to a given problem or story. The studies had both human experiments and computational models. Study participants read a set of simple stories, the Karla the Hawk stories, written by the experimenters and constructed to have specific story-to-story relationships. Subsequent computer models, including the Structure Mapping Engine and the MAC/FAC retrieval module, used propositional-logic-like semantic an-

[7] Take, for example, generating a syntactic analysis in the Penn-Treebank format considered above. Now imagine writing down using your favorite text editor, a syntactic analysis in that format for this whole page of text. For those who haven't done it before it would, no doubt, take a long time, and the answers, compared across multiple informants, would almost certainly not match well.

notations of the stories. The main concern with the modeling experiments is that the representations of the stories were created by the experimenters themselves. Did these representations really reflect how people understood the stories? Did the experimenters inadvertently code the answer into the stimuli? See (Chalmers, French, and Hofstadter 1992) for a discussion of exactly this. Ideally, the subjects themselves would have constructed the semantic annotations. If the Story Workbench had been available such a method would have been possible.

Development of Rhetorical Structure Theory Discourse coherence is the phenomenon that pieces of a text 'hold together' into structures larger than an individual sentence. In a coherent text sentences may 'follow' from previous sentences, or 'lead to' later sentences. One sentence, for example, may be a *consequence* of another; another may be a *quotation* ascribed to some aforementioned entity. In the 1980's Mann & Thompson (1987) developed a representation scheme, Rhetorical Structure Theory (RST), that, with 23 types of relationships, claimed to account for all the different sorts of discourse coherence. How did they arrive at this representation? They started with a proposal for a small number of relationship types that they thought sufficient, and then examined texts to see if that proposal really 'fit' with the data, and then iterated back and forth. If the Story Workbench had been used, this procedure could have been supplemented with actual annotation of texts. Initial constraints and forms of the representations could be integrated into the tool. As they uncovered different relationships, and found the need for tweaks of the representation scheme, the tool's programming could have been modified. There would have been several advantages to this approach. First, during the development stage, they could have continuously monitored inter-annotator agreement to determine if the representation 'made sense' to the annotators. It would have revealed instances of incompatible interpretations, and places where the theory needed to be tightened up. A converging inter-annotator agreement would also give a sense as to when the representation was 'finished'. Second, such an approach would have produced an implementable specification of the representation, something that few representations actually have. Third, at the end of the development they would have had at least a small body of annotated text, and the tools with which anyone could repeat that annotation.

Annotation of PropBank The Proposition Bank, or PropBank (Palmer, Kingsbury, and Gildea 2005), is a dataset of approximately 112,000 verbs annotated for their semantic arguments that has been recently used to great effect to train statistical semantic role labeling systems. The Story Workbench could have been of great use in constructing such a corpus. First, the annotators began annotating only after significant training and under the guidance of a training manual. The training manuals give conventions that should be followed, the required format, and how to recognize and handle edge cases. All of these could have been instantiated inside the Story Workbench programmatically, and so aided both teaching of and adherence to the rules. Second, what if someone wants to replicate the PropBank, or construct a PropBank-style annotation on another set of texts? If the Story Workbench had been used, the tool would have been immediately available for such a replication.

Each of these three examples are specific instances of a more general class of endeavors that would see improvement with the Story Workbench. Studies similar to the Karla the Hawk studies were carried out by Thagard *et al.* on Aesop's fables (Thagard et al. 1990). Hundreds of representations have been developed for use in computational models across Cognitive Science and Artificial Intelligence, and many could have benefitted in the same way as Rhetorical Structure Theory (Markman 1999). The Proposition Bank could be replaced in the above example, with no loss of generality, by WordNet, Semcor (Fellbaum 1998), the Penn Treebank (Marcus, Marcinkiewicz, and Santorini 1993), or the Berkeley FrameNet (Fillmore, Johnson, and Petruck 2003), to name only a select few.

Design Principles, Desiderata and Implementation

What principles and desiderata have guided the design and implementation of the Story Workbench? When possible, I have followed three principles:

1. Build on top of popular tools with large user and developer communities
2. Use open-source and freely-available programming libraries
3. Adopt widely-used and well-documented standards

In application design things change quickly. Today's hot new tool or language is tomorrow's ancient history. All three principles help buffer us from the inexorable advances in the state of the art. The more we adhere to these principles, the more likely our tools or languages are to evolve to the next stage, rather than be thrown out and replaced *in toto*. Similarly, standards usually have precise specifications and the backing of some independent organization; they are unlikely to suddenly disappear forever. If the tools and libraries are freely modifiable, then users need not be constrained by the features provided in the initial implementation, but can implement their own. This is especially important if the tool becomes popular. Finally, a benefit of all three principles is that the more popular a tool, library, or standard, the more resources (such as reference books or tutorials) are available to help users and developers.

In addition to the three principles, there are at least five desiderata:

1. The storage format should be human-readable and tool-independent
2. The storage format should be modular and extensible
3. The tool framework should be highly functional, modular, and extensible
4. The tool should not commit to a specific operating system
5. The tool should not commit to specific NLP technologies

```
<?xml version="1.0" encoding="UTF-8" ?>
<story>
  <rep id="edu.mit.story.char">
    <desc id="0" len="31  off="0">John kissed Mary. She blushed.</desc>
  </rep>
  <rep id="edu.mit.parsing.token">
    <desc id="19" len="4" off="0">John</desc>
    <desc id="47" len="6" off="5">kissed</desc>
    <desc id="61" len="4" off="12">Mary</desc>
    <desc id="71" len="1" off="16">.</desc>
    <desc id="92" len="3" off="19">She</desc>
    <desc id="120" len="7" off="23">blushed</desc>
    <desc id="121" len="1" off="30">.</desc>
  </rep>
  <rep id="edu.mit.parsing.sentence">
    <desc id="94" len="17" off="0" />
    <desc id="122" len="12" off="19" />
  </rep>
  <rep id="edu.mit.parsing.parse">
    <desc id="70" len="17" off="0">(S (NP (NNP John)))(VP (VBD kissed))(NP (NNP Mary))))(. (.)))</desc>
    <desc id="125" len="12" off="19">(S (NP (PRP She)))(VP (VBD blushed)))(. .)))</desc>
  </rep>
</story>
```

Figure 2: Example Story Data File in XML Format

Data Format

The storage format should be human-readable and tool-independent Perhaps the most important design decision is the format of the data that will be produced by the Story Workbench. This data, if useful at all, will long outlive any implementation. Thus it is extremely important that the data format be based on a well-documented and easily-understood standard. It is also imperative that the specification of the format be independent of any implementation. Years later a user needs to be able to understand the data without the benefit of software, and, in principle, recreate an interface to it from scratch.

The data should also be human-readable and, thus, in principle, editable using only a text editor. In the course of research small tweaks need to be made and data needs to be reviewed at a glance. Sometimes a working tool is not available to look at the data. This rules out binary formats and restricts us to something based on a standard character set.

To satisfy this desideratum, I use the eXtensible Markup Language (XML) (Bray et al. 2006) as the base format for the files. XML satisfies both of the above properties: it is a widely-supported, easily-understood international standard, and it is human readable and editable by text editor alone. XML includes in its definition the XML Schema Definition (XSD) language, for formally describing the elements of XML files. One can check to see if a particular XML file is well-formatted with respect to a particular XSD file. The XSD files included standardized places for including documentation and descriptions of the meaning and function of different pieces of the XML document. By choosing XML, the product of our tool, the data, will consist of a number of XML files (the corpus), plus one (or a small number) of XSD files that describe and document the format of the data. This collection of files is then suitable for archiving.

The storage format should be modular and extensible The second important constraint on the data format is that it must admit to additions of as-yet-unforeseen pieces, in other words, it must be *extensible*. I discussed above how there are many different aspects of meaning to a particular text. It is not possible to code them all at once, especially at our current level of knowledge. We should be able to annotate what we need right now, and then, if our needs reveal another aspect of meaning that needs to be layered on top of what has already been done, the format should be able to accommodate this. This holds true, also, for end-users: if they want to take the tool and annotate a new aspect of meaning, they should be able to do this without redesigning the whole format.

Figure 2 is a concrete example of an actual data file, with some of the less important aspects trimmed for ease of exposition. The topmost tag has a number of representation blocks as children. The first representation block is the text itself, and is the single required representation. The following representation blocks are the different layers of meaning that have been annotated for this text. The second representation block is the sentence representation, indicating the extent of each sentence, the next block gives a syntactic analysis of each sentence in a Penn Treebank style format, and the final block is the WordNet word sense, where the data string is the sense key of the referenced synset. Each representation block has zero or more description children that each contain an atomic piece of data. Each description has a number of different XML attributes, the three most important of which are illustrated in the figure. The first is a unique identification number that can be used to refer to that description by other descriptions elsewhere in the document. The second and third are the offset and length of the description, and indicate the span of character indices the description covers. Between the opening and closing description tags is a string of characters, or no characters at all, that represent the data for that description. The format of this string is specific to each representation. This format is modular in that blocks (either whole representations, or individual de-

scriptions) can be added or deleted without modifying other blocks.

Application Framework

The tool framework should be highly functional, modular, and extensible Along with modularity of the data format should come modularity of the Story Workbench itself. Since the XML data format outlined above allows anyone to define their own representations, the application framework on which the Story Workbench is based should support the creation of the infrastructure for producing guesses and checking constraints for that representation, as well as any special graphical user interface widgets that are required. These parts should be wrapped in a module that can be distributed by itself, and can be added to any other instance of the Story Workbench with minimal trouble.

The tool should not commit to a specific operating system The Story Workbench should also run the application on many different platforms. There are many popular operating systems in use today, and if the tool is to be widely adopted, it has to run on as many of those as possible.

I have satisfied these two desiderata by choosing the Java programming language (Gosling et al. 2005) and the Eclipse Application Development Framework (Gamma and Beck 2004) as the fundamental building blocks of the Story Workbench. Java is an inherently cross-platform language, and runs on almost every known operating system. It is open-source, freely-available, and in extremely wide use. There are numerous libraries and utilities available for Java, not just for natural language processing, but for other tasks as well. The Eclipse platform satisfies our demands for both functionality and modularity. Eclipse is open-source, freely-available, implemented almost entirely in Java, and has a large base of users and developers who are constantly at work to improve the tool. It brings ready implementations of the many aspects vitally important to a modern graphical application. It is built out of a set of *plugins*, modules that form a coherent piece of code that add a specific functionality on top of the base program. In the context of the Story Workbench, if an end-user wants to add a new representation or a suite of related representations, they would program their code against the Eclipse API and then package it as a plugin, which can then be distributed to and installed by any user of the Story Workbench. Indeed, the Story Workbench itself is only a collection of plugins that can be distributed to and run by any user of the Eclipse platform.

NLP Technologies

The tool should not commit to specific NLP technologies The Story Workbench makes a commitment to the semi-automatic annotation method. It does not commit to any particular type or implementation of the NLP technology it uses to perform guesses for the annotations. In this regard, the tool is technology-agnostic. To be sure, the default distribution provides implementations of specific tools. But the architecture of the tool is such that end-users can easily introduce new or different tools, just as they can introduce new representations, and when constructing a text can choose between all the different methods on hand. Currently the Story Workbench uses the Stanford Natural Language Group's Java tools (Toutanova et al. 2003; Klein and Manning 2003) to do the heavy lifting for low-level syntactic analysis. I plan to add other freely available NLP tools, such as the OpenNLP library and the various tools that back the GATE system (Cunningham et al. 2002).

A screenshot of the Story Workbench is shown in Figure 3. The text is shown in the center editor. In this example, the Workbench is rightly unable to determine with certainty the attachment of preposition phrase in the the sentence *The boy saw the man with the telescope*. In the configuration shown, every prepositional phrase that could potentially be attached elsewhere in the parse tree is flagged. This is displayed as a warning the *Problems* view, and shows the annotator that there is a *Quick Fix*, namely, a routine correction to this condition. In this case, selecting the Quick Fix will show a dialog that allows the annotator to choose the correct attachment of the phrase.

Experiments Planned or Underway

The Story Workbench enables a number of experiments that previously were conceivable, but too laborious to actually conduct. I outline three experiments that are currently underway in my lab. As an aside, it is important to note that once subjects start to use the Story Workbench, the tool itself becomes a part of the experiment. Thus we have to carefully measure how it performs: how quickly, precisely, and accurately subjects can annotate text with the tool. All of the following experiments are predicated on a thorough understanding of the characteristics of the tool with respect to these measures. We are currently conducting studies to measure these factors for specific representations and graphical user interface elements; for example, we are measuring how well subjects can disambiguate word senses using a wizard specifically designed for the purpose. As each layer of representation, and their concomitant tools, are added to the Story Workbench, they must be tested and benchmarked.

First, I am conducting an experiment on analogical retrieval. I am collecting a large number of structured annotations of texts from the domain of political science to test my theory of intermediate features in retrieval, which treats the differences in the way experts and novices retrieve items from memory. The initial experiments I conducted in this domain exposed to me the problem of doing semantic annotation and led directly to the current work on the Story Workbench (Finlayson and Winston 2006). At the time, I annotated 14 stories using a simple rule-based semantic parser that ran on controlled English input. While the results of the study were intriguing, they were not convincing, mainly because we did not have enough data to test our hypothesis. Now that the tool is operational, I will return to collecting a sufficient amount of data to test the theory of intermediate features as it is applied to analogical retrieval.

Second, I am conducting experiments on how culture affects thought. With the Story Workbench, differences in people's interpretations of various texts can be captured, and can be correlated with aspects of their culture of origin. I are currently investigating whether differences in causal attribution along the individualist vs. collectivist divide (Triandis

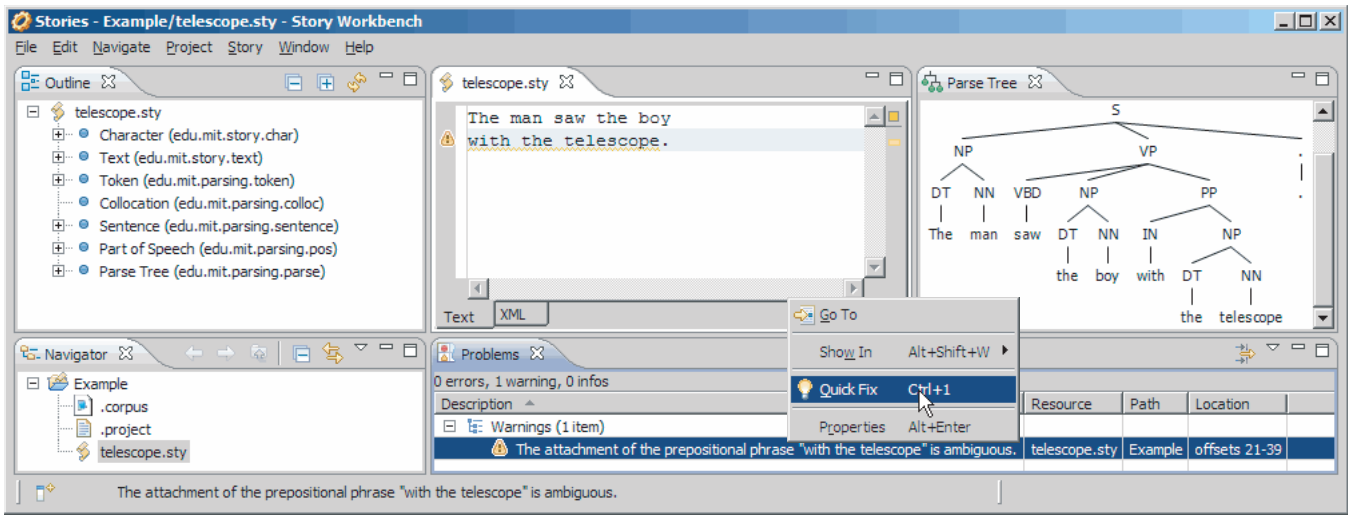

Figure 3: Story Workbench Screenshot

2001) can be detected using data collected from subjects using the Story Workbench.

Finally, I am pursuing, with collaborators in Psycholinguistics, investigation into discourse structure representations. We are annotating a large corpus with which we hope to test falsifiable claims made by certain discourse structure theories about the constraints that apply to certain classes of discourse relationships. See (Wolf and Gibson 2005) for an example. I have designed a number of representations and graphical interfaces for the Story Workbench that allow the annotation of discourse structure; without these tools, the assembly of the discourse annotations would have been insurmountably difficult.

Contributions

The computational understanding of analogical reasoning has been hampered for some time by our inability to quickly and accurately collect large numbers of semantic annotations. I have outlined the manual and automatic approaches and their attendant problems, and have described the semi-automatic approach as an alternate solution. I have proposed that the design of a tool employing the semi-automatic approach should be governed by at least three principles: (1) Build on top of popular tools with large user and developer communities, (2) Use open-source and freely-available programming libraries, and (3) Adopt widely-used and well-documented standards. The desiderata for such a tool are that (1) the storage format should be human-readable and tool-independent, (2) the storage format should be modular and extensible, (3) the tool framework should be highly functional, modular, and extensible, (4) the tool should not commit to a specific operating system, and (5) the tool should not commit to specific NLP technologies. I have demonstrated an implementation of such a tool, called the Story Workbench, that provides the infrastructure that allows us to quickly and accurately collect semantically annotated text. I have outlined several of its key properties, and showed how it satisfies the stated principles and desiderata. Finally, I have described a number of experiments enabled by this tool that are now underway in my lab that, without a tool such as the Story Workbench, would be prohibitively laborious.

Acknowledgements

This work is supported in part by the National Science Foundation under grant number IIS-0413206 and the Air Force Office of Scientific Research under grant number A9550-05-1-0321. Many thanks to Patrick Winston, Whitman Richards, Eliza Chang, Caleb Hug, Michael Klein, and Mac Schwager for their helpful comments during the preparation of this paper.

References

Agirre, E., and Edmonds, P., eds. 2007. *Word Sense Disambiguation*. Text, Speech, and Language Technology. Dordrecht, The Netherlands: Springer.

Bird, S., and Harrington, J. 2001. Special issue: Speech annotation and corpus tools. *Speech Communication* 33:1–174.

Bird, S., and Liberman, M. 2001. A formal framework for linguistic annotation. *Speech Communication* 33:23–60.

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; and Yergeau, F. 2006. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. World Wide Web Consortium (W3C).

Chalmers, D.; French, R.; and Hofstadter, D. 1992. High-level perception, representation, and analogy: a critique of Artificial Intelligence methodology. *Journal of Experimental and Theoretical Artificial Intelligence* 4:185–211.

Cunningham, H.; Maynard, D.; Bontcheva, K.; and Tablan, V. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the Fortieth Annual Meeting of the ACL*, 168–175.

Falkenhainer, B.; Forbus, K. D.; and Gentner, D. 1989. The Structure-Mapping Engine: Algorithm and examples. *Artificial Intelligence* 43:1–63.

Fellbaum, C., ed. 1998. *Wordnet: An Electronic Lexical Database*. Cambridge, MA: MIT Press.

Fillmore, C. J.; Johnson, C. R.; and Petruck, M. R. L. 2003. Background to Framenet. *International Journal of Lexicography* 16:359–361.

Finlayson, M. A., and Winston, P. H. 2006. Analogical retrieval via intermediate features: The goldilocks hypothesis. Technical Report MIT-CSAIL-TR-2006-071, MIT Computer Science and Artificial Intelligence Laboratory.

Fong, S., and Berwick, R. C. 2008. Treebank parsing and knowledge of language: A cognitive perspective. In Love, B. C.; McRae, K.; and Sloutsky, V., eds., *Proceedings of the Thirtieth Annual Meeting of the Cognitive Science Society*, 539–544.

Forbus, K. D.; Gentner, D.; and Law, K. 1994. MAC/FAC: A model of similarity-based retrieval. *Cognitive Science* 19:141–205.

Gamma, E., and Beck, K. 2004. *Contributing to Eclipse: Principles, Patterns, and Plug-ins*. Boston: Addison-Wesley.

Gentner, D., and Landers, R. 1985. Analogical reminding: A good match is hard to find. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, 607–613.

Gentner, D.; Holyoak, K. J.; and Kokinov, B. N., eds. 2001. *The Analogical Mind: Perspectives from Cognitive Science*. Cambridge, MA: MIT Press.

Gentner, D. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive Science* 7:155–170.

Gick, M. L., and Holyoak, K. J. 1980. Analogical problem solving. *Cognitive Psychology* 12:306–355.

Gosling, J.; Joy, B.; Steele, G.; and Bracha, G. 2005. *The Java Language Specification (Third Edition)*. Upper Saddle River, NJ: Addison-Wesley.

Klein, D., and Manning, C. D. 2003. Accurate unlexicalized parsing. In *Proceedings of the Forty-first Meeting of the ACL*, 423–430.

Mann, W. C., and Thompson, S. A. 1987. Rhetorical structure theory: A theory of text organization. Technical Report ISI/RS-87-190, University of Southern California, Information Sciences Institute (ISI).

Marcus, M. P.; Marcinkiewicz, M. A.; and Santorini, B. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* 19:313–330.

Markman, A. B. 1999. *Knowledge Representation*. Mahwah, NJ: Lawrence Erlbaum Associates.

Palmer, M.; Kingsbury, P.; and Gildea, D. 2005. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics* 31:71–105.

Petasis, G.; Karkaletsis, V.; Paliouras, G.; Androutsopoulos, I.; and Spyropoulos, C. D. 2002. Ellogon: A new text engineering platform. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, 72–78.

Pradhan, S.; Hacioglu, K.; Krugler, V.; Ward, W.; Martin, J. H.; and Jurafsky, D. 2005. Support vector learning for semantic argument classification. *Machine Learning* 60:11–39.

Rattermann, M. J., and Gentner, D. 1987. Analogy and similarity: Determinants of accessibility and inferential soundness. In *Proceedings of the Ninth Annual Meeting of the Cognitive Science Society*, 23–35. Lawrence Erlbaum Associates.

Thagard, P.; Holyoak, K. J.; Nelson, G.; and Gochfeld, D. 1990. Analog retrieval by constraint satisfaction. *Artificial Intelligence* 46:259–310.

Toutanova, K.; Klein, D.; Manning, C. D.; and Singer, Y. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the HLT-NAACL*, 252–259.

Triandis, H. C. 2001. Individualism and collectivism: Past, present, and future. In Matsumoto, D., ed., *The Handbook of Culture and Psychology*. Oxford: Oxford University Press. 35–50.

Winston, P. H. 1980. Learning and reasoning by analogy. *Communications of the ACM* 23:689–703.

Wolf, F., and Gibson, E. 2005. Representing discourse coherence: A corpus-based study. *Computational Linguistics* 31:249–287.