A Little Goes a Long Way: Quick Authoring of Semantic Knowledge Sources for Interpretation

Carolyn Penstein Rosé
Carnegie Mellon University
cprose@cs.cmu.edu

Brian S. Hall
University of Pittsburgh
mosesh@pitt.edu

Abstract

In this paper we present an evaluation of Carmel-Tools, a novel *behavior oriented* approach to authoring and maintaining domain specific knowledge sources for robust sentence-level language understanding. Carmel-Tools provides a layer of abstraction between the author and the knowledge sources, freeing up the author to focus on the desired language processing *behavior* that is desired in the target system rather than the linguistic details of the knowledge sources that would make this behavior possible. Furthermore, Carmel-Tools offers greater flexibility in output representation than the context-free rewrite rules produced by previous semantic authoring tools, allowing authors to design their own predicate language representations.

1 Introduction

One of the major obstacles that currently makes it impractical for language technology applications to make use of sophisticated approaches to natural language understanding, such as deep semantic analysis and domain level reasoning, is the tremendous expense involved in authoring and maintaining domain specific knowledge sources. In this paper we describe an evaluation of Carmel-Tools as a proof of concept for a novel *behavior oriented* approach to authoring and maintaining domain specific knowledge sources for robust sentence-level language understanding. What we mean by behavior oriented is that Carmel-Tools provides a layer of abstraction between the author and the knowledge sources, freeing up the author to focus on the desired language processing *behavior* that is desired in the target system rather than the linguistic details of the knowledge sources that would make this behavior possible. Thus, Carmel-Tools is meant to make the knowledge source engineering process accessible to a broader audience. Carmel-Tools is used to author domain specific semantic knowledge sources for the Carmel Workbench (Rosé, 2000; Rosé et al., 2002) that contains broad coverage domain general syntactic and lexical knowledge sources for robust language understanding in English. Our evaluation demonstrates how Carmel-Tools can be used to interpret sentences in the physics domain as part of a content-based approach to automatic essay grading.

Sentence: The man is moving horizontally at a constant velocity with the pumpkin.

Predicate Language Representation:
((velocity id1 man horizontal constant non-zero)
(velocity id2 pumpkin ?dir ?mag-change ?mag-zero)
(rel-value id3 id1 id2 equal))

Gloss: *The constant, nonzero, horizontal velocity of the man is equal to the velocity of the pumpkin.*

Figure 1: Simple example of how Carmel-Tools builds knowledge sources capable of assigning representations to sentences that are not constrained to mirror the exact wording, structure, or literal surface meaning of the text.

While much work has been done in the area of robust semantic interpretation, current *authoring tools* for building semantic knowledge sources (Cunningham et al., 2003; Jay et al., 1997) are tailored for information extraction tasks that emphasize the identification of named entities such as people, locations, and organizations. While regular expression based recognizers, such as JAPE (Cunningham et al., 2000), used for information extraction systems, are not strictly limited to these standard entity types, it is not clear how they would handle concepts expressing complex relationships between entities, where the complexity in the meaning can be real-

ized with a much greater degree of surface syntactic variation. Outside of the information extraction domain, a concept acquisition authoring environment called SGStudio (Wang and Acero, 2003) offers similar functionality to JAPE for building language understanding modules for dialogue systems, with similar limitations. Carmel-Tools is more flexible in that it allows a wider range of linguistics expression that communicate the same idea to match against the same pattern. It accomplishes this by inducing patterns that match against a deep syntactic parse rather than a stream of words, in order to normalize as much surface syntactic variation as possible, and thus reducing the number of patterns that the learned rules must account for. Furthermore, Carmel-Tools offers greater flexibility in output representation than the context-free rewrite rules produced by previous semantic authoring tools, allowing authors to design their own predicate language representations that are not constrained to follow the structure of the input text (See Figure 1 for a simple example and Figure 2 for a more complex example.). See Section 3 and (Rosé, 2000; Rosé et al., 2002) for more details about CARMEL's knowledge source representation.

Note that the predicate language representation utilized by Carmel-Tools is in the style of Davidsonian event based semantics (Hobbs, 1985). For example, in Figure 1 notice that the first argument of each predicate is an identification token that represents the whole predicate. These identification tokens can then be bound to arguments of other predicates, and in that way be used to represent relationships between predicates. For example, the `rel-value` predicate expresses the idea that the predicates indicated by `id1` and `id2` are equal in value.

While language understanding systems with this style of analysis are not a new idea, the contribution of this work is a set of authoring tools that simplify the semantic knowledge sources authoring process.

2 Motivation

While the technology presented in this paper is not specific to any particular application area, this work is motivated by a need within a growing community of researchers working on educational applications of Natural Language Processing to extract detailed information from student language input to be used for formulating specific feedback directed at the details of what the student has uttered. Such applications include tutorial dialogue systems (Zinn et al., 2002; Popescue et al., 2003) and writing coaches that perform detailed assessments of writing content (Rosé et al., 2003; Wiemer-Hastings et al., 1998; Malatesta et al., 2002) as opposed to just grammar (Lonsdale and Strong-Krause, 2003), and provide detailed feedback rather than just letter grades (Burstein et al., 1998; Foltz et al., 1998). Because of the important role of language in the learning process (Chi et al., 2001), and because of the unique demands educational applications place on the technology, especially where detailed feedback based on student language input is offered to students, educational applications present interesting opportunities for this community.

The area of automated essay grading has enjoyed a great deal of success at applying shallow language processing techniques to the problem of assigning general quality measures to student essays (Burstein et al., 1998; Foltz et al., 1998). The problem of providing reliable, detailed, content-based feedback to students is a more difficult problem, however, that involves identifying individual pieces of content (Christie, 2003), sometimes called "answer aspects" (Wiemer-Hastings et al., 1998). Previously, tutorial dialogue systems such as AUTO-TUTOR (Wiemer-Hastings et al., 1998) and Research Methods Tutor (Malatesta et al., 2002) have used LSA to perform an analysis of the correct answer aspects present in extended student explanations. While straightforward applications of bag of words approaches such as LSA have performed successfully on the content analysis task in domains such as Computer Literacy (Wiemer-Hastings et al., 1998), they have been demonstrated to perform poorly in causal domains such as research methods (Malatesta et al., 2002) and physics (Rosé et al., 2003) because they base their predictions only on the words included in a text and not on the functional relationships between them. Key phrase spotting approaches such as (Christie, 2003) fall prey to the same problem. A hybrid rule learning approach to classification involving both statistical and symbolic features has been shown to perform better than LSA and Naive Bayes classification (McCallum and Nigam, 1998) for content analysis in the physics domain (Rosé et al., 2003). Nevertheless, trained approaches such as this perform poorly on low-frequency classes and can be too coarse grained to provide enough information to the system for it to provide the kind of detailed feedback human tutors offer students (Lepper et al., 1993) unless an extensive hierarchy of classes that represent subtle differences in content is used (Popescue et al., 2003). Popescue et al. (2003) present impressive results at using a symbolic classification approach involving hand-written rules for performing a detailed assessment of student explanations in the Geometry domain. Rule based approaches have also shown promise in non-educational domains. For example, an approach to adapting the generic rule based MACE system for information extraction has achieved an F-measure of 82.2% at the ACE task (Maynard et al., 2002). Authoring tools for speeding up and simplifying the task of writing symbolic rules for assessing the content in student essays would make it more practical to take advantage of the benefits of rule based assessment approaches.

3 Carmel-Tools Interpretation Framework

Sentence: During the fall of the elevator the man and the keys have the same constant downward acceleration that the elevator has.

Predicate Language Representation:
((rel-time id0 id1 id2 equal)
(body-state id1 elevator freefall)
(and id2 id3 id4)
(rel-value id3 id5 id7 equal)
(rel-value id4 id6 id6 equal)
(acceleration id5 man down constant non-zero)
(acceleration id6 keys down constant non-zero)
(acceleration id7 elevator down constant non-zero))

Gloss: *The elevator is in a state of freefall at the same time when there is an equivalence between the elevator's acceleration and the constant downward nonzero acceleration of both the man and the keys*

Figure 2: Example of how deep syntactic analysis facilitates uncovering complex relationships encoded syntactically within a sentence

One of the goals behind the design of Carmel-Tools is to leverage off of the normalization over surface syntactic variation that deep syntactic analysis provides. While our approach is not specific to a particular framework for deep syntactic analysis, we have chosen to build upon the publicly available LCFLEX robust parser (Rosé et al., 2002), the CARMEL grammar and semantic interpretation framework (Rosé, 2000), and the COMLEX lexicon (Grishman et al., 1994). This same broad coverage, domain general interpretation framework has already been used in a number of educational applications including (Zinn et al., 2002; VanLehn et al., 2002).

Syntactic feature structures produced by the CARMEL grammar normalize those aspects of syntax that modify the surface realization of a sentence but do not change its deep functional analysis. These aspects include tense, negation, mood, modality, and syntactic transformations such as passivization and extraction. Thus, a sentence and it's otherwise equivalent passive counterpart would be encoded with the same set of functional relationships, but the passive feature would be negative for the active version of the sentence and positive for the passive version. A verb's direct object is assigned the `obj` role regardless of where it appears in relation to the verb. Furthermore, constituents that are shared between more than one verb, for example a noun phrase that is the object of a verb as well as the subject of a relative clause modifier, will be assigned both roles, in that way "undoing" the relative clause extraction. In order to do this analysis reliably, the component of the grammar that performs the deep syntactic analysis of verb argument functional relationships was generated automatically from a feature representation for each of 91 of COMLEX's verb subcategorization tags (Rosé et al., 2002). Altogether there are 519 syntactic configurations of a verb in relation to its arguments covered by the 91 subcategorization tags, all of which are covered by the CARMEL grammar.

CARMEL provides an interface to allow semantic interpretation to operate in parallel with syntactic interpretation at parse time in a lexicon driven fashion (Rosé, 2000). Domain specific semantic knowledge is encoded declaratively within a meaning representation specification. Semantic constructor functions are compiled automatically from this specification and then linked into lexical entries. Based on syntactic head/argument relationships assigned at parse time, the constructor functions enforce semantic selectional restrictions and assemble meaning representation structures by composing the meaning representation associated with the constructor function with the meaning representation of each of its arguments. After the parser produces a semantic feature structure representation of the sentence, predicate mapping rules then match against that representation in order to produce a predicate language representation in the style of Davidsonian event based semantics (Davidson, 1967; Hobbs, 1985), as mentioned above. The predicate mapping stage is the key to the great flexibility in representation that Carmel-Tools is able to offer. The mapping rules perform two functions. First, they match a feature structure pattern to a predicate language representation. Next, they express where in the feature structure to look for the bindings of the uninstantiated variables that are part of the associated predicate language representation. Because the rules match against feature structure patterns and are thus above the word level, and because the predicate language representations associated with them can be arbitrarily complex, the mapping process is decompositional in manner but is not constrained to rigidly follow the structure of the text.

Figure 2 illustrates the power in the pairing between deep functional analyses and the predicate language representation. The deep syntactic analysis of the sentence makes it possible to uncover the fact that the expression "constant downward acceleration" applies to the acceleration of all three entities mentioned in the sentence. The coordination in the subject of the sentence makes it possible to infer that both the acceleration of the man and of the keys are individually in an equative relationship with the acceleration of the elevator. The identification token of the `and` predicate allows the whole representation of the matrix clause to be referred to in the `rel-time` predicate that represents the fact that the equative relationships hold at the same time as the elevator is in a state

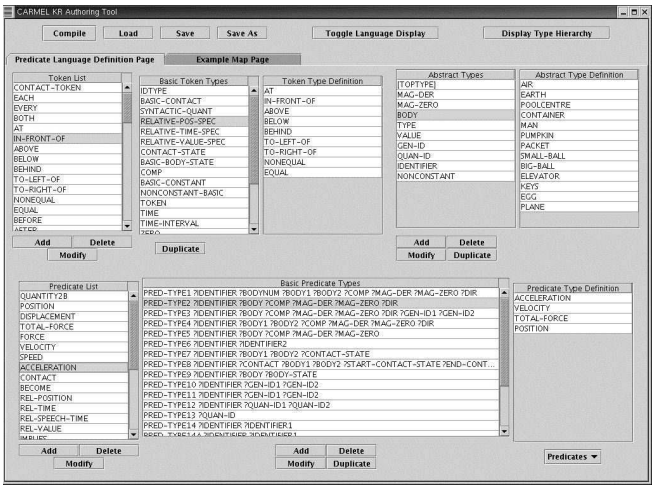

Figure 3: Predicate Language Definition Page

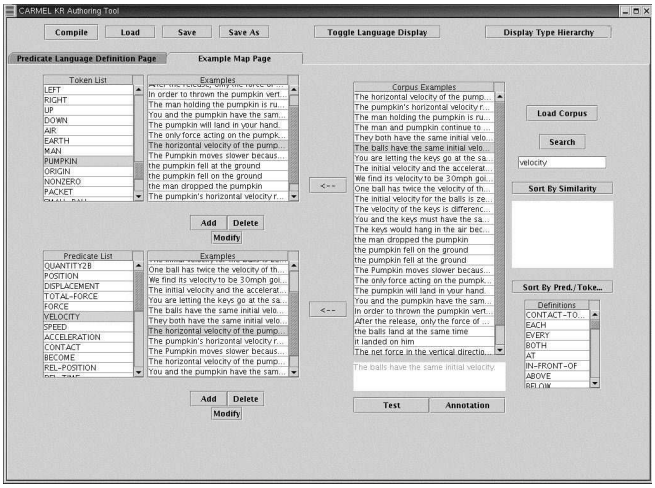

Figure 4: Example Map Page

of freefall. But individual predicates, each representing a part of the meaning of the whole sentence, can also be referred to individually if desired using their own identification tokens.

4 Carmel-Tools Authoring Process

The purpose of Carmel-Tools is to insulate the author from the details of the underlying domain specific knowledge sources. If an author were building knowledge sources by hand for this framework, the author would be responsible for building an ontology for the semantic feature structure representation produced by the parser, linking pointers into this hierarchy into entries in the lexicon, and writing predicate mapping rules. With Carmel-Tools, the author never has to deal directly with these knowledge sources. The Carmel-Tools authoring process involves designing a Predicate Language Definition, augmenting the base lexical resources by either loading raw human tutoring corpora or entering example texts by hand, and annotating example texts with their corresponding representation in the defined Predicate Language Definition. From this authored knowledge, CARMEL's semantic knowledge sources can be generated and compiled. The knowledge source inference algorithm ensures that knowledge coded redundantly across multiple examples is represented only once in the compiled knowledge sources. The authoring interface allows the author or authors to test the compiled knowledge sources and then continue the authoring process by updating the Predicate Language Definition, loading additional corpora, annotating additional examples, or modifying already annotated examples.

The Carmel-Tools authoring process was designed to eliminate the most time-consuming parts of the authoring

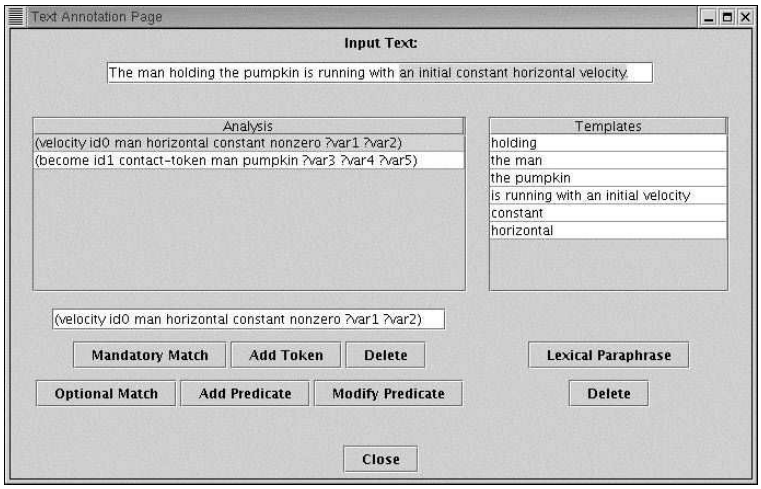

Figure 5: Text Annotation Page Page

process. In particular, its GUI interface guides authors in such a way as to prevent them from introducing inconsistencies between knowledge sources, which is particularly crucial when multiple authors work together. For example, a GUI interface for entering propositional representations for example texts insures that the entered representation is consistent with the author's Predicate Language Definition. Compiled knowledge sources contain pointers back to the annotated examples that are responsible for their creation. Thus, it is also able to provide troubleshooting facilities to help authors track down potential sources for incorrect analyses generated from compiled knowledge sources. When changes are made to the Predicate Language Definition, Carmel-Tools tests whether each proposed change would cause conflicts with any annotated example texts. An example of such a change would be deleting an argument from a predicate type where some example has as part of its analysis an instantiation of a predicate with that type where that argument is bound. If so, it lists these example texts for the author and requires the author to modify the annotated examples first in such a way that the proposed change will not cause a conflict, in this case that would mean uninstantiating the variable that the author desires to remove. In cases where changes would not cause any conflict, such as adding an argument to a predicate type, renaming a predicate, token, or type, or removing an argument that is not bound in any instantiated proposition, these changes are made throughout the database automatically.

4.1 Defining the Predicate Language Definition

The author begins the authoring process by designing the propositional language that will be the output representation from CARMEL using the authored knowledge sources. This is done on the Predicate Language Definition page of the Carmel-Tools interface, displayed in Figure 3. The author is free to develop a representation language that is as simple or complex as is required by the type of reasoning, if any, that will be applied to the output representations by the tutoring system as it formulates its response to the student's natural language input.

The interface includes facilities for defining a list of predicates and Tokens to be used in constructing propositional analyses. Each predicate is associated with a basic predicate type, which is a associated with a list of arguments. Each basic predicate type argument is itself associated with a type that defines the range of atomic values, which may be tokens or identifier tokens referring to instantiated predicates, that can be bound to it. Thus, tokens also have types. Each token has one or more basic token types. Besides basic predicate types and basic token types, we also allow the definition of abstract types that can subsume other types.

4.2 Generating Lexical Resources and Annotating Example Sentences

When the predicate language definition is defined, the next step is to generate the domain specific lexical resources and annotate example sentences with their corresponding representation within this defined predicate language. The author begins this process on the Example Map Page, displayed in Figure 4.

Carmel-Tools provides facilities for loading a raw human tutoring corpus file. Carmel-Tools then makes a list of each unique morpheme it finds in the file and then augments both its base lexicon (using entries from COMLEX), in order to include all morphemes found in the transcript file that were not already included in the base lexicon, and the spelling corrector's word list, so that it includes all morphological forms of the new lexical en-

tries. It also segments the file into a list of student sentence strings, which are then loaded into a Corpus Examples list, which appears on the right hand side of the interface. Searching and sorting facilities are provided to make it easy for authors to find sentences that have certain things in common in order to organize the list of sentences extracted from the raw corpus file in a convenient way. For example, a `Sort By Similarity` button causes Carmel-Tools to sort the list of sentences according to their respective similarity to a given text string according to an LSA match between the example string and each corpus sentence. The interface also includes the Token List and the Predicate List, with all defined tokens and predicates that are part of the defined predicate language. When the author clicks on a predicate or token, the Examples list beside it will display the list of annotated examples that have been annotated with an analysis containing that token or predicate.

Figure 5 displays how individual texts are annotated. The Analysis box displays the propositional representation of the example text. This analysis is constructed using the `Add Token`, `Delete`, `Add Predicate`, and `Modify Predicate` buttons, as well as their subwindows, which are not shown. Once the analysis is entered, the author may indicate the compositional breakdown of the example text by associating spans of text with parts of the analysis by means of the `Optional Match` and `Mandatory Match` buttons. For example, the noun phrase "the man" corresponds to the `man` token, which is bound in two places. Each time a match takes place, the Carmel-Tools internal data structures create one or more templates that show how pieces of syntactic analyses corresponding to spans of text are matched up with their corresponding propositional representation. From this match Carmel-Tools infers both that "the man" is a way of expressing the meaning of the `man` token in text and that the subject of the verb `hold` can be bound to the `?body1` argument of the `become` predicate. By decomposing example texts in this way, Carmel-Tools constructs templates that are general and can be reused in multiple annotated examples. It is these created templates that form the basis for all compiled semantic knowledge sources. Thus, even if mappings are represented redundantly in annotated examples, they will not be represented redundantly in the compiled knowledge sournces. The list of templates that indicates the hierarchical breakdown of this example text are displayed in the Templates list on the right hand side of Figure 5. Note that while the author matches spans to text to portions of the meaning representation, the tool stores mappings between *feature structures* and portions of meaning representation, which is a more general mapping.

Templates can be generalized by entering paraphrases for portions of template patterns. Internally what this accomplishes is that all paraphrases listed can be interpreted by CARMEL as having the same meaning so that they can be treated as interchangeable in the context of this template. A paraphrase can be entered either as a specific string or as a Defined Type, including any type defined in the Predicate Language Definition. What this means is that the selected span of text can be replaced by any span of text that can be interpreted in such a way that its predicate representation's type is subsumed by the indicated type.

4.3 Compiling Knowledge Sources

Each template that is created during the authoring process corresponds to one or more elements of each of the required domain specific knowledge sources, namely the ontology, the lexicon with semantic pointers, and the predicate mapping rules. Using the automatically generated knowledge sources, most of the "work" for mapping a novel text onto its predicate language representation is done either by the deep syntactic analysis, where a lot of surface syntactic variation is factored out, and during the predicate mapping phase, where feature structure patterns are mapped onto their corresponding predicate language representations. The primary purpose of the sentence level ontology that is used to generate a semantic feature structure at parse time is primarily for the purpose of limiting the ambiguity produced by the parser. Very little generalization is obtained by the semantic feature structures created by the automatically generated knowledge sources over that provided by the deep syntactic analysis alone. By default, the automatically generated ontology contains a semantic concept corresponding to each word appearing in at least one annotated example. A semantic pointer to that concept is then inserted into all lexical entries for the associated word that were used in one of the annotated examples. An exception occurs where paraphrases are entered into feature structure representations. In this case, a semantic pointer is entered not only into the entry for the word from the sentence, but also the words from the paraphrase list, allowing all of the words in the paraphrase list to be treated as equivalent at parse time. The process is a bit more involved in the case of verbs. In this case it is necessary to infer based on the parses of the examples where the verb appears which set of subcategorization tags are consistent, thus limiting the set of verb entries for each verb that will be associated with a semantic pointer, and thus which entries can be used at parse time in semantic interpretation mode. Carmel-Tools makes this choice by considering both which arguments are present with that verb in the complete database of annotated examples as well as how the examples were broken down at the matching stage. All non-extracted arguments are considered mandatory. All extracted arguments are considered optional. Each COMLEX subcat

tag is associated with a set of licensed arguments. Thus, subcat tags are considered consistent if the set of licensed arguments contains at least all mandatory arguments and doesn't license any arguments that are not either mandatory or optional. Predicate mapping rules are generated for each template by first converting the corresponding syntactic feature structure into the semantic representation defined by the automatically generated ontology and lexicon with semantic pointers. Predicate mapping rules are then created that map the resulting semantic feature structure into the associated predicate language representation.

5 Evaluation

A preliminary evaluation was run for the physics domain. We used for our evaluation a corpus of essays written by students in response to 5 simple qualitative physics questions such as "If a man is standing in an elevator holding his keys in front of his face, and if the cable holding the elevator snaps and the man then lets go of the keys, what will be the relationship between the position of the keys and that of the man as the elevator falls to the ground? Explain why." A predicate language definition was designed consisting of 40 predicates, 31 predicate types, 160 tokens, 37 token types, and 15 abstract types. The language was meant to be able to represent physical objects mentioned in our set of physics problems, body states (e.g., freefall, contact, non-contact), quantities that can be measured (e.g., force, velocity, acceleration, speed, etc.), features of these quantities (e.g., direction, magnitude, etc.), comparisons between quantities (equivalence, non-equivalence, relative size, relative time, relative location), physics laws, and dependency relations. An initial set of 250 example sentences was then annotated, including sentences from each of a set of 5 physics problems.

Next a set of 202 novel test sentences, each between 4 and 64 words long, was extracted from the corpus. Since comparisons, such as between the accelerations of objects in freefall together, are important for the reasoning in all of the questions used for corpus collection, we focused the coverage evaluation specifically on sentences pertaining to comparisons, such as in Figures 1 and 2. The goal of the evaluation was to test the extent to which knowledge generated from annotated examples generalizes to novel examples.

Since obtaining the correct predicate language representation requires obtaining a correct syntactic parse, we first evaluated CARMEL's syntactic coverage over the corpus of test sentences to obtain an upper bound for expected performance. We assigned the syntactic interpretation of each sentence a score of None, Bad, Partial, or Acceptable. A grade of None indicates that no interpretation was built by the grammar. Bad indicates that parses were generated, but they contained errorfull functional relationships between constituents. Partial indicates that no parse was generated that covered the entire sentence, ut the portions that were completely correct for at least one interpretation of the sentence. Acceptable indicates that a complete parse was built that contained no incorrect functional relationships. If any word of the sentence was not covered, it was one that would not change the meaning of the sentence. For example, "he had the same velocity as you had" is the same as "he had the same velocity as you", so if "did" was not part of the final parse but other than that, the parse was fine, it was counted as Acceptable. Overall the coverage of the grammar was very good. 166 sentences were graded Acceptable, which is about 83% of the corpus. 8 received a grade of Partial, 26 Bad, and 1 None.

We then applied the same set of grades to the quality of the predicate language output. Note that that the grade assigned to an analysis represents the correctness and completeness of the predicate representation the system obtained for that sentence. In this case, a grade of Acceptable meant that all aspects of intended meaning were accounted for, and no misleading information was encoded. Partial indicated that some non-trivial part of the intended meaning was communicated. Any interpretation containing any misleading information was counted as Bad. If no predicate language representation was returned, the sentence was graded as None. As expected, grades for semantic interpretation were not as high as for syntactic analysis. In particular, 107 were assigned a grade of Acceptable, 45 were assigned a grade of Partial, 36 were assigned a grade of Bad, and 14 received a nil interpretation. Our evaluation demonstrates that knowledge generated from annotated examples can be used to interpret novel sentences, however, there are still gaps in the coverage of the automatically generated knowledge sources that need to be filled in with new annotated examples. Furthermore, the small but noticeable percentage of bad interpretations indicates that some previously annotated examples need to be modified in order to prevent these bad interpretations from being generated.

6 Current Directions

In this paper we have introduced Carmel-Tools, a tool set for quick authoring of semantic knowledge sources. Our evaluation demonstrates that the semantic knowledge sources inferred from examples annotated using Carmel-Tools generalize to novel sentences. We are continuing to work to enhance the ability of Carmel-Tools to learn generalizable knowledge from examples as well as to improve the user friendliness of the interface.

References

J. Burstein, K. Kukich, S. Wolff, C. Lu, M. Chodorow, L. Braden-Harder, and M. D. Harris. 1998. Automated scoring using a hybrid feature identification technique. In *Proceedings of COLING-ACL'98*, pages 206–210.

M. T. H. Chi, S. A. Siler, H. Jeong, T. Yamauchi, and R. G. Hausmann. 2001. Learning from human tutoring. *Cognitive Science*, (25):471–533.

J. Christie. 2003. Automated essay marking for content: Does it work? In *CAA Conference Proceedings*.

H. Cunningham, D. Maynard, and V. Tablan. 2000. Jape: a java annotations patterns engine. Institute for Language, Speach, and Hearing, University of Sheffield, Tech Report CS-00-10.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2003. Gate: an architecture for development of robust hlt applications. In *Recent Advanced in Language Processing*.

D. Davidson. 1967. The logical form of action sentences. In N. Rescher, editor, *The Logic of Decision and Action*.

P. W. Foltz, W. Kintsch, and T. Landauer. 1998. The measurement of textual coherence with latent semantic analysis. *Discourse Processes*, 25(2-3):285–307.

R. Grishman, C. Macleod, and A. Meyers. 1994. COMLEX syntax: Building a computational lexicon. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*.

J. R. Hobbs. 1985. Ontological promiscuity. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*.

D. Jay, J. Aberdeen, L. Hirschman, R. Kozierok, P. Robinson, and M. Vilain. 1997. Mixed-initiative development of language processing systems. In *Fifth Conference on Applied Natural Language Processing*.

M. Lepper, M. Woolverton, D. Mumme, and J. Gurtner. 1993. Motivational techniques of expert human tutors: Lessons for the design of computer based tutors. In S. P. Lajoie and S. J. Derry, editors, *Computers as Cognitive Tools*, pages 75–105.

D. Lonsdale and D. Strong-Krause. 2003. Automated rating of esl essays. In *Proceedings of the HLT-NAACL 2003 Workshop: Building Educational Applications Using Natural Language Processing*.

K. Malatesta, P. Wiemer-Hastings, and J. Robertson. 2002. Beyond the short answer question with research methods tutor. In *Proceedings of the Intelligent Tutoring Systems Conference*.

D. Maynard, K. Bontcheva, and H. Cunningham. 2002. Towards a semantic extraction of named entities. In *40th Anniversary meeting of the Association for Computational Linguistics*.

A. McCallum and K. Nigam. 1998. A comparison of event models for naive bayes text classification. In *Proceedings of the AAAI-98 Workshop on Learning for Text Classification*.

O. Popescue, V. Aleven, and K. Koedinger. 2003. A kowledge based approach to understanding students' explanations. In *Proceedings of the AI in Education Workshop on Tutorial Dialogue Systems: With a View Towards the Classroom*.

C. P. Rosé, D. Bhembe, A. Roque, and K. VanLehn. 2002. An efficient incremental architecture for robust interpretation. In *Proceedings of the Human Languages Technology Conference*, pages 307–312.

C. P. Rosé, A. Roque, D. Bhembe, and K VanLehn. 2003. A hybrid text classification approach for analysis of student essays. In *Proceedings of the HLT-NAACL 2003 Workshop: Building Educational Applications Using Natural Language Processing*.

C. P. Rosé. 2000. A framework for robust semantic interpretation. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 311–318.

K. VanLehn, P. Jordan, C. P. Rosé, and The Natural Languag e Tutoring Group. 2002. The architecture of why2-atlas: a coach for qualitative physics essay writing. In *Proceedings of the Intelligent Tutoring Systems Conference*, pages 159–167.

Y. Wang and A. Acero. 2003. Concept acquisition in example-based grammar authoring. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

P. Wiemer-Hastings, A. Graesser, D. Harter, and the Tutoring Res earch Group. 1998. The foundations and architecture of autotutor. In B. Goettl, H. Halff, C. Redfield, and V. Shute, editors, *Intelligent Tutoring Systems: 4th International Conference (ITS '98)*, pages 334–343. Springer Verlag.

C. Zinn, J. D. Moore, and M. G. Core. 2002. A 3-tier planning architecture for managing tutorial dialogue. In *Proceedings of the Intelligent Tutoring Systems Conference*, pages 574–584.