Overcoming the Knowledge Engineering Bottleneck for Understanding Student Language Input

Carolyn P. Rosé, Andy Gaydos, Brian S. Hall, Antonio Roque, and Kurt VanLehn
Learning Research and Development Center, University of Pittsburgh, Pittsburgh PA, 15260

Abstract. In this paper we present Carmel-Tools, a new set of authoring tools for overcoming the knowledge engineering bottleneck for building tutorial dialogue systems that can accept natural language text input from students. Carmel-Tools provides the facilities for speeding up and simplifying the task of creating domain specific knowledge sources for sentence level language understanding. Carmel-Tools can build up domain specific lexical resources automatically from raw human tutoring corpora. It provides a GUI interface for authors to design their own first order propositional representation as well as facilities for annotating example sentences with their corresponding interpretation in this designed representational language. It then generalizes over these annotations and automatically generates all domain specific semantic knowledge sources required for interpreting novel input texts that are similar in content to those that have been annotated.

1 Introduction

In this paper we describe Carmel-Tools[1] a new publicly available resource for the tutorial dialogue systems community. In recent years a tremendous amount of interest has developed in building tutorial dialogue systems. The goal of tutorial dialogue systems is to use dialogue to provide students with a learning environment that exhibits the characteristics that have been shown to correlate with student learning gains, such as student activity. For example, it has been demonstrated that generating words rather than simply reading them promotes subsequent recall of those words (Slamecka and Graf, 1978). Furthermore, encouraging student self-explanation, which includes both generating inferences from material they have read and relating new material to old material, has been shown to correlate with learning (Chi et al., 1981; Pressley et al., 1992).

While tutorial dialogue systems are becoming increasingly prevalent, and this research is mature enough to yield successful evaluations with students [18, 9, 2, 7, 5], the problem of text input understanding has continued to pose serious obstacles. Many systems, such as Paco [16], CATO-Dial [2], Ms. Lindquist [9], and MarCo [20], finesse the problem by avoiding language input altogether in favor of menu-based input or even logical forms. However, entering logical forms is tedious and error prone, and some recent studies have demonstrated that menu-selections are only effective with some students and not for others for stimulating self-explanation leading to learning [4].

[1] To obtain a copy of Carmel-Tools, contact the first author at rosecp@pitt.edu.

The majority of tutorial dialogue systems that accept language input from students have employed shallow techniques. For example, CIRCSIM-TUTOR [6] and Andes-Atlas [18] have been demonstrated to successfully interpret short student answers using shallow semantic grammars. However, such approaches have not yet been demonstrated to scale up to processing longer student turns in a way that is effective for assessing a student's knowledge state. Other systems have processed longer student inputs, even multi-sentential input, using text classification approaches. For example, the AUTO-TUTOR system [7] and the Research Methods Tutor [14] use Latent Semantic Analysis (LSA) [11], which is a "bag of words" approach to text classification. Thus, LSA makes its prediction about what class to assign to an input text based only on which words are included in that text. While AUTO-TUTOR was able to use LSA successfully to handle student input in the Computer Literacy domain, LSA has been demonstrated to have difficulty in highly causal domains such as Research Methods [14].

Text classification approaches are attractive for tutoring systems because they require no sophisticated knowledge engineering effort and are very robust to noisy input. Nevertheless, text classification approaches have some inherent limitations in the context of tutorial dialogue systems. First, text classification is a coarse-grained approach to understanding student input. Thus, it may gloss over subtle details, such as minor vocabulary mistakes or missing pieces of information, that would be desirable to question students about and possibly correct. Secondly, it is limited to a pre-defined set of discrete classifications anticipated by authors.

Thus, it is desirable to be able to construct more detailed analyses of student text input, such as those that can be provided by symbolic language processing approaches. Systems that perform such a sophisticated type of linguistic analysis of student input include BEETLE [22], WHY-ATLAS [21], and the Geometry Tutor [1]. Both BEETLE and WHY-ATLAS use the CARMEL Core Understanding Component [17] for sentence level understanding of student input. Geometry Tutor uses CARMEL's LCFLEX robust parser and the Loom description logic system [12]. One major difficulty in symbolic processing of student text input is the tremendous amount of time and expertise required to build domain specific knowledge sources that can effectively transform the wide range of student text inputs into a form that can be used for reasoning about a student's knowledge state. Thus, in this paper we present Carmel-Tools, a new set of authoring tools for overcoming the knowledge engineering bottleneck for building tutorial dialogue systems that can accept natural language text input from students. Carmel-Tools provides the facilities for speeding up and simplifying the task of creating domain specific knowledge sources for the CARMEL Core Understanding Component.

2 CARMEL

The purpose of the CARMEL Core Input Understanding Component [17] is to provide a general resource for robust sentence level interpretation that can be adapted for use in different systems. CARMEL produces a deep syntactic functional analysis of an input text which is ultimately transformed into a first order predicate representation that can then be used for reasoning about a student's knowledge state. For example, in WHY-ATLAS, CARMEL's representation for each sentence in a student's essay is sent to a discourse level understanding component [13] that uses Tacitus-lite to construct a proof from this predicate language representation.

The CARMEL grammar uses features from the broad coverage COMLEX lexicon [8],

which we have augmented for our purposes. The lexicon provides a clean interface between syntactic and semantic knowledge. In our enhanced version of COMLEX, semantic constructor functions are linked into lexical entries where the entry may act as a lexical head corresponding to the associated semantic concept. Syntactic functional roles are matched to argument positions in the semantic constructor functions. Thus, as syntactic functional roles are assigned at parse time, semantic interpretation can proceed in parallel. CARMEL's deep syntactic analysis can be thought of as a normalized representation of the input sentence, which is convenient for semantic interpretation since the head-argument relations from which the propositional content of the sentence is derived remain consistent across sentences that have the same propositional content but different surface syntactic properties. Thus, the same semantic knowledge used for building an analysis of the propositional content of one surface syntactic variation of a sentence is sufficient for building a representation for all surface syntactic variations of that sentence.

Prior to the existence of Carmel-Tools, despite the fact that CARMEL's grammar and lexicon are domain independent, all domain specific resources required by CARMEL for semantic interpretation had to be authored by hand for each domain. CARMEL requires four different domain specific knowledge sources, all of which are now automatically generated by Carmel-Tools. The first domain specific resource, the lexical ontology, describes the frame based semantic representation produced at parse time. From this, semantic constructor functions are compiled automatically. Pointers to these functions must then be inserted into appropriate entries in the lexicon where they can instruct the parser how to build up semantic representations in parallel to the syntactic ones that are generated by the syntactic grammar. A file of parsed examples that have been properly disambiguated is required for training the parser's statistical disambiguation model. The predicate mapper, which maps the parser's frame based representation into a predicate logic representation, requires a definition of the predicate language as well as rules for mapping between the frame based representation and that predicate language.

3 The Authoring Process

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 author can then 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 process. In particular, its GUI interface guides authors in such a way as to prevent them from introducing inconsistencies between knowledge sources. 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 is able to test 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 everywhere in the database automatically.

3.1 Defining the Predicate Language Definition

Sentence: *The horizontal velocity of the pumpkin is the same as the horizontal velocity of the runner at the time of the release.*

```
((velocity ID1 pumpkin horizontal ?var0 ?var1 ?var2 ?var3)
 (velocity ID2 man horizontal ?var4 ?var5 ?var6)
 (rel-value ID3 ID1 ID2)
 (become ID4 contact man pumpkin attached detached ?var7)
 (rel-time ID5 ID3 ID4 before))
```

Figure 1: **Example CARMEL propositional representation**

The author may begin 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 GUI interface, which is not displayed due to space restrictions.

The author is completely 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. For example, in a very simple system such as Andes-Atlas [18], each sentence could be assigned a very simple atomic representation where each atomic value is associated with an answer class that completely determines the response the system will give the student.

More complex systems require more complex representations. For example, Figure 1 displays CARMEL's analysis for a typical sentence for the WHY-ATLAS system. Thus, we provide a framework in which a Davidsonian style propositional representations can be defined [10]. 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. For example, in the WHY-ATLAS predicate language definition, the predicate `acceleration` is defined as having type `pred-type2`, which has 6 arguments, namely `?identifier`, `?body`, `?comp`, `?Mag-der`, `?mag-zero`, and `?dir`. Based on the Hobbs (1985) formulation of Davidsonian semantics, the first argument, namely `?identifier`, refers to the "condition" that exists when the

predication expressed by the whole instantiated proposition is true. The atomic value bound to this argument can then be used as an identifier to refer to that "condition". This makes the representation more general as it makes it natural to modify propositions with a wide range of adverbials. For instance, in the example in Figure 1 we see that the representation of the adverbial expression "at the time of release" is represented by the instantiated `rel-time` predicate that refers both to the instantiated `rel-value` predicate referring to the state in which the two velocities are equal as well as the instantiated `become` predicate that refers to the event of the pumpkin being released from the man's hands.

Each basic predicate type argument is itself associated with a type that defines the range of atomic values, which may be tokens or identifiers referring to propositions, 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. The interface provides facilities for defining, modifying, and deleting types, tokens, and predicates, although this portion of the interface is not displayed in this paper due to space restrictions. The GUI interface for maintaining the predicate language definition allows the author to quickly take inventory of the defined predicate language, in particular identifying which types belong to which predicates and tokens, and conversely, which predicates and tokens are defined as having which types. A graphical view of the entire type hierarchy is displayed when the author clicks on a `Display Type Hierarchy` button.

3.2 Generating Lexical Resources and Annotating Example Sentences

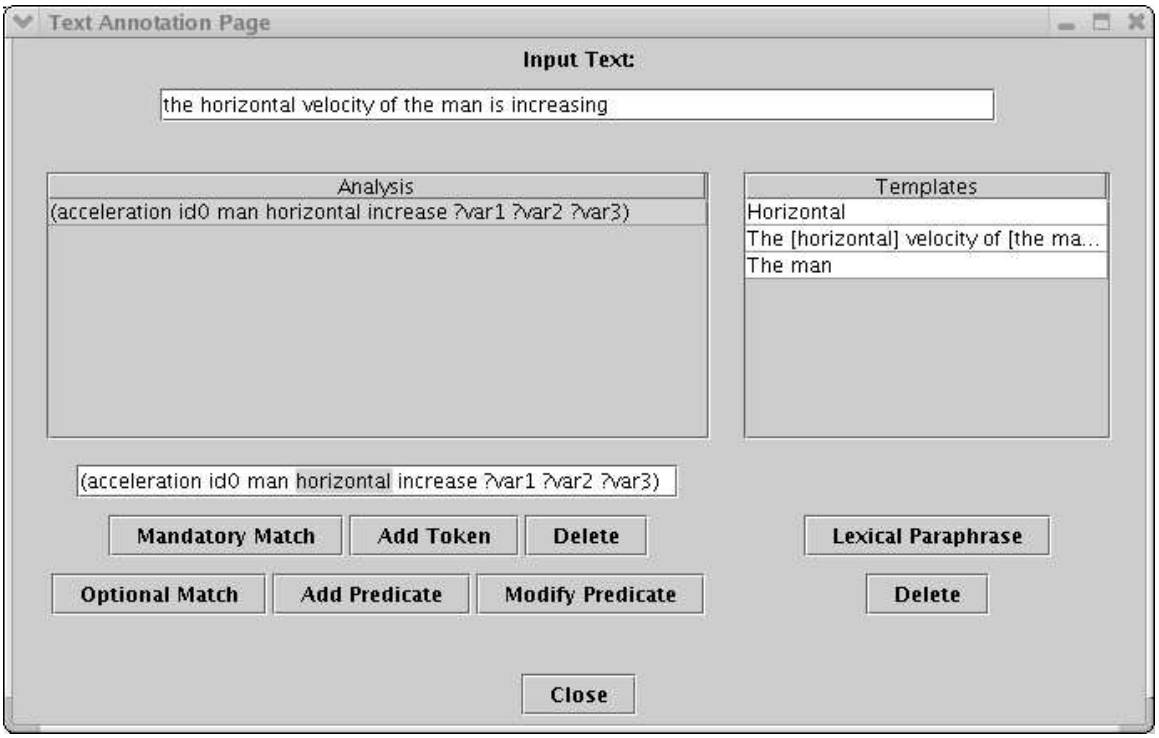

Figure 2: **An Example Analysis**

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, which is not shown due to space restrictions. Note that the author may at any time go back to the Predicate Language Definition page and modify the predicate language definition as desired.

Although the lexicon is one of CARMEL's domain independent resources, it is very large and unwieldy to use as a whole. In addition to keeping the syntactic lexical entries in memory for access by the parser, a word table with all morphological variations of the morphemes in the lexicon is constructed for use with the spelling corrector. If such a table were constructed for the entire COMLEX lexicon, not only would the space requirements be prohibitive, but the spelling corrector's performance would also suffer since many domain inappropriate words would be included there which could easily be incorrectly selected. Thus, Carmel-Tools provides facilities for specializing these lexical resources based on an analysis of raw human tutoring corpora.

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 entries. 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 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. Thus, note that an example sentence likely appears in multiple lists since its representation will include a combination of predicates and tokens.

Figure 2 display 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 sub-windows, 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 learned templates that form the basis for all compiled semantic knowledge sources. 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 2.

3.3 Generalizing Templates

Templates can be made more general 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. This allows a high level of generalization in templates.

4 Conclusions

In this paper we have presented Carmel-Tools, a new publicly available resource for the tutorial dialogue community. While our ultimate goal is to provide tools that do not require any computational linguistics expertise to use, we believe that we have not yet achieved that goal. Nevertheless, Carmel-Tools provides resources for greatly speeding up an simplifying the task of creating domain specific knowledge sources required for symbolic natural language processing.

As a proof of concept, the Predicate Language Definitions for WHY-ATLAS and BEETLE have been entered, and approximately 250 example texts have been annotated. The compiled knowledge sources replicate work that was done by hand over the course of a year. The compiled knowledge sources have been informally checked for correctness by testing them over a regression set as well as on novel input texts similar in content to those that were annotated. This effort took in total about 3 full-time days. One current direction is extending the capabilities of our lexical resource specialization code so that it can build lexical entries for words that are not included in the COMLEX lexicon.

5 Acknowledgments

This research was supported by the Office of Naval Research, Cognitive Science Division under grant number N00014-0-1-0600 and by NSF grant number 9720359 to CIRCLE, Center for Interdisciplinary Research on Constructive Learning Environments at the University of Pittsburgh and Carnegie Mellon University.

References

[1] V. Aleven, O. Popescu, and K. Koedinger. Pedagogical content knowledge in a tutorial dialogue system to support self-explanation. In *Papers of the AIED-2001 Workshop on Tutorial Dialogue Systems*, pages 59–70, 2001.

[2] K. D. Ashley, R. Desai, and J. M. Levine. Teaching case-based argumentation concepts using didactic arguments vs. didactic explanations. In *Proceedings of the Intelligent Tutoring Systems Conference*, pages 585–595, 2002.

[3] M. Chi, N. de Leeuw, M. Chiu, and C. LaVancher. Eliciting self-explanations improves understanding. *Cognitive Science*, 18(3), 1981.

[4] C. Conati and K. VanLehn. Teaching meta-cognitive skills: Implementation and evaluation of a tutoring system to guide self-explanation while learning from examples. In *Proceedings of AI in Education*, 1999.

[5] M. Evans and J. Michael. *One-on-One Tutoring by Humans and Machines*. Lawrence Earlbaum and Associates, in-press.

[6] M. S. Glass. *Broadening Input Understanding in an Intelligent Tutoring System*. PhD thesis, Illinois Institute of Technology, 1999.

[7] A. Graesser, K. Wiemer-Hastings, P. Wiemer-Hastings, R. Kreuz, and the Tutoring Research Group. Autotutor: A simulation of a human tutor. *Journal of Cognitive Systems Research*, 1(1):35–51, 1999.

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

[9] N. T. Heffernan and K. R. Koedinger. An intelligent tutoring system incorporating a model of an experienced tutor. In *Proceedings of the Intelligent Tutoring Systems Conference*, pages 596–608, 2002.

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

[11] T. K. Landauer, P. W. Foltz, and D. Laham. Introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284, 1998.

[12] R. MacGregor. The evolving technology of classification-based knowledge representation systems. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann: San Mateo, CA, 1991.

[13] M. Makatchev, P. Jordan, and K. VanLehn. Discourse processing for explanatory essays in tutorial applications. In *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*, 2002.

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

[15] M. Pressley, E. Wood, V. E. Woloshyn, V. Martin, A. King, and D. Menke. Encouraging mindful use of prior knowledge: Attempting to construct explanatory answers facilitates learning. *Educational Psychologist*, 27:91–109, 1992.

[16] J. Rickel, N. Lesh, C. Rich, and C. L. Sidner. Collaborative discourse theory as a foundation for tutorial dialogue. In *Proceedings of the Intelligent Tutoring Systems Conference*, pages 542–551, 2002.

[17] C. P. Rosé. A framework for robust sentence level interpretation. In *Proceedings of the First Meeting of the North American Chapter of t he Association for Computational Linguistics*, pages 1129–1135, 2000.

[18] C. P. Rosé, P. Jordan, M. Ringenberg, S. Siler, K. VanLehn, and A. Weinstein. Interactive conceptual tutoring in atlas-andes. In *Proceedings of Artificial Intelligence in Education*, pages 256–266, 2001.

[19] N. J. Slamecka and P. Graf. The generation effect: Delineation of a phenomenon. *Journal of Experimental Psychology: Human Learning and Memory*, (4):592–604, 1978.

[20] P. A. Tedesco. Conflicts in group planning: Results from the experimental study of marco. In *Proceedings of the Intelligent Tutoring Systems Conference*, pages 619–629, 2002.

[21] K. VanLehn, P. Jordan, C. P. Rosé, and The Natural Language Tutoring Group. The architecture of why2-atlas: a coach for qualitative physics essay writing. Proceedings of the Intelligent Tutoring Systems Conference, 2002.

[22] C. Zinn, J. D. Moore, and M. G. Core. A 3-tier planning architecture for managing tutorial dialog. In *Proceedings of the Intelligent Tutoring Systems Conference*, 2002.