Lattice-Based Approach to Building Templates for Natural Language Understanding in Intelligent Tutoring Systems

Shrenik Devasani[1], Gregory Aist[1], Stephen Blessing[2], Stephen Gilbert[1]

[1] VRAC, Iowa State University, 1620 Howe Hall, Ames, IA 50011, USA
[2] University of Tampa, 401 W. Kennedy Blvd., Tampa, FL 33606, USA

shrenik@iastate.edu, aist@iastate.edu, sblessing@ut.edu, gilbert@iastate.edu

Abstract. We describe a domain-independent authoring tool, ConceptGrid, that helps non-programmers develop intelligent tutoring systems (ITSs) that perform natural language processing. The approach involves the use of a lattice-style table-driven interface to build templates that describe a set of required concepts that are meant to be a part of a student's response to a question, and a set of incorrect concepts that reflect incorrect understanding by the student. The tool also helps provide customized just-in-time feedback based on the concepts present or absent in the student's response. This tool has been integrated and tested with a browser-based ITS authoring tool called xPST.

Keywords: natural language processing, intelligent tutoring system, authoring tool

1 Introduction

Interpreting textual responses from students by an Intelligent Tutoring System (ITS) is essential if it can come close to matching the performance of a human tutor, even in domains such as Statistics and Physics, since the use of language makes the learning process more natural. Natural language has the advantage of being easy to use for the student, as opposed to learning new formalisms.

Over the past decade, studies have been conducted that confirm the importance of using language in both traditional learning environments and in intelligent tutoring systems. Chi et al [1, 2] have showed that eliciting self-explanations enhances deeper learning and understanding of a coherent body of knowledge that generalizes better to new problems. Aleven et al [3] conducted studies with the PACT Geometry Tutor in which students who provided explanations to solution steps showed greater understanding in the post-test, compared to students who did not provide explanations.

Many ITSs have successfully incorporated natural processing. The CIRCSIM Tutor [4] is a language based ITS for medical students that uses word matching and

finite state machines to process students' natural language input. Rus et al [5] have described an approach of evaluating answers by modeling it as a textual entailment problem. Intelligent tutoring systems such as the AutoTutor [6] and Summary Street [7] use Latent Semantic Analysis (LSA) [8] to evaluate student answers, a technique that uses statistical computation and is based on the idea that the aggregate of all the word contexts in which a word appears determines the similarity of meaning of words to each other. The problem with LSA is that it does not encode word order and it cannot always recognize negation. Another problem with LSA is that it scores students' responses only based on how well it matches the ideal answer, and cannot point out what exactly is wrong with an incorrect response.

Though ITSs today use a variety of techniques to provide support for natural language understanding, user-programming of NLP in ITSs is not common with authoring toolkits. The various techniques described here do not give sufficient power to non-programmers as the NLP is left to expert developers or to machine learning algorithms, and the user is more likely to focus on tutoring strategies. Our approach addresses these issues.

2 The ConceptGrid Approach

ConceptGrid is intended to be used by tutor authors with little or no programming experience. The most crucial aspect about developing an authoring tool that can be used by non-programmers is managing the trade-off between its ease of use and its expressive power. Keeping this in mind, ConceptGrid has been designed such that its ease of use and expressiveness lie between that of simple word matching approaches and complex approaches such as those that use complex machine learning algorithms.

The tutor author develops the natural language understanding component for a tutor by breaking down the expected response to a question into specific concepts. The author then builds templates that describe a set of required concepts (that are meant to be a part of student's response to a question) and a set of incorrect concepts (that reflect incorrect understanding by the student). Every template is mapped to a single user-defined concept name. Since a student can describe a single concept in various forms, several templates can be used to describe different representations of a single concept, in order to recognize and provide feedback to a wider range of student responses (both correct and incorrect). Thus, there is a one-to-many relationship between concepts and templates.

A template consists of one or more atomic checktypes, or check functions, that evaluate a student's input. These particular atomic checktypes are based on well-known algorithms and distance measures. The word "atomic" refers to the fact that these checktypes can be applied to a single word only. The set of atomic checktypes have been described in Table 1.

Apart from these atomic checktypes, we have two more checktypes that help make the template more expressive: Any(n_1, n_2) and Not(n, 'direction', word_list). The checktype "Any" matches any sequence of words that is at least n_1 words long and at most n_2 words. It helps account for words that are not explicitly accounted for using the other checktypes. The "Not" checktype takes care of negation. It makes sure that

the n words appearing to the left or right (specified by ‘direction’) of the word following the checktype do not match the words mentioned in "word_list".

Table 1. Atomic checktypes used in designing a template.

Checktype	Description
Exact(word_list)	Returns true if a literal character-by-character word match with any of the words in word_list is found
Almost(word_list)	Returns true if a literal match, after ignoring vowels, with any of the words in word_list is found
Levenshtein(n, word_list)	Returns true if the least Levenshtein distance between a word in word_list and matched word is <= n
Hamming(n, word_list)	Returns true if the least Hamming distance between a word in word_list and matched word is <= n
Soundex(word_list)	Returns true if a Soundex match with any of the words in word_list is found
Synonym(word_list)	Returns true if an exact match with any of the words in word_list or its synonyms (from WordNet) is found
Stemmer(word_list)	Returns true if a literal match with the stem of the matched word, with any of the words in word_list is found (uses Porter Stemmer)

The checktypes Synonym and Stemmer can be nested within other atomic checktypes to make them more powerful. Levenshtein(Synonym(‘interface’),1), for example, captures the idea that any synonym of the word "interface" is fine, even if it has a spelling mistake.

When the student misses out on a subset of the required concepts, or mentions a subset of incorrect concepts, customized feedback can be given that points out the issue.

3 The ConceptGrid Interface

The web-based interface is designed to allow the user to create templates that describe both required and incorrect concepts, and mention the feedback that needs to be given.

To simplify the process of constructing templates, we have a lattice-style table-driven interface for entering the template’s checktypes and the corresponding parameters (Figure 1). A new template is created either by entering the dimensions of the table or by entering a sample response, from which a table is created and initialized. The table consists of a sequence of multi-level drop-down menus that represent the checktypes. The multiple levels help the author nest different checktypes. Each drop-down menu is associated with a specific number of textboxes that store the parameters associated with it. Each drop-down menu has several

textboxes below it that store the contents of the parameter "word_list" associated with the corresponding checktype. The contingent approach of having the parameters dependent on the specific checktype provides a mild form of just-in-time authoring help. The user can navigate through the table just like a numerical spreadsheet and add or delete new rows and columns.

There are two sets of templates; the first describes required concepts and the second describes incorrect ones. Multiple templates can be mapped onto a single concept. Consider the following question in a statistics problem: “Based on your results, what do you conclude about the conditions of the music?” Let us assume that the correct answer to the question is "Reject the null hypothesis. There is a significant difference in memory recall between the rock music and no music conditions."

Some of the concepts that can be defined for the sample response mentioned above are described in Table 2.

Table 2. Examples of concepts. Conclusion-Correct and Conclusion-Incorrect look at the holistic response and the rest look at the sub-components of the response.

Concept Name	Description
Rejection-Correct	Matches responses that correctly mention whether the null hypothesis has to be rejected or not
Rejection-Incorrect	Matches responses that incorrectly mention whether the null hypothesis has to be rejected or not
Significance-Correct	Matches responses that correctly mention the significance of the result of the statistical test
Significance-Incorrect	Matches responses that incorrectly mention the significance of the result of the statistical test
Ind-Variable-Mention	Matches responses that explicitly mention the independent variable (e.g. type of music)
Dep-Variable-Mention	Matches responses that explicitly mention the dependent variable (e.g. memory recall)
Conclusion-Correct	Matches responses that have the correct conclusion of the statistical test
Conclusion-Incorrect	Matches responses that have the incorrect conclusion of the statistical test

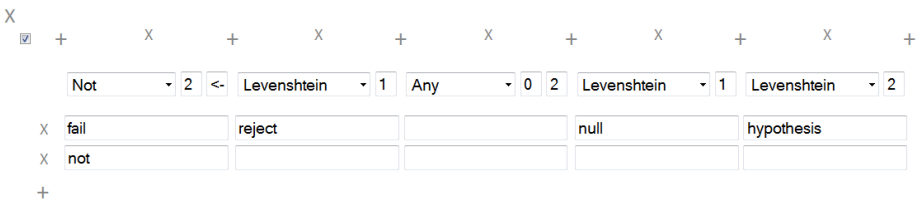

Fig. 1. The lattice-style table-driven interface of ConceptGrid. The template represents the concept “Rejection-Correct”, described in Table 2.

The tutor author then can design a ternary truth table called the Feedback Table (Figure 2) where he or she can enter the feedback that is to be given to the students, based on the truth values of the concepts: true – concept present (green check), false – concept absent (red X), or don't care (yellow dash). The author enters the values of the truth table through tri-state checkboxes. Feedback can be entered for both the absence of required concepts and presence of incorrect ones.

The Feedback Table helps provide feedback in a simple manner for seemingly complicated issues, such as an inconsistent statement (the last row of the Feedback Table in Figure 2) in the example discussed.

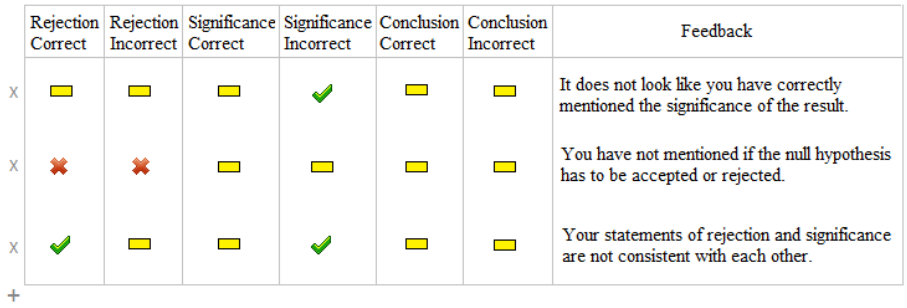

Fig. 2. Feedback Table

There is a provision to create user-defined variables that can be used while building checktypes or mentioning the feedback. This approach helps re-use templates for similar questions. The author can also enter a set of stop words that will be filtered out from the student's response prior to being processed.

Once the templates are designed and the feedback tables are filled, the author can test the templates with sample student responses. The output of the test mentions if the student's response has matched the required concepts. If a match is not found, then it displays the feedback associated with that response. It also displays the truth values of all the concepts defined by the author.

4 Algorithm and Implementation

The implicit sequencing in the lattice approach means that the resulting complex checktypes are finite parsers. That is, progress through the lattice corresponds to progress left-to-right in processing the input.

The templates are represented internally as and-or trees. The algorithm involves a combination of recursion and memoization to efficiently process the input. Since the algorithm might need to backtrack many times, memoization helps speed up the processing by having function calls avoid repeating the calculation of results for previously processed inputs.

Our tool has been integrated with the Extensible Problem Specific Tutor (xPST) - an open source authoring tool that is intended to enable non-programmers to create

ITSs on existing websites and software [9]. Though xPST is a text-based authoring tool, its syntax is not very-code like. ConceptGrid has been customized to generate "code" that is compatible with xPST's syntax, based on the author's templates and Feedback Table, which can be then be inserted into any xPST file.

5 Results: The xSTAT Project

The research question for this paper is whether ConceptGrid could enable an instructor to create a tutor that would score students' free response answers as accurately as he or she manually did. At this point, the question is purely a feasibility issue: can it be done with the ConceptGrid tool? We tested this issue as a part of the xSTAT project at University of Tampa, dedicated to developing an intelligent homework helper for statistics students [10].

For the xSTAT effort, six authors (3 instructors and 3 undergraduates) created multiple tutors each for college level statistics problems. The problems contained real-world scenarios with actual data, followed up by several questions for the student to answer. Each of the problems had a question at the end that asked students to enter the conclusion of the statistics test. To assess these problems, 6 were chosen out of the total pool of 74 and given to students as homework problems. All problems were solved on-line using a standard web browser. Half of the students received feedback on their answers via the xPST intelligent tutor (i.e., answers were marked as either correct or incorrect, and hints and just-in-time messages were displayed), and half did not (i.e., these students simply filled out the web-based form). It is worth noting that these tutors were created without ConceptGrid, so that authors had to explicitly enter the "xPST code" that represents the templates without a graphical user interface. Also, in the absence of visualization through the Feedback Table, subsets of missing and incorrect concepts had to be explicitly mentioned. This non-lattice approach was not very usable by non-programmers. This difficulty motivated the creation of the ConceptGrid lattice approach, which is computationally equivalent and designed to be much more usable by non-programmers.

In all, 41 students solved a total of 233 instances of the six problems across the homework. We built a corpus after collecting all student responses to the end question (both those with tutoring and without). The corpus had 554 unique responses to this final conclusion question across the six homework problems. This corpus includes multiple incorrect responses by the same student to the same problem if they were in the tutored condition. These responses were scored by an instructor and a teaching assistant based on the presence or absence of the concepts defined in Table 2. Then, a tutor author attempted to use ConceptGrid to produce templates that would score the 554 responses similar to those manual scores. The result of that work contained a total of 10 templates common to all six problems, to cover all concepts, except "Ind-Variable-Mention" and "Dep-Variable-Mention". The concepts "In-Variable-Mention" and "Dep-Variable-Mention" required a template each that was unique to each of the six problems. In all, there were 22 templates across all six problems.

Since the manner in which a template tries to match a student's response – a sequence of words is comparable to the manner in which a regular expression matches

a string, it might seem that the results have a lot of false negatives. But, since this approach tries to "understand" responses by looking for smaller concepts and key phrases with the help of checktypes rather than literal word matching, it is much more expressive. The results in Table 3 confirm this observation.

Table 3. Results of the classification of 554 student responses using ConceptGrid

Concept	False Positives	False Negatives	Accuracy
Rejection-Correct	1	34	0.9368
Rejection-Incorrect	6	5	0.9801
Significance-Correct	1	7	0.9856
Significance-Incorrect	12	1	0.9765
Ind-Variable-Mention	1	3	0.9928
Dep-Variable-Mention	4	3	0.9874
Conclusion-Correct	0	24	0.9567
Conclusion-Incorrect	6	0	0.9892

6 Conclusions and Future Work

We have described ConceptGrid, a tool that is intended to help non-programmers develop ITSs that perform natural language processing. It has been integrated into an ITS authoring tool called xPST. We tested it as a part of the xSTAT project and were able to approach the accuracy of human instructors in scoring student responses.

We would like to conduct a study that helps demonstrate that the ConceptGrid tool, a part of xPST, is actually feasible for non-programmers to use on a variety of tasks, as we have done for xPST's core authoring tool [11].

Currently, ConceptGrid does not support a dialogue between the student and tutor. It only evaluates student responses and gives just-in-time feedback. To support more extensive knowledge-construction dialogues, ConceptGrid responses would need to provide information required by the dialogue manager.

Our current approach is non-structural, i.e., it is focused on words and numerical analysis, rather than grammar and logic. The advantage with this approach is that it very simple for non-programmers to use, and is very effective in domains such as statistics where the student responses are expected to follow a general pattern. However, the ConceptGrid approach is domain-independent, one of its biggest advantages.

ConceptGrid could be extended to be structural as well, but that achievement might come at the cost of usability by non-programmers. To include structural matching, either the templates could nest by invoking other templates, or the atomic checktypes could include some checktypes that invoked structural matching. For nested concepts, we could define a concept and then use it within more complex concepts, in the following manner.

GreaterThan(X,Y) = X – "bigger" or "more"or "greater" – "than" – Y

WellFormedConclusion = GreaterThan("weight of the log", "weight of the twig")

This way, the framework can be extended to more powerful natural language processing using a similar approach to the processing that context-free grammars allow. Alternately, the set of ConceptGrid atomic checktypes could be extended to enable structurally-oriented checktypes that would match a nonterminal from a context-free grammar, such as an NP with "twig" as the head in a syntactically oriented grammar, or match the semantics of a section of the utterance.

Acknowledgments. This work is done with the support of the U.S. Air Force Office of Scientific Research.

References

1. Chi, M.T.H, de Leeuw, N., Chiu, M.H., LaVancher, C.: Eliciting Self-Explanations Improves Understanding. Cognitive Science. 18, 439--477 (1994)
2. Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., Glaser, R.: Self-explanations: How Students Study and Use Examples in Learning to Solve Problems. Cognitive Science. 13, 145--182 (1989)
3. Aleven, V., Koedinger, K., Cross, K.: Tutoring Answer Explanations Fosters Learning With Understanding. Proceedings of Artificial Intelligence in Education, AIED '99. pp. 199--206 (1999)
4. Glass, M.: Processing Language Input in the CIRCSIM-Tutor Intelligent Tutoring System. In: Moore, J.D. et al. (eds.), Artificial Intelligence in Education. pp. 210--221 (2001)
5. Rus, V., Graesser, A.: Deeper Natural Language Processing for Evaluating Student Answers in Intelligent Tutoring Systems. American Association for Artificial Intelligence. (2006)
6. Graesser, A., Wiemer-Hastings, P., Wiemer-Hastings, K., Harter, D., Person, N., TRG.: Using Latent Semantic Analysis to Evaluate the Contributions of Students in AutoTutor. Interactive Learning Environemnts. pp. 149--169 (2000)
7. Steinhart, D.: Summary Street: An Intelligent Tutoring System for Improving Student Writing Through the Use of Latent Semantic Analysis. Ph.D. dissertation, Dept. Psychology, University of Colorado, Boulder (2001)
8. Landauer, T.K., Foltz, P.W., Laham, D.: Introduction to Latent Semantic Analysis. Discourse Processes. 25, 259-284 (1998)
9. Blessing, S., Gilbert, S., Blankenship, L., Sanghvi, B.: From SDK to xPST: A New Way to Overlay a Tutor on Existing Software. Proceedings of the Twenty-Second International FLAIRS Conference. (2009)
10. Maass, J., Blessing, S.B.: xSTAT: An Intelligent Homework Helper for Students. Submitted to the 2011 Convention of the Southeast Psychological Association, Jacksonville, FL.
11. Gilbert, S., Blessing, S. B., Kodavali, S.: The Extensible Problem-Specific Tutor (xPST): Evaluation of an API for Tutoring on Existing Interfaces. In: Dimitrova, V. et al. (eds.), Artificial Intelligence in Education. pp. 707--709. IOS Press, Brighton (2006)
12. Boonthum, C., Levinstein, I.B., McNamara, D.S., Magliano, J.P., Millis, K.K.: NLP Techniques in Intelligent Tutoring Systems. IGI Global (2009)
13. Glass, M.S., Evens M.W.: Extracting Information From Natural Language Input to an Intelligent Tutoring System. Far Eastern Journal of Experimental and Theoretical Artificial Intelligence. 1(2), 87--125 (2008)
14. Popescu, Octav.: Logic-Based Natural Language Understanding in Intelligent Tutoring Systems. PhD Thesis, Language Technologies Institute, Carnegie Melon University (2005)