5

Using XLE in an Intelligent Tutoring System

RICHARD R. BURTON

5.1 Introduction

I first met Ron Kaplan in 1973 while he was consulting at Bolt, Beranek and Newman, Inc. on Bill Woods' LUNAR natural language understanding project. He was finishing his dissertation at Harvard at the time and also working with Martin Kay on what would become LFG (Lexical-Functional grammar). I was working with John Seely Brown on intelligent tutoring systems. Roughly, I was trying to get a computer to teach students like good human tutors do. At the time, the state-of-the-art means of communicating with students was teletypes or character-based CRT display terminals. If you wanted to have a free flowing interaction with students, natural language was pretty much the only option. Thus began my interest in understanding natural language.

In the next few years, I built a natural language interface for the intelligent tutoring system SOPHIE that allowed a student to interact with the computer to learn electronic troubleshooting (Brown et al. 1982, Burton and Brown 1986). I was fortunate to be able to work closely with Bill Woods's Natural Language Understanding group. I learned a lot about formalisms for representing information about language and algorithms for manipulating them. They were working on complex linguistic phenomena such as conjunction, relative clauses, and the logical structure of quantification. As we started using SOPHIE with real students, we encountered a different set of linguistic problems.

Intelligent Linguistic Architectures: Variations on themes by Ronald M. Kaplan.
Miriam Butt, Mary Dalrymple, and Tracy Holloway King (eds.).
Copyright © 2006, CSLI Publications.

The students were not using particularly complex sentences. They were, however, using conversational constructs such as ellipsis ("what about T2?") and pronominal reference ("What is it for T1?"), and making plain old spelling mistakes ("What is the voltage at the base of the power limiting transitor?"). Mostly the problems I faced were engineering ones: incomplete language coverage for what students actually said; the system not being fast enough; and poor semantic mapping between existing domains and mine. In the end, the system worked pretty well. Well enough to show that the idea of tutoring students in natural language was possible, at least within a limited domain.

Eventually, I concluded that my goals and hence my problems were sufficiently different from the ones driving natural language understanding that I should be using a different name to label my efforts. I started calling my work natural language engineering. I characterized natural language engineering as using current natural language machinery to provide a "natural language" interface to a computer application. About this time, bit-mapped graphic displays were developed, opening the possibility of graphical interfaces that were inexpensive, easy to program and full of opportunities for instructional interfaces. I stopped pursuing natural language interfaces and began developing graphical user interfaces. I moved to PARC and was fortunate to able to work with Ron for several years developing Interlisp-D. We each saw Interlisp-D as a necessary platform for pursuing our research; natural language for him, graphical user interfaces and intelligent tutoring systems for me.

A few years ago, I was approached by Acuitus, Inc. to provide natural language input capability to a new digital tutor they were developing. I had kept in touch with Ron and knew he had worked very hard refining and developing his ideas about how computers should handle natural language. I jumped at the chance to find out how much progress Ron has made and to see what could be done with the three orders of magnitude more computation that is available from today's computers. This paper describes my experiences using XLE in my most recent natural language engineering effort.

5.1.1 An Intelligent Tutor for Network Administration

The current efforts are focused on building a computer-based course to teach network administration. Our subject matter can be roughly characterized as the networking fundamentals and troubleshooting techniques necessary to find and fix any problems that would prevent a user's computer from being able to browse a web site. Behind this simple description lies a large body of content that includes computer and networking hardware, ethernet and internet protocols and their implementations, networking services, client applications, web servers, and troubleshooting skills.

The course is a mixture of presentation of material, interactive activities to learn about commands, and most importantly, troubleshooting exercises in which students find and fix problems on real (not simulated) systems. The exercises are performed on a three-machine network: a client machine with web browser, a server machine with a web server, and a name server machine. During the exercises, a computer-based tutor monitors their activities and provides help either when the student asks for it or when the tutor decides to step in based on its observations.

Our goal is for our digital tutor to do as well as an excellent human tutor does when working one-on-one with a student. Figure 1 shows a systems-level view of the tutor. It shows how the tutor monitors the students actions and how XLE fits in.

5.2 Why Use Natural Language?

The tutor is written in Java and has (or could have) access to any of Java's interactive graphical and multimedia capabilities. On top of all this, why go to all the trouble of accepting natural language? In fact, the large majority of interactions that the tutor has with students are multiple choice or short answer questions (which do not use natural language). But there are some things that natural language can do that are not available otherwise.

The primary advantage of natural language is that it forces articulation of ideas onto a blank slate. This creates a different learning experience than multiple choice in which the listed choices define the set of allowable answers. The choices shape the student's thinking and allow them to use an elimination process rather than a creation process when they are not sure of the answer. Further, having the students express their thoughts in their own words is an important learning step.

In addition, natural language allows a much larger set of answers than is feasible with multiple choice. Hundreds or thousands of alternative answers can be supported with no change to the interface.

Another reason for using natural language in the interface is that learning how to express concepts and ideas about unix system administration/networking problems is one aspect of the curriculum. The course is teaching students to be system administrators. Part of being a system administrator is being able to write up what you found and changed so that you can communicate with other system administra-

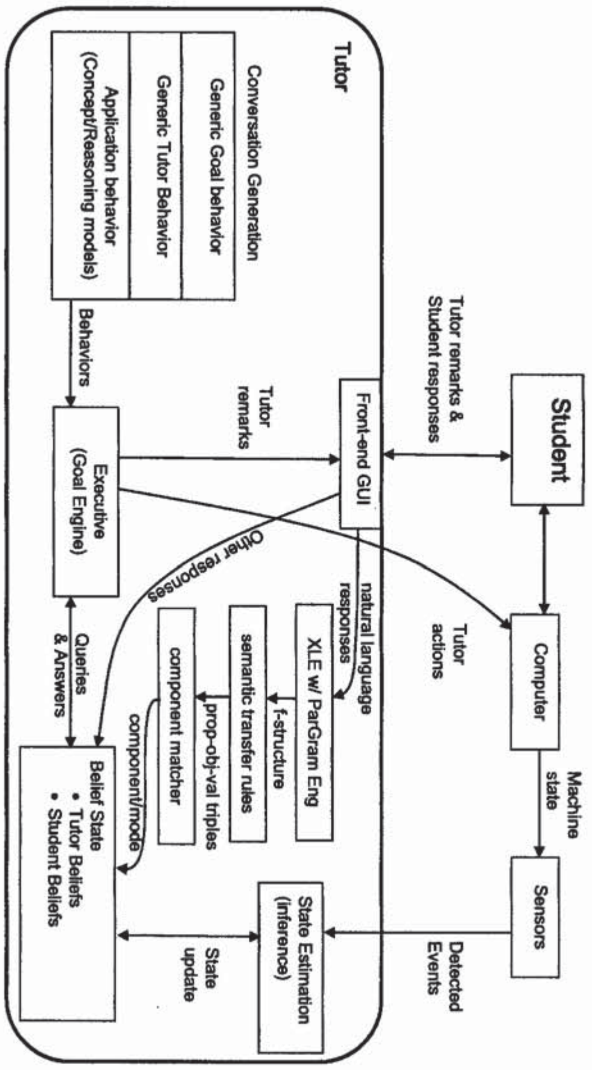

FIGURE 1 System diagram of the digital tutor showing XLE.

tors. Natural language interaction provides students with the opportunity to practice their system administrator speak.

The final reason we pursued natural language is that eventually we will want the tutor to be able to focus on meta-problem solving. Schoenfeld (1992) has demonstrated that a distinguishing characteristic of master mathematical problem solvers is awareness of their own problem solving strategies. He has been able to teach this behavior to undergraduates in a semester course. To get students to focus on meta-problem solving, he reserves the right any time the students are working on a problem to ask the following three questions:

- What (exactly) are you doing? (Can you describe it precisely?)
- Why are you doing it? (How does it fit into the solution?)
- How does it help you? (What will you do with the outcome when you obtain it?)

We eventually want to incorporate this type of behavior into our tutor and believe that interacting in natural language will be required for this ability.

5.2.1 When to Use Natural Language

Having suggested reasons why natural language is critical, it is important to keep in mind that these reasons do not apply to all interactions. Many interactions are well handled by multiple-choice and short answer questions. In fact, our initial prototype of the tutor did not include natural language at all. In the transcripts of trials with this prototype, we looked for places where natural language might be particular useful.

The interaction that stood out as the least satisfying was when the tutor asked the student "what do you think is wrong with the system?". In most sessions, the tutor asks this question early in a problem when the student has performed some tests on the system but has not yet attempted to fix anything. In this situation, the student may be almost ready to fix the problem, totally lost, or anywhere in between. Thus, responding accurately to what the student says is critical for the tutor to start off in the right direction.

Without natural language, this interaction was handled as a multiple choice question with 13 possible answers. This seemed like a good place for natural language. So, we decided to begin our use of XLE by handling student answers to this question.

5.2.2 The Range of Answers

Our tutor has a model of the things that can go wrong with the system. The model's central structure is a hierarchy of functionally-based chunks called components. Since the problems all deal with ways browsing a web page can be broken, the top level component is "the user's browser reads a web page from the server". Just below the top component are several things including "web service works on server". An example of a lower level component is "the client's current IP address." Basically, the hierarchy represents all the things that can be broken in the system.

Each component has a number of ways that it can be faulted. The most common fault is just the generic "broken" but some components can have more specific faults. For example, files are components and may have a fault of "does not exist".

This provides a very nice range of meanings for natural language answers to the question "What's wrong?", that being the set of all components and the ways they can be faulted. Any answer to the question should identify a component and a fault mode.

Example Sentences and Their Interpretations

Here are some examples of responses to the question "what's wrong with the system?" and their interpretation as component/fault mode pairs.

> "You cannot send packets back and forth between the client and the server by using their names." means that the component "the client connects to the server by name" is "broken".
>
> "The http server is not responding to requests." means that the component "web service works on server" is "broken".
>
> Both "The IP address for badmojo is incorrect in the hosts file on goodmojo." and "The apache server's entry in the client's hosts file has the wrong IP address." have the meaning that component "the IP address of the server's entry in the client's hosts file" is "broken".

5.3 How XLE is Used

When doing natural language understanding for SOPHIE in the 1970s, the starting point was a string of characters. This time around, XLE[1] allowed me to start with the deep structure functional groupings and relationships between word meanings in the sentence called f-structures. This is a significant improvement. To get a sense of how large an improvement this is, let's look at an example of an f-structure.

[1]When I use the term XLE, I am referring to both the XLE parsing/generating framework and to the ParGram English grammar (Crouch et al. 2006, Kaplan et al. 2004, Riezler et al. 2002).

5.3.1 Sample F-Structure

Let's consider the sentence "The apache server's entry in the client's hosts file has the wrong IP address." The f-structure produced by XLE is shown in Figure 2 and directly represents the sentence's functional relationships.

There are many relationships shown in Figure 2 but the important ones for our purposes are:

The top level is a 'have' relationship between an 'entry' and an 'address'.

- The 'entry' is for a 'server'.
    - The 'server' is an 'apache' server.
- The 'entry' is 'in' a 'file'.
    - The 'file' is a 'hosts' file.
    - The 'file' is a 'client' file.
- The 'address' is an 'ip' address.
- The 'address' is 'wrong'.

XLE includes the ability to specialize existing grammars, lexicons and morphologies (Kaplan et al. 2002). When I was getting started, Tracy King used this capability to create a grammar with a few domain specific features. For example, the grammar was modified to make NP an acceptable top level constituent. (Since we are asking "what's wrong?", it is reasonable to accept a noun phrase as being the thing that is wrong.) Another modification was an addition to the lexicon to allow unix specific features. As can been seen from Figure 2, the f-structure contains lexical information about the words in the sentence. By far the vast majority of this information comes from standard features in the lexicon such as HUMAN, NUM and PERS. The domain specific features shown in Figure 2 are that 'hosts' is a unix configuration file (unix-cfile) and 'apache' is a unix application (unix-app). The resulting collection of relationships provides a very good beginning to understanding the meaning of this sentence.

Initial Corpus

To guide development of the tutor's natural language understanding component, I created a corpus of 810 sentences. It was basically all the interestingly different ways I could think of to say how each of the fifty components was broken. Then I put the corpus through XLE.[2] After debugging the domain specific grammar, morphology and lexicon additions, only four of the 810 sentences had parsing difficulties. I never

[2]I was using the March 2005 release of XLE with a slightly modified ParGram English grammar from November 2004.

```
"the apache server's entry in the client's hosts file has the wrong IP address"

     PRED        'have<[80:entry], [292:address]>'
                 PRED     'entry'
                                 PRED      'in<[228:file]>'
                                           PRED  'file'
                                                    177 PRED   'hosts'
                                           MOD     1397 NTYPE [NSYN proper]
                                                   1401 HUMAN -, NUM sg, PERS 3, UNIX-TYPE unix-cfile
                                           CHECK [_LEX-SOURCE morphology]
                                           NTYPE [NSYN common]
                                                           167 PRED   'client'
                                 OBJ                       176 CHECK [_LEX-SOURCE countnoun-lex]
                 ADJUNCT                                   142
                                                          1182 NTYPE  NSEM [COMMON count]
                                           SPEC    POSS   1187        NSYN common
                                                          4943
                                                           127 SPEC   DET  PRED      'the'
                                       228                1132             DET-TYPE def
                                      1451                4981 NUM sg, PERS 3
                                      3614                4999
                                      3618
                                      4209 CASE obl, NUM sg, PERS 3
                             105
                            5463 PSEM     (loc)
                            5473 PTYPE    sem
                 CHECK    [_LEX-SOURCE morphology]
                 NTYPE    [NSYN common]
                                     PRED  'server'
                                                16 PRED   'apache'
                                     MOD       638 NTYPE [NSYN proper]
                                 70           8861 HUMAN -, NUM sg, PERS 3, UNIX-TYPE unix-app
                                 79
                 SPEC     POSS   45  CHECK [_LEX-SOURCE countnoun-lex]
                                732
                               8875  NTYPE  NSEM [COMMON count]
                               8880         NSYN common
                                  1
                                585  SPEC   DET  PRED      'the'
                               8884              DET-TYPE def
              80               8898  NUM sg, PERS 3
             947
            1021
            5229
            7744 CASE nom, NUM sg, PERS 3
                 PRED     'address'
                                344 PRED  'ip'
                 MOD           2150 NTYPE [NSYN proper]
                               2049 HUMAN -, NUM sg, PERS 3
                                307 PRED 'wrong'
                 ADJUNCT       1952 ATYPE attributive, DEGREE positive
     OBJ                       1958
             387 CHECK    [_LEX-SOURCE morphology]
            2095
            2492 NTYPE    [NSYN common]
            2687
             292 SPEC     DET  PRED      'the'
     263    1888               DET-TYPE def
    1744    2690 CASE obl, NUM sg, PERS 3
    2698
    2705 CHECK     [_SUBCAT-SOURCE oald-orig]
    7583
    7584 TNS-ASP   [MOOD indicative, PERF -_, PROG -_, TENSE pres]
    7489 CLAUSE-TYPE decl, PASSIVE -, VTYPE main
```

FIGURE 2 A sample f-structure from ParGram English for the sentence "The apache server's entry in the client's hosts file has the wrong IP address."

```
Component(var(1),EntryInFile)
ComponentFile(var(1),hosts)
EntryField(var(1),ipaddress)
OnMachine(var(1),clientmach)
OfMachine(var(1),servermach)
ComponentState(var(1),incorrect)
```

FIGURE 3 The property-object-value representation of the component DestIPEntryInHosts.destination-server.machine-client.

expect to see a student enter any of these four. I considered this to be an powerful testimonial to the capabilities of XLE.

Semantic Interpretation

Armed with usable parses, I began to explore how to get from f-structures into a representation that would mean something to the tutor. F-structures are still a good ways away from the component and fault mode that we are looking for as the meaning for our domain. Covering this distance requires understanding a little bit more about how components are structured.

Components are typed objects. Each type has a specific set of properties. For example, an EntryInFile is a type of component which represents an entry in a configuration file. It has properties:

OnMachine (the machine on which the file resides),
ComponentFile (the name of the file containing the entry),
OfMachine (the machine that the entry is about), and
EntryField (the field of the entry).

A component is determined by its type and the values of its properties. An example of an EntryInFile component is "the IP address in the entry of the server in the hosts file of the client." (Internally, we refer to components by name. This one's name is DestIPEntryInHosts.destination-server.machine-client.) This component's properties are ComponentFile='Hosts', EntryField='IPAddress', OnMachine='client', and OfMachine='server'. It could be described by the English phrases "the server's address in the hosts file on the client machine", or "the client machine's host file entry for the address of the server". The property-object-value triples representation is shown in Figure 3.

The first step in semantic interpretation is to go from the functional relationships of f-structures to property-object-value triples that are used to represent components. As one possible way of doing semantic interpretation, XLE provides a transfer rule language (Crouch 2005, this volume) that rewrites f-structures. We use transfer rules to match

```
PRED(%s,have),
arg(%s,1,%cmp1), +Component(%cmp1,EntryInFile),
arg(%s,2,%cmp2), Component(%cmp2,IPAddress),
Mode(%cmp2,%mode)
  ==> ComponentState(%cmp1,%mode),
      EntryField(%cmp1,IPAddress).
```

FIGURE 4 An example transfer rule.

pieces of the f-structure and rewrite them into domain specific property-object-value triples. Transfer rules are also used to remove f-structure information that is not relevant to our domain. The resulting triples are then used as constraints to determine the component.

The fault mode is represented by the property ComponentState. It is determined by looking at different pieces of the f-structure. Modifiers such as "working" or "good" get transferred into a "faultless" fault mode. Modifiers such as "bad", "broken", "wrong" or "incorrect" are transferred into a "broken" fault mode. If negation is present, the fault mode is switched from "faultless" to "broken" or vice versa. If the student enters a component as a noun phrase without modifiers rather than a complete sentence, the fault mode is left out. (In this case, when the triples are converted into a component, the tutor uses "broken" as a fault mode because the student was asked "what's wrong?".)

5.3.2 Transfer Rules

Figure 4 provides an example of a transfer rule to make things a little more concrete. It handles sentences roughly of the form "<entry> has the <mode> IP address" such as "badmojo's entry in goodmojo's hosts file has the wrong IP address."

Transfer rules[3] consist of a matching part, followed by "==>", followed by the replacement triples. Variables begin with a percent sign (%) e.g., %s or %cmp1. This rule is looking for an f-structure %s that has a PRED='have', a first argument (%cmp1) that is an EntryInFile Component, and a second argument (%cmp2) that is an IPAddress Component which also has a Mode.

When a transfer rule matches, all of the triples in the matching part are removed and the replacement triples are added. That is, the matching part is rewritten as the replacement part. Triples in the matching part are not removed if they are marked with a plus sign (+) as is done in the first Component clause in this rule. The effect of this rule

[3]The XLE documentation (Crouch et al. 2006) contains a complete description of transfer rules and how to use them.

2%	"flatten" depth by changing set representation and removing intermediate features.
7%	recognize machine names and equate terms that are the same within our domain.
16%	determine component parts from nouns and noun-noun modifications.
7%	produce triples from prepositional phrases and possessives.
4%	handle fault modes.
7%	build component triples from noun phrases.
41%	build component and fault mode from main verb.
6%	attach triples that are not connected to main component.
2%	handle negation.
7%	remove f-structure features that are not triples.

FIGURE 5 Functional groupings of transfer rules and the size of each group listed in the order in which they are applied.

would be to add to the EntryInFile Component the properties EntryField='IPAddress' and ComponentState=%mode. %mode will be either 'faultless' or 'broken' depending upon whether address was modified by 'right' or 'wrong' in the sentence.

Transfer rules are ordered. The first rule is applied, then the second rule is applied to the output of the first rule, and so on. In our example, the triples specifying Component and Mode are not part of the f-structure produced by XLE. They are added as a result of earlier transfer rules. The final set of rules removes the features of the f-structure that are not property-object-value triples. The resulting set of domain specific triples specify the component and fault mode.

The system has 1206 transfer rules to cover the corpus. They are summarized in Figure 5. The transfer rule file is slightly more than 225K characters. The execution time is very acceptable. The average time it takes to parse and interpret a sentence is about .6 seconds.

5.3.3 Getting to a Component

The set of triples that comes out of the application of the transfer rules is then matched against the actual set of components to determine which one was being referenced. This allows a separation between natural language issues and the practical issues of tutoring. For example, if you have been following closely, you may have figured out the triples for the phrase "the server's IP address in the host name file on the client" are:

```
Component(var(1),EntryInFile)
ComponentFile(var(1),hostname)
EntryField(var(1),ipaddress)
OnMachine(var(1),clientmach)
OfMachine(var(1),servermach).
```

However, it turns out the client's host name file does not contain IP addresses. It contains names. And it does not contain any information about the server. Having the interface between the natural language processing and the tutor be property triples gives the tutor the chance to see this misconception and, possibly, address it with the student.

This separation is also useful in cases where the set of triples is ambiguous in the sense that it represents more than one component. For example, the sentence "the host name file is wrong" does not say whether it is the client or the server host name file. Depending upon the context, the tutor may want to ask the student which machine, or, if it is clear that the student is focused on one machine, just fill it in.

5.3.4 Mishandled Sentences

After developing rules to get the correct semantics for 810 corpus sentences, the natural language understanding component was incorporated into our tutor. The tutor logs all sentences that it receives. Any that are not correctly handled are examined by hand.

The mishandled sentences we have seen fall into several categories. Some are 'word salad' like "server no ip" or "below application broken". XLE produces useable f-structures for the large majority of the sensical word salad we have seen so far. Our strategy for these has been that if XLE provides a useful f-structure and a human can determine what the student meant, transfer rules are written to pull out the semantics. If either XLE did not produce a usable f-structure or we could not figure out what the student meant, the system is not changed. In these cases, the tutor responds as if the student had said "I don't know" and asks directed questions to get at what the student knows.

Many of the mishandled sentences contain misspellings. For the unambiguously wrong ones, we added a character rewriting pre-pass. For example, 'resoution' gets changed to 'resolution'. This is effective at picking up misspellings that have occurred before. We have left the problem of incorporating a general spelling correction solution to the future. Most of the other mishandled sentences are added to the corpus and handled by adding transfer rules.

Over twelve months of development and testing with students, the corpus has grown to about 1400 sentences. Its growth so far has been nearly linear at a rate of about 50 sentences per month. Since the number of students testing the system has been increasing over time, the rate of new sentences per student is decreasing. This is what we expect. Adding 50 new sentences takes about four days of work. Most of this time is spent adding transfer rules but it also includes substantial time to test the new rules in the system.

5.4 Coverage Issues

5.4.1 Ambiguity

XLE is a very powerful system with an extensive lexicon and grammar for English. While the sentences we have encountered so far do not need all of this power, it is nice to know that if a student types in a complex sentence, XLE will probably handle it. The downside of all the coverage is ambiguity. To simplify the integration of XLE with the tutor, we started with an assumption that the semantics would not deal with multiple interpretations. The system uses the first (most probable) parse (even if one of the later ones may be more correct) and the transfer rules do not build multiple interpretations. This has generally worked well.

A common place where ambiguity arises is nominal compounds such as "http server type entry". In our case this refers to the entry in the httpd configuration file that has 'servertype' as a key. In our approach, we need a transfer rule that puts these four nouns together to create the right semantic triples. As long as the most probable parse always has the same modifying relations, a single rule will work. For this phrase, the most probable parse has 'http' modifying 'server', and 'server' and 'type' modifying 'entry'. Thus we have a rule that matches the most probable parse f-structure and creates the appropriate triples. We have yet to encounter a case where we needed multiple rules for the nominal compounds in our domain. If, in the future, the most probable parses become more problematic, XLE provides a way of calculating the measures used to determine "most probable" and we could specialize it to our corpus.

Just as our students occasionally type in sentences the tutor cannot handle, they more rarely but still occasionally type in a sentence that XLE has trouble with. One recent example is that in the f-structure for "the problem is with the transport layer or farther down" 'down' is associated with the top level 'be' relationship rather than with 'farther'. In this case, a transfer rule finds the 'down' clause and produces the right meaning. This sort of thing has not happened often, and Tracy King has fixed the problems in the next grammar release. Overall, there are less than a dozen rules out of more than 1200 that look for misplaced constituents.

does	-N XLE.
like	-N XLE.
out	-V XLE.
fail	-N XLE.
work	-N XLE.
wrong	-N XLE.
can	-N XLE; -V XLE.
name	-A XLE.
on	-A XLE.
but	-ADV XLE; -N XLE.
or	-ADV XLE; -N XLE.
and	-ADV XLE; -N XLE.

FIGURE 6 List of the word senses that were removed.

5.4.2 Improving parsing by reducing coverage

More troublesome were cases where the first parse used a word sense that clearly makes sense for English in general but not in our domain. For example, the network administration domain does not use 'out' as a verb nor 'can' as a noun, and 'does' is never used as the plural of 'doe'. Fortunately, the XLE lexical routines have a way of removing word senses. Figure 6 lists the words we have had to de-sense. Tracy King suggested that much of the effort to find cases where removing word senses might help could be automated by taking all the technical terms, seeing what their morphological analyses are, and removing any that seem unlikely. At this stage for us, doing it by hand has worked fine.

5.4.3 Do Students Use Proper English?

One of the questions I was asked when I started was "will students type real English sentences into your system?" Based on our experiences, mostly the answer is yes. We have encountered abbreviations that were new to us (e.g. idk for "I don't know") and if text messaging stays popular we expect to get more. It is possible in XLE to create lexical entries for most abbreviations that allows them to be parsed in the normal way. As described earlier, we have also seen some word salad but XLE produced a useable f-structure for most of that. So far, our strategy of treating sentences that the system does not understand as if the student had said "I don't know" is producing appropriate tutorial interactions.

5.5 Conclusion

XLE and the ParGram English grammar are amazing! XLE has never crashed during real use. The grammar has parsed everything we needed it to parse. The transfer rules provide a good mechanism for translating f-structures into domain concepts. There are a lot of them but they are organized enough to continue to be extensible. In summary, our experience with XLE has surpassed our expectations. We believe XLE will continue to work well as our application grows.

My main concern with XLE is the amount that must be known to use it. You need to know morphology and XLE's language for representing morphology. You need to know lexicography and XLE's language for representing it. Thanks to Tracy King, I have not had to modify the grammar but I did need to learn lots of details about the grammar such as what the difference is between an adjunct_x, a mod_x, and an XCOMP. (For our domain, adjunct_x and mod_x are treated the same. Adjunct_x and mod_x are deep structure relations while XCOMP is a surface structure relation. And mostly, we only need consider deep structure relations.) If you want to include domain specific morphology, you need to learn Finite State Morphology, a task that begins with the book of the same name by Beesley and Karttunen (2003). You need to decide how to do semantic interpretation. This will probably involve learning yet another language such as the transfer rule language (which I recommend). XLE will shortly contain a transfer rule based semantics along with its English Grammar that promises to reduce the number of transfer rules needed. This will help.

Much of the application effort for XLE has been targeted at natural language translation. And I suspect that XLE's learning curve in this application is less. From the standpoint of a builder of interactive applications, XLE is a collection of well built, mostly complete parts that can be assembled in different ways. Each application needs to be custom built. We have yet to discover the right point of view on natural language use in interactive applications to make it easier to use. But until we do, XLE has the workbench of tools and parts to make any natural language engineer happy.

Acknowledgments

This paper is dedicated to Ron Kaplan for many years of warmth, friendship and good ideas. Thanks for the progress on all the intractable problems. This work was supported in part by the Defense Advanced Projects Agency within the DARWARS program (Contract #N00014-030C-0295). I would like to express gratitude to Ralph Chatham for his support. Special thanks to Tracy King for her extensive, quick, knowledgeable support and for helping me get started with a customized grammar. Thanks to Tracy King, Rich Levinson and an anonymous reviewer for comments on this paper.

References

Beesley, Kenneth R. and Lauri Karttunen. 2003. *Finite State Morphology*. Stanford, CA: CSLI Publications.

Brown, John S., Richard R. Burton, and Johan deKleer. 1982. Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III. In D. Sleeman and J. S. Brown, eds., *Intelligent Tutoring Systems*, pages 227–282. New York, NY: Academic Press.

Burton, Richard R. and John S. Brown. 1986. Toward a natural language capability for computer-aided instruction. In B. J. Grosz, ed., *Readings in Natural Language Processing*, pages 605–625. Los Altos, CA: Morgan Kaufmann.

Crouch, Richard. 2005. Packed rewriting for mapping semantics to KR. In *Proceedings of the 6th International Workshop on Computational Semantics (IWCS-6)*, pages 103–114. Tilburg, The Netherlands.

Crouch, Richard, Mary Dalrymple, Ronald M. Kaplan, Tracy H. King, John T. Maxwell, III, and Paula Newman. 2006. XLE Documentation. Palo Alto Research Center.

Kaplan, Ronald M., Tracy H. King, and John T. Maxwell, III. 2002. Adapting existing grammars: The XLE experience. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02), Workshop on Grammar Engineering and Evaluation*, pages 29–35. Taipei, ROC.

Kaplan, Ronald M., Stefan Riezler, Tracy H. King, John T. Maxwell, III, Alexander Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the Human Language Technology Conference and the 4th Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'04)*, pages 97–104. Boston, MA.

Riezler, Stefan, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, pages 271–278. Philadelphia, PA.

Schoenfeld, Alan H. 1992. Learning to think mathematically: Problem solving, metacognition, and sense-making in mathematics. In D. A. Grouws, ed., *Handbook for Research on Mathematics Teaching and Learning*, chap. 15, pages 334–370. New York, NY: MacMillan.