Construction and Analysis of an LKB grammar

Chris Becker

1 Introduction

My goal for this project was to research the construction of an LKB grammar and determine the feasibility for its future use in other projects, such as Rapaport and Kibby's CVA project. In this paper I discuss some of the aspects of constructing a new grammar and modifying a pre-existing one. I describe my own efforts doing each, and provide a comparison of the results. Finally, I will go into some of the results from the current implementation of an LKB to SNePS converter.

2 Choosing a grammar model

Copestake (2002) provides a discussion on creating new grammars for the LKB. She suggests that "it is best to start from one of the sample grammars rather than to build a new grammar completely from scratch"; however if a particular project requires having a grammar in a completely different framework from HPSG, then it may be necessary to start from scratch.

The first step I took was to select a grammar with an adequate ability to parse text with an intermediate level of complexity, yet simple enough to be understood by someone who had never worked on an LKB grammar before. The grammars I initially looked at were the g5lex and g8gap from the book "Implementing Typed Feature Structure Grammars"; the "textbook" grammar, based on the grammar in Sag, Wasow, and Bender's "Syntactic Theory"; and finally, the Ergo grammar, developed by the CSLI LinGO Lab.

The first two grammars, g5lex and g8gap had the simplicity to be easily understood, but not the complexity to handle a satisfactory number of grammatical English constructions. G5lex provides an adequate overview of how types, rules, and feature structures are put together in an LKB grammar. G8gap, in addition to adding to the types of grammatical constructions that the G5lex handles, also uses a different format to represent the lexicon, which bears some similarities to that used in the ergo grammar. In this aspect, g8gap is probably a good place to start if one is interested in trying to understand how the ergo grammar is written.

Another drawback of the above two grammars is that they lack a number of lexical types necessary for creating complex English sentences. Both lack definitions for adjectives and adverbs, and thus also any grammar rules to account for them. Since these grammars are so small as they are, modifying them to account for these parts of speech would probably require modifications to its entire overall design. Rather than do this, I decided it would be optimal to find a grammar that accounted for these things already.

The next grammar I looked at was CSLI's ergo grammar. With its lexicon of almost 10,000 words, and scores of grammatical and lexical rules to follow, it is without a doubt the optimal grammar to use for a project that requires a robust parser. However, for my purposes it was far too large and complex to completely reverse engineer. Instead, I opted to use it as a baseline grammar, so that with some slight additions to the lexicon I could use it to compare to the results of a simpler grammar that I could put together.

I finally found a good balance of simplicity and functionality in the textbook grammar. All the necessary lexical types were accounted for in the type hierarchy and rule definitions, and it was nowhere near as large or complex as the ergo. Additionally, since it was based on the Sag, Wasow, and Bender book, I was able to use it as a reference when analyzing the grammar. I decided to use this grammar in my analysis since it presented a fairly standard construction of an HPSG framework.

3 Grammar construction

The primary goal in the creation of this grammar was to have it parse a majority of the 37 sentences in the test corpus, and be able to explain why any remaining sentences would not parse. The test sentences used come from the simplified version of the "brachet" passage used for the CVA project, which is derived from Thomas Malory's Morte d'Arthur. I chose this as the test corpus because work as been done to create an ATN grammar for this text as well, and so this would be useful for comparison. The sentences used are listed in appendix A.

These sentences provided good test material because of the variation of grammatical complexity they contained; from simple declaratives to conjoined clauses and relative clauses. In total, this test data was. Their breakdown into different parts of speech were as follows: 29 verbs, 11 prepositions, 4 proper nouns, 27 common nouns, 4 determiners, 4 conjunctions, and 4 adjectives.

In order to best understand how a grammar is constructed, I took apart the textbook grammar piece by piece, and then put it back together, using only what was necessary to make it fully functional, and then added features as necessary to account for certain constructions in the test corpus.

One aspect of this process involved documenting each type definition in the code based on four features: the types it inherits from, the types that inherit from it, the types that use it in their definition, and the types it uses in its definition. For each of these types I list the name and the file it is defined in. With large grammars this type of documentation is essential for allowing someone new to the LKB to quickly follow its construction. This could be important when making any modifications or additions to the type hierarchy so that the use of each component can be easily traced. An example of the format of this documentation is given below for the type arg_pred.

```
;;; arg_pred
;;; inherits from:
;;;   predication:  types.tdl
;;; inherited by:
;;;   fiv_pred:     semantics.tdl
;;;   srv_pred:     semantics.tdl
;;; used by:
;;;   modifier-sem: semantics.tdl
;;; uses:
;;;   sit-index:    types.tdl

arg_pred := predication &
 [ ARG sit-index ].
```

The overall file structure of the grammar is as follows:

```
irregs.lisp             irregular spelling rules
binding.tdl             implementations of Argument Realization Principle
forms.tdl               forms for different parts of speech
grules.tdl              grammar rules
grule-types.tdl         types required for grammar rules
inflr.tdl               inflection rules
lexemes.tdl             definition of lexical types
lexicon.tdl             definition of word types
lrules.tdl              lexical rules
lrule-types.tdl         types required for lexical rules
parse-nodes.tdl         definition of constituent types
parts-of-speech.tdl     defines part of speech types
roots.tdl               defines root
semantics.tdl           semantics related type definitions
types.tdl               syntax related type definitions
```

Prior to rebuilding a grammar there are some issues that have to be addressed:

1. what lexical entries are already in the grammar?
2. what word classes are already in the grammar?
3. what lexical items need to be added?
4. what word classes need to be added?
5. what rules are defined already?
6. what rules need to be defined to parse the entire corpus?

As I mentioned earlier, the textbook grammar already had all the necessary word classes and a basic set of rules capable of handling most of the types of constructions in the test corpus. However there were a number of phenomena that it did not account for. In this section I will explain what they were and how I accounted for them.

possessives with ”’s”

The original textbook-based grammar did not support possessives ending with ” ’s ”. To fix this, I either had to define ” ’s ” in the lexicon, as well as the rules to combine it with other constituents, or represent the entire possessive as a single unit, which is somewhat less efficient but easier to do on a small lexicon. I opted for the latter choice, and turned to the ergo grammar for an idea of what type this combined lexeme should have; it had to be a type already present in the textbook-based grammar.

It turns out that ergo treats the ” ’s ” as a determiner, and the combined NP + ” ’s” is also labeled as a determiner in the final parse tree. In my own grammar I did not have the rules for combining these parts together, and creating one would likely require modifications in numerous places, which is no simple task since this grammar is not written in a very modular, decoupled fashion. So, instead I represented the entire possessive as a single type of determiner. Since there were only two instances of this construction in the test corpus, it was not time consuming to implement.

One side effect of this was that a possessive could not follow a standard determiner. For example, ”The brachet bites the hart’s buttock.” Would not parse because it would not allow a determiner to follow another determiner, e.g. ”the hart’s”. As a result the circumvention for ” ’s ” was only effective for proper nouns, e.g. as in ”King Arthur’s hall”.

Verb particles

Another obstacle was accounting for verb particles such as ”up” in ”pick up”. Here too I followed the example of the ergo grammar, and simply represented this combination as a single lexical item. This allowed the successful parse of sentence (8), ”The knight picks up the brachet.”

”or else”

The compound conjunction ”or else” posed problems in both the textbook-based grammar and the modified ergo grammar. Since ”or else” has the same meaning as ”or” (”else” adds nothing to the phrase in this context), I represented this as a single lexical item in both grammars, using the same semantic and syntactic supertypes as ”or”.

”either”

The word ”either” also posed a problem for both grammars because it was not used as a standard conjunction, but rather at the head of the conjoined phrase. As a result, the manner in which the clause containing it should parse is fairly unique to the use of this one word. As such, neither grammar had any rules to account for this particular use of ”either”. Sentence (17) which contained this construction was the only sentence that was not parsed by either the textbook-based or ergo grammar.

Titles and adjectives

Titles such as ”sir” or ”king”, which are used frequently in the test corpus presented a problem because there was no predefined lexical type that could account for its use. One possibility could be to use it as a determiner, however this would conflict with the fact that they must take a proper noun as its complement. A better alternative would be to represent it as an (attributive) adjective. The only problem with this was that attributive adjectives could not be parsed by the textbook-based grammar. The only use of adjectives that its rules allowed for was as the complement of the main verb. I was unable to add the necessary rules to fix this problem, as this structure would be determined from a number of separate type definitions in the grammar, and due to the strong interlinking of everything it would not be possible to make any sweeping changes without breaking some other component.

Fortunately, the ergo grammar is able to handle titles and adjectives just fine, although due to the great structural differences between it and the textbook-based grammar, it would not be possible to copy the rules over.

Genitives

The original design of the textbook grammar could not account for possessive constructions such as ”the X is Y’s”. One way I attempted to fix this was to define certain genitives such as ”his” and ”hers” as adjectives, since the textbook grammar was capable of parsing such forms with an adjective in the position of ”Y”. This proved to work, so I then created a genitive lexeme type based roughly on the adjective type definition, so that the resulting parse would be more accurate.

WH-words

Wh-words presented a problem in the textbook-based grammar because they were not represented in the original textbook grammar lexicon. The solution to this was to follow the structure of other nominal pronouns (e.g he, she) and create a type for neutral gender. Although the sentences containing ”who” in the test corpus were not parsed, this is mainly due to other unparsable grammatical structures in the test corpus.

The ergo grammar was able to parse relative clauses, however, it did so with no small amount of trouble. The sentences containing these turned out to be the most computationally expensive ones to parse, and often ended up generating errors or crashing the system. However, this might have also been due to the overall length of the sentence, which was larger than the ones without a relative clause. Sentence (22), although it parsed, did not do so correctly. Three of the four sentences that initially caused errors were parsed once the global variable for maximum-number-of-edges was increased dramatically. System memory may have been the only issue preventing the remaining sentence (16) from parsing. For reference, the system I used had 512 Mb RAM, and a clock speed of 1000Mhz.

Modifying the ergo grammar

The ergo grammar already dealt with all of the above issues, with the exception of ”or else” and ”either” in the grammatical contexts they were in. As such, the main benefit of using the ergo grammar is that it

requires very little modification since it has such a large lexicon and collection of rules. Currently this grammar has 9920 words, 6606 unique word stems, and 82 semantic categories.

Adding words to the lexicon is a simple process since all the necessary word classes are defined already. Each item in the lexicon inherits from a lexical type and a semantic relation, following from a standard HPSG representation. The simplest way to add a new item to the lexicon is to base its definition on that of a similar word that is already in the lexicon. Should this not be possible, then the word must be matched with a lexical type; to determine the semantic relation for a new lexical entry to inherit, it would be necessary to study the definition of each type in the type heirarchy. Unfortunately the designers of the ergo did not include much internal documentation in the grammar so determining how intertwined the code is from the bottom up would require extensive work.

The lexical entries added to the ergo grammar are listed in appendix C.

4 Results

The textbook-based grammar

The results of parsing the test corpus with the textbook-based grammar is as follows:

19/37 were parsed.
9/37 returned no parse.
9/37 returned errors.

The resulting parse trees for the sentences parsed are given in appendix B. Overall I was satisfied that it parsed more than 50% of the sentences parsed, and did so correctly as well. The main factors that prevented the remaining sentences from parsing include those that I mentioned in the previous section that I was unable to circumvent.

The results show that the grammar is capable of handling relative clauses to a limited extent, as seen in the parses of sentences (11), (12), and (13). However, the inability for it to parse the other sentences with relative clauses may have been due to additional elements that were not parsable by this grammar, such as attributive adjectives.

Modified Ergo

With the addition of the necessary lexical items, the ergo grammar was able to correctly parse almost all of the test corpus with no difficulty. The results of the initial test run were as follows:

Initial results:

4/37 returned an error regarding *maximum-number-of-edges*
(Sentences: 14,16,22,37)
33/37 parsed

The only errors returned related to exceeding the value of *maximum-number-of-edges*. This global variable specifies ”a limit on the number of edges that can be created in a chart, to avoid runaway grammars taking over multi-user machines” (Copestake 2002). By default, this value is set to 500; in the ergo, this variable is set to 4000. I performed two more runs with the modified ergo grammar, with the value of *maximum-number-of-edges* increased first to 8000, and then to 16000. The results of each of these runs are given below.

After increasing *maximum-number-of-edges* from 4000 to 8000:

2/37 returned an error regarding *maximum-number-of-edges*. (S.14,22)
1/37 returned no parse. (S.16)
34/37 parsed. (S.37 parsed now)

After increasing *maximum-number-of-edges* from 8000 to 16000:

1/37 returned error an regarding *maximum-number-of-edges*. (S.16)
36/37 parsed.
1/36 parsed incorrectly. (S.22)

The resulting parse trees are given in appendix D. Overall, the ergo grammar had the most difficulty parsing relative clauses in terms of accuracy and processing time. Sentence (22), which contains several embedded relative clauses returned an incorrect parse tree. One uncertainty, however, is whether or not in this case the tree turned out wrong while the actual parse was correct. To resolve this we could analyze the MRS, but I will have to leave this as future work to be done.

5 Implementation with SNePS

As part of my initial goal of researching the possible uses of the LKB in other projects, I examined the current progress of the ”snepsifier” currently being worked on by Anthony Ekeh and David Pierce. The snepsifier uses the MRS output of a parsed sentence to generate the appropriate representation of that sentence in a SNePS network. Currently the program only works on a limited number of cases, and is able to produce basic representations of the main predicate arguments in a sentence.

Of the 37 sentences in the test corpus, 36 of which were parsed by the LKB using the modified ergo grammar, 10 were successfully converted to the SNePSUL code needed to generate the appropriate nodes in SNePS. The sentences that were able to generate a SNePS representation successfully are listed below.

Sentence	SNePS translation
(1) A hart runs into King Arthur's hall.	(BUILD OBJECT "hall" REL "hall" POSSESOR "arthur") (BUILD AGENT "hart" ACT (BUILD ACTION "run" OBJECT "hall")
(4) The hart runs next to the Round Table.	(BUILD AGENT "hart" ACT (BUILD ACTION "run")
(5) The brachet bites the hart's buttock.	(BUILD OBJECT "buttock" REL "buttock" POSSESOR "hart") (BUILD AGENT "brachet" ACT (BUILD ACTION "bite" OBJECT "buttock")
(6) A knight arises.	(BUILD AGENT "knight" ACT (BUILD ACTION "arise")
(7) The knight picks up the brachet.	(BUILD AGENT "knight" ACT (BUILD ACTION "pick" OBJECT "brachet"))
(8) The knight mounts his horse.	(BUILD AGENT "knight" ACT (BUILD ACTION "mount" OBJECT "horse")
(9) The knight rides his horse.	(BUILD AGENT "knight" ACT (BUILD ACTION "ride" OBJECT "horse"))
(10) The knight carries the brachet.	(BUILD AGENT "knight" ACT (BUILD ACTION "carry" OBJECT "brachet"))
(26) The lady is sleeping in the pavilion.	(BUILD AGENT "lady" ACT (BUILD ACTION "sleep"))
(29) The brachet bays in the direction of Sir Tor.	(BUILD AGENT "brachet" ACT (BUILD ACTION "bay"))

In general, the snepsifier was able to generate a representation for sentences of the form: *det noun verb PP/NP*, and only in cases where the final PP or NP contained a fairly simple structure. For some reason, the program failed to generate a SNePS representation for sentences beginning with a proper noun. Sentences containing a relative clause did not work as well.

Another issue to take note of is that the snepsifier only works on the MRS produced by the ergo grammar. When running the snepsifier in conjunction with the textbook-based grammar, it did not generate any SNePS representations.

6 Future work

Future work on LKB grammars will probably focus closely on their ability to be coordinated with other systems, such as SNePS. I foresee little need to focus on the construction of grammars since the most efficient way of using the LKB is to adapt an existing grammar to a project's specific needs. There is probably little need to look into any other grammar besides the ergo grammar, as this is the most comprehensive grammar available. The only modifications that need to be made in order to adapt it for a specific purpose is to replace the files lexicon.tdl and semrels.tdl with custom made ones. As long as the customization is limited to these two files, they should be portable to future iterations of the ergo grammar by LinGO.

7 Conclusion

My overall goal was to provide a sufficient overview of the use of an LKB grammar so that other researchers may assess the potential for this system to play a role in their projects. In order to accomplish this, I broke down and built up a moderately sized grammar based on the pre-existing "textbook" grammar, and modified the ergo grammar. Results on the test corpus showed that the textbook grammar was able to account for roughly half of the grammatical constructions present in it, while the ergo grammar was able to account for all of them.

The grammar that I built, based on the textbook grammar was not meant to be very extendible. My main goal in building it was to make the code for the grammar easy to trace through. In modifying the ergo grammar my goal was to determine the benchmark for successful parses that the LKB could provide.

The most efficient method to using the LKB is to choose a grammar with the most complete and sound grammatical coverage, and then customize the lexicon. The structure for most words can be derived from words that are already in the lexicon. In the ergo grammar, nearly all lexical types are accounted for, so little modification would have to be made to adapt it to a specific project.

Among the projects that would benefit from the LKB is Rapaport and Kibby's CVA project. The process of converting natural language input into a SNePS representation is something that would greatly benefit this project as it would greatly speed up the process of testing the definition algorithms on the semantic representations of various passages. Currently the majority of time spent by students in the CVA project is used to hand-code the semantics of passages into SNePS. With a robust all-purpose grammar, this part of the research could be done with much greater ease, and work on this project could be focused

where it should be, on studying the algorithms themselves. Currently the ”snepsifier” works with mixed success, and so much work will still need to be done on it before it can be employed in this project.

References

[1] Copestake, Ann. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications, Stanford University.

[2] Sag, Ivan A., Wasow, Thomas and Bender, Emily M. 2003. *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford University.

[3] Carroll, John; Copestake, Ann; Malouf, Robert, and Oepen, Stephan. 1991-2002 The LKB System. http://www-csli.stanford.edu/ aac/lkb.html.

[4] Copestake, Ann, et al. LinGO English Resource Grammar http://lingo.stanford.edu/erg.html

[5] Callison-Burch, Chris and Guffey, Scott. Textbook Grammar Implementation. As part of the LKB package.