Using a Logic Grammar to Learn a Lexicon

Manny Rayner, Åsa Hugosson, Göran Hagert

Swedish Institute of Computer Science
Box 1263
S-164 28 KISTA
Sweden

Tel: +46-8-7521500

Summary

It is suggested that the concept of "logic grammar" as relation between a string and a parse-tree can be extended by admitting the lexicon as part of the relation. This makes it possible to give a simple and elegant formulation of the process of infering a lexicon from example sentences in conjunction with a grammar. Various problems arising from implementation and complexity factors are considered, and examples are shown to support the claim that the method shows potential as a practical tool for automatic lexicon acquisition.

Keywords: Logic programming, Prolog, logic grammar, learning, lexicon.

Topic Area: Theoretical issues

1. Introduction

The basic idea is as follows: a logic grammar [1] can be viewed as the definition of a relation between a string and a parse-tree. You can run it two ways: finding the parse-trees that correspond to a given string (parsing), or finding the strings that correspond to a given parse-tree (generating). However, if we view the lexicon as part of this relation, we get new possibilities. More specifically, we can compute the *lexicons* that correspond to a given string; this can in a natural way be viewed as a formalization of "lexicon learning from example sentences". In terms of the "explanation-based learning" paradigm, this makes the associated parse-tree the "explanation" (See diagram 1).

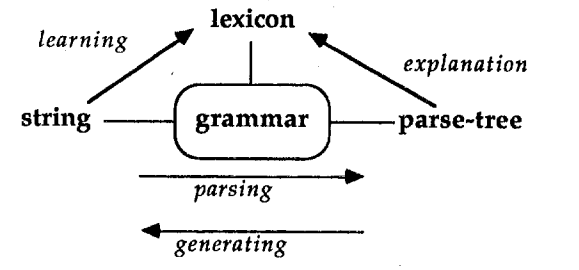

Diagram 1

In what comes below, we are going to consider the following questions:

1) We are learning from positive-only examples. What can't be learned like this?

2) The basic structural constraint, the thing that makes it all work, is the assumption that a word can *usually* only be interpreted as one part of speech. If we assume that this is always going to be true, then things really go pretty well (Section 2). However, this rule is broken sufficiently often that a realistic system has to able to deal with it. How?

3) How important is the order in which examples are presented? Can the system select a good order itself, if it is important?

4) What kind of complexity features are there? How scalable is it in terms of number of sentences, number of grammar rules, number of words to learn?

2. Learning with the "one entry per word" assumption.

This is the simplest variant of the idea: assume that there is one entry per word, and represent the lexicon as an association-list (alist) with one entry for each word. Each sentence now constrains the possible values of these entries to be ones which allow it to be parsed; the hope is that a conjunction of a suitably large number of such constraints will be enough to determine the lexicon uniquely.

In concrete Prolog programming terms, what this means is the following. In the

initial lexicon, the entries are all uninstantiated. We use this to parse the first sentence, which fills in some entries; the resulting partially instantiated lexicon is sent to the second sentence, which either refutes it or instantiates it some more, and the process is repeated until we get to the end. If at any stage we are unable to parse a sentence, we just backtrack. If we want to, we can continue even after we've got to the end, to generate all possible lexicons that are consistent with the input sentences and the grammar (and in fact we ought to do this, so as to know which words are still ambiguous). This procedure can be embodied as a one-page Prolog program (see diagram 2), but despite this it is still surprisingly fast on small examples (a grammar with 15-30 rules, 10-15 sentences with a total of 30-40 words to learn). We performed some experiments with this kind of setup, and drew these conclusions:

1) Certain things can't be learned from positive-only examples. For example (at least with the grammars we have tried), it is impossible to determine whether *belongs* is a verb which takes a PP complement with preposition *to*, or is an intransitive verb which just happens to have a PP modifier in all the sentences where it turns up. However, things of this kind seem fairly rare.

2) Order is fairly critical. When examples are presented at random, a run time of about 100 seconds for a 10-12 sentence group is typical; ordering them so that not too many new words are introduced at once drops this to about 5 seconds, a factor of 20. This gets worse with more sentences, since a lot of work can be done before the system realizes it's got a wrong hypothesis and

backtracks.

```
learn(Sents,L):-
 start_lex(Sents,L),
 learn_1(Sents,L).

learn_1([],L).
learn_1([F|R],L) :-
 parse(F,L),
 learn_1(R,L).

parse(Sent,L) :- s(Sent,[],L).

start_lex(Sents,L) :-
 setof([W,_],S^(member(S,Sents),
              member(W,S)),L).

lex_lookup(Word,Lex,Class) :-
 member([Word,Class],Lex).

% Example grammar:

s(L) ---> np(L),vp(L).
np(L) ---> det(L),noun(L).
vp(L) ---> iv(L).
vp(L) ---> tv(L),np(L).
det(L) ---> [D],{lex_lookup(D,L,det)}.
noun(L) ---> [N],{lex_lookup(N,L,noun)}.
iv(L) ---> [V],{lex_lookup(V,L,iv)}.
tv(L) ---> [V],{lex_lookup(V,L,tv)}.
```

Diagram 2

3) A more important complexity point: *structural* ambiguities needn't be *lexical* ambiguities; in other words, it is quite possible to parse a sentence in two distinct ways which still both demand the same lexical entries (in practice, the most common case by far is NP/VP attachment ambiguity). Every such ambiguity introduces a spurious duplication of the lexicon, and since these multiply we get an exponential dependency on the number of sentences. We could conceivably have tried to construct a grammar which doesn't produce this kind of ambiguity (cf. [2], pp. 64-71), but instead we reorganized the algorithm so as to collect after each step the set of all possible lexicons compatible with the input so far. Duplicates are then eliminated from this, and the result is passed to the next step. Although the resulting program is actually considerably more expensive for small examples, it wins in the long run. Moreover, it seems the right method to build on when we relax the "one entry per word" assumption.

3. Removing the "one entry per word" assumption.

We don't actually remove the assumption totally, but just weaken it; for each new sentence, we now assume that, of the words already possessed of one or more entries, at most one may have an unknown alternate. Multiple entries are sufficiently rare to make this reasonable. So we extend the methods from the end of section 2; first we try and parse the current sentence by looking up known entries and filling in entries for words we so far know nothing about. If we don't get any result this way, we try again, this time with the added possibility of *once* assuming that a word which already has known entries in fact has one more.

This is usually OK, but sometimes produces strange results, as witness the following example. Suppose the first three sentences are *John drives a car, John drives well,* and *John drives*. After the first sentence, the system guesses that *drives* is a transitive verb, and it is able to maintain this belief after the second sentence if it also assumes that *well* is a pronoun. However, the third sentence forces it to realize that *drives* can also be an intransitive verb. Later on, it will presumably meet a sentence which forces *well* to be an adverb; we now have an anomalous lexicon where *well* has an extra entry (as pronoun), which is not actually used to explain anything any longer. To correct situations like this one, a two-pass method is necessary; we parse through all the sentences a second time with the final lexicon, keeping count of which entries are actually used. If we find some way of going through the whole lot without using some entry, it can be discarded.

4. Ordering the sentences

As remarked above, order is a critical factor; if words are introduced too quickly, so that the system has no chance to disambiguate them before moving on to new ones, then the number of alternate lexicons grows exponentially. Some way of ordering the sentences automatically is essential.

Our initial effort in this direction is very simple, but still seems reasonably efficient; sentences are pre-ordered so as to minimize the number of new words introduced at each stage. So the first sentence is the one that contains the smallest number of distinct words, the second is the one which the smallest number of words not present in the first one, and so on. We have experimented with this approach, using groups of between 20 and 40 sentences and a grammar containing about 40 rules. If the sentences are randomly ordered, the number of alternate lexicons typically grows to over 400 within the first 6 to 10 sentences; this slows things down to the

point where further progress is in practice impossible. Using the above strategy, we get a fairly dramatic improvement; the number of alternates remains small, reaching peak values of about 30. This is sufficient to be able to process the groups within sensible times (less than 15 seconds per sentence average). In the next two sections, we discuss the limitations of this method and suggest some more sophisticated alternatives.

5. Increasing efficiency

It is rather too early to say how feasible the methods described here can be in the long term. As far as we can see, scalability is good as far as grammar-size is concerned; we have increased the number of rules from 15 in the first version to about 40 in the current one with little performance degradation. Scalability with respect to number of sentences is more difficult to estimate. Using the methods described in sections 3 and 4, we have sucessfully processed groups of up to 50 sentences (about equally many words), with run times typically in the region of 10-15 minutes. An example is shown in the appendix. It is reasonable to suppose that the system as it stands would be capable of dealing with groups up to four or five times this size (i.e. 200-250 words to learn), but it has a limit; the problem is that there are always going to be a few words in any given corpus which occur insufficiently often for their lexical class to be determinable. Although these words are typically fairly rare, the ambiguities they introduce multiply in the usual way, leading to an eventual

breakdown of the system. The following tentative ideas represent some approaches to this problem which we are currently investigating.

What appears to be necessary is to find some intelligent way of utilising the fact that the various alternate lexicons all agree on the majority of entries; typically, less than 10% are ambiguous after any given step in the processing. The current system completely ignores this, representing each lexicon as a separate entity. If we are to improve this state of affairs, we can envisage two possible plans. Firstly, we could simply remove the "difficult" words, hoping that there are sufficiently few for this not to matter. More ambitiously, we can try to share structure between lexicons, so that the common part is not duplicated. We now expand on these two ideas in more detail.

5.1. Removing "difficult" entries

At regular intervals the group of alternate lexicons is analyzed: the normal state of affairs is that they are identical excepting the entries for a few words, the potential "troublemakers". What one could do would be simply to remove these entries, making them once again uninstantiated; then all sentences containing the offending words would be removed from the subgroup marked as already having been processed, and saved for possible future use. The overall effect would be to reduce the group of alternate lexicons to a single "lowest common denominator", which would represent the "reliable" information so far acquired, this at the expense of losing some partial information on the "dubious" words.

We have carried out a few simple experiements along these lines, using a variant of the idea which at each "check-point" removes all ambigous words for which there are no further sentences awaiting processing. This seems at first sight very reasonable, but unfortunately it turns out that there are problems. Although one might easily think that an ambiguous word is going to stay ambiguous if it doesn't occur in any of the remaining sentences, in actual fact this is not so; a word can be disambiguated "indirectly", as a result of other words being disambiguated. To give a simple example: suppose that the first sentence is *The zebra laughed*. This can give rise to a number of possibilities: for example, *the* and *laughed* could be pronouns, and *zebra* a transitive verb. If the word *zebra* didn't occur again, one would thus wrongly conclude that there was no way of determining whether it was a common noun or a transitive verb. But this can easily be accomplished if *the* or *laughed* are later assigned to their proper classes, which will then remove the incorrect interpretation and indirectly make *zebra* unambiguous too. Clearly, a more sophisticated implementation is required if this idea is going to work.

5.2. "Lexicon compaction" using Prolog constraints

Here, we discuss the idea of exploiting the similarity between different alternate lexicons to "merge" or "compact" them. The technical tool we will be using to perform this operation is the Prolog "constraint" mechanism [3], [4]. What we propose is illustrated in diagram 3, which shows two alternate lexicons, differing in a single entry. These can be combined into the third lexicon without any loss of information.

Simple compaction of two lexicons

Two alternate lexicons for the sentence: `the dog belongs to the man`

```
[[the:d],[dog:n],[belongs:v(intrans)],
 [to:prep],[man:n]]

[[the:d],[dog:n],[belongs:v(prep(to))],
 [to:prep],[man:n]]
```

These can be compacted into the following single lexicon

```
[[the:d],[dog:n],
 [belongs:<X:X=v(prep(to);X=v(intrans)>],
 [to:prep],[man:n]]
```

Diagram 3

The technique is potentially very powerful, and in favourable circumstances can be used to compact together large numbers of alternates, as diagram 4 illustrates.

Compacting four lexicons into one in a two-stage process.

```
lex1:  [ ... [belongs:v(intrans)], ...
[plays:v(intrans)], ...]

lex2:  [ ... [belongs:v(prep(to))], ...
[plays:v(intrans)], ...]

lex3:  [ ... [belongs:v(intrans)], ...
[plays:v(prep(with))], ...]

lex4:  [ ... [belongs:v(prep(to))], ...
[plays:v(prep(with))], ...]
```

In the first stage, we compact lex1 and lex2 to make lex12, and lex3 and lex4 to make lex34.

```
lex12: [ ... [belongs:
<X:X=v(prep(to);X=v(intrans)> ], ...
             [plays:v(intrans)], ...]

lex34: [ ... [belongs:
<X:X=v(prep(to);X=v(intrans)> ], ...
             [plays:v(prep(with))], ...]
```

Then we compact lex12 and lex34 to get the final result.

```
      [ ... [belongs:
<X:X=v(prep(to);X=v(intrans)> ], ...
[plays: <Y:Y=v(prep(with);Y=v(intrans)> ],
...]
```

Diagram 4

What makes the "compaction" method so attractive is that it appears to get the best of both worlds: no information is lost, but substantial efficency gains can be attained. The method draws its power from the fact that it is "intelligent" about divergences between lexicons: if the sentence to be parsed contains none of the "constrained" words, then the compacted lexicon will behave as though it were a single, unambiguous, lexicon; but if "constrained" words are present, then the lexicon will be "split" again, to exactly the extent required by the various parsings of the sentence. It is to be noted that all this of course requires a Prolog constraint mechanism which is both efficient and logically complete, something that has

only recently become possible [4]. We are currently in the process of implementing the method within our system.

6. Conclusions and further directions

We have described a series of experiments which investigate the feasibility of automatically infering a lexicon from a logic grammar and a set of example sentences; this stands in fairly sharp contrast to most work done so far within the field of automatic language acquisition, where the emphasis has been either on grammar induction e.g. [5], [6], [7], or learning of word senses [8]. In view of the fact that much recent linguistic research has been moving towards unification-based formalisms where the bulk of the information is stored in the lexicon, we think that ideas like the ones we propound here should have a rich field of application. For example, Pollard and Sag's HPSG framework [9] has at only a couple of dozen grammatical rules, all of which are extremely general; the rest of the information is lexical in nature.

Although we think that progress to date has been extremely encouraging, it is still a little too early to make any firm claim that our methods are going to be usable in a practical system. As discussed above, there are some non-trivial efficiency problems to be overcome: it also seems likely that we will need a more sophisticated ordering algorithm than that described in section 4, probably incorporating some notion of giving higher priority to sentences containing ambiguous words. Other important topics which we so far have not had time to devote attention to are the use of morphological information and the development of some way of handling incorrect sentences (maybe just ignoring them is enough; but our feeling is that things will be a little trickier). These and other related questions will, we hope, provide fruitful ground for continued research in this area.

References

[1] F.C.N. Pereira,*Logic for Natural Language Analysis* SRI Technical Note 275, 1983

[2] F.C.N. Pereira & D.H.D. Warren, *Definite Clause Grammars Compared with Augmented Transition Networks*, Research Report, Dept. of AI, Edinburgh University 1978 (also in Artificial Intelligence, 1980)

[3] A. Colmerauer, *Prolog-II, Manuel de reference et model theorique*, Groupe d'Intelligence Artificielle, Universite Aix-Marseille, 1982

[4] M. Carlsson, *An Implementation of "dif" and "freeze" in the WAM*, SICS Research Report, 1986

[5] S.F. Pilato & R. Berwick *Reversible Automata and Induction of the English Auxiliary System*, Proc. 23rd ACL, Chicago, 1985

[6] R.M. Wharton, *Grammar Enumeration and Inference*, Information and Control, Vol 33, 253-272, 1977

[7] J.R. Andersson, *A Theory of Language Learning Based on General Learning Principles*, Proc. 7th IJCAI, Vancouver, 1981

[8] R.C. Berwick, *Learning Word Meanings From Examples* IJCAI 1983

[9] C. Pollard & I. Sag *Information-Based Syntax and Semantics, Vol. 1*,CSLI 1987

We enclose two appendices. The first shows some sample runs; the second, the grammar used in the examples.

```
Appendix 1

SICStus V0.5 - July 31, 1987
Copyright (C) 1987,
Swedish Institute of  Computer Science.
All   rights reserved.
| ?- ['start.pl'].
[consulting /khons/asa/learning/start.pl..]
[compiling /khons/asa/learning/xgproc.pl...]
[xgproc    compiled   in   14480  msec.]
[consulting /khons/asa/learning/xgrun.pl...]
[xgrun reconsulted  in 159 msec.]
[consulting /khons/asa/learning/utilities.pl.]
[utilities.pl reconsulted in 1360 msec.]
[compiling /khons/asa/learning/prettyprint.pl.]
[prettyprint.pl  compiled in 4680 msec.]
[consulting /khons/asa/learning/top.pl...]
[top.pl reconsulted  in 5920 msec.]
[consulting /khons/asa/learning/sent.pl...]
[sent.pl reconsulted in 2340 msec.]

** Grammar from file grammar.pl : 0 words **

[consulting /khons/asa/learning/read-file.pl.]
[read-file.pl reconsulted in 1420 msec.]
[start.pl consulted in 32100 msec.]

%-----------------------------------------
% A  simple   test with  six  sentences.
%-----------------------------------------

yes
| ?- test_group(5).

Order before sorting: [1,26,2,3,4,5]
Order after sorting: [1,2,26,3,4,5]

%-----------------------------------------
% The format of each line is:
% Sentence number (in test sentence),
% sentence,number of lexicons left.
%-----------------------------------------

1.  the cat saw the dog 8
2.  the dog saw a cat 2
26. that man saw the dog 3
3.  a man saw the nice dog 2
4.  the nice dog likes the man 2
5.  the man likes the dog that the cat saw 1

Run time = 13420. Compiling statistics ...

%-----------------------------------------
% The system asks the user which  of the
% alternate  lexicons is the correct one.
% Here there is only one possibility left.
%-----------------------------------------

a: det
cat: noun(_48268)
dog: noun(_48270)
likes: verb(trans)
man: noun(_48273)
nice: adj
saw: verb(trans)
that: det rel_pro
 the: det

 Is this correct? yes.

 No mistakes
 yes

 %-----------------------------------------
 % A rather more complicated example.
 %-----------------------------------------
```

```
| ?- test_group(0).

Order before sorting:
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,
 17,18,19,20,21,22,23,24,25,26,27,28,29,
 30,31,32,33,34,35,36,37,38,39,40,41,42,43]

Order after sorting:
[1,2,3,4,5,26,27,13,14,6,15,39,11,9,19,18,
 21,20,10,12,17,7,23,33,16,8,22,28,29,30,
 31,32,25,24,35,38,34,36,37,40,41,42,43]

1.  the cat saw the dog 8
2.  the  dog saw a cat 2
3.  a man saw  the nice dog  2
4.  the nice dog likes the man 2
5.  the man likes the dog that the cat saw 1
26. that man saw the dog 1
27. the man has a cat 1
13. the dog belongs to the  man 4
14. the dog likes most men 8
6.  most men like the dog 4
15. the men like john 4
39. the man hoped that john likes the dog 24
11. the dog hoped that the man
                        read the newspaper 16
9.  the man read the newspaper 16
19. john has read the newspaper today  16
18. john read the newspaper today 16
21. the man read the newspaper before
                        john saw the cat 16
20. john has not read the newspaper 16
10. the dog brought the man the newspaper 16
12. john threw the newspaper to the dog 12
17. john threw the newspaper on the table 12
7.  the dog sat on the table 20
23. the cat sat on the car  20
33. john saw a glass on the table 16
16. the dog sat with john 8
8.  the table belongs to the man who
                                owns the dog 8
22. the man who owns the cat drives the car 8
28. the man who has a cat has no dog 8
29. the cat ate a fish  8
30. john ate the beans 8
31. the man ate a can of beans 16
32. the man brought the cat a can
                                   of catfood 16
25. the man can drive the car 72
24. the dog can not  drive the car 16
35. the man drank the whisky 16
38. john hoped the dog drank the water 4
34. john drank a glass of water 2
36. john poured the water on the cat 2
37. john poured a can of water on the cat 2
40. mary knows that john owns a dog 2
41. john believes that mary drives a car  2
42. mary believes john knows that peter
                                 has a cat 4
43. peter can not believe that mary
                             ate the fish 2


Run time  = 949120. Compiling statistics
...
%-----------------------------------------------
% This is the first lexicon of two. The
% divergences are summarized by the system
% further down. Note that "can" and "has"
% are correctely assigned to two different
% classes, and "that" to three.
%-----------------------------------------------
% Nouns can be classified as either
% "count" or "measure". Most of them
% could be either, but nouns occuring
% in partative constructions ("can of
% catfood", "glass of whisky") are
% forced to be "measure".
%-----------------------------------------------
a: det
ate: verb(trans)
beans: noun(measure)
before: sub_conj
believe: verb(s_comp)
believes: verb(s_comp)
belongs: verb(intrans)
brought: verb(doubly_trans)
can: noun(_342148) verb(aux)
car: noun(_342150)
cat: noun(_342152)
catfood: noun(measure)
dog: noun(_342155)
drank: verb(trans)
drive: verb(trans)
drives: verb(trans)
fish: noun(_342160)
glass: noun(_342162)
has: verb(trans) verb(aux)
hoped: verb(s_comp)
john: name
knows: verb(s_comp)
like: verb(trans)
likes: verb(trans)
man: noun(_342170)
mary: name
men: noun(_342173)
most: det
newspaper: noun(_342176)
nice: adj
no: det
not: negator
of: partative_marker
on: prep
owns: verb(trans)
peter: name
poured: verb(trans)
read: verb(trans)
sat: verb(intrans)
saw: verb(trans)
table: noun(_342189)
that: rel_pro det comp
the: det
threw: verb(trans)
to: prep
today: adv
water: noun(measure)
whisky: noun(_342197)
who: rel_pro
with: prep

Is this correct? yes.
belongs : 1 mistakes [verb(pobj([to,45]))]
yes
| ?- halt.


user time 983.600000
```

Appendix 2

```
% Here is the grammar to the learning system:

s --> np,vp.

np --> det,np1(_).
np --> name.

np1(Type) --> adjs,
              n(Type),
              optional_pp,
              rel.

adjs --> [].
adjs --> adj,adjs.

vp --> v(Verb),
       lex(Verb,verb(V_type)),
       v_comps(V_type),v_mods.

v_comps(intrans) --> [].
v_comps(trans) --> np.
v_comps(doubly_trans) --> np,np.
v_comps(pobj(Prep)) --> pp(Prep).
v_comps(s_comp) --> comp,s.
v_comps(s_comp) --> s.

v_mods --> pp(Prep).
v_mods --> adv.
v_mods --> sc.
v_mods --> [].

sc --> sub_conj,s.

optional_pp --> pp(_).
optional_pp --> partative_marker,
                np1(measure).
optional_pp --> [].

pp(Prep) --> [Prep],lex(Prep,prep),np.

rel --> [].
rel --> rel_pro,s.

det --> [Word],lex(Word,det).

adv --> [Word],lex(Word,adv).

adj --> [Word],lex(Word,adj).

sub_conj --> [Word],lex(Word,sub_conj).

n(Type) --> [Word],lex(Word,noun(Type)).

name --> [Word],lex(Word,name).

comp --> [Word],lex(Word,comp).

partative_marker --> [Word],
                     lex(Word,partative_marker).

rel_pro ... np --> [Word],lex(Word,rel_pro).

v(Verb) --> [Verb],lex(Verb,verb(_)).
v(Verb) --> aux,[Verb],lex(Verb,verb(_)).

aux --> [Verb],lex(Verb,verb(aux)).
aux --> [Verb,Negator],
        lex(Verb,verb(aux)),
        lex(Negator,negator).
```