GRAMMAR LEARNING USING INDUCTIVE LOGIC PROGRAMMING

Stephen Pulman
Oxford University
Somerville College
Oxford OX2 6HD
`Stephen.Pulman@somerville.ox.ac.uk`

James Cussens
Dept of Computer Science
York University
YO10 5DD
`jc@cs.york.ac.uk`

January 2001

Abstract

This paper gives a brief introduction to a particular machine learning method known as inductive logic programming. It is argued that this method, unlike many current statistically based machine learning methods, implies a view of grammar learning that bears close affinity to the views linguists have of the 'logical problem of language acquisition'.

Two experiments in grammar learning using this technique are described, using a unification grammar formalism, and positive-only data.

What is Inductive Logic Programming?

Inductive Logic Programming [MDR94] is a machine learning technique that builds logical theories (here,(full) first order logic) to explain observations. 'Explain' here means that it is possible to deduce the evidence from the axioms of the theory (and not be able to deduce negative evidence). ILP is best introduced via the following schema, and a consequent derivation:

1. Background & Hypothesis $\models$ Evidence

We do not assume a tabula rasa: for reasons that every linguist will be familiar with, it is necessary to assume a fairly rich set of background assumptions to constrain the space of possible hypotheses. Given this background, and the evidence, the task is to come up with a hypothesis such that when it is conjoined with the background, the evidence can be deduced from it. Each of the components in the above schema is represented as a set of logical statements.

Notice that schema 1 is logically equivalent to 2, since if P $\models$ Q then P $\rightarrow$ Q (the deduction theorem), and P $\rightarrow$ Q $\equiv$ $\neg$Q $\rightarrow$ $\neg$P (contraposition):

2. Background & $\overline{\text{Evidence}}$ $\models$ $\overline{\text{Hypothesis}}$

where the overline indicates negation. Since Background & $\overline{\text{Evidence}}$ is by hypothesis, consistent, it will be the case by Herbrand's theorem[1] (provided that we restrict the form of H and E) that there is some finite set of ground clauses that are true in every model of that expression. Step 3 of the derivation is:

3. Find set of clauses $\overline{\text{C}}$ true in every model of:
   Background & $\overline{\text{Evidence}}$.

Notice that we represent this set of clauses as a negation, to make succeeding steps tidier. Since this set of clauses is true in every model of Background & $\overline{\text{Evidence}}$, then the following step of the derivation holds:

4. Background & $\overline{\text{Evidence}} \models \overline{\text{C}} \models \overline{\text{Hypothesis}}$

Note that $\overline{\text{Hypothesis}}$ will be a subset of $\overline{\text{C}}$.

The remaining two steps of the derivation follow simply:

5. $\overline{\text{C}} \models \overline{\text{Hypothesis}}$

6. Hypothesis $\models$ C

From step 6 we can now 'invert entailment' to work out candidate hypotheses. Clearly C is one such candidate; generalisations of C constitute others. [Mug95] gives an algorithm for enumerating likely candidates, which can then be subject to various preference measures (simplicity, coverage etc.)

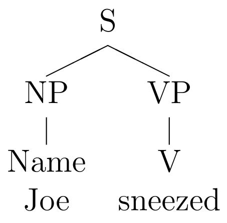
Chart Parsing

Chart parsing [Win83] is a well known technique for finding all possible parses of a sentence with respect to a particular grammar: usually one with a context-free backbone, although richer formalisms can also be accommodated. Constituents are represented as predicates relating positions ('vertices') between words in the input sentence: thus a parsed sentence like 'Joe sneezed' could be represented as:

Id	Constituent	From	To	Containing
c1	Name	0	1	Joe
c2	V	1	2	sneezed
c3	NP	0	1	c1
c4	VP	1	2	c2
c5	S	0	2	c3, c4

The equivalent tree representation would be:

[1]Herbrand's theorem states that a formula has a model iff it has a Herbrand model, where this is artifically constructed from the constants occurring in the formula itself. In the case where there are no functions in the formula, the model will be finite. See e.g. [CL73], Chapter 4.

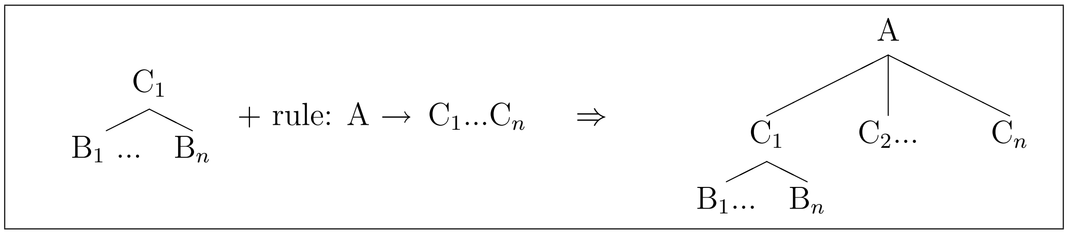


The advantage of a representation like this is that ambiguity can be economically represented, as can incomplete constituents, by the addition of an extra column indicating which components are still needed.

Abstracting away from details of data structures and control regimes we can represent chart parsing as a deductive operation. The 'rules of inference' can be represented as follows, where the first line represents the required input premises, and the second line represents the conclusion:

$$\text{Propose: } \boxed{\frac{\text{C}_1 \text{ from X to Y, and Rule A} \rightarrow \text{C}_1\ \text{C}_2...\text{C}_n}{\text{A} \rightarrow \text{C}_1\bullet\text{C}_2...\text{C}_n \text{ from X to Y}}}$$

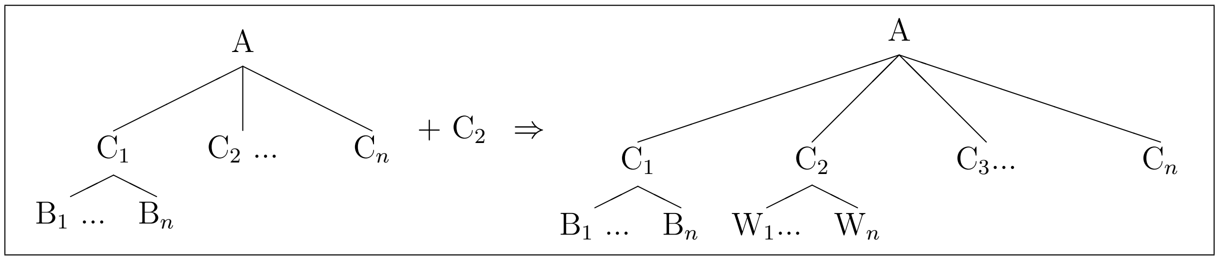
Propose is the rule that hypothesises a new constituent based on the existence of its leftmost daughter. It says: if there is a constituent C_1 from position X to position Y, and there is a grammar rule which builds an A from a series of constituents beginning with a C_1, then begin a new constituent, an incomplete A, consisting just of the recognised C_1, but expecting to find the remainder of the constituents. The notation A $\rightarrow$ $C_1\bullet C_2...C_n$ is to be read as 'something which will be an A when we have found $C_2...C_n$'. We can picture this as:



Other formulations of Propose are possible, reflecting different parsing strategies: e.g. right to left (the version above is left to right), head-driven etc.

The other main rule is 'Combine' which combines an existing incomplete constituent with an existing complete one.

$$\text{Combine: } \boxed{\frac{\text{A} \rightarrow \text{C}_1...\bullet\text{C}_i...\text{C}_m \text{ from X to Y, and C}_i \text{ from Y to Z}}{\text{A} \rightarrow \text{C}_1...\text{C}_i\bullet\text{C}_{i+1}...\text{C}_m \text{ from X to Z}}}$$



If we initialise by putting in constituents for each lexical entry for each word, and apply the above two rules repeatedly in any order as many times as possible, then we will find all the parses for the input sentence according to the given grammar. Even if there is no complete parse for the sentence, we will find all the complete subconstituents, and as many incomplete subconstituents as can be recognised given the current formulation of Propose.

The deductive formulation of chart parsing ([SSP95]) makes clear that we are reasoning from an initial set of axioms (lexical entries for a sequence of words, and some grammar rules) to a particular conclusion (that the sequence of words is a sentence). Different derivations of this conclusion represent alternative syntactic structures for the sentence.

Hypothesising missing rules

The basis of the grammar learning algorithm is as follows. We assume an initial small grammar which will correctly parse some sentences. We then try to parse a corpus of sentences which may contain constructs not covered in the original grammar. The chart parsing algorithm gives us the constituents of the sentence that can be analysed. Using the information in the grammar, and the record of constituents in the chart, we then try to hypothesise which rules could be added to obtain a complete parse.

We can formulate this process as a deductive one. The first rule below propagates 'needs', i.e. hypotheses about what constituents are missing.

$$\text{Needs: } \boxed{\frac{\text{rule: A} \rightarrow \text{ B C, X}\langle\text{Y}\langle\text{Z, need A from X to Z, got B from X to Y}}{\text{need C from Y to Z}}}$$

We start off the process of propagating needs (in the simplest case) by assuming that the sentence in question is grammatical, and that we therefore need an S from the initial to the final vertex. Applying the Needs rule will tell us what constituents we are lacking to make a completely parsed sentence.

The second rules uses the needs to make hypotheses about possible missing rules:

Hypothesise:

$$\boxed{\frac{\text{rule: A} \rightarrow \text{ B C, need A from X to Z, got B from X to Y, got } \mathrm{D}_i\text{...}\mathrm{D}_n \text{ from Y to Z}}{\text{hypothesise C} \rightarrow \mathrm{D}_i\text{...}\mathrm{D}_n}}$$

Note that both Needs and Hypothesise actually should be formulated in a more general way to account for the possibility of rules with varying numbers of daughters, and for the varying positions of already parsed subconstituents with respect to the hypothesised needs and rules.

We can illustrate the operation of Need and Hypothesise with respect to the following simple grammar:

Example

S	$\rightarrow$	NP VP
VP	$\rightarrow$	V
V	$\rightarrow$	snores
V	$\rightarrow$	sees
NP	$\rightarrow$	John, Mary

This grammar contains a transitive verb but has no rule for the corresponding verb phrase. After trying to parse 'John sees Mary' we will have a chart that contains (among others) the following complete and incomplete constituents, which will give rise to the needs and hypotheses indicated:

```
           S                    Need S from 1 to 4...
          / \
         /   VP                 so need VP from 2 to 4.
        /                       Got V from 2 to 3,
      NP      V     NP          got NP from 3 to 4,
      |       |     |           so hypothesise (1) VP -> V NP
   1 John 2 sees 3 Mary 4
```

Note that there will be several other logically possible hypothesised rules arising from the application of Needs and Hypothesise, including:

(2) S $\rightarrow$ NP V NP
(3) S $\rightarrow$ NP VP NP, etc.

Hypothesis 2 builds a sentence without a VP constituent, and hypothesis 3 builds a sentence with an intransitive VP followed by an object. Nothing in what we have done so far says that these are in some sense unlikely candidates compared to hypothesis 1.

Can we cast what we have done so far within the ILP schema? Our evidence is the observation that some sequence of words forms a sentence. Our background is the existing grammar, and the subconstituents that have been found in the chart. To arrive at our hypotheses, we are essentially reasoning backwards (inverting entailment) from the evidence to find candidate rules such that if we added them to the grammar, the conclusion that the sequence of words constituted a sentence would follow.

So far, any logically possible hypothesis will count as a valid one. As every linguist knows, this is too unconstraining a position. We need to enrich our background knowledge with a notion of a 'humanly possible rule' so as to favour natural candidates and eliminate logically possible but linguistically impossible ones. In order to do this we need a precise definition of the linguistic formalism in which our grammars are couched. Recent theories of grammar within the Chomskyan tradition unfortunately do not provide definitions - or even examples - at the level of formal detail necessary for computational implementation. Instead we turn to a simple form of unification grammar [Shi86, Pul96] which has proved itself rich enough to serve as the basis for large scale grammatical descriptions in its own right, as well as serving as a target language for the compilation of many current feature based formalisms like HPSG ([CP95]).

Meta-grammar of rules

In our experiments syntax rules consist of a mother category and zero or more daughter categories. A category consists of a label like S, NP, etc. with a set of feature-value equations. Values can be atomic, boolean combinations of atomic values, categories, lists of categories, or variables. A typical rule in this formalism is:

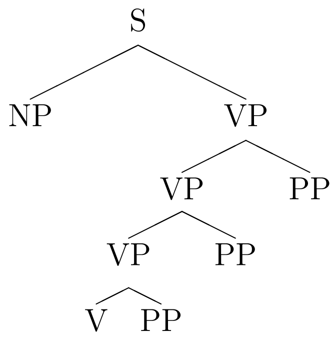
```
s_np_vp  s:[gaps=A,mor=B,type=C,inv=n] ==>
             [np:[gaps=[ng:[],ng:[]],mor=B,type=C,case=subj],
              vp:[gaps=A,mor=B,aux=_]].
```

This rule says, roughly speaking, that a non-inverted sentence (of various types: main, relative etc) consists of an NP and VP, which must agree in their relevant morphological features ('mor') and the sentence node also inherits these values: 'B' is a variable which must have the same value wherever it occurs. The value of 'mor' is a boolean combination of number, person, and verbal features. The type of the sentence is determined by the type of the subject noun phrase (wh, relative, normal, etc.) The 'gaps' feature implements movement via a threading mechanism ([Per81]) which has the same ancestor as the 'slash' feature of GPSG ([GKPS85] and HPSG ([PS87]). The subject NP must not contain a gap.

A linguistically possible rule in this formalism must obey various conventions. We assume rules have no more than three daughters. We axiomatise a 'head' relation: thus verbs are the heads of verb phrases, and verb phrases are the heads of sentences, and so on. We prefer rules that conform to something like X-bar conventions. Various features like morphology are constrained by a version of a head feature convention. Gap features are rather heavily constrained, since their logic is very complex ([CP00a]). 'Useless' rules are declared illegal: a useless rule is one that could lead to a non-terminating unary tree derivation: e.g A $\rightarrow$ B where we already have B $\rightarrow$ A.

These above-mentioned extra components of our background knowledge about natural or possible rules can be used to filter out impossible hypothesised rules and to rank the remaining ones in order of preference.

However, we are still not in a position to produce completely natural rule candidates. Recall that our Needs and Hypothesise rules act only on a single sentence. This has two disadvantages: firstly, since the partially parsed sentence will have information from particular lexical entries in it, the hypotheses produced will be too specific, containing information about agreement or other properties that are dependent on particular lexical items and not an essential part of the the rule that is being learned. Secondly, and this is a rather more subtle and possibly serious point, we will not produce sensible hypotheses for sentences that would require multiple recursive applications of the rule(s) that are being hypothesised. To illustrate this latter point, imagine that we have a sentence NP V PP PP PP which should have the structure illustrated:



We have the partial grammar S $\rightarrow$ NP VP, and VP $\rightarrow$ V PP. A linguistically natural rule to produce the desired parse would be VP $\rightarrow$ VP PP, but this will not be hypothesised by our procedure: instead, we will postulate rules like VP $\rightarrow$ VP PP PP, or VP $\rightarrow$ V PP PP PP.

To attempt to overcome both of these problems we run our learning algorithm over a corpus of sentences twice. The first time we produce candidate rules, filtered and ranked according to the criteria above. Then we carry out two further operations: firstly, we collect together candidates that have the same phrasal skeleton, and produce their 'least general generalisation'. This is another simple form of inverting entailment: we are producing from sentence-specific candidates a more general candidate which implies all its more specalised versions. The aim here is to be able to abstract away from information specific to particular words and which varies across different instances of what our algorithm regards as constituents built by the same rule. We find the 'least general' generalisation because we want to retain information that stays constant across the candidates.

Secondly, we reparse the corpus having added the hypothesised rules to the grammar, and rank the hypotheses according to how many previously unparsed sentences they are used to parse successfully. In order to solve the second problem referred to above, we have to hope that any reasonable sized corpus would contain examples which required only one application of a missing rule and that these might be more numerous than those that require more than one. Under these circumstances we ought to find the the correct rule is hypothesised on the basis of the examples requiring only one application, and this hypothesis should become highly valued because it will occur multiple times in the analyses of the more complex examples. However, the inaccurate multiple-application hypotheses will only apply once to each such complex case and thus should be ranked lower.

Some preliminary experiments.

In order to test the feasibility of this method we conducted an experiment [CP00b, CP00a]. We took an existing unification grammar, originally developed with the approximate coverage of the fragment of English described in Montague's PTQ fragment ([Mon74]), and generated sentences of various lengths randomly with this grammar to form a corpus of several hundred sentences. Since the grammar contained no selectional restriction mechanism these sentences were frequently strange, but that that was not important for the task:

```
[a,heavy,manual,wont,have,continued].
[slowly,the,person,with,nlpcom,starts].
[the,smooth,new,computers,dropped,under,the,things].
```

Next we removed selected rules from the grammar, one at a time, and using the generated corpus applied the procedure described above to try to hypothesise candidates to replace the missing rules that would enable all of the sentences to be parsed.

Here are some examples of rules removed and learned:

Removed: VP → VP Modifier (ran quickly,ran in the park etc.)

```
vp:[gaps=[A,B],mor=C,aux=n]  ==>
                    [vp:[gaps=[A,D],mor=C,aux=n],
                     mod:[gaps=[D,B],of=or(s,vp),type=_]].
```

Learned:

```
vp:[gaps=[_286,ng:[]],mor=or(inf,pl),aux=n] ==>
            [vp:[gaps=[_286,_270],mor=or(inf,pl),aux=n],
             mod:[gaps=[_270,ng:[]],of=or(nom,vp),type=n]]
```

The rule learned here is too specialised, since it will not pass on a gap, only accepting VPs with no gap, or where a gap is found, not VPs which are daughters of other VPs containing a gap elsewhere. Similarly, the agreement features are too refined, presumably reflecting the nature of the corpus. The 'of' feature which specifies the kind of modifier that is possible has also been incorrectly learned: the rule would allow an 'of NP' prepositional phrase as a VP modifier, and would not allow sentential adverbs.

Removed: Nom → Nom Modifier (man in a car etc. Nom=N')

```
nom:[mor=A]  ==>
    [nom:[mor=A],
     mod:[gaps=[ng:[],ng:[]],of=nom,type=or(n,q)]].
```

Learned:

```
nom:[mor=or(pl,s3)]  ==>
       [nom:[mor=or(pl,s3)],
        mod:[gaps=[_339,_339],of=or(nom,vp),type=or(n,q)]]
```

The rule for postnominal modifiers is learned quite successfully: in one respect it is better than the original, for that would allow a first or second person singular nom, if there was such a thing, to be postmodified, whereas the learned form requires the nom to be 3rd singular, or plural.

Removed: VP → V NP (transitive verb phrases)

```
vp:[gaps=A,mor=B,aux=C]  ==>
    [v:[mor=B,aux=C,inv=n,subc=[np:[gaps=_,mor=_,type=_,case=_]]],
     np:[gaps=A,mor=_,type=or(n,q),case=nonsubj]].
```

Learned:

```
vp:[gaps=[_418,_420],mor=or(inf,or(ing,s3)),aux=n]==>
    [v:[mor=or(inf,or(ing,s3)),aux=n,inv=n,
          subc=[np:[gaps=_,mor=_,type=or(n,q),case=nonsubj]]],
     np:[gaps=[_418,_420],mor=or(pl,s3),type=n,case=_]]
```

This rule gets the gap threading correct, but is not general enough in some of the features, again presumably reflecting the fact that the corpus did not contain examples with other feature specifications. Nevertheless, we feel that all three examples are quite impressively close to correct given that only a few dozen relevant examples were contained in the corpus.

In a second experiment we concocted a mini-corpus containing examples of two constructions not covered in the original grammar:

```
[smith,owns,a,computer,company].
[jones,read,some,client,company,reports].
[jones,owned,a,big,telephone,machine].
[computer,machines,stopped].
[no,car,telephone,computer,failed].
[the,telephone,cars,have,stopped].
[computer,machines,are,stopping].
[jones,owns,all,car,telephones].
[computers,fail].
[jones,likes,telephones].
```

Compound nominals ‘client company reports’ are not covered, and nor are bare plural NPs ‘Computers fail’.

After running the algorithm the most highly valued candidate rules (rewritten in a more readable format) are:

```
r217 nom:[mor=or(pl,s3),mor=X] ===>
   nom:[mor=s3]
   nom:[mor=X]

r810 np:[gaps=[ng:[],ng:[]],mor=pl,type=or(n,q),case=X] ===>
   nom:[mor=pl]
```

These are pretty much the rules that we would have written by hand for these cases.

Alternatives that were not so highly valued included things like

NP $\rightarrow$ Nom Nom
NP $\rightarrow$ Det Nom Nom
S $\rightarrow$ Nom Nom VP
NP $\rightarrow$ NP Nom

which have much less intuitive plausibility.

Conclusion

We have demonstrated that the inductive chart parsing approach is capable of generating natural rule hypotheses. However, we do not yet have a fully automatic unsupervised learning method. In the current state of the system it is more plausibly seen as a tool for helping with rapid grammar development, suggesting first-cut hypotheses which the linguist can then test and refine further by hand. Our next step will be to try out the system in this mode by adapting the current small grammar to some more realistic corpus, such as one of the well known air travel inquiry domains. If it proves possible to get the grammar to an acceptable degree of coverage with less effort than the traditional hand-crafted (although corpus supported) methods, then we will have constructed a useful practical tool.

References

[CL73] C-L. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

[CP95] B. Carpenter and G. Penn. Compiling typed attribute-value logic grammars. In Bunt. H and Tomita. M, editors, *Current Issues in Parsing Technologies*. Kluwer Academic Press, 1995.

[CP00a] J. Cussens and S. G. Pulman. Experiments in inductive chart parsing. In J. Cussens and S. Džerowski, editors, *Learning Language in Logic*, number 1925 in Lecture Notes in Computer Science, pages 143–156. Springer, 2000.

[CP00b] James Cussens and Stephen Pulman. Incorporating linguistics constraints into inductive logic programming. In *Proceedings of CoNLL2000 and LLL2000*, pages 184–193, Lisbon, September 2000. ACL.

[GKPS85] G. Gazdar, E. Klein, G. Pullum, and I. Sag. *Generalised Phrase Structure Grammar*. Oxford: Basil Blackwell, 1985.

[MDR94] S. Muggleton and L De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[Mon74] Richard Montague. The proper treatment of quantification in English. In Thomason R., editor, *Formal Philosophy*. Yale University Press, New York, 1974.

[Mug95] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[Per81] Fernando C.N. Pereira. Extraposition grammars. *Computational Linguistics*, 7:243–256, 1981.

[PS87] C. Pollard and I Sag. *Information-based Syntax and Semantics: Volume 1: Fundamentals*. CSLI Lecture Notes Number 13. Chicago University Press, 1987.

[Pul96] Stephen G. Pulman. Unification encodings of grammatical notations. *Computational Linguistics*, 22/3:295–328, 1996.

[Shi86] S. Shieber. *An Introduction to Unification Approaches to Grammar*. CSLI Lecture Notes Number 4. Chicago University Press, 1986.

[SSP95] S. Shieber, Y. Schabes, and F.C.N . Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 1995.

[Win83] T. Winograd. *Language as a Cognitive Process*. Addison Wesley, 1983.