A Learning Natural Language Parser

Duncan McKee
Kurt Krebsbach
Department of Computer Science
Lawrence University
Appleton, WI 54911
mckeed@lawrence.edu

Abstract

A natural language parser is described that analyzes the syntactic structure of an input sentence in relation to a specified grammar and generates all possible syntax trees of the sentence, along with estimates of the probability of each being the correct parse. The grammar used is based on X-bar theory, and the parsing algorithm is a chart parse – a top-down parser which uses dynamic programming for efficiency in cases where the grammar leads to ambiguities. The parser has a database of the frequency of application of each syntax rule in the grammar as well as a lexicon of known words and their lexical categories and frequency of use in each category. These are used in a probabilistic context-free grammar model to yield the likelihood judgments of the candidate parses and are updated by user feedback, leading to more accurate subsequent estimations.

Introduction

Approaches to natural language understanding have usually modeled language as having levels of structure corresponding to traditional levels of linguistic analysis. Thus, speech input undergoes acoustic and phonetic analysis, then phonological, morphological, syntactic, and finally semantic analysis. Each level in this schema imposes a layer of structure on the input with the goal of eventually arriving at a logical representation of the information it imparts.

Natural language does not bend conveniently to these structural analyses, unfortunately. Ambiguity is prevalent at every level of analysis, and some ambiguity cannot even be resolved by a native speaker. When humans process language, structure and meaning are inferred using contextual information. The first problem this poses to computers is that, given a string of words, there are no hard and fast rules to govern the syntactic structure of that string. Information from the semantic level of analysis is required to disambiguate many grammatical ambiguities. Since, in other respects, semantic analysis takes place *after* syntactic analysis, this poses a problem for the level-by-level strategy, necessitating backtracking.

Many recent trends in linguistics and natural language processing move away from the multi-level, structure-heavy approach in favor of methods based on statistical analysis and positional relationships. These ideas have been very productive, but there is still room for structural approaches. By combining them with the statistical and probabilistic strategies, structural analyses can be made flexible enough to deal with ambiguities. This paper examines an example of how this can be accomplished, applied specifically to the syntactic level of analysis: parsing.

Parsing is the logical first step to computer understanding of natural language text. Most approaches to determining the meaning of a natural language statement first analyze the syntactical structure of the statement. The syntactical analysis of a sentence contains the information of what roles the words in the sentence are playing as well as the phrase structure of the sentence – that is, what parts constitute the subject and predicate of the sentence and what the adjectives, adverbs, and prepositional phrases refer to.

Unlike programming languages, which have relatively simple formal grammars, natural languages have not been successfully formalized, so modifications must be made to adapt traditional parsing algorithms to them. One major challenge is that natural language grammars frequently generate multiple possible parses for a single sentence. These are ambiguities that humans resolve through the application of a wealth of semantic and contextual knowledge that computers are not yet able to replicate.

This project's goal was to develop a system to parse grammatical English sentences into phrase structures in a way that deals gracefully with ambiguity. The resulting parser first generates all possible phrase structures using the chart parser algorithm. It then uses a probabilistic grammar to assign each of these possibilities a likelihood measure, which is the estimated probability of that parse being the correct representation of the sentence. These measures are normalized probabilities of the incidence of each result based on the sentences that the system has seen in the past. Thus, as the parser is exposed to more sentences, it learns what are the most common and the least common grammatical constructions in the language, and therefore improves its judgments of ambiguous cases.

Background

X-Bar Grammar

The output of a parser such as the one described here is a parse tree, a construct describing the phrase structure of the input sentence in relation to a specified grammar. The internal nodes of a parse tree represent non-terminal symbols in the grammar and the leaf nodes are the words of the input sentence. The grammar used in this parser is a context-free grammar based on X-bar theory. Although X-bar grammar grew out of the Chomskyan generative grammar tradition that frowns upon probabilistic methods (Russell & Norvig, 2003), it still has many features that are useful in computational analysis. It lends itself well to abstraction, as its rules are at most binary branching and are categorized in terms of a few abstract patterns. It also serves as the base framework for much work on describing syntactic phenomena such as agreement, pronoun reference, and structural transformations (Haegeman, 1994).

```
XP ⟶ Spec  X'

X' ⟶ X  |
      X'  Comp  |  Comp  X'  |
      X'  Adjunct  |  Adjunct  X'

X  ⟶  <word>
```

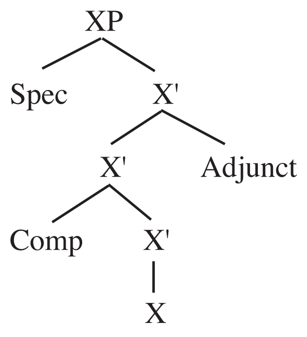


Figure 1: X-bar theory’s categorization of non-terminal symbols means that the syntax rules conform to a set of abstract patterns. On the right is a generic XP schema.

There are a few types of non-terminal symbols in an X-bar grammar. First, there are lexical category symbols, which represent the grammatical functions of words, such as noun, verb, preposition, etc. These are always the parent nodes of the terminal symbols – the words themselves – in the parse tree. For each lexical category symbol X, there is an X-phrase symbol, XP, which represents a phrase headed by a word of category X. The motivation for grouping words in this way arises from the perception that an XP fills the same role in a sentence that a single X fills; often, an entire phrase can be replaced by a single word without changing the essence of the sentence, whereas this cannot be done with groups of consecutive words that do not make up phrases. For example, in the sentence, "The computer with the file crashed," the subject is the noun phrase "the computer with the file" and can be replaced by a pronoun to make the sentence "It crashed."

X-bar theory postulates another type of non-terminal symbol, usually written X' and pronounced 'X-bar'. One or more of these symbols comes between an X node and its corresponding XP node; they provide branching locations for complements and adjuncts to X – words or phrases that support or modify X. Since both of these are optional, X' nodes often have only a single child node.

S ⟶ NP VP

NP ⟶ DET N' | N'

N' ⟶ N | N' PP

VP ⟶ AUX V' | V'

V' ⟶ V | V' NP

PP ⟶ P'

P' ⟶ P | P NP

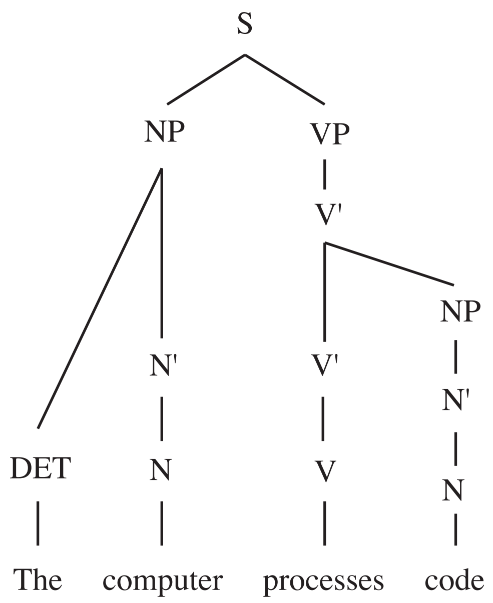


Figure 2: On the left is a small, incomplete X-bar grammar. On the right is a sentence and its parse tree according to this grammar. The root symbol of the sentence, 'S', is not technically a proper X-bar symbol, but it is used here for simplicity.

Chart Parser

The algorithm used to find the parse trees is a top-down chart parser, a search algorithm that uses dynamic programming techniques. Chart parsers are commonly used in natural language processing because they deal efficiently with ambiguous grammars, which can be combinatorially problematic (Russell & Norvig, 2003; Allen, 1995). This algorithm uses a data structure called a chart to store partial parses, which are parse trees of substrings of the input sentence. It builds up the set of all partial parses by extending and combining them as it processes the words of the sentence one at a time. The variant described here is adapted from Russell & Norvig (2003) and is similar to the Earley parser (Earley, 1970).

For an *n*-word sentence, the chart is a multigraph with *n*+1 vertices, where a partial parse of the *i*th to *j*th words would be associated with an edge from vertex *i* to vertex *j*+1. This edge is added to the chart during the processing of the *j*th word by one of three methods: PREDICTOR, SCANNER, and EXTENDER. Figures 3-5 show how these methods would produce edges when parsing the example sentence from Figure 2. When an edge is added for a partial parse tree with a non-terminal symbol X as a leaf node, the PREDICTOR method adds a new edge for each rule in the grammar for X, representing the possible branchings that could extend the tree (Figure 3). For each word *w*, the SCANNER method extends the edges with partial parse trees that need a word at this position of a lexical category that *w* belongs to (Figure 4). When an edge is added for a complete parse of some phrase Y that spans words *i* through *j,* the EXTENDER method combines it with any edges ending at *i* that were lacking a Y subtree; this adds a new edge that spans the combined spans of the edges that produce it (Figure 5). If the grammar admits a parse tree for the entire sentence, the algorithm succeeds by producing an edge from the first vertex to the last vertex that corresponds to a parse tree with the start symbol of the grammar as the root node and the words of the sentence as the only leaf nodes.

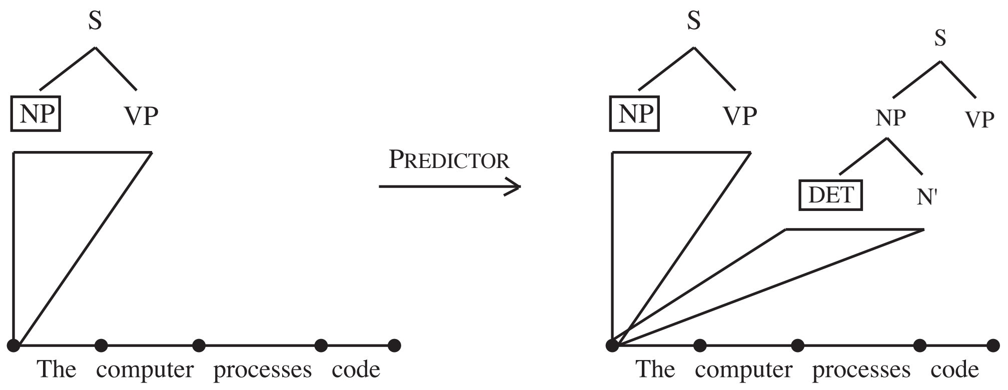


Figure 3: An edge added by the PREDICTOR method. The box indicates the next branch to be extended or specified.

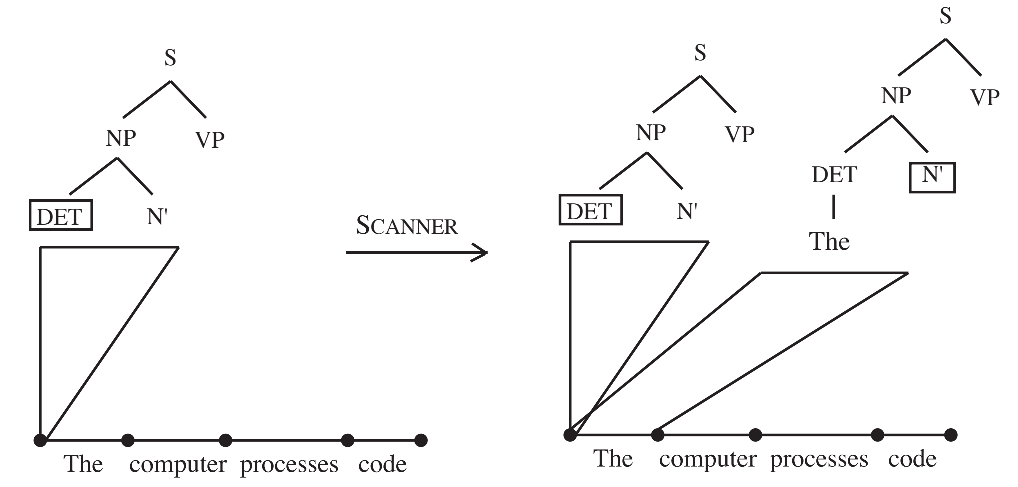


Figure 4: An edge added by the SCANNER method. Note: not all edges that would be in the chart at this point in the algorithm are shown.

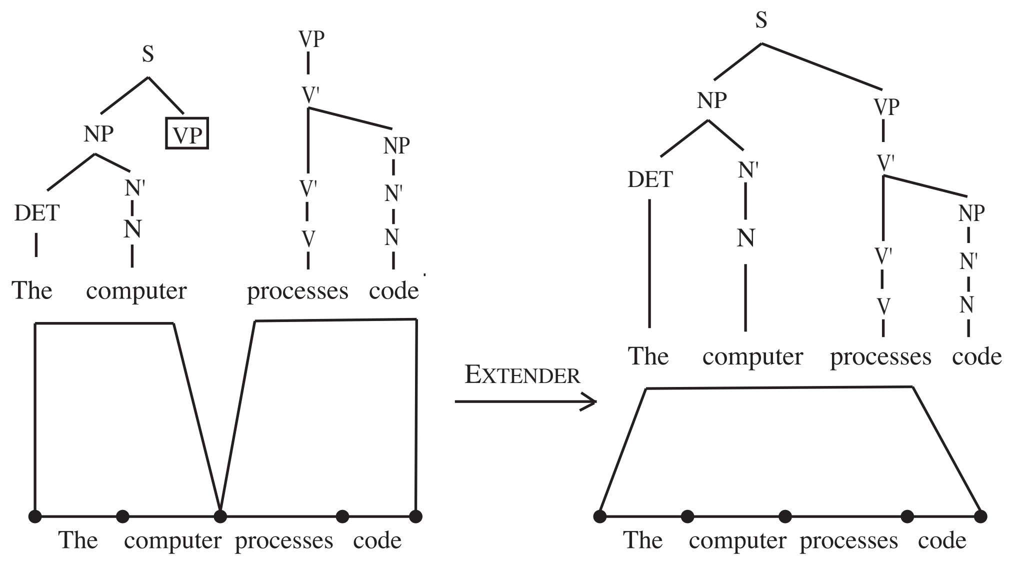


Figure 5: An edge added by the EXTENDER method. Note that the edges and partial parse trees shown on the left remain in the chart and can extend other edges – they are omitted on the right for clarity.

Probabilistic Context-Free Grammar

The statistical learning feature of the system uses a form of probabilistic context-free grammar, a model of syntactic structure that is more conducive to learning and more robust in ambiguous cases (Manning & Shütze, 1999). In a PCFG, every syntax rule – including those that attach words to lexical categories – has a probability associated with it. For each non-terminal symbol X, the rules in the grammar of the form X –> Y Z partition the set of possibilities for the structure of an X component. Thus, the normalization constraint dictates that the probabilities of the rules in this set sum to one.

The probability of a given complete parse tree is simply the product of the individual probabilities of each node in the tree. This probability can be understood as production probability; if the PCFG is used to generate a legal sentence by starting with the symbol 'S' and recursively replacing each non-terminal symbol with one of the relevant rules (chosen randomly using the given probability distribution), the probability of arriving at some sentence is the product of the probabilities of each rule choice.

The Parser

The PCFG technique is used in this parser not for making the parsing itself more efficient or flexible, but rather as a means by which to store and use information about the grammar that is collected over many applications of the parser, and hence provide a more accurate evaluation of future input. The parser is able to produce more than a list of the possible parse trees for an input sentence; it also judges their relative grammatical acceptability. This does not take semantic information into account, as this is solely the syntactic level of analysis. In theory, the output from the parser could be passed on to a semantic processor, which could use the parser's evaluations as a starting point and update the likelihood measures based on semantic information.

The system uses a database for knowledge persistence. When an input sentence is entered into the parser, each word is looked up in a database of all words known to the system and their possible lexical categories (e.g. noun, verb, etc.). For each word/category pair, the lexicon database has a count of how many times the word has been observed in that category. The count of each category is divided by the sum of the counts of all categories for that word – this yields the probability that the word exemplifies that category. This implementation of probability inference is crude but effective.

The chart parser algorithm is then used to generate all the possible parses of the input sentence, using a fully-specified X-bar grammar. This algorithm is efficient in most situations, but, due to the tree structures of the generated parses, there is a possibility of an exponential number of results. Fortunately, such sentences are not common in actual usage.

In addition to word category counts, the occurrence frequency of each grammar rule that is used in the candidate parses is selected from the database, which keeps a count of past appearances of each grammar rule. These counts are converted to probabilities in the same way as are the word/category counts. The number of times a certain nonterminal symbol has been observed to be rewritten by each rule is divided by the sum of the counts of all rewrites of that rule, effectively yielding the probability of that syntax tree node appearing in that environment.

After generating all the probabilities of the components of the parses, they are multiplied as is typical in the use of probabilistic context-free grammars, resulting in occurrence probabilities of each of the possible parse trees. These are then normalized against each other, representing the assumption that the correct parse is among the possible parses generated by the chart parser, and these can be compared as measures of their relative strength as solutions.

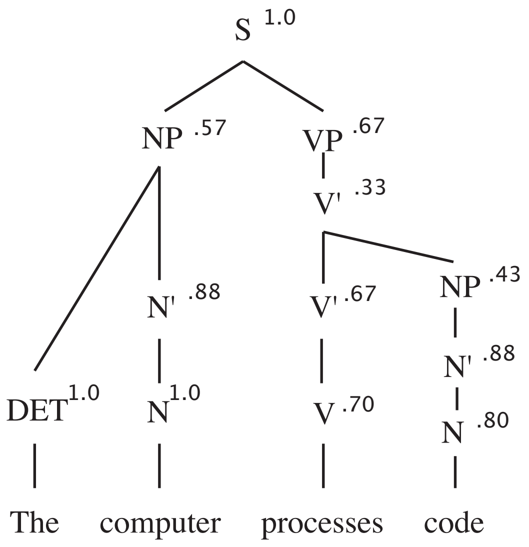
Figure 6 shows an example of the parser's analysis of a sentence with multiple parses. It portrays a hypothetical state of its database after having seen 30 input sentences. Via the ratio of the PCFG probabilities of the parse trees, the parser assigns a 90% probability to the solution on the left and a 10% probability to the solution on the right (under the assumption that one of them is the intended interpretation).

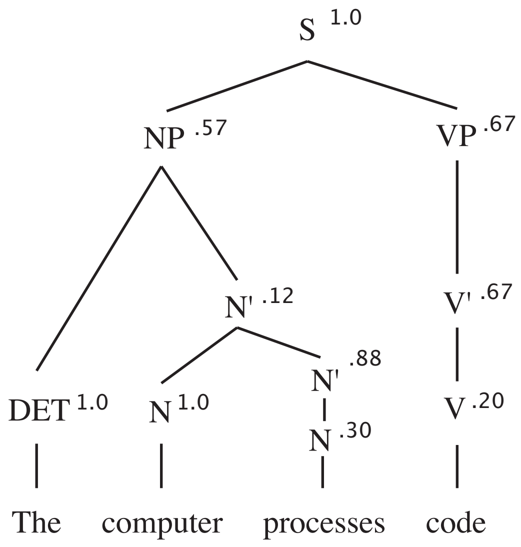
The user can then indicate which result was the intended structure, which will increment the occurrence counts in the database for the rules and lexical category assignments used in that parse. If the same input is fed to the parser a second time, it will favor the correct one slightly more than it did previously. Over time, the system learns what structures are most used in the language, and hence makes better guesses in ambiguous situations.

In practice, this has produced some positive results; the parser is often able to distinguish between the obvious, straightforward parse of a sentence and an anomalous result brought about by a grammar rule that exists to deal with some rare construction. For example, it favors the correct interpretation of the sentence "I can code" over the interpretation that implies "I put code into cans" by a 99.3% to 0.7% margin. Still, there are some instances where the incorrect parse scores higher, usually because determining the correct interpretation requires semantic information.

It is important to note that although the input is only being analyzed on the basis of syntactic structure, the way the system favors some rules over others is, to a degree, based on semantic information learned from human agents via its feedback system. Furthermore, it is being trained only on the dialect and sentence domain of the input it receives, which could make it inappropriate for input from a novel source.

rules	frequencies	probabilities
S $\longrightarrow$ NP VP	30	1.0
NP $\longrightarrow$ DET N' | N'	20 | 15	0.57 | 0.43
N' $\longrightarrow$ N | N N'	35 | 5	0.88 | 0.12
VP $\longrightarrow$ AUX V' | V'	10 | 20	0.33 | 0.67
V' $\longrightarrow$ V | V' NP	30 | 15	0.67 | 0.33





Probability: .88^2 * .67^2 * .80 *.70 * .43 * .33 * .57 = 0.01575
Normalized: 0.907

Probability: .88 * .67^2 * .12 * .20 * .30 * .57 = .0016212
Normalized: 0.093

Figure 6: A rule for compound nouns has been added to the grammar, producing multiple parses for the example sentence. The table above shows a hypothetical state of the database after it has seen 30 input sentences. The nodes in the parse trees are marked with their individual probabilities and the tree probability calculations are shown below.

Conclusion

Without semantic and contextual knowledge, consistently determining the correct syntactic interpretation of a sentence is not possible. However, the strategy of generating multiple candidate interpretations and providing as much information as possible about the syntactic feasibility of each holds the promise of providing an advantageous starting point for semantic and logical analysis without necessitating backtracking to the syntactic analysis stage. In any case, the complete understanding of natural language – while currently an unsolved problem – has inspired much beneficial research.

Acknowledgments

Many thanks to Lawrence University Provost David Burrows for supporting this work through a Lawrence *Enhancing Academic Distinctiveness* grant. This program was written in PLT Scheme and made use of Jay McCarthy's Sqlite package, available at the PLaneT Package Repository (http://plt-scheme.org).

References

Allen, James. 1995. *Natural Language Understanding*. Redwood City, CA: Benjamin/ Cummings.

Earley, J. 1970. "An efficient context-free parsing algorithm", Communications of the Association for Computing Machinery, 13:2:94-102.

Haegeman, Liliane. 1994. *Introduction to Government and Binding Theory*. Oxford, UK: Blackwell.

Manning, C. & Schütze, H. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: The MIT Press.

Russell, S & Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Pearson Education.